

# Capítulo 3–Circuitos Sequenciais

Otimização e tradeoffs:

Codificação de estados

Profa. Eliete Caldeira

# Codificação de estados

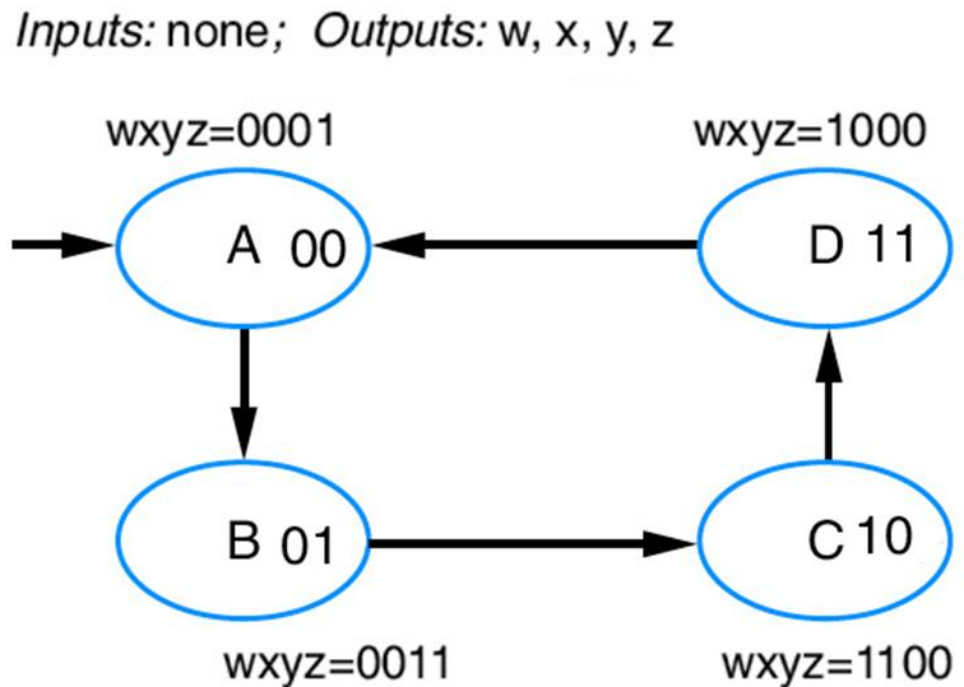
- ▶ É a tarefa de atribuir uma representação única de bits para cada um dos estados de uma FSM
- ▶ Algumas codificações de estados podem otimizar o circuito do bloco de controle:
  - Reduzindo o tamanho do circuito, ou
  - Permitindo *tradeoff* entre tamanho e desempenho

# Codificação de estados

- ▶ Codificação binária usando o menor número possível de bits.
- ▶ Para obter o número de flip-flops  $n$  para uma máquina com  $m$  estados, usa-se a equação
  - $2^n \geq m$
- ▶ Codifica-se os estados consecutivos de acordo com a sequência binária.

# Codificação de estados

- Exemplo: para codificar S0, S1, S2 e S3 são necessários dois flip-flops ( $s_0 \Rightarrow 00$ ,  $s_1 \Rightarrow 01$ ,  $s_2 \Rightarrow 10$ ,  $s_3 \Rightarrow 11$ )



# Codificação de estados

- ▶ Codificação binária usando o menor número possível de bits.
  - Codifica-se os estados consecutivos de acordo com a sequência binária. Exemplo: para codificar S0, S1, S2 e S3 são necessários dois flip-flops ( $s_0 \Rightarrow 00$ ,  $s_1 \Rightarrow 01$ ,  $s_2 \Rightarrow 10$ ,  $s_3 \Rightarrow 11$ )
- ▶ Codificações binárias alternativas com largura de bits mínima
  - Há muitas maneiras de se mapear as codificações binárias com largura mínima de bits para um conjunto de estados. Para 4 estados, existem  $4! = 24$  codificações possíveis
  - Uma codificação pode resultar em uma lógica combinacional menor do que outra.
  - As ferramentas automatizadas podem tentar diversas codificações diferentes para reduzir a lógica combinacional do bloco de controle

