3. PORTAS LÓGICAS

Olá, prezado cursista! Bem-vindo ao nosso terceiro capítulo relativo a BCC265!

Você sabe como representar um circuito lógico, propriamente dito, a partir de uma expressão booleana? Nesta unidade, abordaremos a maneira pela qual podemos obter circuitos lógicos usando componentes reais. Uma outra questão que lhe pode ocorrer seria: as propriedades e leis da álgebra booleana serão empregadas aqui neste capítulo? Sim, a álgebra boolena será necessária para que possamos utilizar as portas lógicas – que constituem na abstração mais básica dos sistemas digitais, representando os operadores lógicos.

Saiba que a álgebra digital nos será útil para, por exemplo, realizar simplificações de expressões booleanas. Mas como montar a expressão lógica (booleana) para construir um circuito que atenda a um certo objetivo? Falaremos também sobre como extrair, a partir de uma tabela-verdade, por exemplo, que representa o comportamento do circuito a ser criado, a expressão booleana.

Para que possamos caminhar por esses pontos, iniciaremos com a implementação de circuitos lógicos a partir de expressões booleanas conhecidas. Quando estivermos familiarizados com a forma de implementar circuitos, migrando as expressões para um mundo mais prático, será o momento de extrairmos as expressões a partir de tabelas-verdade. Como a extração de expressões pode implicar em expressões não otimizadas, fecharemos esta unidade falando sobre como otimizar (ou simplificar as expressões) usando a álgebra booleana e, por fim, usando Mapas de Karnaugh. Com essa etapa de simplificação, obteremos circuitos reduzidos, otimizados.

Preparado para colocar as mãos na massa, ou melhor, nos circuitos? Então, vamos lá!

3.1 Obtenção do circuito lógico a partir de expressões booleanas

Você já deve saber como transformar as expressões em diagramas esquemáticos usando as portas lógicas básicas (“NOT”, “AND”, “OR”, “NAND”, “NOR”, “XOR” e “XNOR”), certo? Vamos, contudo, dar uma reforçada nesse ponto, pois é a base para a implementação física dos sistemas lógicos digitais.

3.1.1 Famílias lógicas

Para a implementação física, vamos relembrar da precedência na álgebra booleana, pois

teremos que segui-la para que possamos proceder à interligação das portas lógicas. Dessa forma, as ligações ocorrerão na seguinte ordem (TOCCI, 2018):

- 1º: parênteses;
- 2º negação (operador "NOT");
- 3º operador "AND";
- 4 º operador "OU", "XOR", "XNOR".

Em segundo lugar, é interessante conhecermos quais os componentes que utilizaremos para a montagem de nossos circuitos. Em linhas práticas e gerais, existem dois grandes grupos de circuitos integrados (componentes eletrônicos que encapsulam vários transistores e, no nosso caso, implementam os operadores lógicos): os componentes baseados na tecnologia TTL ("*Transitor-transitor Logic*", ou Lógica de transitor-transitor) e os baseados em CMOS ("*Complementary Metal-Oxide Semiconductors*" – Semicondutor de metal-óxido complementar).

Além dessas duas famílias, podemos citar:

- ECL (*emitter-coupled logic*);
- BiCMOS (*bipolar complementary metal-oxide semicondutor*);
- GaAs (*arseneto de gálio*).

Cada tecnologia tem as suas características básicas, como a velocidade de trabalho, a potência dissipada e a voltagem requerida para seu funcionamento.

Você quer ler?

Em função das características distintas entre TTL e CMOS, os circuitos lógicos não devem misturar essas duas tecnologias, ou seja, utiliza-se apenas TTL ou apenas CMOS. Para saber Porém, caso haja realmente necessidade de "interfacear" circuitos CMOS ↔ TTL, o material artigo de IFSC (2018) mostra como devemos proceder. Disponível em: <https://wiki.ifsc.edu.br/mediawiki/index.php/AULA_16_-_Eletr%C3%B4nica_Digital_1_-_Gradua%C3%A7%C3%A3o>.

Em função de praticidade, utilizaremos a linha TTL, que apresenta os seguintes

parâmetros básicos:

- *Fan-out* (número de portas lógicas que poderemos conectar à saída de outra porta): geralmente 10.
- Tensão de alimentação: 5V.
- Faixas de tensões para a representação dos níveis lógicos:
  - Nível lógico “0”: de 0V a 0.8V.
  - Nível lógico “1”: de 2.4V a 5V.

Outro motivo para se adotar o TTL consiste na sugestão da utilização do *TinkerCad* (https://www.tinkercad.com/) para que possamos testar os circuitos implementados. O ambiente em questão contempla um simulador *online* de circuitos, facilitando seu manuseio em diversas situações e locais. Antes de iniciarmos nossa montagem, é interessante mencionar sobre as “pinagens” dos circuitos integrados que poderão ser utilizados. A figura a seguir ilustra dois circuitos integrados da linha TTL: o 7404 (implementa portas “NOT”) e o 7408 (que contempla portas “AND”).

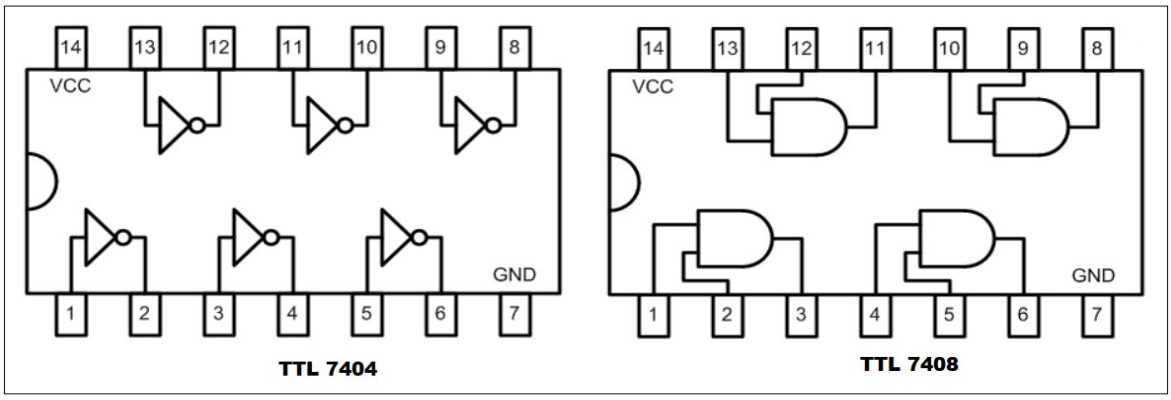


Figura 1 – Diagrama das pinagens dos circuitos integrados TTL modelos 7404 (com seis portas “NOT”) e 7408 (com qautro portas “AND”).
Fonte: Extraída de *datasheets* dos fabricantes, 2020.

Na figura, podemos observar que os circuitos integrados 7404 e 7408 possuem 14 pinos, sendo que o pino 7 deve ser ligado ao terra (GND – *Ground*) e o pino 14 ao terminal positivo da alimentação (VCC). Os pinos são numerados no sentido anti-horário e iniciam sua contagem no lado esquerdo de uma marca (chanfro – componente visto por cima). Observa-se, ainda, que o 7404 possui, encapsuladas, seis portas inversoras, e o 7408, quatro portas “AND”.

Mas, como implementar, de fato, as expressões booleanas utilizando-se portas lógicas?

Conversaremos sobre isso a seguir.

3.1.2 – Implementação de sistemas lógicos digitais a partir de expressões booleanas

Para iniciar nossa trajetória implementando circuitos a partir de expressões booleanas, vamos supor que desejamos construir um circuito cuja expressão booleana hipotética consiste em:

S = A . B + C . (~A + B)

Usando a sequência de precedência, teremos os passos listados a seguir.

