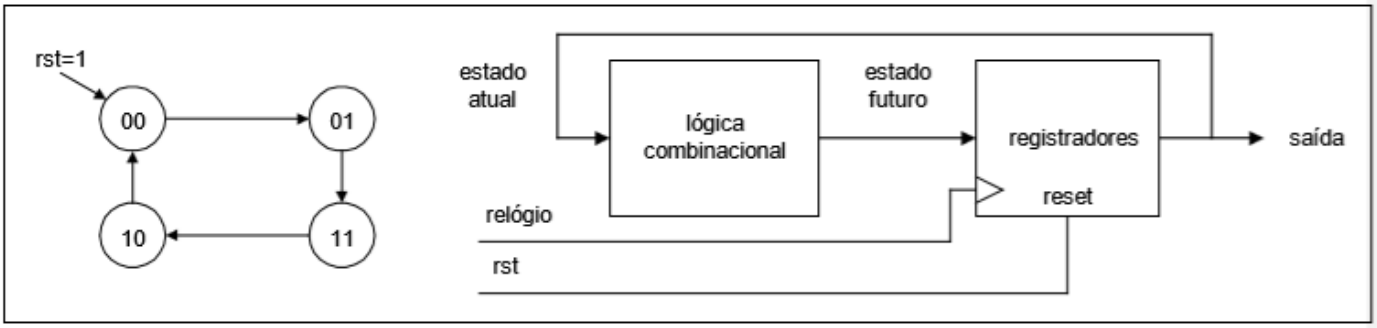
Máquinas de estado finito – exemplo

- **Contador**
- **A saída dos registradores corresponde ao valor de saída**
  - código do estado = valor de saída
    - não é necessário decodificar

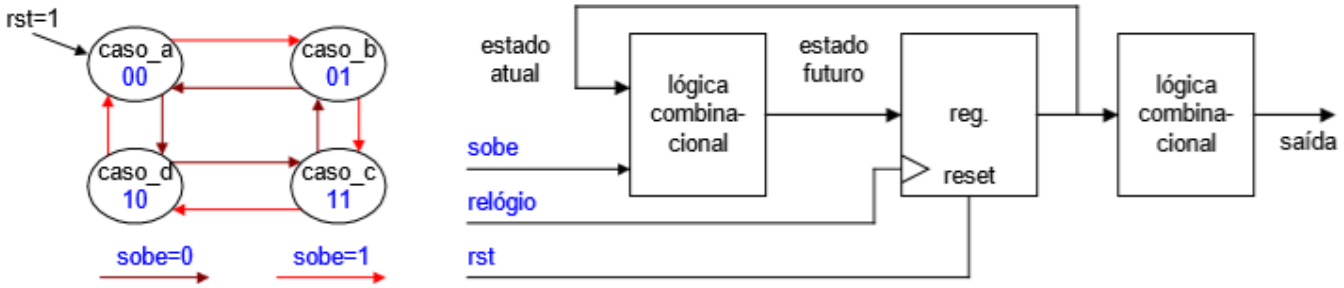


Máquinas de estado finito – descrição

```
 1 ENTITY maq_est1 IS
 2   PORT (ck      :  IN     BIT;                      -- relogio borda subida
 3         rst     :  IN     BIT;                      -- rst=1, q=00
 4         q       :  BUFFER BIT_VECTOR (1 DOWNTO 0)); -- saida
 5 END maq_est1;
 6
 7 ARCHITECTURE teste OF maq_est1 IS
 8
 9 BEGIN
10   abc: PROCESS (ck, rst)
11   BEGIN
12     IF rst = '1' THEN                       -- estado inicial
13       q <= "00";
14     ELSIF (ck'EVENT and ck ='1') THEN       -- ciclo de estados
15       CASE q IS
16         WHEN "00" => q <= "01";
17         WHEN "01" => q <= "11";
18         WHEN "11" => q <= "10";
19         WHEN "10" => q <= "00";
20        END CASE;
21     END IF;
22   END PROCESS abc;
23 END teste;
```

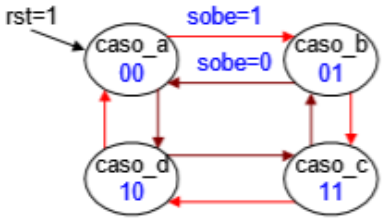
Máquinas de estado finito – contador crescente/decrescente

- Evolução dos estados segundo o esquema:



Máquinas de estado finito – contador crescente/decrescente

- **Evolução dos estados**:



- **Tipo enumerado definido na descrição**:

```
TYPE st IS (caso_d, caso_c, caso_b, caso_a); -- novo tipo definido
SIGNAL estado : st;                          -- sinal pode assumir os
                                             -- valores definidos em st
```

- **4 estados definidos** → st: caso_d caso_c caso_b caso_a
- **valores da saída q correspondentes** → q: 10 11 01 00
- **Correlação estado ↔ saída**:
  - pode ser necessária uma decodificação
- **Sinal estado armazena o estado atual da máquina**

```
  --           q=2     q=3     q=1     q=0
  TYPE st IS (caso_d, caso_c, caso_b, caso_a);  -- novo tipo definido
  SIGNAL estado : st;                           -- sinal estado tipo "st"
BEGIN
  abc: PROCESS (ck, iniciar)
  BEGIN
    IF iniciar = '1' THEN                 -- estado inicial
      estado <= caso_a;                   -- q=0
    ELSIF (ck'EVENT and ck ='1') THEN     -- ciclo de estados
      CASE estado IS
        WHEN caso_a =>                                -- q=0
          IF sobe = '1' THEN estado <= caso_b;  -- q=1
          ELSE               estado <= caso_d;  -- q=2
          END IF;
        WHEN caso_b =>                                -- q=1
          IF sobe = '1' THEN estado <= caso_c;  -- q=3
          ELSE               estado <= caso_a;  -- q=0
          END IF;
        WHEN caso_c =>                                -- q=3
          IF sobe = '1' THEN estado <= caso_d;  -- q=2
          ELSE               estado <= caso_b;  -- q=1
          END IF;
        WHEN caso_d =>                                -- q=2
          IF sobe = '1' THEN estado <= caso_a;  -- q=0
          ELSE               estado <= caso_c;  -- q=3
          END IF;
       END CASE;
    END IF;
  END PROCESS abc;

  WITH estado SELECT   -- decodifica estado
    q <= "00" WHEN caso_a, -- q=0
         "01" WHEN caso_b, -- q=1
         "11" WHEN caso_c, -- q=3
         "10" WHEN caso_d; -- q=2
END teste;
```

Controle de um motor de passo

Vamos aplicar esse procedimento de projeto em uma situação prática — o controle de um *motor de passo*, que gira em passos discretos, geralmente 15° por passo, em vez de girar em movimento contínuo. Os enrolamentos dentro do motor devem ser energizados e desenergizados em uma sequência específica para produzir movimentos em passos discretos. Sinais digitais são normalmente usados para controlar a corrente em cada enrolamento do motor. Motores de passo são bastante utilizados em situações em que o controle preciso de posição é necessário, como, por exemplo, no posicionamento de cabeças para leitura/escrita de discos magnéticos, no controle de cabeças de impressoras e em robôs.

A Figura 7.30(a) mostra um diagrama de um típico motor de passo de quatro enrolamentos. Para que o motor gire de modo correto, os enrolamentos 1 e 2 devem estar sempre em estados opostos; isto é, quando o enrolamento 1 está energizado, o enrolamento 2 não está, e vice-versa. Da mesma maneira, os enrolamentos 3 e 4 devem estar sempre em estados opostos. As saídas de um contador síncrono de dois bits são usadas para controlar a corrente nos quatro enrolamentos. A e $\overline{A}$ controlam os enrolamentos 1 e 2, e B e $\overline{B}$ controlam os enrolamentos 3 e 4. Os amplificadores de corrente são necessários porque as saídas dos flip-flops não podem gerar a corrente exigida pelos enrolamentos.

