

IMPLEMENTANDO FLIP-FLOPS E REGISTRADORES DE DESLOCAMENTO EM VHDL

1 CIRCUITOS SEQUENCIAIS

Circuitos digitais podem ser divididos em duas categorias: os circuitos combinacionais e os circuitos sequenciais (Figura 1).

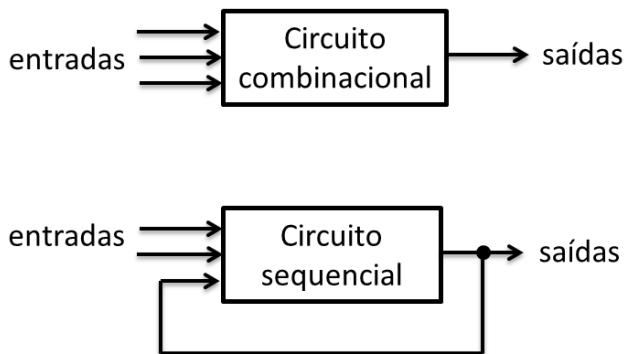


Figura 1 – Diferença entre circuitos combinacionais e circuitos sequenciais.

Circuitos combinacionais são aqueles em que as saídas dependem somente das entradas. Mudando-se as entradas, mudam-se as saídas. Mas, voltando-se as entradas para um valor anterior, a saída também volta para o mesmo resultado anterior. A sequência com que os valores de entrada foram sendo alterados até o momento presente não afeta de forma alguma a saída atual do circuito. Alguns exemplos de circuitos combinacionais são: somas de mintermos, somador parcial, somador completo, somadores de palavras binárias, multiplexadores, decodificadores e detectores de paridade.

Circuitos sequenciais são aqueles em que as saídas dependem não só das entradas, mas também das saídas anteriores. Assim, o valor da saída atual não depende só da entrada atual, mas também da sequência com a que os valores de entrada foram sendo alterados até o momento presente. É possível que uma entrada “010” produza uma saída “0” em determinado momento, mas, depois de algumas alterações nos valores de entrada, voltar a entrada para “010” produza uma saída “1”. Diz-se, portanto, que o circuito tem “memória”. Isso é implementado por meio do que chamamos de realimentação: uma ou mais saídas do circuito é conectada (ou realimentada) junto às entradas do circuito. Alguns exemplos de circuitos sequenciais são: latches, flip-flops, registradores de deslocamento, contadores e máquinas de estados.

2 FLIP-FLOPS

Um flip-flop pode ser considerado como uma memória de 1 bit. O flip-flop tem terminais que permitem, por meio de determinada combinação, mudar o valor da saída para “0” ou “1”. O valor da saída é mantido em “0” ou “1” mesmo após a combinação de entrada ter sido alterada, e permanecerá até que se use a combinação de entrada apropriada para trocar novamente o valor da saída.

Flip-flops são as unidades básicas com que se constroem memórias. Um flip-flop é construído por meio de um arranjo de portas lógicas, no qual as saídas de algumas das portas lógicas são realimentadas nas entradas de outras portas lógicas em camadas anteriores do circuito. Um conjunto de 8 flip-flops é uma memória de 8 bits, ou seja, de 1 byte. Com 8192 flip-flops é possível construir uma memória de 8 kbits, ou seja, de 1 kB. Para se construir uma memória de 1 GB são necessários 8.589.934.592 flip-flops (mais de 8 bilhões de flip-flops!)

Flip-flops costumam ter um terminal de entrada chamado de “clock” ou “relógio”. O sinal de clock permite controlar quando as entradas dos demais terminais serão levadas em consideração. Por exemplo, em alguns flip-flops é possível alterar a saída somente quando o clock é igual a 1; em outros, isso acontece somente na borda de subida do clock, isto é, na transição de “0” para “1” do clock.

Os demais terminais de entrada de um flip-flop podem ser classificados entre síncronos e assíncronos. Os terminais síncronos são aqueles que só tem efeito se determinada condição do clock for satisfeita (por exemplo, só tem efeito na borda de subida do clock). Os terminais assíncronos são aqueles que têm efeito independentemente do clock.

Os terminais assíncronos mais comuns são o “preset” e o “clear”. O terminal preset altera a saída do flip-flop para “1” de forma independente do clock. O terminal clear altera a saída do flip-flop para “0” de forma independente do clock. Note que, em alguns circuitos, os terminais de preset e clear podem funcionar de forma síncrona, isto é, podem depender do clock para ter efeito.

