

5. CIRCUITOS LÓGICOS SEQUENCIAIS

Introdução

Olá, prezado estudante! Seja bem-vindo à quinta unidade de **BCC265**!

A partir de seus conhecimentos sobre os circuitos combinacionais, chegou a hora de dialogarmos sobre a outra classe dos circuitos digitais: os circuitos lógicos sequenciais. Dentro dos circuitos sequenciais veremos, inicialmente, os *latches* e os *flip-flops* para que, depois, possamos entrar no mundo dos registradores e contadores. Para que servem os *latches*, *flip-flops* e registradores? A funcionalidade básica destes elementos consiste no armazenamento de informações. Os *latches* e *flip-flops* são capazes de armazenar um *bit* cada componente. Por sua vez, os registradores armazenam palavras de informações. Com as palavras armazenadas, poderemos realizar algumas operações e empregar os registradores em diversas aplicações. E, o que vêm a ser os contadores? Como o próprio nome diz, são circuitos responsáveis por estabelecer uma sequência de contagem. E será que as contagens devem ser sempre lineares, ou seja, com os números consecutivos? Não, por isso, abordaremos os contadores síncronos e assíncronos. Os assíncronos permitem que sejam produzidas apenas contagens lineares, por sua vez, com os síncronos, podemos definir sequências não lineares.

Com as contagens produzidas pelos contadores, o que podemos visar? Realizar contagens não se aplica somente às aplicações que demandam contagem puramente. Podemos, também, visar, por exemplo, divisão de frequência e controle inerente a uma máquina de estados. Com os componentes digitais, é possível montar um circuito contador? Sim, porém, para isso, vamos descrever outros dois tipos de *flip-flops*: os *flip-flops* tipo “JK” e tipo “T”. Tenha em mente que eles são a base dos circuitos contadores, tanto assíncronos quanto síncronos.

Assim como os demais circuitos combinacionais e sequenciais, os contadores também podem ser implementados utilizando Linguagens de Descrição de Hardware (HDL – *Hardware Description Language*) ou utilizando circuitos integrados que implementam os *flip-flops* ou que implementam os próprios contadores.

Para que possamos conversar sobre o assunto, vamos primeiro abordar os novos tipos de *flip-flops* (tipo “JK” e tipo “T”) para que possamos aplicá-los nos contadores assíncronos. Tratando sobre os contadores assíncronos, após vermos os conceitos envolvidos e a sua forma de implementação, falaremos de como limitar a sua contagem. Em seguida, dialogaremos sobre os contadores síncronos, sobre como poderemos implementá-los, bem como sobre suas vantagens sobre os assíncronos. Por fim, para que possamos conversar sobre o emprego de contadores nas

máquinas de estado, conceituaremos tais máquinas de estados e onde poderemos utilizá-las.

5.1 Conceitos Básicos dos Circuitos Lógicos Sequenciais

De acordo com (TOCCI; WIDMER; MOSS, 2018), circuitos lógicos sequenciais são aqueles capazes de armazenar informações. As informações a serem armazenadas podem depender do estado atual do componente; é devido a essa sequência de informações que os circuitos lógicos sequenciais são assim denominados.

A ideia básica para a construção de circuitos sequenciais consiste de 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.

5.1.1 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:

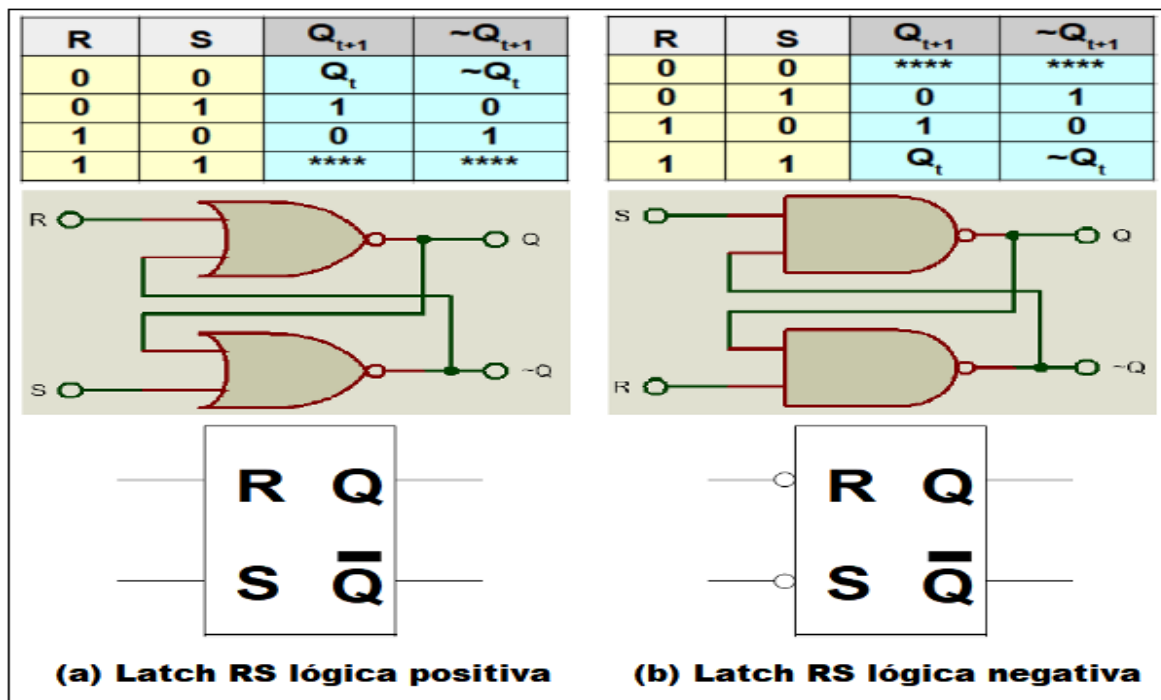


Figura 1 – Tabela-verdade e implementação básica de latches do tipo RS usando a lógica positiva (a) e usando a lógica negativa (b).

Fonte: Elaborada pelo autor, 2020.

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”.

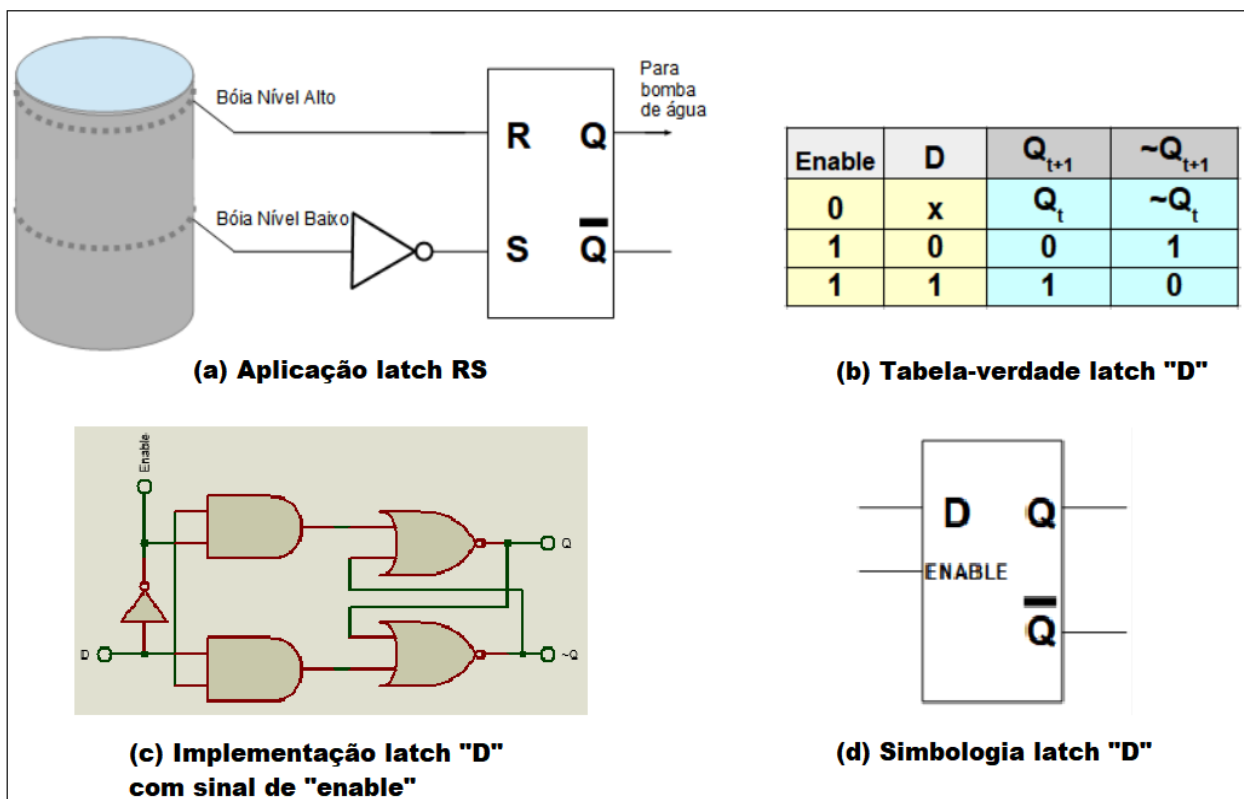


Figura 2 – Temos, em (a), um exemplo de aplicação do *latch* “RS”. Em (b), a tabela-verdade do *latch* tipo “D” e a sua implementação em (c) e a sua simbologia em (d).