- 1º Passo: aplicar a negação na variável "**A**", para depois poder usar a saída da porta "NOT" na parcela "**~A + B**".
- 2º Passo: desenvolver, paralelamente, as parcelas "**A . B**" e "**(~A + B)**".
- 3º Passo: utilizar uma porta "AND" para integrar a variável "**C**" e a saída de "**(~A + B)**".
- 4º Passo: utilizar uma porta "OR", recebendo as saídas da porta "AND" do terceiro passo com a saída da porta "AND" da parcela "**A . B**".

Por meio da aplicação da sequência exposta acima, temos o diagrama esquemático presente na figura a seguir. A figura contém as ligações realizadas no simulador "*TinkerCad*", em que foram utilizados os circuitos integrados tais quais são encontrados no mercado.

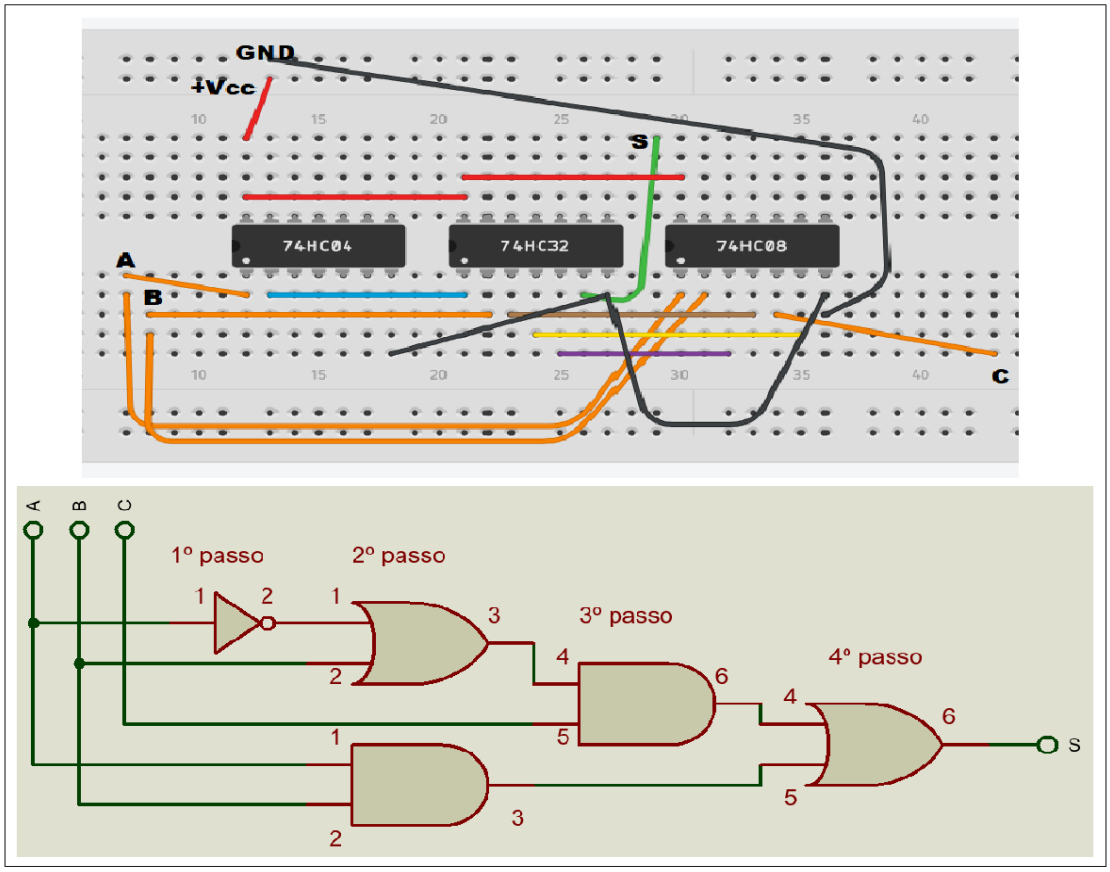


Figura 2 – Diagrama esquemático e ligações físicas (feitas no simulador tinkercad.com) relativos à expressão booleana **`S = A . B + C . (~A + B).`**

Fonte: Elaborada pelo autor, 2020.

Na figura anterior, temos o diagrama esquemático e as ligações físicas relativas à expressão **`S = A . B + C . (~A + B)`**. No diagrama esquemático, os números associados às portas lógicas representam os números dos pinos associados aos circuitos integrados (7404 = portas "NOT"; 7432 = portas "OR"; 7408 = portas "AND"). As colunas (linhas verticais) do *protoboard* (placa para ligações) são barramentos, ou seja, constituem os mesmos pontos para ligações. Os fios em laranja são as entradas e o fio verde, a saída "**`S`**". Os demais fios são responsáveis pelas ligações entre as portas. Por exemplo, o fio azul, ligado no pino 2 do 7404 representa a saída de uma porta "NOT". A outra extremidade do fio azul está conectada ao pino 1 do 7432, ou seja, em uma entrada de uma das portas "OR" encapsuladas nesse circuito integrado.

Em relação à implementação em Verilog, como ficaria a expressão **`S = A . B + C . (~A + B)`**? As codificações a seguir ilustram duas formas de implementação – uma versão baseada apenas em uma abordagem estrutural e utilizando-se abordagem comportamental.

```
//implementação estrutural

module Teste01(a,b,c,s);

   input a,b,c;
   output s;

   assign s = a&b | c&(~a|b);

endmodule // Teste01
```

```
//implementação comportamental

module Teste01(a,b,c,s);

   input a,b,c;
   output s;
   reg    s;

   always @(*)
     begin
       s = a&b | c&(~a|b);
     end

endmodule // Teste01
```

Em nosso encontro virtual comentaremos o que cada linha da codificação representa...

Caso

Um projetista e implementador de sistemas lógicos digitais foi incumbido de realizar um certo projeto. Porém, ele deseja fazer um teste do circuito produzido a fim de verificar se o circuito funciona como definido em relação às questões de tempo e de propagação de sinais. Ele também deseja testar algumas versões distintas do projeto, envolvendo tecnologias distintas de circuitos integrados e também envolvendo uma versão desenvolvida em HDL (*Hardware Description Language* – Linguagem de Descrição de *Hardware*).

Como existem várias alternativas, e para não perder tempo e dinheiro construindo vários circuitos lógicos para que seja tomada uma decisão, ele resolveu simular todas as possibilidades. Com o resultado da simulação, pôde verificar o verdadeiro comportamento das versões implementadas. Como lição desse caso, sugerimos que, independentemente da complexidade e tamanho do circuito a ser produzido, é conveniente realizar simulações prévias do circuito sob desenvolvimento.

Mas, como implementar a parte de *testbench* das codificações acima colocadas? O trecho a seguir (que também será comentado em nosso encontro virtual) contém uma sugestão de implementação.

```
//módulo para simulação - testbench

module top;

   reg t_a, t_b,t_c;

   wire t_s;

   initial
     begin: simul_stop
      #8 $stop; $dumpflush;
     end

   initial
     begin: dump_file
      $dumpfile("teste01.dump");
      $dumpvars(0,t_a,t_b,t_c,t_s);
      $dumpon;
      $display("Time  t_a  t_b  t_c  t_s");
      $monitor(" %0d   %b   %b   %b  %b", $time, t_a, t_b, t_c, t_s);
     end

   initial
     begin: start
      t_a = 0;
      t_b = 0;
      t_c = 0;
     end

   always
     begin: process
      #1 {t_a,t_b,t_c} = {t_a,t_b,t_c} + 1;
     end

   Teste01 t(.a(t_a),.b(t_b),.c(t_c),.s(t_s));

endmodule // top
```

Já que mencionamos a implementação em simuladores, mencionamos aqui uma dica de como produzir os valores das variáveis por meio de um botão. O circuito a seguir ilustra uma das formas que poderá ser usada no circuito; no caso, seriam necessários três circuitos idênticos, cada um produzindo uma das variáveis de entrada.