A escrita síncrona em um flip-flop é feita por meio dos seus terminais síncronos. Os nomes desses terminais mudam de acordo com o tipo de flip-flop. Por exemplo, os flip-flops RS têm como entradas síncronas os terminais R e S; os flip-flops JK têm como entradas síncronas os terminais J e K; etc.

2.1 Flip-flop tipo D

Um flip-flop tipo D é aquele em que a escrita síncrona é feita por meio de um terminal de entrada (“D”) que funciona da seguinte forma: se D = “0” quando a leitura está habilitada (na borda de subida do clock, por exemplo), então a saída (“Q”) vai para “0”; se D = “1”, então Q vai para “1”.

Um exemplo de flip-flop D é mostrado na Figura 2 e sua tabela verdade é mostrada na Tabela 1. Note que, nesse flip-flop, a entrada clear é assíncrona (funciona independentemente do clock), enquanto que a entrada D, que é síncrona, só tem efeito na borda de subida do clock. Quando o clock está parado em “0” ou “1”, ou mesmo em sua borda de descida, a entrada D é desconsiderada e o flip-flop simplesmente mantém a sua saída atual.

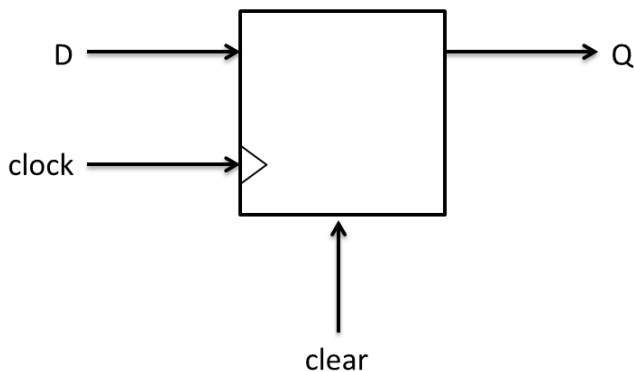




Figura 2 – Diagrama de blocos de um flip-flop D.

Tabela 1 – Tabela verdade de um flip-flop D.

entradas			saída
Clear	clock	D	Q
1	x	x	0
0		0	0
0		1	1
0	outros	x	mantém

Um exemplo de código VHDL que implementa o flip-flop D descrito pela Tabela 1 é mostrado na Figura 3. Note que a entrada D não foi incluída na lista de sensibilidade do process, pois a saída só será atualizada se clear ou clock mudarem. A condição “borda de subida” do clock foi implementada usando o comando `rising_edge()`, que indica se houve uma transição do sinal de “0” para “1”.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity flipflopD is
5      Port ( D,clk,clr : in  STD_LOGIC;
6            Q : out  STD_LOGIC);
7  end flipflopD;
8
9  architecture flipflopD_arch of flipflopD is
10 begin
11     process(clr,clk)
12     begin
13         if clr = '1' then Q <= '0';
14         elsif rising_edge(clk) then Q <= D;
15         end if;
16     end process;
17 end flipflopD_arch;
```

Figura 3 – Código VHDL que implementa o flip-flop D descrito na Tabela 1.

2.2 Flip-flop tipo T

Um flip-flop tipo T é aquele em que a escrita síncrona é feita por meio de um terminal de entrada (“T”) que funciona da seguinte forma: se T = “0” quando a leitura está habilitada (na borda de subida do clock, por exemplo), então a saída (“Q”) mantém seu estado atual; se T = “1”, então Q troca seu estado atual (de “0” para “1” ou de “1” para “0”).

Um exemplo de flip-flop T é mostrado na Figura 4 e sua tabela verdade é mostrada na Tabela 2. Note que, neste exemplo, a entrada clear foi omitida; a entrada T é síncrona, pois só tem efeito na borda de subida do clock. Quando o clock está parado em “0” ou “1”, ou mesmo em sua borda de descida, a entrada T é desconsiderada e o flip-flop simplesmente mantém a sua saída atual.

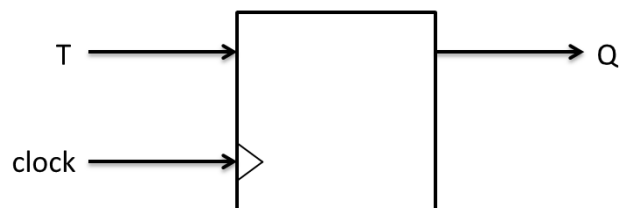




Figura 4 – Diagrama de blocos de um flip-flop D.

Tabela 2 – Tabela verdade de um flip-flop T.

entradas		saída
clock	T	Q
	0	mantém
	1	inverte
outros	x	mantém

Um exemplo de código VHDL que implementa o flip-flop T descrito pela Tabela 2 é mostrado na Figura 5. Note que a entrada T não foi incluída na lista de sensibilidade do process, pois a saída só será atualizada se clock mudar. A condição “borda de subida” do clock foi novamente implementada usando o comando `rising_edge()`, que indica se houve uma transição do sinal de “0” para “1”.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity flipflopT is
5      Port ( T,clk : in  STD_LOGIC;
6            Q : out  STD_LOGIC);
7  end flipflopT;
8
9  architecture flipflopT_arch of flipflopT is
10     signal Qbuf : STD_LOGIC;
11     begin
12         process(clk)
13         begin
14             if rising_edge(clk) then
15                 if T = '1' then Qbuf <= not(Qbuf);
16                 else Qbuf <= Qbuf; -- opcional
17                 end if;
18             end if;
19         end process;
20         Q <= Qbuf;
21     end flipflopT_arch;
```