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
S0	00	00	00	00	00	00	01	01	01	01	01	01	10	10	10	10	10	10	11	11	11	11	11	11
S1	01	01	10	10	11	11	00	00	10	10	11	11	00	00	01	01	11	11	00	00	01	01	10	10
S2	10	11	01	11	01	10	10	11	00	11	00	10	01	11	00	11	00	01	01	10	00	10	00	01
s3	11	10	11	01	10	01	11	10	11	00	10	00	11	01	11	00	01	00	10	01	10	00	01	10

# Codificação de estados

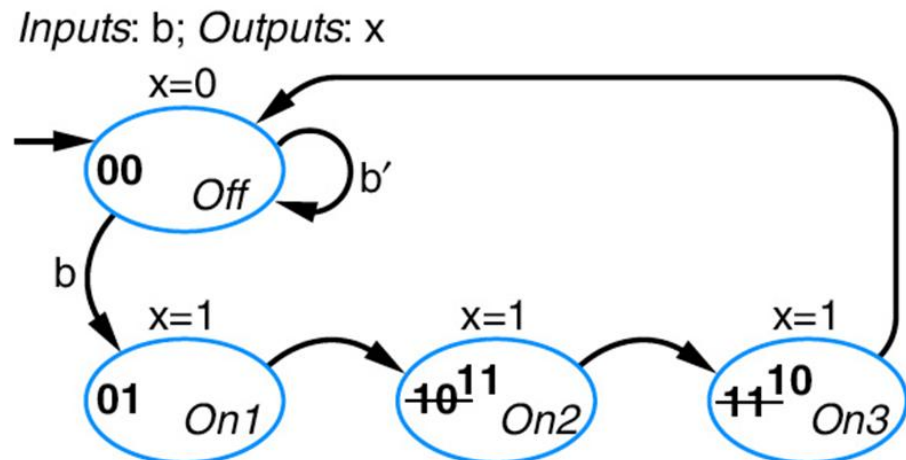
- ▶ Exemplo do laser ligado em 3 ciclos depois do aperto do botão

TABLE 6.3 State table for laser timer controller with alternative encoding.

	Inputs			Outputs		
	s1	s0	b	x	n1	n0
Off	0	0	0	0	0	0
	0	0	1	0	0	1
On1	0	1	0	1	1	1
	0	1	1	1	1	1
On2	1	1	0	1	1	0
	1	1	1	1	1	0
On3	1	0	0	1	0	0
	1	0	1	1	0	0

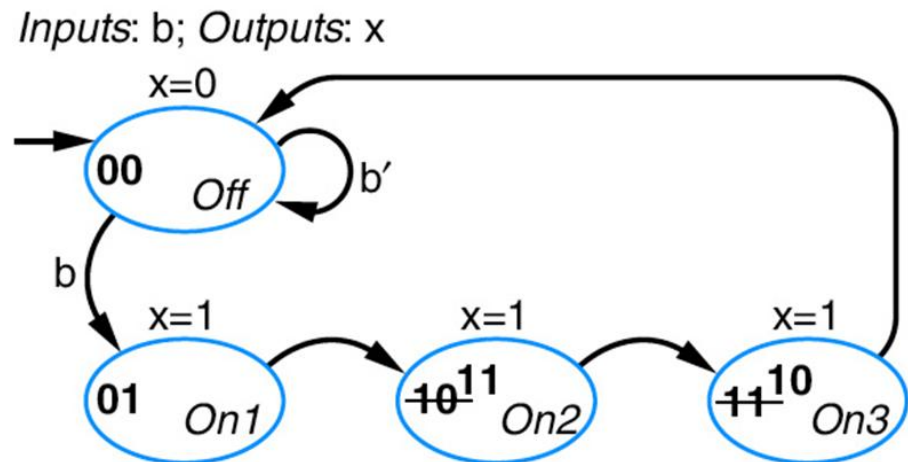
- ▶ Com codificação Gray:

- ▶  $x = s1 + s0$
- ▶  $n1 = s0$
- ▶  $n0 = s1'b + s1's0$



# Codificação de estados

- ▶ Exemplo do laser ligado em 3 ciclos depois do aperto do botão
- ▶ Com codificação original:
  - $x = s1 + s0$
  - $n1 = s1's0 + s1s0'$
  - $n0 = s1's0'b + s1s0'$
- ▶ Com codificação Gray:
  - $x = s1 + s0$
  - $n1 = s0$
  - $n0 = s1'b + s1's0$





# Codificação de estados

- ▶ Embora seja possível usar a codificação com menor número de flip-flops, muitas vezes são usados mais flip-flops do que o mínimo
- ▶ Esta codificação requer um registrador de estado mais largo, mas pode requerer menos lógica.
- ▶ Pode-se codificar quatro estados usando três bits.
  - Por exemplo, A:000, B:011, C:110 e D:111.



# Codificação de estados

- ▶ Codificação usando um bit por estado: esquema popular de codificação que usa 1 bit por estado (**one-hot encoding**)
  - Por exemplo, para os quatro estados, A:0001, B:0010, C:0100 e D:1000.
- ▶ Vantagens: velocidade e simplicidade de projeto
- ▶ Estado pode ser detectado a partir de um bit apenas, não é necessário decodificá-lo usando uma porta AND
- ▶ Próximo estado do bloco de controle e a lógica de saída podem envolver menos portas e/ou portas com menos entradas
- ▶ Apresenta menor atraso

# Codificação de estados

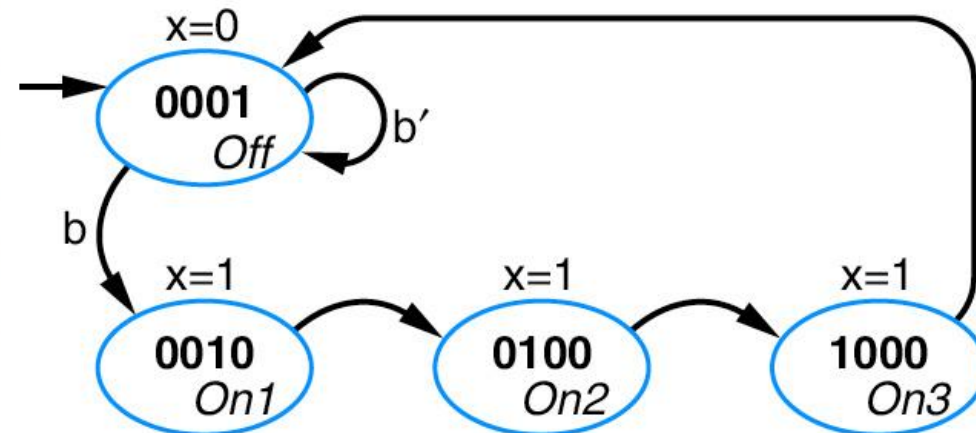
## ► Exemplo do laser usando codificação one-hot

**TABLE 6.6** State table for laser timer controller with one-hot encoding.

	Inputs					Outputs				
	s3	s2	s1	s0	b	x	n3	n2	n1	n0
Off	0	0	0	1	0	0	0	0	0	1
	0	0	0	1	1	0	0	0	1	0
On1	0	0	1	0	0	1	0	1	0	0
	0	0	1	0	1	1	0	1	0	0
On2	0	1	0	0	0	1	1	0	0	0
	0	1	0	0	1	1	1	0	0	0
On3	1	0	0	0	0	1	0	0	0	1
	1	0	0	0	1	1	0	0	0	1