Fonte: Elaborada pelo autor, 2020.

Na figura, temos, em (a), uma aplicação prática para o *latch* “RS”. Essa aplicação visa ligar uma bomba de água somente quando o nível de água, em uma caixa de água, atingir um patamar mínimo (identificado pela “boia nível baixo”). Ao ser atingido esse nível, o sinal “S” do *latch* será acionado ligando-se a bomba que permanecerá ligada até que seja atingido o nível da “boia nível alto”. Em tal situação, o pino “R” será acionado, desligando-se a bomba que ficará desligada 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). Nota-se que, caso *latch* 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*:

- **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: escutei que a memória *cache* do computador (SRAM – Static RAM) é baseada em “*flip-flops*”. Mas o que vem a ser esse componente? Qual a diferença em relação aos *latches*? Responderemos a esse questionamento a seguir. Continue acompanhando!

5.1.2 Flip-Flops

Mencionamos que os *latches* têm o objetivo de armazenar informações. Porém, existe um outro componente, denominado “*flip-flop*”, que também tem por objetivo 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 como 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”. Por meio da figura a seguir, poderemos observar que um *flip-flop* tipo “D” poderá ser implementado utilizando-se dois *latches* do tipo “D”.

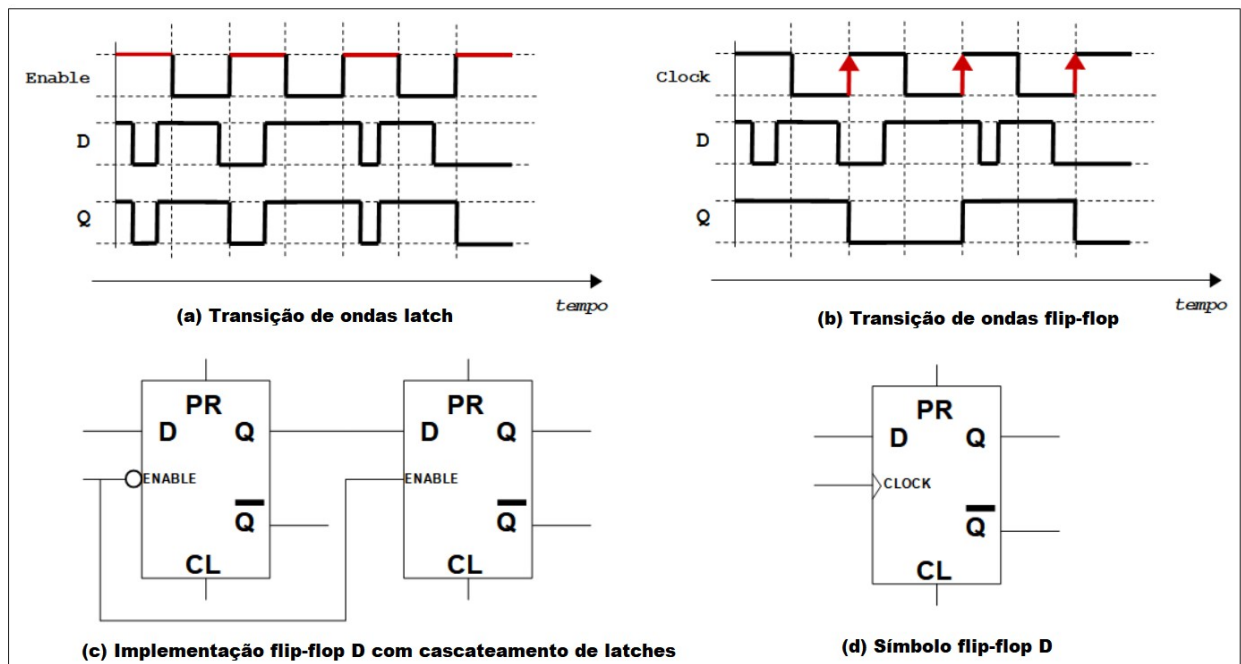


Figura 3 – Formas de ondas relativas a um *latch* “D” (a) e a um flip-flop “D” (b). Implementação de um flip-flop “D” usando *latches* “D” (c). Simbologia flip-flop “D” (d).

Fonte: Elaborada pelo autor, 2020.

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 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 acima 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.

A codificação a seguir, em Verilog, retrata um *flip-flop* tipo D com entradas “*preset*” e “*clear*” assíncronas e síncronas.

```

module FF_D_Assinc (D,Q,PR,CL,CLK) ;
    input D,PR,CL,CLK;
    output reg Q;

    initial
        begin
            Q = 0;
        end

    always @(posedge CLK or posedge PR or posedge CL)
        begin
            if (PR) Q <= 1;
            else if (CL) Q <= 0;
            else Q <= D;
        end

endmodule // FF_D_Assinc

module FF_D_Sinc (D,Q,PR,CL,CLK) ;
    input D,PR,CL,CLK;
    output reg Q;

    initial
        begin
            Q = 0;
        end

    always @(posedge CLK)
        begin
            if (PR) Q <= 1;
            else if (CL) Q <= 0;
            else Q <= D;
        end

endmodule // FF_D_Sinc

```

No código acima podemos observar que a diferença básica entre o tratamento dos sinais “*preset*” e “*clear*” relaciona-se com o momento de ativação. Na ativação assíncrona, estes sinais são tratados junto ao sinal do *clock* na expressão de ativação do “*always*”. No tratamento síncrono, o único teste do “*always*” refere-se ao próprio sinal de *clock*. Convém salientar que quando se utiliza a sinalização “*posedge*” ou “*negedge*” no *always*, a atribuição deve ser não bloqueante – no caso, utiliza-se o sinal de atribuição “*<=*”.

Mas como armazenar palavras em vez de armazenar apenas 1 bit? A resposta a essa questão está na utilização de registradores. O que veremos a seguir.

5.2 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*.