Figura 5 – Código VHDL que implementa o flip-flop T descrito na Tabela 2.

No exemplo da Figura 5, é preciso chamar a atenção para a forma com que a troca de estado foi implementada (linha 15). Na implementação do flip-flop D, fizemos simplesmente “`Q <= D;`” (ver Figura 3, linha 14). Aqui, gostaríamos de fazer “`Q <= not(Q);`”. Entretanto, esse comando geraria uma mensagem de erro durante a verificação de sintaxe, algo como “`identifier Q is not readable`”. Isso acontece porque Q foi definido como uma saída do circuito, portanto não é possível ler seu valor, somente escrever nele; isto é, Q só pode aparecer do lado esquerdo de uma atribuição, nunca do lado direito. Para contornar esse problema, criamos um sinal Qbuf (Figura 5, linha 10) para atuar como um Q “intermediário”. Como sinais podem ser usados tanto para ler quanto para escrever, estes podem aparecer tanto do lado direito quanto do lado esquerdo de uma atribuição, evitando a mensagem de erro. Fazemos todas as operações usando Qbuf e, na linha 20, fazemos “`Q <= Qbuf`”.

Note que a atribuição “`Q <= Qbuf`” poderia ter sido feita antes da estrutura process, sem prejuízo para o funcionamento do circuito, pois é uma atribuição concorrente que acontece concomitantemente com as atribuições dentro do process. Poderia

ainda ter sido movida para dentro da estrutura process, sem nenhum prejuízo.

Com relação à atribuição na linha 15 da Figura 5: “`Qbuf <= not(Qbuf);`”, note que esse comando, se escrito fora de uma estrutura process, não seria sintetizável, pois descreveria uma porta inversora no qual a saída está curto-circuitada com sua entrada, gerando uma contradição: $A = \bar{A}$. Entretanto, quando essa atribuição aparece dentro de um process, após um comando “`if rising_edge...`”, o sintetizador interpreta o lado esquerdo da atribuição como sendo a saída de um flip-flop tipo D e o lado direito como sendo sua entrada (Figura 6). O comando, portanto, faz sentido e é sintetizável.

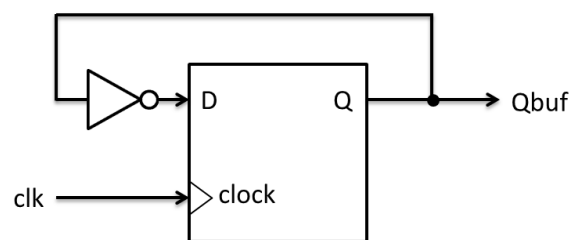


Figura 6 – Diagrama de blocos que representa a forma como o comando “`Qbuf <= not(Qbuf);`” é interpretado pelo sintetizador quando essa atribuição aparece dentro de um process, após um comando “`if rising_edge(clk)`”. O sintetizador interpreta o lado esquerdo da atribuição como sendo a saída de um flip-flop D e o lado direito como sendo sua entrada.