- $x = s3 + s2 + s1$
- $n3 = s0$
- $n2 = s1$
- $n1 = s0.b$
- $n0 = s0.b' + s3$

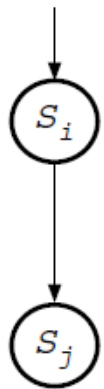
Inputs: b; Outputs: x



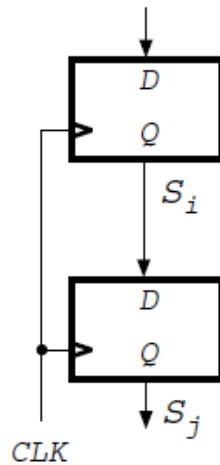
**Figure 6.45** One-hot encoding of laser timer.

# Codificação de estados

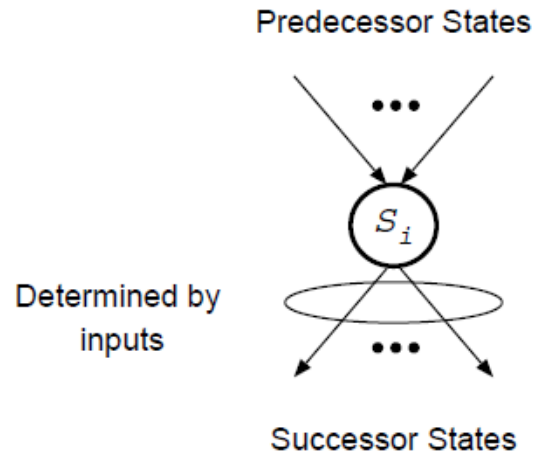
## ► Implementação one-hot



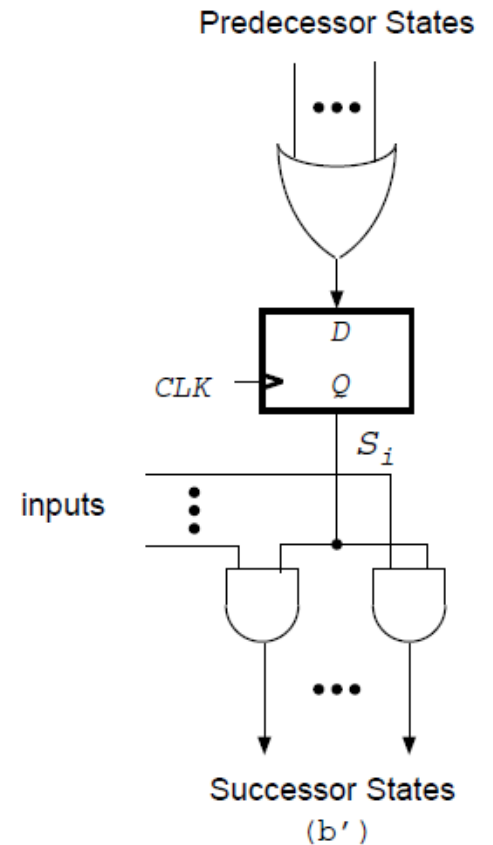
(a)



(a')

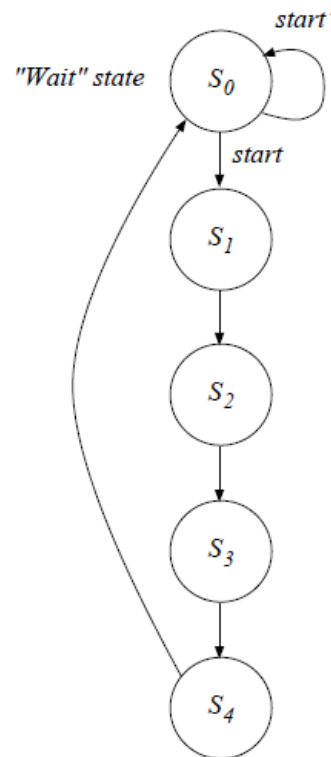


(b)