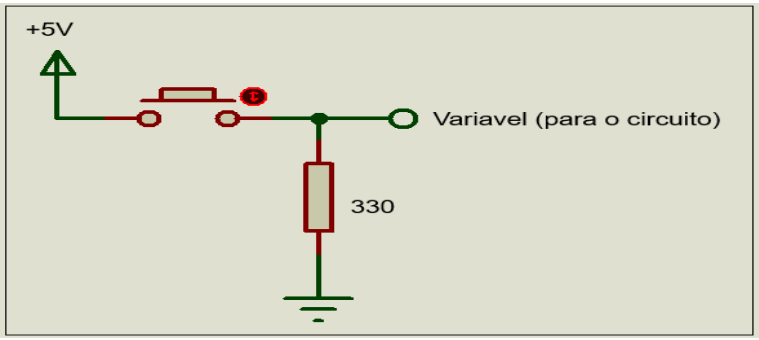


Figura 3 – Forma de inserir um botão para gerar os níveis lógicos “1” ou “0” relativos a uma variável a ser manipulada pelo circuito digital.

Fonte: Elaborada pelo autor, 2020.

Perceba que quando o botão for pressionado, a tensão de +5V é externada do circuito, fazendo com que a variável represente o nível lógico "1". Por outro lado, quando o botão não estiver pressionado, o circuito produzirá o nível lógico "0" em que o referencial de terra é externado por intermédio do resistor de 300 ohms. Tal resistor é denominado como "*pull-down*" (em uma tradução literal, "puxar para baixo") devido à sua derivação do referencial terra (GND – Ground). Caso o botão estivesse ligado diretamente ao "terra" e o resistor ligado ao +5V, teríamos o caso de um resistor de "*pull-up*".

Até aqui, fizemos a implementação de uma expressão booleana que já se encontrava formada. Mas como obter uma expressão booleana? É o que você saberá a seguir.

Vamos praticar?

Para essas sugestões, crie uma conta no *TinkerCad* para que você possa implementar as seguintes expressões booleanas:

a) S = (~(A + B) . (C + D)) . B

b) S = (A + ~(B.C)) + (B . ~(A.C))

3.2 Obtenção da expressão booleana pela interpretação do circuito lógico

Você já deve saber que uma tabela-verdade representa o comportamento de um sistema lógico diante de todas as combinações possíveis das variáveis de entrada envolvidas. Mas, como efetivamente extrair uma expressão booleana a partir da tabela-verdade para que possamos, depois, implementar fisicamente o circuito? Veremos esse ponto a seguir, porém, antes, conversaremos sobre formas padrões de expressões booleanas.

3.2.1 Formas padrões de expressões booleanas

Antes de conversarmos sobre o processo de obtenção da expressão, vamos colocar o conceito de formas padrões: **soma de produtos** e **produto de somas** (IDOETA, 2019).

- **Soma de produtos**: a expressão booleana representada na forma de soma de produtos é constituída pela interconexão, via porta "OR", de parcelas. Cada parcela, denominada como "mintermo", é formada por variáveis interconectadas por portas "AND". Como exemplo, temos: `S = A.B.C + ~A.B.C + A.~B.C`.

- **Produto de somas**: a expressão booleana representada na forma de produto de somas é constituída pela interconexão, via porta "AND", de parcelas. Cada parcela, denominada como "maxtermo", é formada por variáveis interconectadas por portas "OR". Como exemplo, temos: `S = (A+B+C) . (~A+B+C) . (A+~B+C)`.

Quando temos todas as variáveis presentes em todas as parcelas (mesmo que elas apareçam de forma complementadas), dizemos que a expressão está em sua forma canônica. Expressões derivadas diretamente de tabelas-verdade (antes de qualquer manipulação algébrica) sempre estarão em sua forma canônica. Exemplos e contraexemplos de formas canônicas:

- Exemplos de expressões canônicas:
  - `S = A.B.C + ~A.B.C + ~A.B.~C`
  - `S = (A+B+C) . (~A+~B+C) . (A+~B+C)`

- Contraexemplos de expressões canônicas:
  - `S = A.B.C + ~A.B.C + ~A.~C`
  - `S = (A+B+C) . (~A+C) . (A+~B+C)`

Mas como obter a expressão a partir da tabela-verdade? Dialogaremos, agora, sobre essa questão.

3.2.2 Extração da expressão booleana a partir da tabela-verdade

Conversamos sobre tabelas-verdade e sobre formas padrões de representação das expressões booleanas. Mas, como juntar esses dois assuntos para que possamos extrair expressões a partir de tabelas-verdade?

Cada linha da tabela-verdade gera um "mintermo" ou um "maxtermo", conforme ilustra a tabela a seguir, em que temos uma tabela-verdade de duas variáveis.

A	B	mintermos	maxtermos
0	0	~A . ~B	A + B
0	1	~A . B	A + ~B
1	0	A . ~B	~A + B
1	1	A . B	~A + ~B

Tabela 1 – Mintermos e maxtermos associados a cada linha de uma tabela-verdade envolvendo duas variáveis.

Fonte: Elaborada pelo autor, 2020.

Podemos notar, na tabela-verdade, que para construirmos um mintermo temos que complementar as variáveis que estiverem associadas ao valor lógico “0” — deixando sem inverter as variáveis que estiverem sinalizadas como “1”. Por outro lado, para construirmos um maxtermo, temos que complementar as variáveis que estiverem associadas ao valor lógico “1”, deixando sem inverter as variáveis que estiverem sinalizadas como “0”.

Para exemplificarmos o processo de obtenção de uma expressão booleana a partir de uma tabela-verdade, imagine que há a necessidade de criar um circuito que produzirá o resultado de uma votação envolvendo três pessoas votantes “**A**”, “**B**” e “**C**”. O primeiro passo será a criação da tabela-verdade. Em tal tabela-verdade, vamos fazer algumas considerações:

- Para cada votante, o valor “0” denota um voto contrário e, portanto, “1” representa o voto a favor.
- Para o resultado, o resultado valendo “1” significa que o assunto votado foi aprovado, e que o valor “0” representa que a votação finalizou com a rejeição do ponto apreciado.

A partir dessas considerações, vamos construir a tabela-verdade. Aproveitando a oportunidade, para facilitar nossa conversa, já associaremos o mintermo e o maxtermo a cada linha da tabela-verdade. A figura a seguir mostra a tabela-verdade resultante.

	A	B	C	V	mintermos	maxtermos
0	0	0	0	0	~A . ~B .~C	A + B + C
1	0	0	1	0	~A . ~B . C	A + B +~C
2	0	1	0	0	~A . B .~C	A + ~B + C
3	0	1	1	1	~A . B . C	A + ~B +~C
4	1	0	0	0	A . ~B .~C	~A + B + C
5	1	0	1	1	A . ~B . C	~A + B +~C
6	1	1	0	1	A . B .~C	~A + ~B + C
7	1	1	1	1	A . B . C	~A + B + C

Tabela 2 – Mintermos e maxtermos associados a um circuito para a exibição do resultado de votação envolvendo três pessoas votantes (“**A**”, “**B**” e “**C**”).

Fonte: Elaborada pelo autor, 2020.

Na tabela-verdade, perceba, temos a saída “**V**” sinalizada em “1” quando tivermos dois ou

mais votantes que votaram a favor (nível lógico “1”). Para extrairmos a expressão booleana, escolhemos as linhas que tiverem resultado em “**V**” sinalizado em “1” para obter a expressão na forma de soma de produtos, ou escolhemos as linhas cujo valor “**V**” seja “0” para extrairmos a expressão na forma de produto de somas.

A cada linha escolhida, faremos com que o mintermo ou o maxtermo integre a expressão resultante. Assim, teremos:

Soma de produtos:

- Linhas escolhidas: 3, 5, 6 e 7.
- Expressão resultante: **`V = ~A.B.C + A.~B.C + A.B.~C + A.B.C`**

Produto de somas:

- Linhas escolhidas: 0, 1, 2 e 4.
- Expressão resultante: **`V = (A+B+C).(A+B+~C).(A+~B+C).(~A+B+C)`**

Como mencionado anteriormente, toda expressão obtida da tabela-verdade é uma expressão escrita em sua forma canônica. Assim, é passível de simplificação. Veremos, a seguir, como poderemos simplificá-lo.