Na verdade, uma interpretação possível para a estrutura process do exemplo mostrado na Figura 5 é uma em que a entrada de um flip-flop D é conectada a saída de um multiplexador 2 para 1, em que a entrada de seleção é T e as entradas de dados são Qbuf e \bar{Qbuf} , respectivamente (Figura 7), pois note que, quando T = “0”, Qbuf receberá Qbuf e, quando T = “1”, Qbuf receberá \bar{Qbuf} .

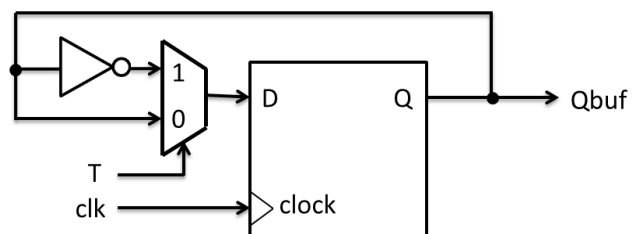


Figura 7 – Diagrama de blocos que representa a forma como a estrutura process no código VHDL mostrado na Figura 5 pode ser interpretada pelo sintetizador: um multiplexador 2 para 1 em série com um flip-flop tipo D. Note que, quando T = “0”, a saída receberá Qbuf e, quando T = “1”, esta receberá \bar{Qbuf} .

Por fim, note que a linha 16 do código VHDL mostrado na Figura 5 é opcional. Excluir essa linha do código não alteraria em nada o funcionamento do circuito implementado, pois, se a estrutura process não alterasse o valor do sinal Qbuf, este simplesmente manteria seu valor atual, que é exatamente o que a atribuição na linha 16, “Qbuf <= Qbuf;”, tenta fazer.

Uma forma alternativa de implementar este flip-flop T seria declarando Qbuf como uma “variável” ao invés de um “sinal” (Figura 8). Note que, como variáveis só existem dentro do process onde foram declaradas, a atribuição “Q <= Qbuf” teve que ser movida para dentro do process (linha 19). Note ainda que os símbolos de atribuição para Qbuf foram trocados de “<=” para “:=” (linhas 15 e 16), por se tratar agora de uma variável, e não de um sinal. Entretanto, sempre que possível, recomenda-se usar sinais ao invés de variáveis quando se pretende implementar o circuito em FPGA, pois variáveis são, muitas vezes, não sintetizáveis, enquanto que sinais geralmente representam fios no circuito sintetizado.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity flipflopT is
5      Port ( T,clk : in  STD_LOGIC;
6            Q : out  STD_LOGIC);
7  end flipflopT;
8
9  architecture flipflopT_arch of flipflopT is
10 begin
11     process(clk)
12         variable Qbuf : STD_LOGIC;
13     begin
14         if rising_edge(clk) then
15             if T = '1' then Qbuf := not(Qbuf);
16             else Qbuf := Qbuf; -- opcional
17             end if;
18         end if;
19         Q <= Qbuf;
20     end process;
21 end flipflopT_arch;
```

Figura 8 – Código VHDL que implementa o flip-flop T descrito na Tabela 2, desta vez declarando Qbuf como uma “variável” ao invés de como um “sinal”.

Embora os códigos da Figura 5 e da Figura 8 sejam sintetizáveis e funcionem normalmente em um FPGA, sua simulação não funcionará. Isso acontece pois o estado inicial do flip-flop (de Qbuf, no caso) nunca foi definido. Para fins de simulação, é necessário inicializar Qbuf ainda na declaração, fazendo