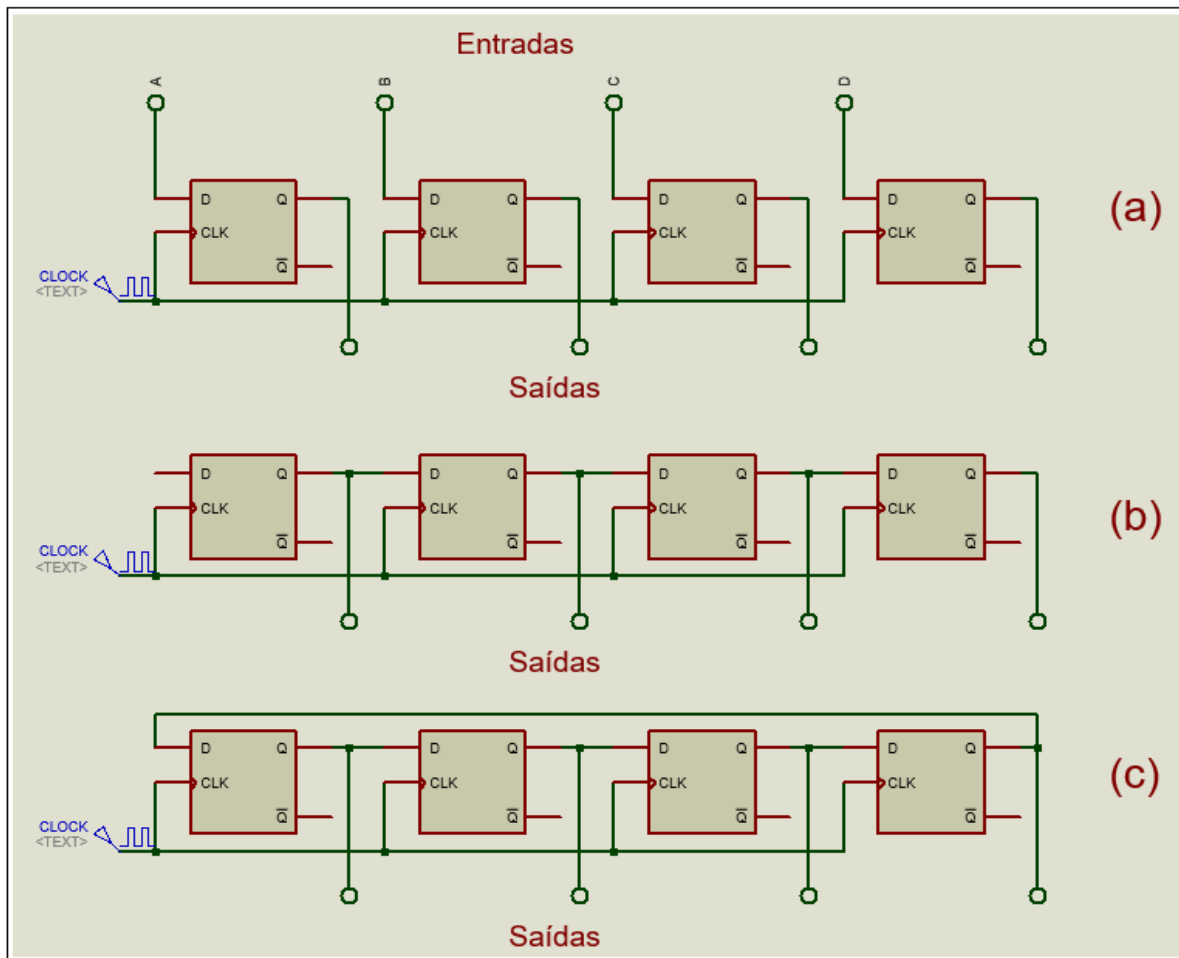


Figura 4 – Três modos de interligação dos *flip-flops* nos registradores: armazenamento básico (a), registrador de deslocamento (b), registrador de deslocamento em anel (c)

Fonte: Elaborada pelo autor, 2020.

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 é fornecida e 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 o 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.

A codificação a seguir ilustra a implementação, em Verilog, de um registrador de deslocamento em anel.

```
module RegDeslocAnel(rst,clk,D,Q,LS);

    input rst,clk,LS;
    input [7:0] D;

    output reg [7:0] Q;

    always @(posedge clk)
    begin
        if(rst) Q <= 8'b00000000;
        else if (LS == 1'b0) Q <= D;
        else Q <= {Q[0],Q[7:1]};
    end

endmodule // RegDeslocAnel
```

Na codificação acima temos, em Verilog, um registrador de deslocamento em anel (*right shift* – deslocamento para a direita). Os pinos “rst”, “clk” e “LS” são, respectivamente: “reset”, “clock” e “load/store”. No evento de transição positiva do sinal do *clock*, o valor de “Q” é instanciado em 0 caso o sinal de *reset* esteja ativado ou instanciado com o valor de “D” caso o pino “LS” esteja valendo 0. Em nenhuma destas condições, o deslocamento ocorre através da concatenação do *bit* Q[0] com a subpalavra Q[7:1].

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 utilizando-se tanto circuitos lógicos combinacionais quanto 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, você assista ao vídeo de (IFPB, 2018), disponível em: <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 funcionalidades. 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 configuração de registradores apresentados na figura 12?

5.3 Contadores binários assíncronos

Como mencionamos, para que possamos conversar sobre contadores, teremos que falar sobre dois novos tipos de *flip-flops*. Relembrando: *flip-flops* são componentes da lógica digital sequencial capazes de armazenar informações. Para que ocorra a carga do bit, os *flip-flops* são sensíveis às transições do sinal do *clock* (tanto de subida quanto de descida).

5.3.1 *Flip-flop* tipo “JK” e tipo “T”

Um *flip-flop* tipo “JK” lembra um pouco o *flip-flop* tipo “RS”, ou seja, os seus terminais desempenham as funcionalidades de “SET” e “RESET”. Porém não há a preocupação, como nos “RS”, de evitar que os dois terminais sejam ativados simultaneamente. A figura a seguir mostra a tabela-verdade e o diagrama esquemático do *flip-flop* tipo “JK”.

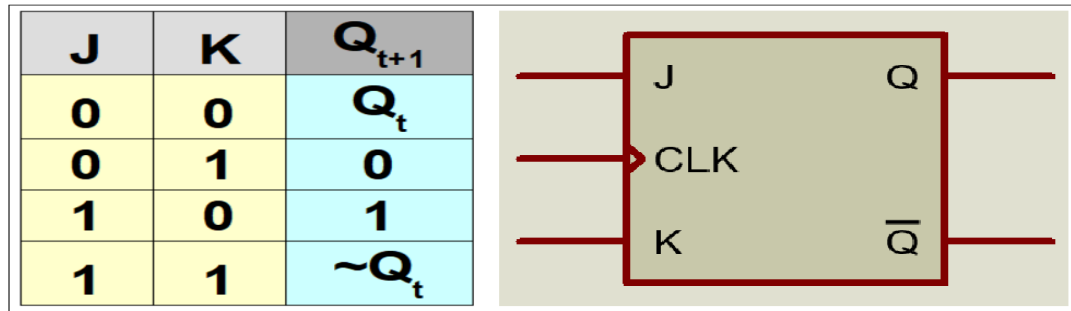


Figura 5 – Tabela-verdade e simbologia do *flip-flop* tipo “JK”. A saída “ Q_{t+1} ” reflete o momento futuro do valor armazenado em função do valor corrente no tempo “ t ” (valor “ Q_t ”).

Fonte: Elaborada pelo autor, 2020.

Como você pode observar na tabela-verdade apresentada na figura, quando os terminais “J” e “K” recebem, simultaneamente, o valor lógico “0”, o valor armazenado no *flip-flop* “JK” permanece inalterado mesmo no pulso do *clock*. Por sua vez, quando ativados isoladamente, os terminais “J” e “K” fazem com que o valor armazenado se torne “1” e “0”, respectivamente. A diferença do “JK” com o “RS” encontra-se na última linha. Enquanto que, no “RS” deve-se evitar essa combinação com ambos os terminais “R” e “S” no nível lógico “1”, no *flip-flop* “JK” essa combinação faz com que o valor armazenado seja complementado, ou seja, caso esteja armazenado o valor “1”, o próximo valor será “0” e vice-versa (IDOETA, 2012).

Você o conhece?

O *flip-flop* “JK” foi assim batizado em homenagem ao criador do circuito integrado: Jack Kilby, um engenheiro da Texas Instruments. Para saber um pouco da história dele, você poderá acessar o artigo escrito por (DINGMAN, 2013), que está disponível em: <https://pcworld.com.br/a-lenda-de-jack-kilby-e-os-55-anos-do-circuito-integrado/>.

Uma variação do *flip-flop* “JK” consiste no *flip-flop* “T”. O tipo “T” (*Toggle* – alternância) pode ser implementado a partir de um “JK” com os seus terminais interconectados. Assim, o *flip-flop* assume apenas duas possibilidades: ou mantém o valor armazenado ou o inverte. A figura a seguir mostra a implementação do *flip-flop* “T”, bem como sua tabela-verdade e um exemplo de uso. Confira!

