# SISTEMAS DIGITAIS

## UNIDADE 3 – CIRCUITOS LÓGICOS

Fernando Cortez Sica

# Introdução

Olá, prezado estudante! Seja bem-vindo a esta unidade de **Sistemas Digitais**. 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 tipos de componentes podem ser usados para a montagem física dos circuitos? Bem, saiba, desde já, que grande parte dos circuitos que apresentaremos, assim como seus 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 e em nosso cotidiano, nas mais diversas situações.

Outra questão que poderá vir à sua mente ao estudar este conteúdo: com os pontos aqui abordados, poderei implementar circuitos mais complexos? Um circuito, ou sistema, corresponde à interligação de módulos com funcionalidades bem focadas e específicas. Assim, com a bagagem adquirida aqui, você poderá, sim, modelar e implementar circuitos mais complexos. Faremos também menções técnicas usadas na indústria para aproximar os pontos teóricos dos pontos práticos.

Conversaremos, inicialmente, sobre os conceitos envolvidos nos circuitos lógicos, formando uma base teórica que permitirá a você compreender os 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. Para finalizar, abordaremos os circuitos lógicos sequenciais, destacando e diferenciando seus componentes básicos: *latches* e *flip-flops*.

Preparado? Então vamos lá!

# 3.1 Circuitos Lógicos Combinacionais

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 esses circuitos? Descubra a seguir!

## *3.1.1 Explicações iniciais*

De acordo com Idoeta e Capuano (2012), um circuito é combinacional quando 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 é mostrar o resultado de uma votação entre três pessoas (“A”, “B” e “C”). Será considerado “aprovado” se tivermos dois ou três votantes favoráveis ao ponto sob votação.

## 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/ (https://exame.abril.com.br/economia/e-brincando-que-se-trabalha/).

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 dessas considerações, temos, na figura a seguir, a tabela-verdade, a expressão e o circuito resultante.

Figura 1 - Tabela-verdade, expressão e circuito para gerar o resultado de votação entre três votantes.
Fonte: Elaborada pelo autor, 2019.

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).**

## CASO

Um desenvolvedor recebeu um projeto para automatizar um processo vinculado a uma horta hidropônica familiar. Como mecanismos, tinha que implementar as seguintes funcionalidades: fechar a cobertura caso houvesse incidência de sol muito forte; de tempos em tempos, liberar uma solução composta por água e nutrientes; e verificar o nível da água em circulação, de modo a injetar mais caso necessário. Inicialmente, ele pensou em implementar o sistema totalmente baseado em *software*. Porém, ele percebeu que seria necessário disponibilizar um computador dedicado a tais funcionalidades — consumindo uma quantidade razoável de energia elétrica. Ele percebeu, também, que as funcionalidades eram muito básicas, de modo que não compensava adotar uma solução baseada em Arduino ou algo semelhante. A solução dele, então, foi montar a lógica totalmente baseada em sistemas digitais, manipulando os dados exportados e atuando sobre componentes eletrônicos básicos. O projeto resultante teve como características: baixo custo de implementação; alta disponibilidade; e fácil manutenção.

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

### *3.1.2 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 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.

Figura 2 - Tabela-verdade, expressões booleanas e diagrama esquemático de um decodificador decimal-binário.
Fonte: Elaborada pelo autor, 2019.

A figura ilustra a implementação de um decodificador decimal para binário. As entradas, representando o dígito decimal a ser convertido, são identificadas 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 valor 1 caso o valor fornecido como entrada seja 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, este 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

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 “co Gray”. Por exemplo, caso estejamos manipulando um número de 2 bits, como a a seguir.

**Note que, a cada transição do código Gray, tem variação de apenas um bit da palavra. Como implementar um codificador que converte um número em BCD8421 para o formato Gray?**

### *3.1.3 Multiplexadores*

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 seu valor propagado para a saída.
Para melhor compreender seu funcionamento e sua constituição física, vamos observar a figura a seguir:

Figura 3 - Funcionamento básico, tabela-verdade e diagrama esquemático de um multiplexador de quatro entradas e uma saída.
Fonte: Elaborada pelo autor, 2019.

Na figura, perceba, temos a composição de um multiplexador composto por quatro entradas (**$E_0$**, **$E_1$**, **$E_2$** e **$E_3$**) e uma saída (**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 **$S_0$**:

$\overline{S_1}$ e $\overline{S_0}$ = 00

$\overline{S1}$ e $\overline{S0}$ = 01

S1 e 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 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.