```
signal Qbuf : STD_LOGIC := '0';
ou então
variable Qbuf : STD_LOGIC := '0';
```

Inicializações, como as acima, não são sintetizáveis, sendo simplesmente ignoradas pelo sintetizador quando o circuito está sendo implementado em um FPGA, não tendo efeito algum no circuito físico.

2.3 Flip-flop tipo JK

Um flip-flop tipo JK é aquele em que a escrita síncrona é feita por meio de dois terminais de entrada (“J” e “K”) que funciona da seguinte forma:

- se J = “0” e K = “0” quando a leitura está habilitada (na borda de subida do clock, por exemplo), então a saída (“Q”) mantém seu estado atual;
- se J = “0” e K = “1” quando a leitura está habilitada, então Q muda seu estado para “0”;
- se J = “1” e K = “0” quando a leitura está habilitada, então Q muda seu estado para “1”;
- se J = “1” e K = “1” quando a leitura está habilitada, então Q troca seu estado atual (de “0” para “1” ou de “1” para “0”).

Um exemplo de flip-flop JK é mostrado na Figura 9 e sua tabela verdade é mostrada na Tabela 3. Note que, neste exemplo, as entradas clear e preset são assíncronas (funcionam independentemente do clock), enquanto que as entradas J e K, que são síncronas, só têm efeito na borda de subida do clock. Quando o clock está parado em “0” ou “1”, ou mesmo em sua borda de descida, as entradas J e K são desconsideradas e o flip-flop simplesmente mantém a sua saída atual. Note ainda que, neste exemplo, a entrada preset tem precedência sobre a entrada clear, isto é, quando preset é “1” a entrada clear é simplesmente ignorada. De modo geral, a situação em que tanto preset quanto clear estão em “1” deve ser evitada pelo usuário do flip-flop, pois consiste em uma contradição.

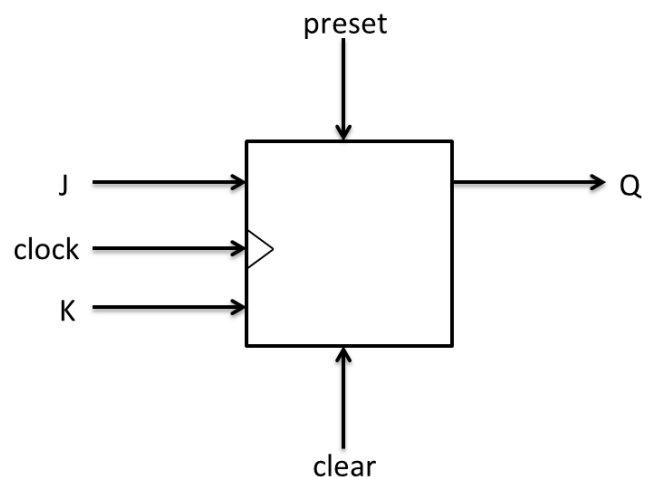






Figura 9 – Diagrama de blocos de um flip-flop JK.

Tabela 3 – Tabela verdade de um flip-flop JK.

entradas					saída
preset	clear	clock	J	K	Q
1	x	x	x	x	1
0	1	x	x	x	0
0	0		0	0	Mantém
0	0		0	1	0
0	0		1	0	1
0	0		1	1	inverte
0	0	outros	x	x	mantém

Não apresentaremos aqui o código VHDL que implementa o flip-flop JK descrito pela Tabela 3 e mostrado na Figura 9. Mas note que este pode ser implementado facilmente usando os conceitos apresentados nos exemplos anteriores (flip-flops D e T). É preciso atenção com relação aos comandos preset e clear, de modo a garantir que o preset tenha precedência ao clear, conforme especificado.

Com relação aos comandos J e K, é possível implementar a lógica usando a estrutura “if-elsif-else”, como nos exemplos anteriores, ou mesmo usando a estrutura “case-when”, conforme exemplificado na Figura 10. No exemplo, note que, como “J” e “K” são entradas de 1 bit independentes, é preciso criar um sinal (vetor “JK” de 2 bits) para usar como entrada da estrutura “when-case”.