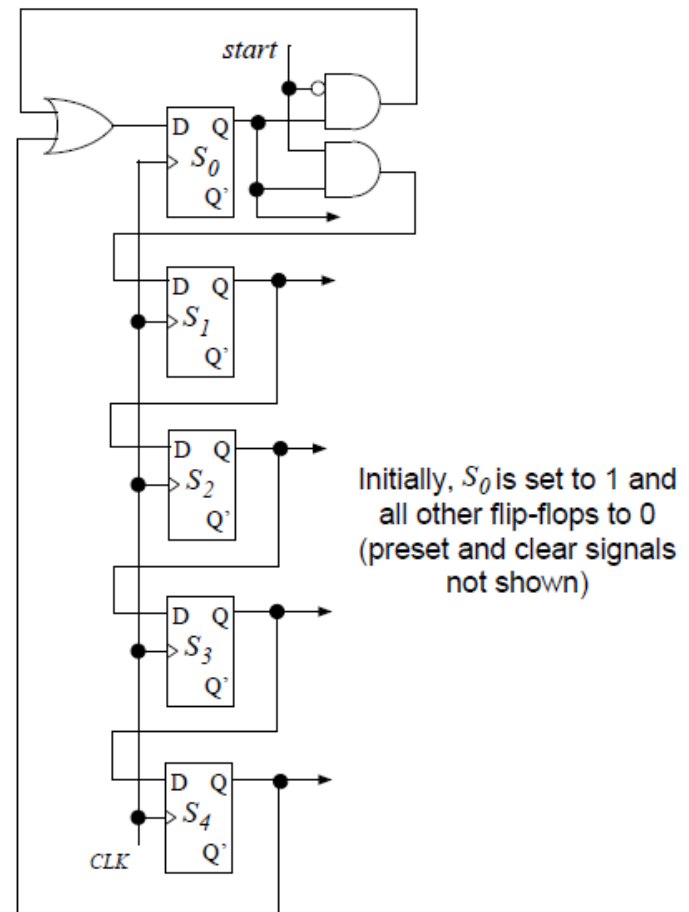
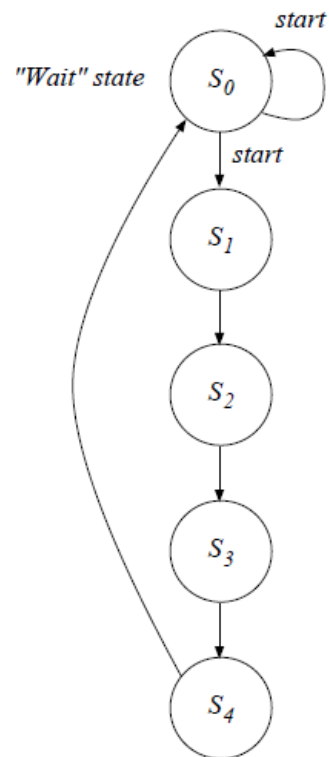
# Codificação de estados