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:

**1**

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.

**2**

CMOS → 4052; TTL → 74153: dual MUX de quatro entradas cada. Os bits de seleção são comuns a dois multiplexadores implementados no circuito integrado.

**3**

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 uma melhor compreensão, vamos retomar o exemplo da votação apresentado no início deste capítulo e reimplementá-lo utilizando 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, 2019.

Na figura, perceba, podemos visualizar a implementação do circuito de votação, utilizando o 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 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*, ou terra). Por fim, temos os bits de seleção nos pinos 11, 10 e 9 (sendo o pino "A", o mais significativo).
Para utilizar um MUX como gerador de funções, devemos estabelecer uma correspondência direta entre os pinos de entrada do MUX e a coluna de saída da tabela-verdade. Assim, quando tivermos o valor "0" na tabela-verdade, devemos associar o pino correspondente ao "terra", e quando 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 o artigo de Curvello (2018), disponível em: https://www.embarcados.com.br/introducao-do-mundo-do-hardware-reconfiguravel-conhecendo-as-fpgas/ (https://www.embarcados.com.br/introducao-do-mundo-do-hardware-reconfiguravel-conhecendo-as-fpgas/).

Já falamos sobre como selecionar uma entrada para que 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, assim, de um demultiplexador (demultiplex ou, ainda, DEMUX).

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

Figura 5 - Ideia básica de funcionamento, tabela-verdade e diagrama esquemático de um DEMUX de quatro saídas.
Fonte: Elaborada pelo autor, 2019.

Na figura, perceba, 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 a 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.

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, podemos fazer uso da união de MUXs ou 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.

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, 2019.

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.

## *3.1.4 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:

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

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

Para a implementação, vamos abstrair o mesmo processo que realizamos em 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 em que há 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.

Figura 7 - Tabelas-verdade e diagramas esquemáticos dos circuitos relativos ao meio somador, meio subtrador, somador completo e subtrador completo.
Fonte: Elaborada pelo autor, 2019.

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.
- "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 (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, 2019.

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 "*overflow*" (estouro). Na operação de soma, se esse bit for igual a "1", 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 vêm sendo realizados no sentido de encontrar soluções que, por exemplo, consumam menos energia. O trabalho realizado por Faria (2014) relata a implementação de um somador do tipo "bit-serial", e está 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 (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)>. Para saber mais sobre esse tipo de implementação, você também poderá ler o artigo de Magalhães (2016), disponível em: https://www.embarcados.com.br/somador-carry-lookahead-de-4-bits/ (https://www.embarcados.com.br/somador-carry-lookahead-de-4-bits/).

Até o momento, conversamos sobre circuitos lógicos combinacionais. Mas o que seriam os sequenciais? A seguir, abordaremos esse tema.

## VAMOS PRATICAR?

Falamos em circuitos combinacionais para efetuar a aritmética básica: subtração. Mas e se tivéssemos apenas um circuito para efetuar a operação de de subtração? Neste caso, teríamos um bit de controle (denominado, por “SOMASUB”) para indicar a operação a ser realizada.

Por exemplo: caso o “SOMASUB” seja associado ao valor lógico “0”, o retornará, em sua saída, o valor relativo à soma. Caso seja associado ao valo “1”, a saída corresponderá à subtração dos valores fornecidos às suas entrada implementar esse circuito?

# 3.2 Circuitos Lógicos Sequenciais

De acordo com Tocci, Widmer e Moss (2018), circuitos lógicos sequenciais são aqueles capazes de armazenar informações, as quais vão depender do estado atual do componente. É justamente devido a essa sequência de informações que os circuitos lógicos sequenciais são assim denominados.

A ideia básica para a construção de circuitos sequenciais consiste em duas portas inversoras interconectadas formando um ciclo: a saída de uma porta NOT é ligada à entrada da outra porta NOT. Com essa realimentação, consegue-se fazer com que a informação seja mantida enquanto o dispositivo permanecer ligado. Como exemplos de circuitos sequenciais, podemos citar: registradores; registradores de deslocamento; contadores assíncronos; e contadores síncronos. Todos esses circuitos são implementados utilizando “*latches*” e “*flip-flops*”, conceitos que veremos a seguir.