```
architecture ...
...
    signal JK : STD_LOGIC_VECTOR(1 downto 0);
begin
    JK <= J & K;
    process(pr,clr,clk)
    begin
        if ...
        elsif ...
        elsif rising_edge(clk) then
            case JK is
                when "00" => ...
                when "01" => ...
                when "10" => ...
                when "11" => ...
                when others => ...
            end case;
        ...
        end if;
    end process;
    ...
end ...
```

Figura 10 – Descrição VHDL de um flip-flop JK usando a estrutura “when-case”. Note que, como “J” e “K” são entradas de 1 bit independentes, é preciso criar um sinal (vetor “JK” de 2 bits) para usar como entrada da estrutura “when-case”.

3 REGISTRADORES DE DESLOCAMENTO

Um registrador de deslocamento é um circuito que implementa uma memória de dois ou mais bits (uma palavra binária), de modo que os bits dessa palavra binária possam ser deslocados para a direita e(ou) para a esquerda, à medida em que novos bits são inseridos nessa palavra binária de forma serial (isto é, um bit de cada vez). A entrada serial normalmente é controlada por meio de um sinal de clock, de modo que um novo bit é inserido na palavra binária a cada ciclo de clock (por exemplo, na borda de subida do clock). Em um registrador de deslocamento ativado pela borda de subida do clock, por exemplo, quando o clock está parado em “0” ou “1”, ou mesmo em sua borda de descida, o registrador de deslocamento simplesmente mantém a sua saída atual.

Registradores de deslocamento podem ser unidirecionais — em que os deslocamentos acontecem somente em uma direção, de modo que novos bits são inseridos sempre exclusivamente pela direita ou exclusivamente pela esquerda — ou bidirecionais — em que um pino de entrada indica se o deslocamento deve acontecer para a direita ou para a esquerda e, portanto, novos bits podem ser inseridos tanto no bit menos significativo da palavra binária quanto no bit mais significativo.

Além da(s) entrada(s) serial(is) — em que novos bits são inseridos um a um, seja pela direita ou pela esquerda — registradores de deslocamento podem ter também uma entrada paralela, que permite inserir a palavra binária inteira, todos os bits de uma vez. Essa escrita paralela acontece quando um pino de controle (normalmente chamado de “load”) é habilitado, e pode ser síncrona (a escrita acontece na borda de subida do clock, por exemplo) ou assíncrona (por exemplo, acontece sempre que load é igual a “1”, independentemente do estado do clock).

O registrador de deslocamento pode ainda ter um pino que, quando habilitado, zera todos os bits da palavra binária. Esse pino, normalmente chamado de “reset”, também pode funcionar de maneira síncrona (o reset acontece na borda de subida do clock, por exemplo) ou assíncrona (por exemplo, acontece sempre que clear é igual a “1”, independentemente do estado do clock).

Um exemplo de registrador de deslocamento de 4 bits é o descrito pela Tabela 4. Note que esse registrador é bidirecional, sendo que a escrita serial acontece no bit menos significativo por meio da entrada L e no bit mais significativo por meio da entrada R. Quem controla a direção do deslocamento é a entrada “dir”: quando dir é igual a “0”, o