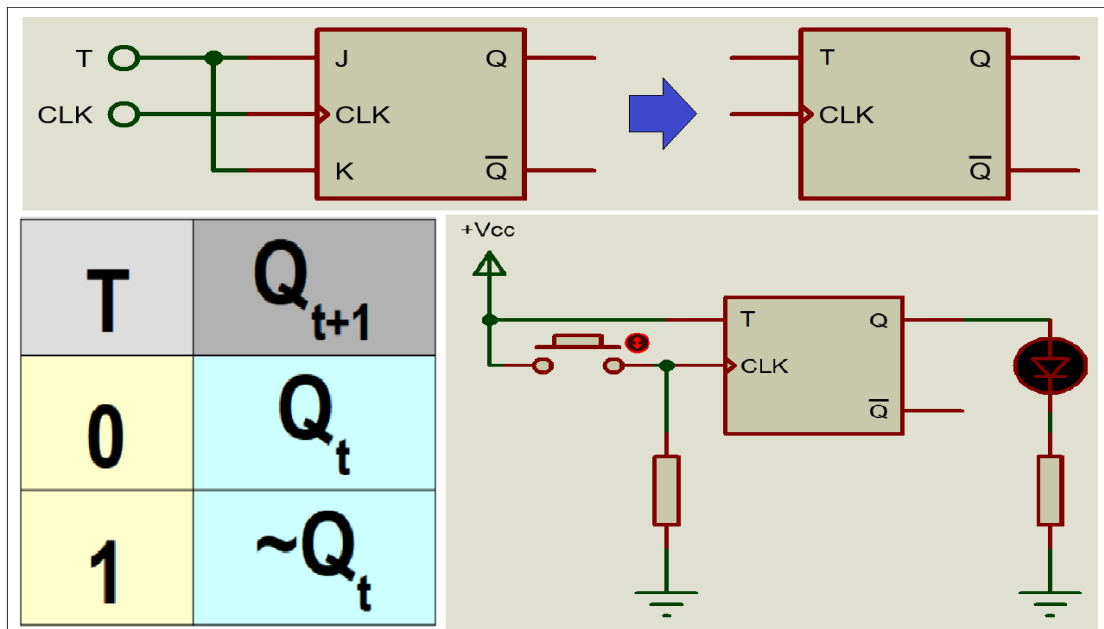


Figura 6 – Implementação de um *flip-flop* tipo “T” a partir de um tipo “JK” (parte superior), tabela-verdade e exemplo de aplicação do *flip-flop* tipo “T”.

Fonte: Elaborada pelo autor, 2020.

Na figura temos, na parte superior, a forma de implementação de um *flip-flop* tipo “T” a partir de um “JK”. Podemos notar que os terminais “J” e “K” estão conectados entre si, fazendo com que o tipo “T” tenha apenas duas possibilidades, expressas na tabela-verdade inserida na figura: ou mantém-se o valor armazenado (quando “T” for associado ao valor lógico “0”) ou inverte o valor armazenado (quando “T” receber o valor lógico “1”).

Para uma melhor abstração, temos, ainda na figura, uma aplicação envolvendo o *flip-flop* tipo “T”. Trata-se de uma chave “liga-desliga”. A cada toque no botão, um pulso é enviado ao *clock* do *flip-flop*, fazendo com que haja a alternância do estado do circuito controlado pelo *flip-flop* (no caso do exemplo, um LED). Assim, podemos ligar e desligar o circuito (por exemplo, um aparelho) ao pressionar um botão.

O código a seguir contém as implementações, em Verilog, dos *flip-flops* “JK” e “T”.

```

module FF_JK(J,K,Q,NQ,clk,rst);

    input J,K,clk,rst;
    output NQ;
    output reg Q;
    assign NQ = ~Q;

    always @(posedge clk)
    begin
        if(rst) Q <= 1'b0;
        else
            begin
                case ({J,K})
                    2'b00: Q <= Q;
                    2'b01: Q <= 1'b1;
                    2'b10: Q <= 1'b0;
                    2'b11: Q <= ~Q;
                endcase // case ({J,K})
            end
        end // always @ (posedge clk)

    endmodule // FF_JK

module FF_T(T,Q,NQ,clk,rst);

    input T,clk,rst;
    output NQ;
    output reg Q;

    assign NQ = ~Q;

    always @(posedge clk)
    begin
        if(rst) Q <= 1'b0;
        else if (T == 1'b1) Q <= ~Q;
    end

    endmodule // FF_T

```

Na codificação acima temos os *flip-flops* “JK” e “T” implementados em Verilog. Em ambos os casos, o sinal de *reset* (rst) é síncrono. Foi utilizado o “*assign*” para designar a saída “não Q” (denotado por NQ) pois podemos sempre considerá-la como o inverso da saída “Q”.

Mas como utilizar efetivamente o *flip-flop* “JK” ou “T” nos circuitos contadores? Conversaremos agora sobre esse assunto.

5.4 Contadores assíncronos

Como conversamos há pouco, os *flip-flops* tipo “JK” e tipo “T” têm a particularidade de permitir uma alternância do valor armazenado. Essa alternância será diretamente aproveitada nos

contadores. Para começarmos a falar sobre a relação entre esses *flip-flops* e o contador assíncrono, podemos adiantar que, se notarmos, uma contagem linear consiste na produção de valores consecutivos. Valores consecutivos se alternam entre positivos e negativos. Lembrando em representação binária, qual é o último bit (o menos significativo – último bit à direita) quando o número é par? E quando o número é ímpar? Já podemos começar a notar essa alternância: o último bit varia entre “0” e “1” para representar os valores pares e ímpares, respectivamente.

Você já começou a imaginar como fica a implementação? Podemos iniciar nossa implementação envolvendo um *flip-flop* “JK” ou “T” no bit menos significativo do contador. A cada pulso de *clock* (que poderá ser, por exemplo, a cada pessoa que passe em uma catraca), o valor é alternado, expressando os pares e ímpares. Mas como ficariam os demais bits da palavra sob contagem? Para melhor abstrairmos, vamos analisar a tabela-verdade expressa na figura a seguir. Na mesma figura, perceba, encontra-se também a implementação de um contador para uma palavra de três bits, ou seja, realiza a contagem do valor $000_{(2)}$ ($0_{(10)}$) até $111_{(2)}$ ($7_{(10)}$).

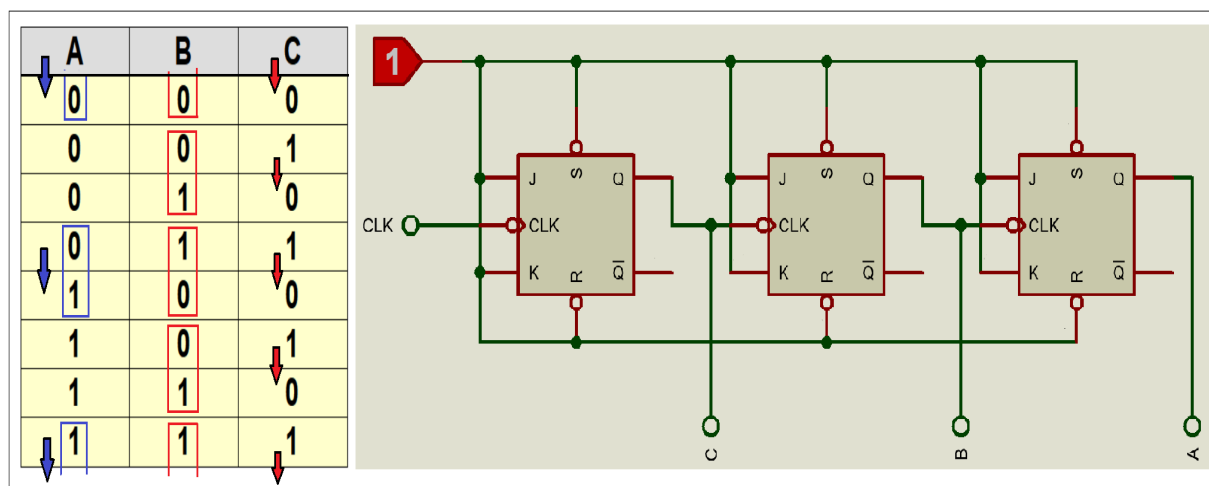


Figura 7 – Tabela de transições dos valores em um contador binário assíncrono e a respectiva implementação utilizando *flip-flops* “JK”.

Fonte: Elaborado pelo autor, 2020.

Na figura, percebemos que a mudança de um bit da palavra binária se dá apenas quando houver a transição de descida do bit de ordem imediatamente inferior. Por exemplo, o bit “B” apenas varia quando o bit “C” passa do nível lógico “1” para o nível lógico “0”. O mesmo se aplica ao bit “A”, que depende da transição de descida do bit “B”. Esse é o motivo de utilizarmos *flip-flops* com a lógica negativa no sinal do *clock* (o símbolo de inversão junto à entrada do *clock* sinaliza o uso da lógica negativa). Assim, os *flip-flops* apenas são ativados na transição de descida do sinal do *clock*. Quando ativado, o *flip-flop* “JK” (poderia ter sido usado o tipo “T”)

alterna seu valor armazenado devido ao fato de que seus terminais “**J**” e “**K**” estão conectados ao nível lógico “1”. No nível lógico “1”, estão também conectados todos os terminais “**S**” (set) e “**R**” (reset) dos *flip-flops*. Tais terminais, quando ativados, fazem com que o valor armazenado assuma o valor “1” ou “0”, respectivamente, independentemente do sinal do *clock* ou dos terminais “**J**” e “**K**”. Os terminais “**S**” e “**R**”, nesse caso, também operam na lógica negativa, ou seja, são ativados quando alimentados com o nível lógico “0”.