## *3.2.1 Latches*

Os *latches* são os componentes básicos da lógica digital sequencial. Em sua implementação básica, são formados por portas lógicas dotadas de inversão (NOR ou NAND) conforme a figura a seguir:

Figura 9 - Tabela-verdade e implementação básica de latches do tipo RS usando a lógica positiva (a) e usando a lógica negativa (b).
Fonte: Elaborada pelo autor, 2019.

Na figura, perceba, temos dois *latches* RS: um sendo implementado com portas NOR (a – lógica positiva) e outro com portas NAND (b – lógica negativa). Os valores inseridos em seus terminais "**R**" ("*reset*") e "**S**" ("*set*") definem o comportamento do componente. Na lógica positiva, esses terminais são ativados no nível "1" e, na lógica negativa, ativados em "0". Quando os dois terminais não são ativados (na primeira linha da tabela-verdade de (a) ou na última linha da tabela-verdade de (b)), o valor armazenado no componente fica inalterado. Ao se ativar o terminal "**R**" (terceira linha de (a) ou segunda linha de (b)), é carregado para ser armazenado o valor "0". Na ativação do "**S**" (segunda linha de (a) ou terceira linha de (b)), o componente passa a armazenar o valor "1". Para esse tipo de componente, deve-se evitar que os dois terminais "**R**" e "**S**" sejam ativados simultaneamente. Em tal situação, o *latch* entra no estado de indeterminação.

Os valores armazenados poderão ser utilizados por intermédio de seus pinos de saída "**Q**" e "**~Q**" (o nome "**Q**" é derivado da palavra "*queue*" – fila, ou a própria sequência). Essas saídas serão sempre complementares, ou seja, "**Q**" apresenta o próprio valor armazenado e "**~Q**" externa o valor armazenado de forma negada.

Mas como evitar a ativação simultânea dos terminais "**RS**"? Existem algumas outras versões de *latch*, como o *latch* do tipo "D". A figura a seguir mostra uma utilização prática do *latch* "**RS**" e a implementação do *latch* tipo "D".

Figura 10 - Temos, em (a), um exemplo de aplicação do latch "RS". Em (b), a tabela-verdade do latch tipo "D" e a sua implementação em (c) e a sua simbologia em (d).
Fonte: Elaborada pelo autor, 2019.

Na figura, temos, em (a), uma aplicação prática para o *latch* "**RS**". Essa aplicação visa ligar uma bomba de água somente quando o nível, em uma caixa d'água, atingir um patamar mínimo (identificado pela "boia nível baixo"). Ao ser atingido esse nível, o sinal "**S**" do *latch* será acionado e a bomba será ligada, permanecendo assim até que seja atingido o nível da "boia nível alto". Em tal situação, o pino "**R**" será acionado, desligando-se a bomba, que permanecerá assim até que o nível de água atinja novamente o nível baixo.

Ainda na mesma figura, temos em (b) a tabela-verdade de um *latch* tipo "D" com sinal de "**enable**" (habilitação). Note que, caso esteja desabilitado ("**enable**" igual a 0), o *latch* permanecerá armazenando o valor atual independentemente do valor da entrada "**D**". Quando o "**enable**" estiver em seu nível alto, o valor presente na entrada "**D**" do *latch* será copiado e armazenado pelo componente e poderá ser utilizado por intermédio de seus terminais de saída. Essa tabela-verdade é refletida no item (c) da figura, em que se nota que o *latch* "D" é implementado a partir do *latch* "**RS**". Finalmente, temos em (d) a representação do *latch* "D".

Além da entrada de habilitação, podemos encontrar outros sinais de controle nos *latches* (clique para ler):

| | |
|---|---|
| **PRESET** | O sinal de "**PRESET**", quando ativado, faz com que o valor armazenado se tornasse "1", independentemente dos valores de "**enable**" e de "D". |
| **CLEAR** | Esse sinal, independentemente dos valores presentes em "**enable**" e em "D", faz com que o valor armazenado se torne "0". |

Uma questão que você pode, agora, refletir sobre: a memória *cache* do computador (SRAM – Static RAM) é baseada em "*flip-flops*". Mas o que vem a ser esse componente? Qual é a diferença em relação aos *latches*? Responderemos esse questionamento a seguir, portanto, continue acompanhando com atenção!