deslocamento acontece para a esquerda (portanto, a entrada serial habilitada é L e a escrita é feita no bit menos significativo); quando dir é igual a “1”, o deslocamento acontece para a direita (portanto, a entrada serial habilitada é R e a escrita é feita no bit mais significativo). Esse registrador tem ainda uma entrada de dados paralela e síncrona D (que por sua vez é habilitada pela entrada “load”) e uma entrada de “reset” síncrona. Por fim, note que o

procedimento de reset tem prioridade sobre o procedimento de escrita paralela, o qual, por sua vez, tem prioridade sobre o procedimento de escrita serial. Assim, a escrita serial e (consequentemente) os deslocamentos só acontecem quando tanto reset quando load forem iguais a “0”. O funcionamento desse registrador de deslocamento é ilustrado na Figura 11.

Tabela 4 – Tabela verdade de um registrador de deslocamento bidirecional de 4 bits com “reset” síncrono e entrada paralela síncrona.

entradas							saída
clock	reset	Load	D	dir	L	R	Q
	1	X	xxxx	x	x	x	0000
	0	1	D ₃ D ₂ D ₁ D ₀	x	x	x	D ₃ D ₂ D ₁ D ₀
	0	0	xxxx	0	0	x	Q ₂ Q ₁ Q ₀ 0
	0	0	xxxx	0	1	x	Q ₂ Q ₁ Q ₀ 1
	0	0	xxxx	1	x	0	0 Q ₃ Q ₂ Q ₁
	0	0	xxxx	1	x	1	1 Q ₃ Q ₂ Q ₁
outros	x	X	xxxx	x	x	x	Q ₃ Q ₂ Q ₁ Q ₀

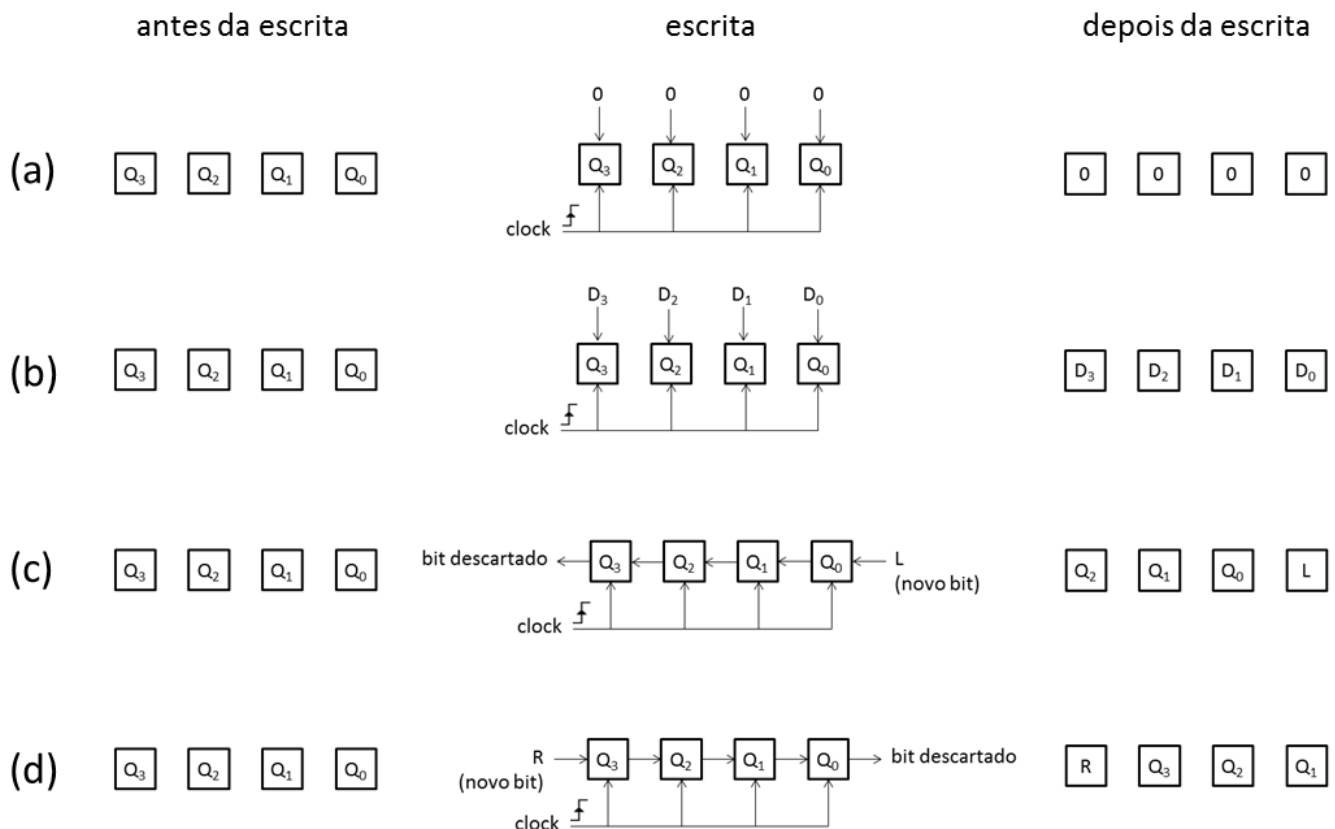




Figura 11 – Funcionamento de um registrador de deslocamento bidirecional com reset síncrono e entrada paralela síncrona, mostrando o estado do registrador antes e depois do procedimento de escrita: (a) procedimento de reset; (b) procedimento de escrita paralela; (c) procedimento de escrita serial no bit menos significativo (deslocamento para a esquerda); (d) procedimento de escrita serial no bit mais significativo (deslocamento para a direita).

Para ilustrar a implementação de um registrador de deslocamento em linguagem VHDL, vamos utilizar como exemplo um registrador de deslocamento mais simples: unidirecional e sem entrada de clear e sem escrita paralela. A implementação de um registrador de deslocamento mais complexo é uma simples extensão deste exemplo. O funcionamento desse registrador é descrito na Tabela 5.

Tabela 5 – Tabela verdade de um registrador de deslocamento unidirecional de 4 bits.

entradas		saída
clock	L	Q
	0	$Q_2Q_1Q_0\ 0$
	1	$Q_2Q_1Q_0\ 1$
outros	x	$Q_3Q_2Q_1Q_0$

Um exemplo de código VHDL que implementa o registrador de deslocamento descrito na Tabela 5 é mostrado na Figura 12. Note que a entrada L não foi incluída na lista de sensibilidade do process, pois a saída só será atualizada se clock mudar. A condição “borda de subida” do clock foi implementada usando o comando `rising_edge()`, que indica se houve uma transição do sinal de “0” para “1”.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity shiftreg is
5      Port ( L,clk : in  STD_LOGIC;
6            Q : out  STD_LOGIC_VECTOR(3 downto 0));
7  end shiftreg;
8
9  architecture shiftreg_arch of shiftreg is
10     signal Qbuf : STD_LOGIC_VECTOR(3 downto 0);
11 begin
12     process(clk)
13     begin
14         if rising_edge(clk) then
15             Qbuf <= Qbuf(2) & Qbuf(1) & Qbuf(0) & L;
16         end if;
17     end process;
18     Q <= Qbuf;
19 end shiftreg_arch;
```