Vamos praticar?

Suponha uma tabela-verdade de 8 linhas (envolvendo as variáveis “A”, “B” e “C”) cuja coluna de saída é: 01101101 (o bit “0” mais à esquerda corresponde à linha “A=0”, “B=0” e “C=0” e, consequentemente, o valor “1” mais à direita, corresponde à linha “A=1”, “B=1” e “C=1”). Extraia a expressão a partir da tabela-verdade na forma de soma de produtos e, depois, na forma de produto de somas. Em seguida, monte o circuito correspondente no *TinkerCad* e em *Verilog*.

3.3 Simplificação de circuitos lógicos

O processo de simplificação de circuitos lógicos visa diminuir o número de portas lógicas ou tentar, pelo menos, diminuir o tempo de transição do sinal por entre os componentes que fazem parte do circuito. Uma redução do número de portas ou da extensão do caminho que o sinal deverá trafegar proporciona várias consequências, como:

- diminuição no custo dos circuitos;
- diminuição do espaço utilizado pelo circuito;
- diminuição do consumo e da potência dissipada;
- possibilidade de utilizar frequências mais altas de operação;
- diminuição de possíveis diferenças de atrasos de propagação em ramos distintos do circuito.

Mencionamos, a pouco, “diminuição de possíveis diferenças de atrasos de propagação em ramos distintos do circuito”. Mas, o que isso significa? Todo sinal que é propagado por um circuito ou por um fio sofre atraso de propagação. A figura a seguir ilustra o caso ideal e o caso real em um sinal aplicado sobre uma porta inversora.

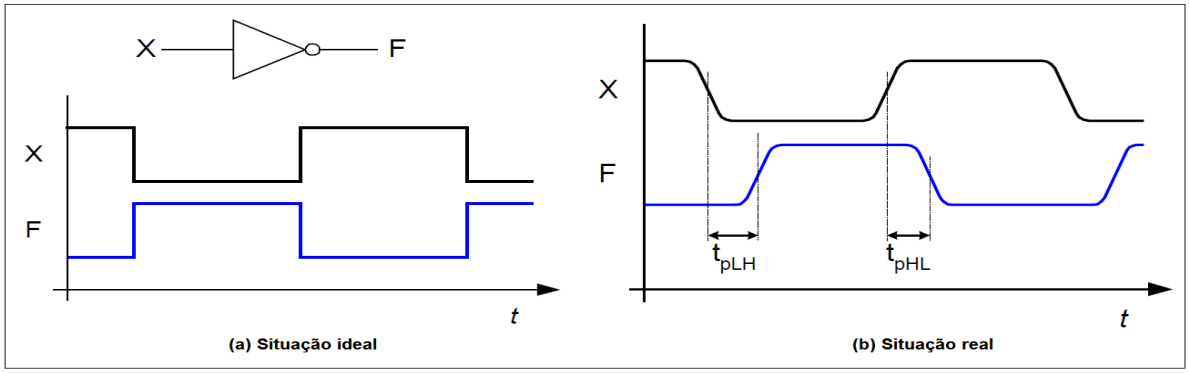


Figura 4 – Atraso de propagação sobre uma porta inversora.

Fonte: Elaborada pelo autor, 2020.

Nota-se, na figura acima que, em (b), temos um atraso de propagação entre a entrada “X” e a saída “F”. Esse atraso é denotado por t_{pLH} (tempo para transição do nível baixo (L – low) para o nível alto (H – high)) e t_{pHL} (tempo para a transição do nível alto (H) para o nível baixo (L). Esses tempos poderão ser distintos e dependem das características de construção do componente. Desta forma, para se estimar o tempo de propagação (t_{pd}), temos:

$$t_{pd} = Max(t_{pdLH}, t_{pdHL})$$

Em um circuito composto por várias portas lógicas e vários caminhos, teremos que analisar o pior caminho (caminho no qual tem-se o maior tempo de propagação) para se estimar o tempo de propagação máximo e, consequentemente, calcular a frequência máxima de trabalho do circuito. Para uma melhor abstração, vamos supor que uma porta “AND” e uma porta “OR” tenham o t_{pd} seu valendo 10ns e, uma porta “NOT”, com um t_{pd} igual a 7ns. Sendo assim,

teremos o caminho crítico do circuito ilustrado na figura a seguir:

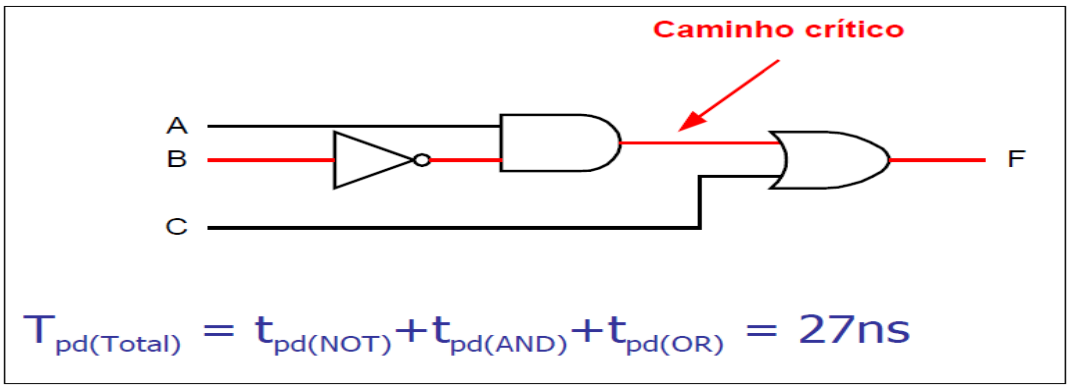


Figura 5 – Caminho crítico do circuito F = C | A.~B.

Fonte: Elaborada pelo autor, 2020.

Na figura acima, temos um caminho crítico que proporciona um atraso de 27ns. Mas, além do atraso temporal do sinal, o que um caminho crítico pode provocar? Para responder a essa pergunta, vamos analisar a expressão **`F = A . ~A`**. Idealmente, aplicando a álgebra booleana, teríamos o valor de "F" sendo constante "0". Porém, na prática, o valor de "F" é representado pela figura a seguir:

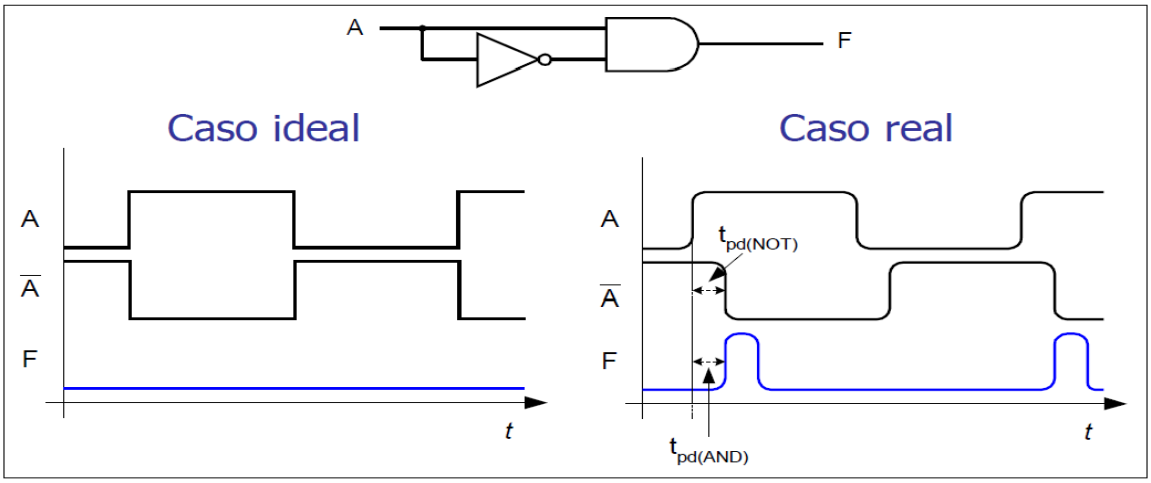


Figura 6 – Ruído introduzido pelo atraso de propagação no caminho crítico.