### *3.2.2 Flip-Flops*

Mencionamos que os *latches* têm o objetivo de armazenar informações. Porém, existe um outro componente, denominado "*flip-flop*", cujo objetivo também é o armazenamento de dados. A diferença básica entre eles consiste em sua forma de habilitação (ativação). Enquanto que os *latches* permanecem ativos durante todo o período no qual o sinal de habilitação estiver ativo, os *flip-flops* são ativados apenas nas transições dos valores. Tais transições são denominadas "rampas".

- Rampa (transição) de subida, rampa positiva, ou lógica positiva: representa a transição do valor "0" para o valor "1".
- Rampa (transição) de descida, rampa negativa, ou lógica negativa: representa a transição do valor "1" para o valor "0".

Mas como implementar um *flip-flop*? Vamos tomar como exemplo um *flip-flop* tipo "D". Na figura a seguir, observe que um *flip-flop* tipo "D" poderá ser implementado utilizando-se dois *latches* do tipo "D".

Figura 11 - Formas de ondas relativas a um latch "D" (a) e a um flip-flop "D" (b). Implementação de um flip-flop "D", usando latches "D" (c). Simbologia flip-flop "D" (d).
Fonte: Elaborada pelo autor, 2019.

Na figura, os momentos de ativação encontram-se destacados na cor vermelha (itens (a) e (b) da figura). Podemos notar que, no *latch*, o sinal "**D**" continua sendo propagado para dentro do componente enquanto o sinal de habilitação estiver ativo (nesse exemplo, está sendo utilizada a lógica positiva). Por sua vez, no *flip-flop*, ocorre a cópia do valor de "**D**" apenas no período de transição do sinal de habilitação (denominação foi renomeada para "*clock*" para adequação ao *flip-flop*). Essa lógica pode ser acompanhada se observarmos a implementação em (c). Note que foi utilizado, no primeiro estágio, um *latch* que utiliza a lógica negativa para o sinal de habilitação (cópia de "**D**" habilitada no nível "0"). O segundo estágio realiza a cópia nos momentos em que o "enable" vale "1" (lógica positiva). Essa sequência de ativação ("0" no *latch* à esquerda e "1" no *latch* à direita) faz com que o valor "**D**" seja efetivamente copiado na transição de "0" para "1". Caso tivéssemos invertido a ordem de ativação dos *latches*, teríamos a cópia durante a transição de "1" para "0", ou seja, estaríamos manipulando a rampa de descida.
Para finalizar, o item (d) da figura mostra a simbologia de um *flip-flop* tipo "D". A diferenciação da representação de um *latch* é feita pelo uso do sinal triangular no terminal de *clock*, indicando a ativação pela rampa. Caso o componente fosse ativado na lógica negativa, teríamos, na entrada do sinal de *clock*, a simbologia de uma porta inversora.
Mas como armazenar palavras em vez de armazenar apenas 1 bit? A resposta a essa questão está na utilização de registradores, os quais veremos a seguir.

### *3.2.3 Registradores*

Registradores são organizações de *flip-flops*, de modo que todos possam ser ativados simultaneamente, ou seja, no mesmo sinal de *clock* (VAHID; LASCHUK, 2011).
Na figura a seguir, podemos ver três configurações de registradores quanto à forma de interligação dos *flip-flops*:

Figura 12 - Três modos de interligação dos flip-flops nos registradores: armazenamento básico (a), registrador de deslocamento (b), registrador de deslocamento em anel (c)
Fonte: Elaborada pelo autor, 2019.

Na figura, temos a exemplificação da utilização dos *flip-flops* "D" para a implementação de registradores. Podemos observar que o sinal de "**clock**" é comum a todos os *flip-flops*. Na versão (a), temos um registrador para realizar, exclusivamente, armazenamento de uma palavra de quatro bits. A palavra fornecida é armazenada ao pulso do sinal de *clock*. Em (b) e em (c), temos a possibilidade de deslocar a informação armazenada (no caso, deslocamento para a direita). O valor armazenado em um *flip-flop* é copiado para seu vizinho da direita. Poderíamos também realizar deslocamento para a esquerda, no caso, o valor armazenado em um *flip-flop* deveria ser copiado para o seu vizinho da esquerda. A diferença entre as versões (b) e (c) reside no fato de que, em (b), o *flip-flop* mais à esquerda recebe um valor externo, enquanto que, em (c), recebe o valor armazenado no *flip-flop* mais à direita; fazendo-se, assim, uma realimentação dos bits.
Os registradores podem ser usados em diversas situações, dentre as quais podemos destacar:

- armazenamento básico;
- multiplicação/divisão por base 2;
- conversor serial/paralelo e paralelo/serial;
- controlador sequencial;
- divisor de frequência.