Você quer ver?

Comercialmente, existem diversas implementações de contadores binários por meio de circuitos integrados. Dentro das implementações, podemos encontrar aqueles que fazem tanto a contagem crescente quanto a contagem decrescente, em que a seleção da funcionalidade é feita por meio de um pino de controle. Para saber um pouco sobre uma implementação comercial de contador, você poderá assistir ao vídeo do canal Eletrônica Fácil (2015), disponível em: <https://www.youtube.com/watch?v=v2AW8mwLSg>.

Você deve estar se perguntando: “Mas e para realizar contagens decrescentes?”. Bem, para que a contagem seja realizada decrescentemente, basta realizar uma das seguintes opções:

- coletar os valores das saídas negadas: os valores decrescentes correspondem aos bits complementados dos valores crescentes;
- usar a lógica positiva no sinal do *clock*: pode-se, para realizar a contagem decrescente, utilizar a lógica positiva junto ao sinal do *clock*. Dessa forma, as alterações dos valores armazenados nos *flip-flops* ocorrerão na transição de subida do sinal de *clock*. Nesse caso, deve-se manter as saídas da palavra sob contagem atreladas aos pinos “**Q**” dos *flip-flops*.

Mas e se uma aplicação, como os segundos de um relógio, requerer uma contagem que não chega até a capacidade máxima de contagem do dispositivo? Como proceder para limitar uma contagem? Veremos isso a seguir.

Vamos praticar?

Um gerador de onda triangular pode ser obtido por intermédio da ligação de um conversor D/A (digital analógico) na saída de um contador binário crescente-decrescente. Implemente um contador binário que alterne, automaticamente, o seu modo de contagem, ou seja, em um certo momento, ele se comporta como um contador crescente (variando os valores, por exemplo, de 0

a 7) e, ao atingir o topo da contagem, ele passa a ser decrescente (variando os seus valores de 7 a 0). Essa alternância deve ser automática quando a contagem atingir o seu limite.

5.4.1 Limitação de contagem nos contadores assíncronos

Vamos supor que precisamos implementar o campo das unidades e das dezenas de segundos de um relógio. Sabemos que as unidades dos segundos variam de 0 até 9 e, por sua vez, o campo das dezenas dos segundos varia de 0 até 5. Inicialmente, podemos observar que, para expressar valores de 0 até 9, precisaremos de 4 bits. Porém, com os 4 bits, poderemos ter uma contagem de 0 até 15 ($2^4 - 1$). Assim, não podemos deixar com que a contagem chegue até o valor máximo, ou seja, devemos reiniciá-la antes. Mas como fazer isso? Nesse ponto, surge uma das aplicações dos sinais de “SET” e “RESET” dos *flip-flops*. Quando a contagem alcançar o primeiro valor indesejável (no caso, o valor 10), deve-se ativar os terminais “RESET” de todos os *flip-flops*, fazendo com que o valor armazenado volte a valer 0.

A figura a seguir mostra como limitar as contagens e, ainda, como estabelecer a relação entre a contagem das unidades e as dezenas dos segundos de um relógio.

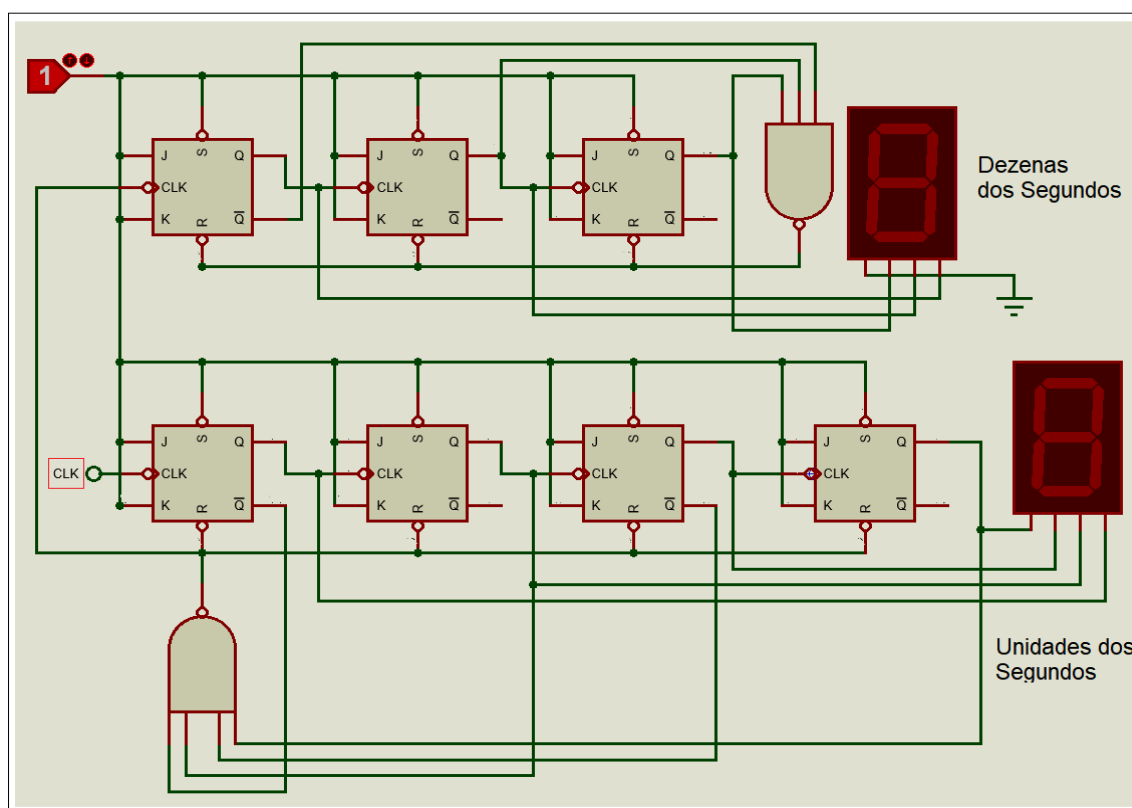


Figura 8 – Exemplificação da limitação de contagem dos contadores assíncronos na geração das dezenas e das unidades dos segundos em um relógio.

Fonte: Elaborada pelo autor, 2020.

Na figura, perceba que temos dois circuitos contadores assíncronos: um relativo à contagem de 0 a 9 (unidade dos segundos) e o outro responsável pela contagem de 0 a 5 (dezenas dos segundos). O *clock* externo, atribuído ao contador das unidades de segundos deve ser de 1 pulso por segundo (1 Hz). Por sua vez, o *clock* do contador das dezenas de segundos é o próprio sinal de “reset” do contador das unidades. A cada reciclagem da contagem das unidades, incrementa-se a dezena dos segundos.

Podemos notar, ainda em relação à figura anterior, que a limitação da contagem é realizada por intermédio de uma porta “NAND” (devido à lógica negativa do sinal de “reset” dos *flip-flops*, quando atingir um valor inválido, por exemplo, $6_{(10)}$ ($110_{(2)}$) no contador das dezenas, a porta “NAND” resultará no nível lógico “0”, ativando os terminais de “reset” e, consequentemente, zerando o valor armazenado no contador.

O processo de limitação da contagem pode ser aplicado para se obter divisão de frequência. No caso das unidades dos segundos, para que o sinal de “reset” seja emitido, são necessários 10 pulsos de *clock*. Dessa forma, temos um divisor de frequência por 10, também denominado como “divisor de década”.

Na codificação a seguir, ilustramos a implementação, em Verilog, de um contador de 4 *bits* limitado ao valor 5 (inclusive).

```
module Cont0_5(Q,rst,clk);  
  
    input  rst,clk;  
    output reg [3:0] Q;  
  
    initial  
    begin  
        Q = 0;  
    end  
  
    always @(posedge clk)  
    begin  
        if((rst) || (Q == 5)) Q <= 0;  
        else Q <= Q + 1;  
    end  
  
endmodule // Cont0_5
```

No código acima temos a implementação, em Verilog, de um contador de 4 *bits* de módulo 5. Podemos notar que a limitação pode ser realizada por intermédio de um comando condicional “*if*” e o incremento da contagem pela simples operação de soma.