Fonte: Elaborada pelo autor, 2020.

Notamos, na figura acima, o ruído introduzido pelo caminho crítico no circuito obtido a partir da expressão **`F = A . ~A`**. Desta forma, a análise de propagação do sinal é de suma importância para que se garanta um circuito menos suscetível a falhas em decorrência de ruídos introduzidos pela não sincronização da propagação dos sinais em seus ramos.

Voltando ao assunto sobre como simplificar expressões lógicas, veremos, a seguir, como aplicar as propriedades da lógica booleana e como utilizar o Mapa de Karnaugh.

3.3.1 Simplificação pela aplicação de propriedades e teoremas da Álgebra de Boole

Mas como efetuar a simplificação do circuito? Aplicando-se os teoremas e propriedades da lógica booleana. Em algumas ocasiões, vale, também, o bom senso para aplicar as propriedades, como:

- tentar colocar os termos complementados juntos, dessa forma, consegue-se obter: "**~A+A=1**" ou "**~A.A=0**".
- tentar fazer com que apareça alguma variável isolada de modo a colocá-la em evidência e anular as demais parcelas nas quais tal variável se faça presente.

Vamos iniciar a exemplificação da simplificação, tomando por base o nosso circuito da votação entre três votantes. Para tanto, vamos partir da expressão obtida pelo uso dos mintermos, ou seja, a expressão na forma de soma de produtos.

V = ~A.B.C + A.~B.C + A.B.~C + A.B.C

Inicialmente, podemos observar que alguns membros são comuns a algumas parcelas. Assim, poderemos colocar em evidência:

(I) V = ~A.B.C + A.~B.C + A.B.~C + A.B.C
(II) V = ~A.B.C + A.~B.C + A.B.~C + A.B.C
(III)V = ~A.B.C + A.~B.C + A.B.~C + A.B.C

Notamos que existem três possibilidades de colocação em evidência. Em (I), podemos colocar em evidência "**B.C**"; em (II), podemos colocar "**A.C**"; e, em (III), podemos evidenciar "**A.B**". Será que podemos fazer uso das três possibilidades? A resposta é positiva. Lembrando da propriedade da identidade: "**A+A=A**", podemos fazer o caminho contrário, ou seja, "**A=A+A**". Assim, poderemos replicar a parcela que é referenciada três vezes na evidência ("**A.B.C**"):

V = ~A.B.C + A.~B.C + A.B.~C + A.B.C + A.B.C + A.B.C

Agora, poderemos colocar em evidência "**B.C**", "**A.C**" e "**A.C**":

$$V = \sim A.B.C + A.\sim B.C + A.B.\sim C + A.B.C + A.B.C + A.B.C$$
$$V = B.C(\sim A + A) + A.C(\sim B + B) + A.B(C + \sim C)$$

Observando o conteúdo dos parênteses, notamos que as variáveis aparecem de forma complementada, portanto, poderemos aplicar o postulado do complemento "**~x+x=1**":

$$V = B.C.1 + A.C.1 + A.B.1$$

Por fim, aplicamos o postulado do elemento neutro "**x.1=x**":

$$V = B.C + A.C + A.B$$

Nesse ponto, temos a expressão final, simplificada, para o circuito de votação envolvendo três votantes:

$$V = B.C + A.C + A.B$$

Você quer ver?

Para efetuar simplificações de expressões lógicas, existem diversas ferramentas automáticas disponíveis, tanto gratuitas quanto pagas. A videoaula disponibilizada por (Maganha, 2017) aborda algumas alternativas para facilitar a vida dos projetistas/desenvolvedores de circuitos baseados em lógica digital. Disponível em: <https://www.youtube.com/watch?v=H9qKIHYHV8k>.

Para melhor apresentarmos o processo de simplificação, vamos partir para um outro exemplo de expressão booleana passível de ser otimizada:

$$S = \sim X \ . \ (X + Y) + \sim Z + Z.Y$$

Inicialmente, a fim de colocarmos juntos termos complementados, podemos aplicar a distribuição do termo "**~X**" sobre "**(X+Y)**" e, também, aplicar a distributiva "**~Z + Z.Y**":

$$S = \sim X \ . \ (X + Y) + \sim Z + Z.Y$$
$$S = \sim X \ . \ X + \sim X.Y + (\sim Z + Z).(\sim Z + Y)$$

Aplica-se o postulado do complemento em "**~X . X**" e em "**(~Z + Z)**":

```
S = ~X . X + ~X.Y + (~Z + Z).(~Z + Y)
S = 0 + ~X.Y + (1).(~Z + Y)
```

Podemos retirar os elementos neutros (“0” no operador “OR” e “1” no operador “AND”):

```
S = 0 + ~X.Y + (1).(~Z + Y)
S = ~X.Y + (~Z + Y)
S = ~X.Y + ~Z + Y
```

Agora, podemos colocar o “**Y**” em evidência:

```
S = ~X.Y + ~Z + Y
S = Y(~X + 1) + ~Z
```

Por fim, aplicamos o postulado da absorção sobre o operador “OR” no termo “**~X + 1**”:

```
S = Y(~X + 1) + ~Z
S = Y + ~Z
```

Para realizar a simplificação de uma expressão booleana, podemos, ainda, utilizar outra técnica de simplificação: o “Mapa de Karnaugh” – assunto que conversaremos no próximo tópico.

3.3.2 Simplificação pela aplicação de Mapa de Karnaugh

Como mencionado há pouco, podemos utilizar o Mapa de Karnaugh para simplificar expressões escritas como soma de produtos. Mas o que vem a ser o Mapa de Karnaugh?

De acordo com (Vahid, 2008), os Mapas de Karnaugh constituem uma ferramenta visual, na forma de uma matriz, para a simplificação de expressões que possuem poucas variáveis. Para expressões que manipulam cinco ou mais variáveis, torna-se inviável a utilização dessa ferramenta. Os Mapas de Karnaugh permitem um entendimento básico de outros métodos existentes para a simplificação de expressões booleanas.

Você sabia?

Tenha em mente que o problema de otimização pode ser resolvido por diversas técnicas e ferramentas além dos Mapas de Karnaugh. Uma alternativa para simplificar expressões com um número maior de variáveis consiste na utilização do método de Quine-McCluskey. Porém, a simplificação pode ir além, inclusive empregando técnicas baseadas em inteligência artificial. No trabalho de monografia de (Oliveira, 2015), você encontrará uma solução baseada em evolução artificial. Disponível em: <http://bdm.unb.br/bitstream/10483/11045/1/2015_VitorCoimbraDeOliveira.pdf>.

Um Mapa de Karnaugh faz um mapeamento direto da tabela-verdade. Cada célula da matriz representada pelo mapa corresponde a uma linha da tabela-verdade. A figura a seguir ilustra os mapas para 2, 3 e 4 variáveis.