- ▶ Exemplo: Projete o sistema do grafo usando codificação one-hot



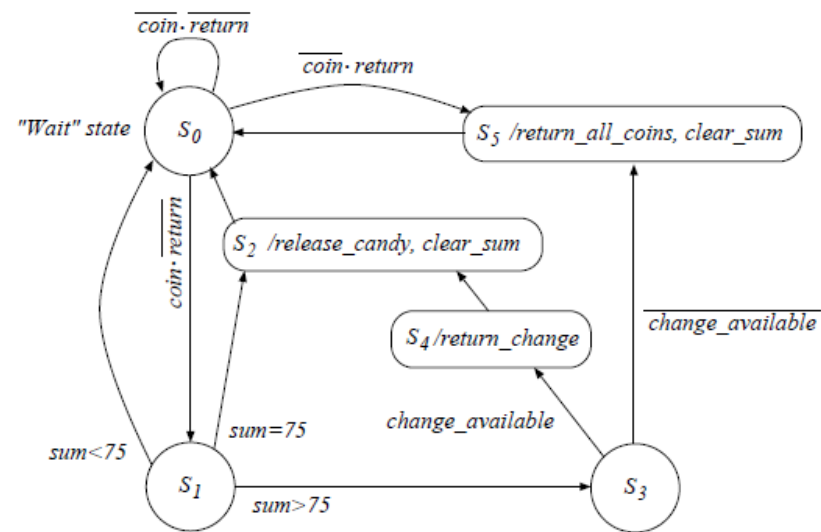
# Codificação de estados

- ▶ Exemplo: Projete o sistema do grafo usando codificação one-hot



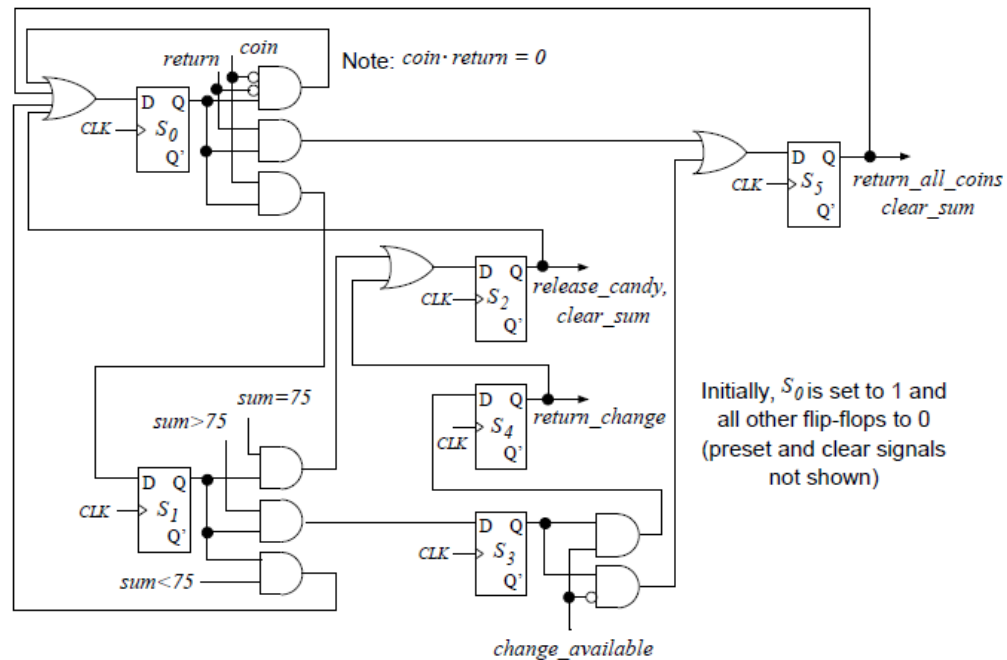
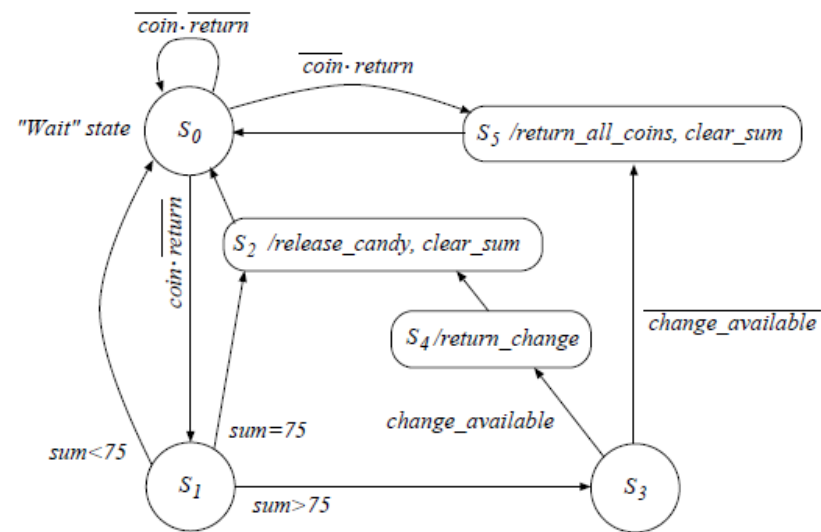
# Codificação de estados