Os contadores assíncronos são assim chamados pois cada *flip-flop* atua em uma

frequência distinta. O primeiro *flip-flop* atua na frequência “**F**”, o segundo já na frequência “**F/2**”, o terceiro em “**F/4**” e assim por diante. Uma limitação inerente aos contadores assíncronos consiste no tempo para que, dado um pulso de *clock*, tenhamos o valor de forma estável. Para a estabilidade do valor, os pulsos de *clock* devem ser propagados de *flip-flop* em *flip-flop*, causando lentidão na operação. Além disso, caso façamos a coleta do valor antes que todos os sinais de *clock* tenham sido propagados, podemos coletar um valor errôneo, representando, assim, um ruído do valor.

Para se resolver isso, podemos implementar um contador denominado “**contador síncrono**”, o qual abordaremos a seguir.

Vamos praticar?

Mencionamos que podemos limitar a contagem de um contador binário. Existe um divisor de frequência denominado como divisor de década. Tal divisor limita a sua contagem no valor 10. Implemente um contador com essa particularidade.

5.5 Contadores binários síncronos

Como mencionado, um contador assíncrono apresenta, como ponto negativo, a possibilidade de um valor errôneo de contagem durante a transição do sinal do *clock* entre os seus *flip-flops*. Para resolver esse problema, podemos fazer uso dos contadores binários síncronos.

5.5.1 Tabelas de transições de valores dos *flip-flops* do tipo “JK”

Os contadores síncronos são assim denominados como tal pelo fato de que todos os *flip-flops* integrantes recebem o sinal de *clock* ao mesmo tempo (VAHID; LASCHUK, 2008). Dessa forma, as suas ativações ocorrem simultaneamente. Mas, devido à ativação ao mesmo tempo, como definir o que cada *flip-flop* deve fazer? Para isso, cada entrada “**J**” e “**K**” deve ser associada a um circuito combinacional que fornece um valor de acordo com o estado atual de contagem. Isso possibilita, também, a geração de contagens não lineares, ou seja, valores de contagem não necessitam ser sequenciais.

Mas em que embasar a geração dos valores dos circuitos combinacionais cujas saídas serão atribuídas aos terminais “**J**” e “**K**” dos *flip-flops*? Para isso, teremos que olhar as transições

de cada bit da palavra referente à contagem entre cada linha subsequente. Antes de exemplificarmos esse processo, apresentaremos, na figura a seguir, as possibilidades de transições e os valores associados às entradas “J” e “K”.

0 → 1	1 → 0
J = 1	J = X
K = X	K = 1
0 → 0	1 → 1
J = 0	J = X
K = X	K = 0

Figura 9 – Valores das entradas “J” e “K” associados às transições relativas à contagem demandada. As transições devem ser observadas em cada bit, na sequência de linha por linha.

Fonte: Elaborada pelo autor, 2020.

E como usar as transições? Vamos, agora, como podemos, efetivamente, implementar as contadores síncronos.

5.5.2 Implementação dos contadores binários síncronos

Para que possamos conversar sobre como podemos implementar um contador binário síncrono, vamos supor que necessitamos construir um contador cuja contagem é: $0 \rightarrow 3 \rightarrow 2 \rightarrow 4 \rightarrow 5$. Como o valor máximo da contagem é 5, necessitamos de 3 bits; consequentemente, 3 *flip-flops*. A figura a seguir ilustra a tabela-verdade, contemplando as transições dos bits da palavra sob contagem, assim como os valores associados aos terminais “J” e “K” dos *flip-flops* constituintes do contador.

	A	B	C	Ja	Ka	Jb	Kb	Jc	Kc
0	0	0	0	0	X	1	X	1	X
3	0	1	1	0	X	X	0	X	1
2	0	1	0	1	X	X	1	0	X
4	1	0	0	X	0	0	X	1	X
5	1	0	1	X	1	0	X	X	1

Figura 10 – Tabela-verdade referente à contagem “ $0 \rightarrow 3 \rightarrow 2 \rightarrow 4 \rightarrow 5$ ”, destacando-se os valores de “Ja”, “Ka”, “Jb” e “Kb” associados às transições dos bits “A” e “B”.

Fonte: Elaborada pelo autor, 2020.

Na figura acima, temos a formação dos valores relativos aos terminais “Ja”, “Ka”, “Jb”, “Kb”, “Jc” e “Kc”, relativos às transições dos bits “A”, “B” e “C” que compõem a palavra sob contagem. Por exemplo, na transição do valor da contagem 0 para 3, temos no bit “A” a transição “0 → 0”. Assim, temos os valores “Ja” e “Ka”, relativos a essa transição, valendo “0” e “X”, respectivamente. Por sua vez, por exemplo, na transição dos valores de 2 para 4, temos, em “B”, a transição do valor lógico “1” para o valor lógico “0”. Dessa forma, “Jb” e “Kb”, para a referida transição, devem corresponder a “X” e “1”, respectivamente.

Extraindo todas as expressões e realizando o processo de simplificação, temos as seguintes expressões booleanas:

- $J_a = \sim A \cdot B \cdot \sim C$
- $K_a = A \cdot \sim B \cdot C$
- $J_b = \sim A \cdot \sim C$
- $K_b = \sim A \cdot \sim C$
- $J_c = \sim B \cdot \sim C$
- $K_c = A \oplus B$

Essas expressões poderão ser observadas na figura a seguir, que ilustra a implementação do contador síncrono, que realiza a sequência “0 → 3 → 2 → 4 → 5”.

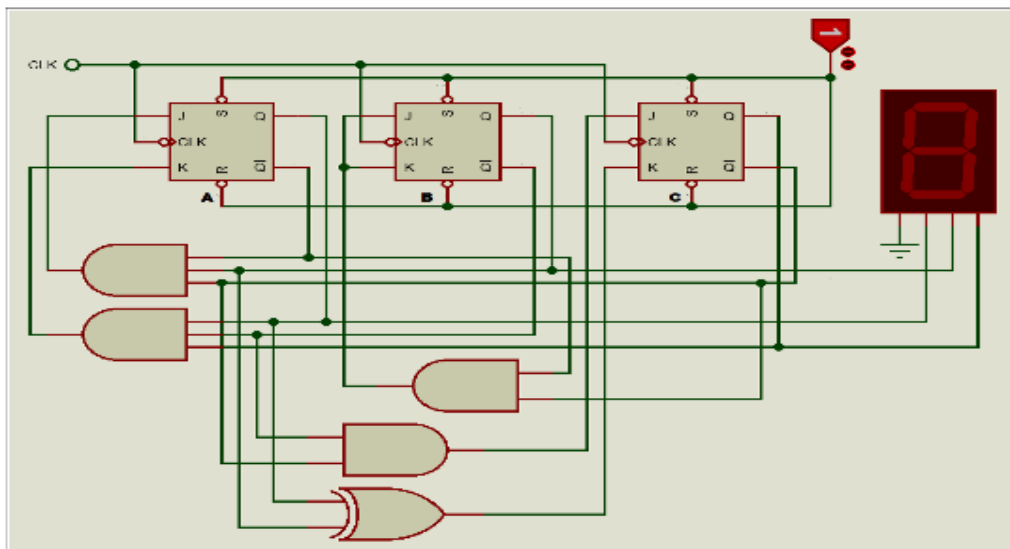


Figura 11 – Implementação do contador síncrono para a exibição, no display, da sequência “0 → 3 → 2 → 4 → 5”. Nota-se que, pelo fato do sinal do *clock* chegar simultaneamente a todos os *flip-flops*, a ordem dos bits de saída é irrelevante.

Fonte: Elaborada pelo autor. 2020.

Existem alguns números que não foram contemplados na sequência. No caso do exemplo anterior, temos os valores 1, 6 e 7. Para evitar transtornos em possíveis desvios de contagem (causados, por exemplo, por distúrbios externos ou fadiga dos próprios componentes utilizados), sugerimos adotar uma das três possibilidades relacionadas a seguir.

- Voltar imediatamente a um valor da contagem de forma assíncrona. Nesse caso, aciona-se imediatamente os terminais “PRESET” (ou “set”) e “CLEAR” (ou “reset”) dos *flip-flops*.
- Criar transições entre cada elemento fora da contagem para um valor da contagem. Por exemplo, no nosso caso: “1 → 0”, “6 → 0” e “7 → 0”. Nessa possibilidade, a volta à contagem determinada é realizada de forma síncrona (espera-se o pulso do *clock* para voltar à sequência correta).
- Criar uma sequência à parte envolvendo os elementos fora da contagem para que, no final, convirja para algum valor da contagem estabelecida. No nosso caso, teríamos, por exemplo: “7 → 6 → 1 → 0”. A volta à sequência correta, nesse caso, é também realizada de forma síncrona.