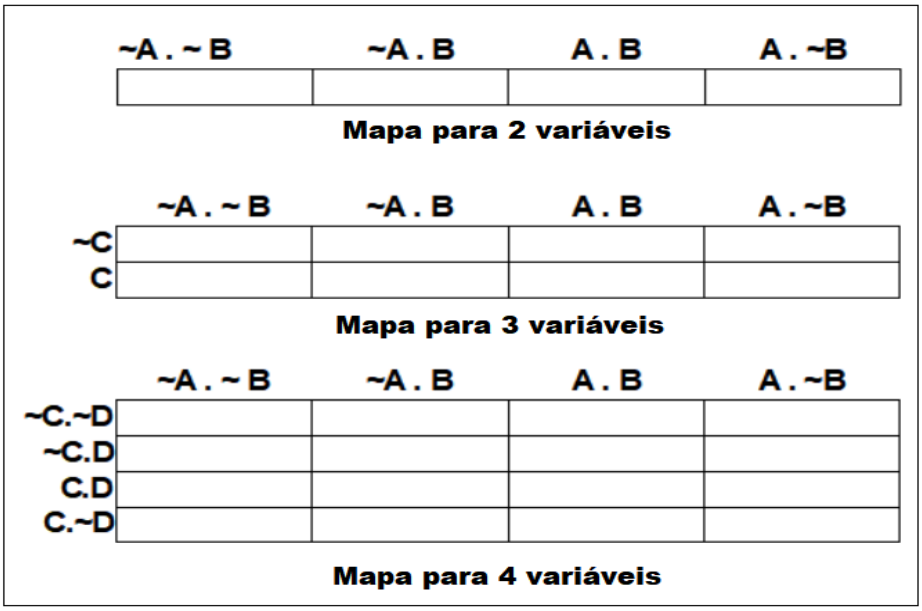


Figura 7 – Mapas de Karnaugh para 2, 3 e 4 variáveis. A sequência das variáveis é fixa, podendo apenas as células serem rotacionadas, sem alterar o sentido de células vizinhas.

Fonte: Elaborada pelo autor, 2020.

Para realizar simplificações utilizando o Mapa de Karnaugh, precisamos conhecer a ideia de vizinhança e de grupo. Para isso, tomemos um mapa para quatro variáveis, conforme ilustrado a seguir.

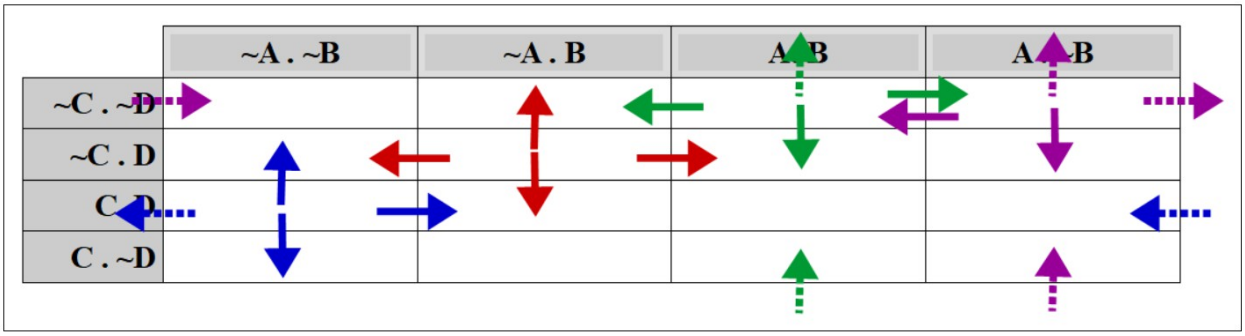


Figura 8 – Exemplos de vizinhos em um mapa para quatro variáveis. As vizinhanças são exibidas pelas colorações das setas. Por exemplo, a célula "**~A.B.~C.D**" tem, como vizinhos: "**~A.~B.~C.D**", "**A.B.~C.D**", "**~A.B.~C.~D**" e "**~A.B.C.D**".

Fonte: Elaborada pelo autor, 2020.

Na figura acima, podemos notar que a linha "**~C.~D**" é vizinha à linha "**C.~D**" e a coluna "**~A.~B**" é vizinha à coluna "**A.~B**". O motivo de serem vizinhos consiste no fato de que o mapa de Karnaugh não é planar, ou seja, como se fosse criarmos uma esfera conectando as extremidades. Sendo assim, cada célula terá, no caso do mapa para 4 variáveis, sempre 4 vizinhos. A ideia de conectar as colunas da extremidade também de aplica aos mapas para 2 e 3 variáveis.

Você o conhece?

Muito se fala dos Mapas de Karnaugh para realizar o processo de simplificação de expressões booleanas. Para saber um pouco mais sobre seu idealizador, Maurice Karnaugh, leia o artigo de (Martins, 2009), disponível em: <https://mauromartins.wordpress.com/2009/05/31/maurice-karnaugh-e-os-mapas-de-karnaugh/>.

Outro conceito relacionado aos mapas de Karnaugh é relacionado à formação de grupos. Veremos que serão transpostos os valores "1" presentes na coluna de saída da tabela-verdade. Os agrupamentos a serem realizados tem o objetivo de envolver as células que contiverem o valor "1". Considerando, ainda, um mapa para quatro variáveis, veremos a ideia de grupos na figura a seguir.

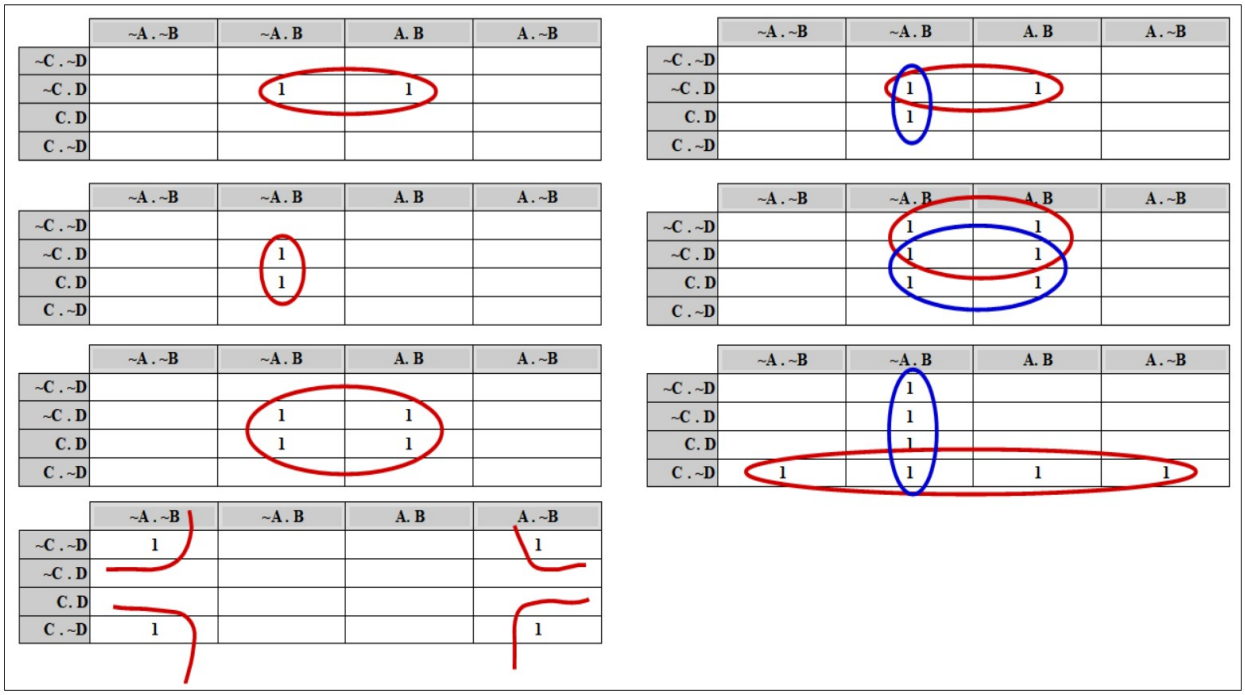


Figura 9 – Exemplos de agrupamentos em um mapa de Karnaugh para quatro variáveis. Nota-se que um elemento "1" pode pertencer a mais de um grupo.

Fonte: Elaborada pelo autor, 2020.

Na figura, temos exemplos de alguns agrupamentos em um Mapa de Karnaugh para quatro variáveis. Algumas observações poderão ser feitas em relação à construção dos grupos:

- cada grupo pode ter 2^i elementos "1", ou seja, cada grupo poderá ser formado por um elemento "1" sozinho, por 2, 4, 8 ou 16 integrantes;
- um grupo não pode ter "arestas", por exemplo, ou a forma da letra "L" e nem a forma da letra "T";
- cada elemento "1" pode pertencer a mais de um grupo, desde que ele agrupe algum outro "1" que não foi ainda englobado por outro grupo;
- criar o menor número de agrupamentos e cada grupo deve ter o maior número possível de elementos "1". Cada grupo representará uma parcela na expressão simplificada resultante, e em cada parcela serão eliminadas k variáveis. Sendo $k=log_2(N)$, em que N representa a quantidade de elementos "1" dentro do grupo.