Diversas aplicações poderão ser feitas tanto com circuitos lógicos combinacionais quanto com sequenciais. Praticamente todas as funcionalidades vistas neste capítulo podem ser encontradas comercialmente por meio dos circuitos integrados.

## VOCÊ QUER VER?

Atualmente, a implementação de sistemas digitais está mudando o foco para a utilização de *HDL* (*Hardware Description Language* – Linguagem de Descrição de Hardware). Para você saber um pouco sobre esse assunto, assista ao vídeo de (IFPB, 2018), disponível em: https://www.youtube.com/watch?v=9rEOvt5cCQM (https://www.youtube.com/watch?v=9rEOvt5cCQM).

Uma outra vertente que poderá ser utilizada na implementação de circuitos é baseada na utilização de *HDL* (*Hardware Description Language* – Linguagem de Descrição de Hardware), tal como Verilog e VHDL.

## VAMOS PRATICAR?

Foi mencionado que um registrador pode desempenhar várias funcionalidade Foram destacadas as seguintes:

- armazenamento básico;
- multiplicação/divisão por base 2;
- conversor serial/paralelo e paralelo/serial;
- controlador sequencial;
- divisor de frequência.

Você seria capaz de associar cada funcionalidade aos tipos de configu registradores apresentados na figura 12?

# Síntese

Chegamos ao fim desta unidade. Neste 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.

Nesta unidade, você teve a oportunidade de:

- ter um contato inicial com os componentes e circuitos básicos digitais;
- diferenciar sistemas lógicos combinacionais dos sistemas lógicos sequenciais;
- identificar e implementar aplicações utilizando sistemas digitais;
- fazer um paralelo com algumas funcionalidades dos processadores e sistemas computacionais.

# Bibliografia

CURVELLO, A. **Introdução do Mundo do Hardware Reconfigurável**: Conhecendo as FPGAs, 28 nov. 2018. Disponível em: (https://www.embarcados.com.br/introducao-do-mundo-do-hardware-reconfiguravel-conhecendo-as-fpgas/)https://www.embarcados.com.br/introducao-do-mundo-do-hardware-reconfiguravel-conhecendo-as-fpgas/ (https://www.embarcados.com.br/introducao-do-mundo-do-hardware-reconfiguravel-conhecendo-as-fpgas/). Acesso em: 19/09/2019.

FARIA, R. M. **Utilização de Aritmética bit-serial para Redução de Consumo de Energia**. Dissertação (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)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 (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). Acesso em: 19/09/2019.

GATTO, E. C. **Circuitos Digitais**: Paridade Parte 1, 25 jun. 2014. Disponível em: (https://pt.slideshare.net/elainececiliagatto/circuitos-digitais-paridade-parte-1)https://pt.slideshare.net/elainececiliagatto/circuitos-digitais-paridade-parte-1 (https://pt.slideshare.net/elainececiliagatto/circuitos-digitais-paridade-parte-1). Acesso em: 19/09/2019.

IDOETA, I. V.; CAPUANO, F. G. **Elementos de Eletrônica Digital**. 41. ed. São Paulo: Érica, 2012.

IFPB Eletrônica. **Eletrônica Digital – Aula 6 – Introdução ao SystemVerilog**, 02 mar. 2018. Disponível em: (https://www.youtube.com/watch?v=9rEOvt5cCQM)https://www.youtube.com/watch?v=9rEOvt5cCQM (https://www.youtube.com/watch?v=9rEOvt5cCQM). Acesso em: 18/09/2019.

MAGALHÃES, G. V. N. **Somador carry-lookahead de 4 bits em VHDL**, 20 jan. 2016. Disponível em < (https://www.embarcados.com.br/somador-carry-lookahead-de-4-bits/)https://www.embarcados.com.br/somador-carry-lookahead-de-4-bits/ (https://www.embarcados.com.br/somador-carry-lookahead-de-4-bits/)>. Acesso em: 19/09/2019.

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

VAHID, F.; LASCHUK, A. **Sistemas Digitais**: Projeto, otimização e HDLs. Porto Alegre: Bookman, 2011.