Figura 12 – Código VHDL que implementa o registrador de deslocamento unidirecional de 4 bits descrito na Tabela 5.

No exemplo da Figura 12, é preciso chamar a atenção para a forma com que a troca de estado foi implementada (linha 15). Lembre-se que Q foi definido como uma saída do circuito, portanto não é possível ler seu valor, somente escrever nele; isto é, Q só pode aparecer do lado esquerdo de uma atribuição, nunca do lado direito. Para contornar esse problema, criamos um sinal Qbuf (linha 10) para

atuar como um Q “intermediário”. Como sinais podem ser usados tanto para ler quanto para escrever, estes podem aparecer tanto do lado direito quanto do lado esquerdo de uma atribuição, evitando a mensagem de erro. Fazemos todas as operações usando Qbuf e, na linha 18, fazemos “Q <= Qbuf”. Note que a atribuição “Q <= Qbuf” poderia ter sido feita antes da estrutura process, sem prejuízo para o funcionamento do circuito, pois é uma atribuição concorrente que acontece concomitantemente com as atribuições dentro do process. Poderia ainda ter sido movida para dentro da estrutura process, sem nenhum prejuízo.

Uma forma alternativa de implementar este registrador de deslocamento seria declarando Qbuf como uma “variável” ao invés de um “sinal” (Figura 13). Note que, como variáveis só existem dentro do process onde foram declaradas, a atribuição “Q <= Qbuf” teve que ser movida para dentro do process (linha 17). Note ainda que os símbolos de atribuição para Qbuf foram trocados de “<=” para “:=” (linha 15), por se tratar agora de uma variável, e não de um sinal. Entretanto, sempre que possível, recomenda-se usar sinais ao invés de variáveis quando se pretende implementar o circuito em FPGA, pois variáveis são, muitas vezes, não sintetizáveis, enquanto que sinais geralmente representam fios no circuito sintetizado.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity shiftreg is
5      Port ( L,clk : in  STD_LOGIC;
6            Q : out  STD_LOGIC_VECTOR(3 downto 0));
7  end shiftreg;
8
9  architecture shiftreg_arch of shiftreg is
10 begin
11     process(clk)
12         variable Qbuf : STD_LOGIC_VECTOR(3 downto 0);
13     begin
14         if rising_edge(clk) then
15             Qbuf := Qbuf(2) & Qbuf(1) & Qbuf(0) & L;
16         end if;
17         Q <= Qbuf;
18     end process;
19 end shiftreg_arch;
```

Figura 13 – Código VHDL que implementa o registrador de deslocamento unidirecional de 4 bits descrito na Tabela 5, desta vez declarando Qbuf como uma “variável” ao invés de como um “sinal”.

4 IMPLEMENTANDO CLOCK MANUAL NO FPGA

Após a síntese de um código VHDL, acontece a etapa de implementação, durante a qual pinos do FPGA são associados a entradas e saídas da entidade “top level”.



Ao se tentar mapear um pino do FPGA associado a uma chave (ex: SW0) do kit de desenvolvimento de modo a associá-lo com a entrada de clock de um flip-flop ou de um registrador de deslocamento, o ISE Webpack acusará um erro. Isso acontece porque o FPGA tem pinos reservados para sinais de clock, os quais garantem que diferentes partes do circuito recebem exatamente o mesmo sinal de clock (sem atrasos) e, assim, funcionem de forma síncrona. Ao usar como clock do circuito um pino do FPGA que não tem essa característica, perde-se essa garantia, podendo os sinais de clock que chegam a diferentes partes do circuito sofrerem diferentes atrasos.

Para este experimento, tais atrasos não seriam um empecilho. Por outro lado, o uso de uma chave como clock torna o experimento mais didático.

Assim, para podermos usar uma chave como clock sem que o ISE Webpack acuse erro, será necessário incluir a seguinte linha de código no arquivo UCF:

```
NET clk CLOCK_DEDICATED_ROUTE = FALSE;
```

em que “clk” é o nome da entrada da entidade “top level” que age como clock do circuito. Fazendo isto, a mensagem de erro é convertida em um “warning” e o ISE Webpack consegue concluir a implementação com sucesso.