A partir das observações feitas, vamos, agora, apresentar a sequência de passos que devem ser feitos para o processo de simplificação de uma expressão booleana.

- **Passo 1**: transcrever os valores "1" da coluna de saída da tabela-verdade. Cada valor

"1" deverá ser colocado na célula correspondente à coordenada de sua linha.

- **Passo 2**: formar grupos com os elementos vizinhos, observando-se as regras descritas anteriormente.
- **Passo 3**: em cada grupo, eliminar as variáveis que apareçam de forma complementada, ou seja, preservar aquelas que apareçam de forma idêntica em todas as células envolvidas. Por exemplo, caso tivermos uma dupla nas coordenadas "**~A.B.~C.D**" e "**A.B.~C.D**", deve-se eliminar 1 variável (***k=log₂(2)=1***). No caso, a variável "A" aparece negada em uma célula e não negada na outra. Assim, o resultado da simplificação dessa dupla é "**B.~C.D**". Outro exemplo, caso tivéssemos uma quádrupla formada pelas células "**~A.B.~C.D**", "**A.B.~C.D**","**~A.B.C.D**" e "**A.B.C.D**", encontramos, "**A**" e "**C**" como variáveis que apareçam complementadas – desta forma, a eliminaremos. A partir desta eliminação de "**A**" e "**C**", restam, então, somente as variáveis "**B.D**" na composição final da parcela.. Neste segundo caso, como formamos uma quádrupla, foram eliminadas 2 variáveis (***log₂(4)=2***).

Vamos, agora, voltar ao exemplo da votação envolvendo qautro votantes. A seguir, na figura, temos a tabela-verdade, o Mapa de Karnaugh e o diagrama esquemático do circuito relacionado ao nosso projeto.

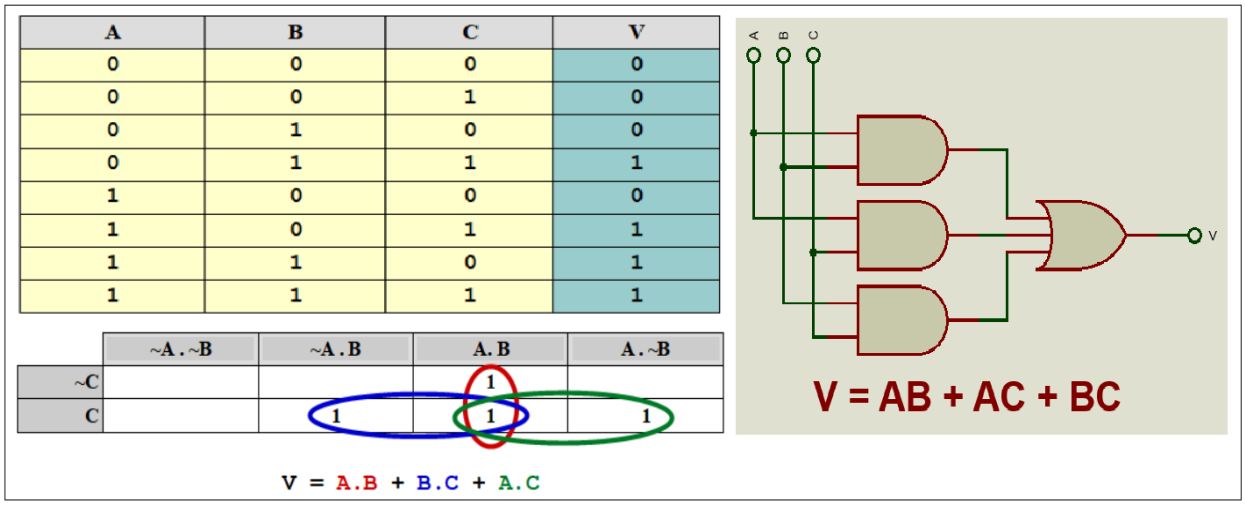


Figura 10 – Processo de simplificação do circuito de votação retratando a tabela-verdade, o Mapa de Karnaugh, além da expressão e o diagrama esquemático do circuito resultante.

Fonte: Elaborada pelo autor, 2020.

Perceba que a célula com a coordenada "**A.B.C**" pertence a três agrupamentos;

exatamente o que acontece quando simplificamos a expressão por meio da álgebra booleana, replicando a parcela para que aparecesse três vezes. Nos sistemas lógicos digitais, além dos níveis "0" e "1", também podemos usar a identificação de valores "**X**" e "**Z**". Mas o que são eles?

- **Valor lógico** "**X**": esse valor, também chamado de "tanto faz", serve para identificar casos nos quais uma certa informação é irrelevante para o resultado da expressão lógica e, consequentemente, para o circuito.
- **Valor lógico** "**Z**": o "**Z**" indica o estado de "alta impedância", ou seja, um desacoplamento de parte do circuito com o restante. Esse estado é utilizado, por exemplo, em sistemas de memória nos quais encontramos os "*buffers 3-state*". Os três estados são representados por: "*operação de leitura*", "*operação de escrita*" e "*não utilizado/desacoplado*". Esse último estado é exatamente o estado "**Z**".

Para um melhor esclarecimento, vamos apresentar um caso no qual é utilizado o valor lógico "**X**". Imagine que um sistema que manipula duas variáveis ("**A**" e "**B**"), porém, por questões de implementações externas, nunca receberá essas duas variáveis valendo "1" simultaneamente. Desse modo, o projetista do circuito não precisa se preocupar com a situação com "**A=1**" e "**B=1**" simultaneamente. A figura a seguir ilustra a tabela-verdade, o Mapa de Karnaugh e o diagrama esquemático desse circuito hipotético.

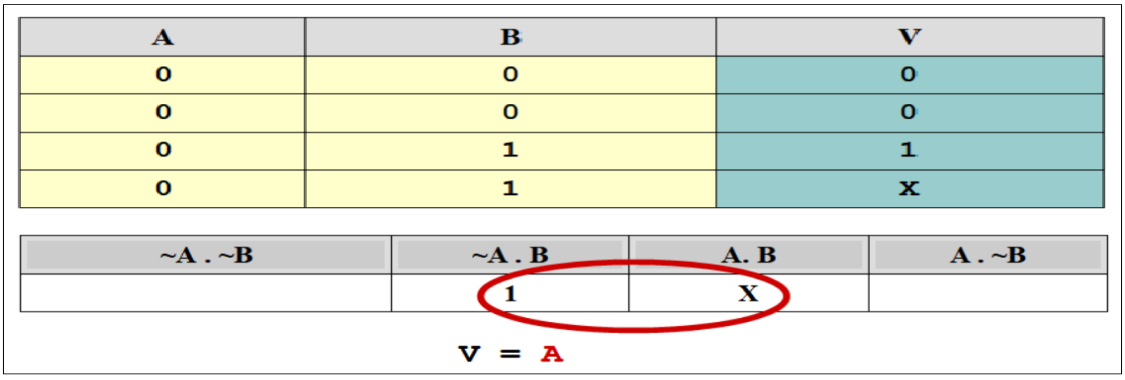
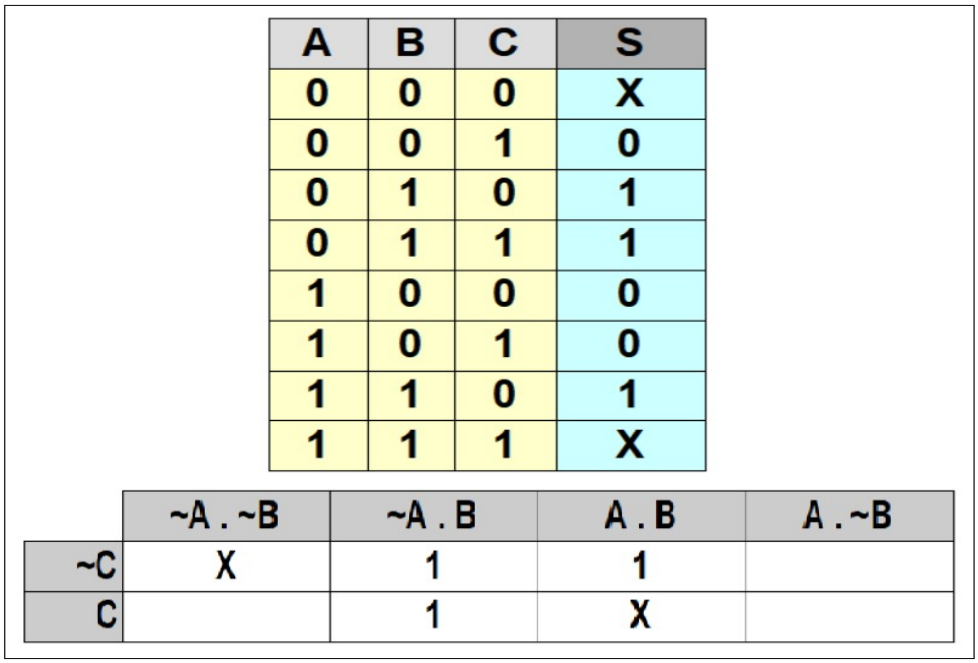