A codificação a seguir ilustra a implementação, em Verilog, do contador síncrono tomado como exemplo para realizar a contagem $0 \rightarrow 3 \rightarrow 2 \rightarrow 4 \rightarrow 5$.

```
module ContAssincrono(Q,rst,clk);
    input rst,clk;
    output reg [3:0] Q;

    initial
        begin
            Q = 0;
        end

    always @(posedge clk)
        begin
            if(rst) Q <= 0;
            else
                case(Q)
                    0: Q <= 3;
                    3: Q <= 2;
                    2: Q <= 4;
                    4: Q <= 5;
                    5: Q <= 0;
                    default: Q <= 0;
                endcase // case (Q)
            end
        endmodule // ContAssincrono
```

No código acima nota-se que a sequência de um contador síncrono pode ser realizada, para simplificar a implementação, através de um comando “*case*”. No caso do exemplo acima, caso seja assumido algum valor não válido, a contagem é reiniciada em 0 de forma síncrona.

Existem outras aplicações para utilizarmos os contadores síncronos? Veremos, a seguir, uma aplicação direta. Trata-se da manipulação de máquinas de estado.

Vamos praticar?

Conversamos que um contador síncrono permite realizar uma contagem com uma sequência não linear. Implemente, no caso, um contador para realizar a seguinte sequência: $0 \rightarrow 3 \rightarrow 2 \rightarrow 1 \rightarrow 0 \rightarrow 3 \dots$

5.6 Máquinas de Estados

De acordo com Tocci, Widmer e Moss (2018), máquinas de estados são circuitos com a capacidade de determinar um próximo estado em função do estado corrente e das variáveis de entrada. Mas o que são estados? Estados correspondem às funcionalidades implementadas – podemos, inclusive, fazer uma analogia com uma função se pensarmos no mundo da programação. Dessa forma, temos funções em execução – mas como passar de uma função para outra, ou seja, como passar de um estado para o outro? A passagem de estados é decorrente da ação de eventos. Um evento pode ser, por exemplo, a variação do valor de uma variável ou resposta de um sensor. Com essa breve conceituação, vamos ver na prática o que é uma máquina de estados e como podemos implementá-la?

5.6.1 Implementação de Máquinas de estados

Para que possamos dialogar sobre como implementar uma máquina de estados, vamos imaginar uma máquina automática para a venda de refrigerantes. Como estados podemos mencionar, por exemplo:

- Estado Inicial: Representa o estado de “*standby*”, ou seja, a máquina fica aguardando uma ação. A única ação plausível de acontecimento para sair do estado de “*standby*” poderá ser de inserção de dinheiro.
- Inserindo Dinheiro: Esse estado representa a ação de contabilização das quantias

depositadas na máquina de refrigerantes. A cada inserção ocorre a somatória do valor inserido para a totalização do saldo. Para sair deste estado, o usuário poderá pressionar o botão de cancelamento da compra ou o botão para a seleção do produto.

- **Liberando Produto:** Denota o procedimento para a liberação do produto selecionado, atuando sobre as partes mecânicas da máquina. Enquanto o produto ainda estiver em liberação, a máquina permanecerá neste estado. Para simplificar o nosso exemplo, não está sendo testado se o saldo inserido é suficiente para a compra do produto. A máquina finalizará esse estado somente quando a liberação do produto for concluída.
- **Devolvendo Troco:** O estado “Devolvendo Troco” envolve as ações para a devolução do saldo ainda disponível após a compra ou após o usuário ter solicitado o cancelamento da compra. A máquina permanecerá neste estado enquanto ainda tiver saldo a ser liberado. Após a liberação total da quantia devida ao usuário, a máquina retornará ao estado de “*standby*”.

Como podemos notar, os estados correspondem às ações que poderão ser realizadas pelo sistema.

Caso

Um projetista recebeu a incumbência de implementar um circuito para detectar uma sequência de dados recebidos por uma rede de computadores. Essa sequência é relacionada com o protocolo utilizado, de forma a identificar sinais de controle. Tratando-se de verificar sequência, ele logo percebeu que a melhor forma de se implementar é utilizando uma máquina de estados. Porém, ele sabe que existem dois modelos para se implementar: a máquina de Moore e a máquina de Mealy. Pesquisando, ele tomou conhecimento que, nas máquinas de Moore, as saídas são decorrentes apenas do estado atual, e, nas máquinas de Mealy, as saídas são decorrentes tanto do estado atual quanto dos valores assumidos pelas entradas. Tratando-se do recebimento de informações pela rede, ele ficou com receio de que as variações das informações das entradas pudessem afetar o valor da saída de forma assíncrona, ou seja, caso as entradas fossem modificadas, poderiam ser provocados ruídos nos valores obtidos na saída do circuito. Porém, por outro lado, as respostas geradas por uma máquina de Moore são mais lentas em relação à Mealy, que proporciona, também, uma implementação menos complexa e, consequentemente, mais barata em relação a Moore. Porém, após refletir sobre a questão, o projetista resolveu adotar o modelo de Moore em função da questão relacionada ao ruído que poderia aparecer na máquina de Mealy para esse tipo de aplicação.

A passagem de um estado para o outro ocorre na ocorrência de eventos. Como eventos, ainda utilizando a máquina automática para venda de refrigerantes, podemos citar algumas ações.

- **Dinheiro Inserido:** corresponde à ação de inserção de dinheiro e a respectiva passagem da cédula ou moeda no sensor que identifica o valor depositado.
- **Botão Seleciona Produto Pressionado:** ação decorrente do pressionamento, pelo usuário, do botão para a seleção de um produto.
- **Botão Cancelamento Pressionado:** denota a ação de pressionamento, pelo usuário, do botão para o cancelamento da operação de compra de refrigerantes.
- **Produto em Liberação:** evento que indica que o produto ainda não completou o seu curso para a liberação. Indicativo fornecido, por exemplo, do sensor de passagem do produto.
- **Produto Liberado:** evento que indica que o produto já foi liberado.
- **Ainda tem Saldo a Devolver:** evento associado ao saldo restante a ser devolvido ao usuário.
- **Saldo Zerado:** evento disparado quando o saldo estiver zerado.
- **Nada:** por se tratar de uma máquina síncrona, ou seja, sincronizada pelos pulsos de *clock*, o evento “nada” é associado quando nenhuma ação externa foi realizada no momento do pulso do *clock*.

Na máquina de estados, os estados são representados por nós (elipses ou circunferências) e os eventos são representados por arcos ou *links* (setas), interligando os estados. Juntando os estados e os eventos mencionados, podemos estabelecer a máquina completa, conforme ilustra a figura a seguir:

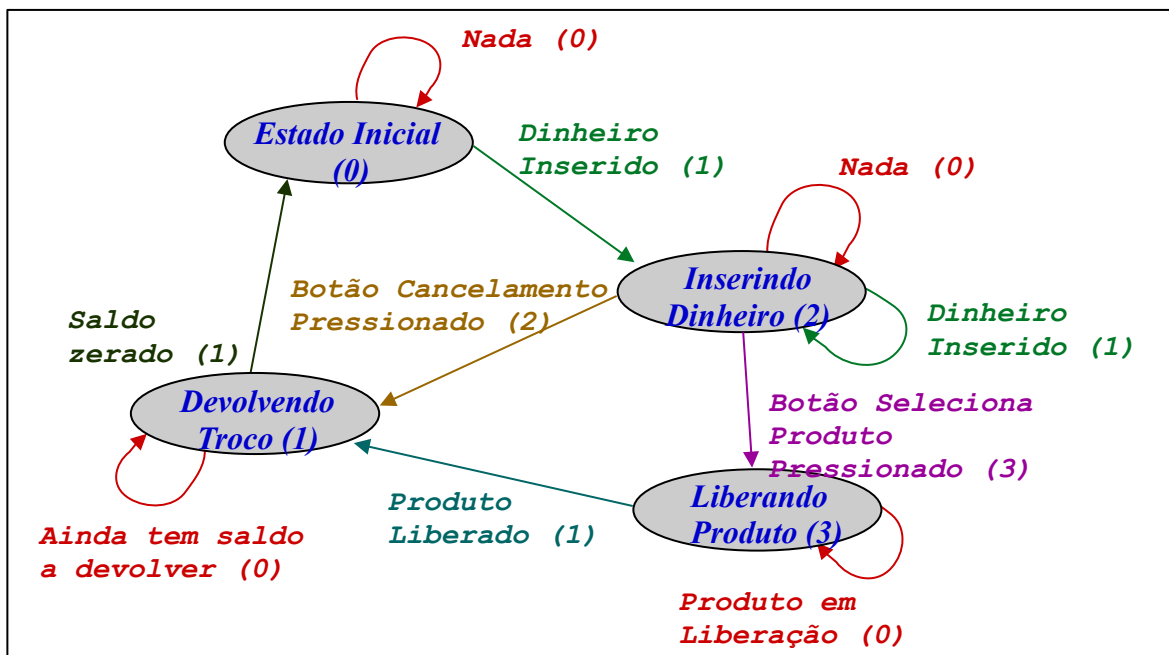


Figura 12 – Exemplificação de uma máquina de estados para a venda automática de refrigerantes. Os “nós” representam os estados e os arcs representam os eventos.