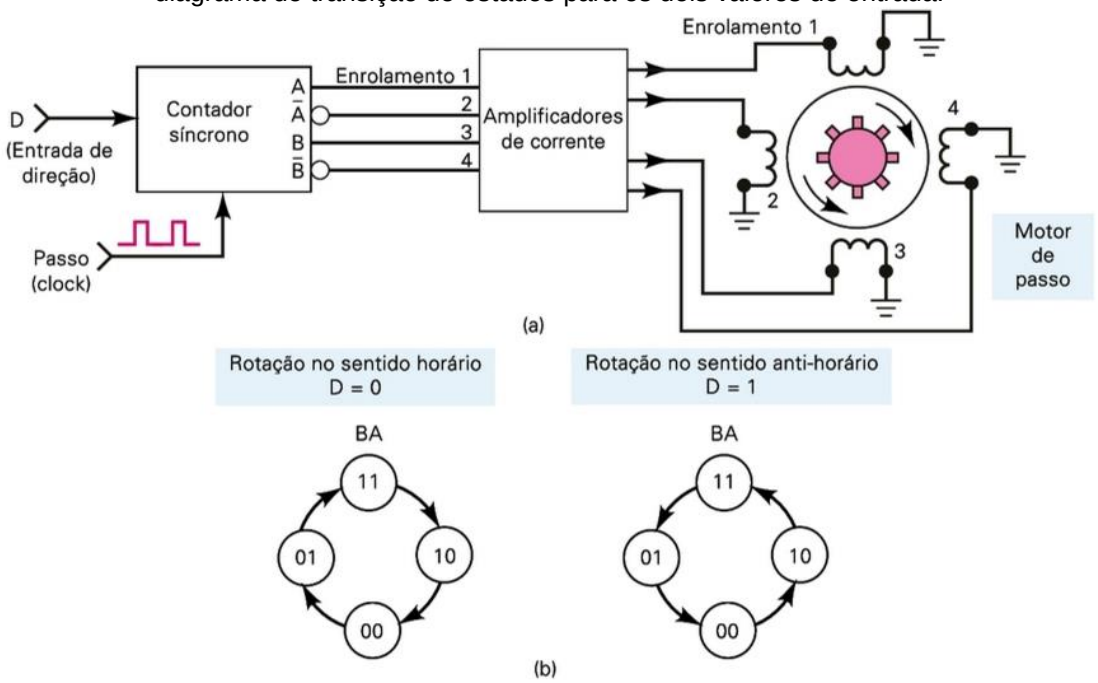
Uma vez que o motor de passo pode girar em sentido horário ou anti-horário, temos uma entrada de direção, D, usada para controlar a direção de rotação. Os diagramas de estados para as duas situações podem ser vistos na Figura 7.30(b). Para termos a rotação em sentido horário, devemos ter $D = 0$ e o estado do contador, BA, tem de seguir a ordem 11, 10, 00, 01, 11, 10,..., e assim por diante, conforme disparado pela entrada de sinal. Para rotação no sentido anti-horário, $D = 1$ e o contador tem de seguir a sequência 11, 01, 00, 10, 11, 01, ..., e assim por diante.

Estamos, agora, prontos para seguir os seis passos para o projeto de um contador síncrono. Os passos 1 e 2 já foram dados e podemos proceder aos passos 3 e 4. A Tabela 7.7 mostra cada ATUAL estado possível para D, B e A e o PRÓXIMO estado desejado juntamente com os níveis lógicos necessários para as entradas J e K alcançarem as transições. Observe que, em todos os casos, a entrada de direção, D, não muda do ATUAL para o PRÓXIMO estado; isso acontece porque ela é uma entrada independente que é mantida em nível ALTO ou BAIXO à medida que o contador avança em sua sequência.

O passo 5 do procedimento de projeto é apresentado na Figura 7.31, na qual a informação da Tabela 7.7 foi transferida para os mapas de Karnaugh que mostram como cada sinal J e K está relacionado ao estado ATUAL de D, B e A. Fazendo os agrupamentos apropriados, as expressões lógicas simplificadas para cada sinal são obtidas.

O passo final é mostrado na Figura 7.32, em que o contador síncrono de dois bits é implementado usando as expressões para J e K obtidas nos mapas K.

Figura 7.30 (a) um contador síncrono fornece a sequência apropriada de saídas para acionar o motor de passo; (b) diagrama de transição de estados para os dois valores de entrada.



1) Como seria em VHDL o código para controle do problema do motor de passo descrito acima?

2) Escreva o código VHDL implementando uma máquina de estados para um simples máquina de lavar, seguindo as especificações:

2.1 A máquina fica ociosa até que o botão de início seja apertado.

2.2 Quando o botão for apertado, a máquina se enche de água até o tanque estar completo.

2.3 Como o tanque completo, o agitador entra em funcionamento até que o temporizador assinale o final da lavagem.

2.4 Por fim, o tanque gira até a água ser eliminada e, então, volta ao estado de desligada.

As entradas da entidade serão: clock, start, full, timeup, dry

As saídas das entidades serão: water_valve, agitate_mode, spin_mode