Figura 11 – Tabela-verdade, Mapa de Karnaugh e diagrama esquemático de um circuito hipotético em que as variáveis de entrada nunca assumirão o valor "1" simultaneamente.
Fonte: Elaborada pelo autor, 2020.

Perceba que o valor "**X**" foi utilizado para auxiliar na simplificação da célula "**~A.B**". Porém, caso a célula vizinha já tivesse sido englobada por algum outro agrupamento, ou caso o elemento "**X**" estivesse isolado, faríamos com que o "**X**" valesse "0". Assim, podemos ressaltar que o elemento "**X**" somente valerá "1" caso ele seja útil na simplificação de algum elemento "1"

que faça vizinhança.

Para possibilitar uma melhor abstração de quando é viável a utilização do "**X**" no processo de agrupamentos, vamos analisar a figura abaixo que contempla uma tabela-verdade de um circuito hipotético que manipula três variáveis:



A	B	C	S
0	0	0	X
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	X

	~A . ~B	~A . B	A . B	A . ~B
~C	X	1	1	
C		1	X	

Figura 12 – Dois momentos da utilização do valor booleano "**X**" ("tanto faz") no processo de simplificação utilizando Mapa de Karnaugh.

Fonte: Elaborada pelo autor, 2020.

Na figura, o "**X**" localizado na coordenada "**A.B.C**" foi utilizado como elemento "1", pois ele possibilita uma maior otimização dos itens presentes nas coordenadas "**~A.B.~C**", "**A.B.~C**" e "**~A.B.C**", estabelecendo-se, assim, uma quádrupla. Por outro lado, caso tivéssemos feito o "**X**" presente na coordenada "**~A.~B.~C**", faríamos com que aparecesse uma parcela a mais na expressão resultante otimizada. Essa parcela seria redundante, pois a célula contendo o "1" da coordenada "**~A.B.~C**" já pertence a um agrupamento.

Como mencionado, o mapa de Karnaugh é ideal para manipular um número pequeno de variáveis, pois a partir da quinta variável, um novo plano deve ser aberto, de modo que teremos que identificar grupos de elementos "1" vizinhos nos planos que estão sendo manipulados. Para finalizar, é conveniente passar a dica de que podemos realizar agrupamentos dos elementos "0" em vez de agrupamentos de "1". Porém, nesse caso, deveremos inverter a expressão resultante. Esse macete é útil quando se tem uma grande quantidade de elementos "1" e poucos elementos "0" bem posicionados.

Por exemplo, imagine que, em um mapa de quatro variáveis (16 células), temos apenas dois elementos “0”, formando uma dupla. Podemos obter a expressão booleana dessa dupla, e, depois, complementar a expressão resultante.

Vamos praticar?

a) Extraia a expressão na forma de soma de produtos e na forma de produto de somas a partir de uma tabela-verdade de 8 linhas (envolvendo as variáveis “A”, “B” e “C”) cuja coluna de saída é: 11101010 (o bit “1” mais à esquerda corresponde à linha “A=0”, “B=0” e “C=0” e, consequentemente, o valor “0” mais à direita, corresponde à linha “A=1”, “B=1” e “C=1”). Extraia a expressão a partir da tabela-verdade na forma de soma de produtos e, depois, na forma de produto de somas. Após extrair as duas expressões, aplique as propriedades da álgebra booleana para provar que as duas expressões são equivalentes. Após a simplificação, construa o circuito equivalente no TinkerCad e em Verilog.

b) Utilizando-se Mapa de Karnaugh, simplifique a expressão boolena cuja tabela-verdade é expressa pela seguinte coluna de saída: 10X011010010XX0X (o bit “1” mais à esquerda corresponde à linha “A=0”, “B=0”, “C=0” e “D=0” e, consequentemente, o valor “X” mais à direita, corresponde à linha “A=1”, “B=1”, “C=1” e “D=1”).Após a simplificação, construa o circuito equivalente no TinkerCad e em Verilog.

Síntese

Chegamos ao fim desta unidade. Nessa nossa conversa, pudemos colocar um pouco de prática em nossos circuitos, pois pudemos ver como passar os sistemas lógicos digitais das expressões para os circuitos propriamente ditos. Pudemos, ainda, extrair as próprias expressões booleanas a partir das tabelas-verdade para, depois, aplicarmos técnicas de simplificação de modo a otimizar os circuitos obtidos. Com os pontos abordados, você está apto a desenvolver os seus primeiros circuitos digitais, podendo comprovar suas funcionalidades por meio de simuladores ou realizando a sua implementação física.

Aqui, você teve a oportunidade de:

- ter um contato inicial com os componentes físicos que implementam os componentes básicos da eletrônica digital.
- colocar em prática as teorias vistas até o momento em Sistemas Digitais.
- saber identificar e manusear os componentes da eletrônica digital.

- projetar e modelar circuitos através de suas tabelas-verdade e expressões booleanas.
- aplicar as teorias acerca da lógica booleana para simplificar expressões lógicas.

Bibliografia

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).

IFSC. **Aula 16 – Eletrônica Digital 1 – Graduação**. Publicado em 22/06/2018. Disponível em <https://wiki.ifsc.edu.br/mediawiki/index.php/AULA_16_-_Eletr%C3%B4nica_Digital_1_-_Gradua%C3%A7%C3%A3o>. Acesso em 19/10/2020.

MAGANHA, G. V. S. **Eletrônica Digital #61: Software para simplificar expressões booleanas**. Publicado em 22/05/2017. Disponível em <https://www.youtube.com/watch?v=H9qKIHYHV8k>. Acessado em 19/10/2020.

MARTINS, M. **Maurice Karnaugh e os Mapas de Karnaugh**. Publicado em 31/05/2009. Disponível em <https://mauromartins.wordpress.com/2009/05/31/maurice-karnaugh-e-os-mapas-de-karnaugh/>. Acessado em 19/10/2020.

OLIVEIRA, V.C. **Projeto e otimização de circuitos digitais por técnicas de evolução artificial**. Monografia (Graduação em Bacharelado em Ciência da Computação), Instituto de Ciências Exatas, Universidade de Brasília, Brasília, 2015. Disponível em <http://bdm.unb.br/bitstream/10483/11045/1/2015_VitorCoimbraDeOliveira.pdf>. Acessado em 19/10/2020.

TINKERCAD. <https://www.tinkercad.com>. Acessado em 18/10/2020.

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).