Fonte: Elaborada pelo autor, 2020.

Como ilustrado na figura, podemos notar que os eventos interligam os estados. Os eventos apenas atuam sobre o estado corrente da máquina de estados caso estiverem associados ao próprio estado corrente. Por exemplo, caso o usuário pressione o botão de seleção de produto enquanto a máquina estiver no estado inicial, nada acontecerá, pois o referido evento não está associado ao estado inicial.

Ainda em relação à figura anterior, podemos notar que tanto os estados quanto os eventos encontram-se numerados. A numeração dos eventos é reiniciada dentro de cada estado. A numeração se faz presente pelo fato de que, com ela, montaremos a nossa tabela de transições. Você percebe alguma aproximação com algum tópico já conversado neste capítulo? Se notar melhor, na máquina de estado, você verá uma sequência de valores, por exemplo, do estado “0” para o estado “3”. Dessa forma, podemos já identificar a sequência “0 → 3”. Ficou mais claro que estamos tratando de um circuito contador binário síncrono, certo? Mas como podemos montar a tabela completa contemplando os estados e os eventos?

Nesse caso, poderemos juntar a identificação dos estados com a identificação dos eventos, formando, portanto, uma palavra que corresponderá aos bits de entrada da tabela-verdade do contador síncrono. Mas quais serão os bits de saída? Para a questão da máquina de refrigerantes, desejamos obter, como saída, o próprio estado corrente.


Assim, construiremos uma tabela-verdade tendo, como entradas, os bits “ E_1 ” e “ E_0 ” para

denotar os estados, e também os bits “ V_1 ” e “ V_0 ” para representar os eventos. Tratando de um contador binário síncrono, as saídas da tabela-verdade corresponderão aos bits dos terminais “ J ” e “ K ” dos *flip-flops* “JK” associados aos bits dos estados. Podemos, então, notar que o estado atual influenciará na obtenção do novo estado.

Você sabia?

Você sabia que máquinas de estado não são implementadas somente a nível de *hardware*? Os sistemas computacionais baseados em *software* também podem ser modelados utilizando-se as máquinas de estado. Para saber mais, acesso o artigo escrito por Bertoleti (2015), disponível em: <https://www.embarcados.com.br/maquina-de-estado/>.

A figura a seguir contempla a tabela-verdade usada para a obtenção das transições relativas à máquina automática para venda de refrigerantes, realçando as transições presentes na máquina de estado.



E1	E0	V1	V0	JE1	KE1	JE0	KE0
0	0	0	0	0	X	0	X
0	0	0	1	1	X	0	X
0	1	0	0	0	X	X	0
0	1	0	1	0	X	X	1
1	0	0	0	X	0	0	X
1	0	0	1	X	0	0	X
1	0	1	0	X	1	1	X
1	0	1	1	X	0	1	X
1	1	0	0	X	0	X	0
1	1	0	1	X	1	X	0

Figura 13 – Tabela-verdade relativa às transições entre os estados para a obtenção dos valores dos terminais dos *flip-flops* representativos do estado corrente. As cores dos valores de cada “ J ” e “ K ” correspondem às cores das transições.

Fonte: Elaborada pelo autor, 2020.

Podemos notar, pela figura, que os valores relativos aos terminais “ J ” e “ K ” são obtidos observando-se as transições ocasionadas pelo estado atual, em conjunto com os eventos acontecidos. Por exemplo, a primeira linha da tabela-verdade refere-se ao “estado 0/evento 0”. Para essa combinação de estado/evento, tem-se a transição apontando para a própria linha.

Assim, tanto “**E1**” quanto “**E2**” possuem a transição “ $0 \rightarrow 0$ ”, resultando no valor lógico “0” para “**JE1**” e “**JE0**” e no valor lógico “X” para “**KE1**” e “**KE0**”.

Por sua vez, a segunda linha (estado 0 / evento 1) tem a sua transição para a quinta linha (estado 2). Portanto, “**E1**” tem uma transição “ $0 \rightarrow 1$ ” e, “**E0**”, uma transição “ $0 \rightarrow 0$ ”, resultando, portanto, nos valores “1”, “X”, “0”, “X” para os terminais “**JE1**”, “**KE1**”, “**JE0**” e “**KE0**”, respectivamente.

Realizando a extração das expressões a partir da tabela-verdade contida na figura, temos:

- $JE1 = \sim E0 \cdot \sim V1 \cdot V0$
- $KE1 = E0 \cdot \sim V1 \cdot V0 + E1 \cdot \sim E0 \cdot \sim V0$
- $JE0 = E1 \cdot \sim E0 \cdot V1$
- $KE0 = \sim E1 \cdot \sim V1 \cdot V0$

Notamos que necessitamos extrair as expressões apenas dos *flip-flops* relativos à geração do estado corrente (bits “**E1**” e “**E0**”), devido ao fato de que os bits “**V1**” e “**V0**” são variáveis de entrada, provenientes dos sensores da máquina automática de venda de refrigerantes. Assim, os sensores são multiplexados de forma que apenas aqueles relacionados com o estado atual possam ter os seus valores manipulados pelo contador binário síncrono.

Vamos praticar?

Mencionamos que uma máquina de estado tem por objetivo representar uma sequência de estados. Por sua vez, um contador também visa estabelecer uma sequência de estados que são os próprios valores sob contagem. Usando máquinas de estados, implementar um contador Gray de 2 bits. Lembrando que a contagem Gray segue a seguinte sequência: 00, 01, 11 e 10.

Síntese

Chegamos da nosso última unidade. Nesse nosso encontro, pudemos dialogar sobre circuitos, com a propriedade de realizar contagens, os chamados “**circuitos contadores binários**”. Pudemos conceituar e analisar as diferenças dos contadores binários assíncronos e os contadores binários síncronos. Pudemos, ainda, ver uma aplicação direta do contador síncrono que consiste na máquina de estados.

Dessa forma, você, com os conhecimentos aqui adquiridos, poderá implementar sistemas

lógicos digitais para a manipulação de palavras binárias, seja como elemento principal de processamento, seja como circuitos auxiliares ou de interfaceamento lógico. Os pontos aqui abordados também servirão de base para um entendimento de como funciona um sistema lógico digital, ou, ainda, servir como bagagem para a tomada de decisões nos momentos de sua implementação – quer usando circuitos integrados ou quer usando HDL (*Hardware Description Language* – Linguagem de Descrição de *Hardware*).

Aqui, você teve a oportunidade de:

- ter contato com a teoria relacionada com os contadores binários;
- saber diferenciar e aplicar soluções baseadas em contadores binários assíncronos e contadores binários síncronos;
- desenvolver máquinas de estado;
- implementar sistemas lógicos digitais que envolvam contadores binários.

Bibliografia

BERTOLETI, P. **Máquina de Estado**. Publicado em 07/08/2015. Disponível em <<https://www.embarcados.com.br/maquina-de-estado/>>. Acessado em 14/12/2020.

DINGMAN, H. **A lenda de Jack Kilby e os 55 anos do circuito integrado**. Publicado em 13/09/2013. Disponível em <<https://pcworld.com.br/a-lenda-de-jack-kilby-e-os-55-anos-do-circuito-integrado/>>. Acessado em 14/12/2020.

ELETRÔNICA FÁCIL. **Eletrônica Digital 33** - Contador crescente e decrescente usando o CI 4510. Publicado em 12/05/2015. Disponível em <<https://www.youtube.com/watch?v=v2AW8mwxLSg>>. Acessado em 14/12/2020.

IDOETA, I.V.; CAPUANO, F. G. **Elementos de Eletrônica Digital**. 42 Ed. São Paulo: Érica, 2019. Disponível na Minha Biblioteca. Acessível via Minha UFOP – Biblioteca Digital).

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).