- ▶ Exemplo 2: Projete o sistema do grafo usando codificação one-hot



# Codificação de estados

- Exemplo 2: Projete o sistema do grafo usando codificação one-hot





# Codificação de estados

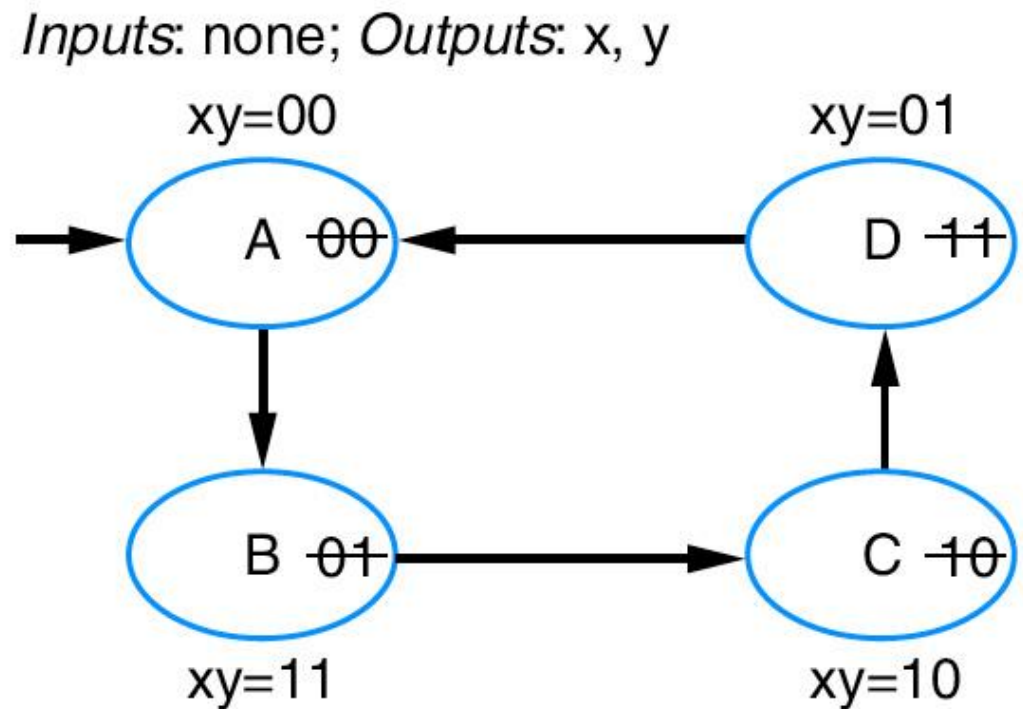
- ▶ Codificação one-hot (1 bit por estado) em sistemas com mais estados:
  - Reduções no caminho critico podem ser ainda maiores
  - Reduções no tamanho da lógica podem ser mais pronunciadas
- ▶ Por outro lado, a codificação de um bit por estado pode resultar em um registrador de estado grande demais:
  - FSM com 1000 estados: 10 bits vs. 1000 bits
- ▶ Nestes casos, considerar codificações que usam um número de bits intermediário

# Codificação de estados

- ▶ Codificação de saída (estados iguais as saídas)
- ▶ Algumas descrições de problemas podem exigir que uma dada sequência de valores seja gerada em um conjunto de saídas.

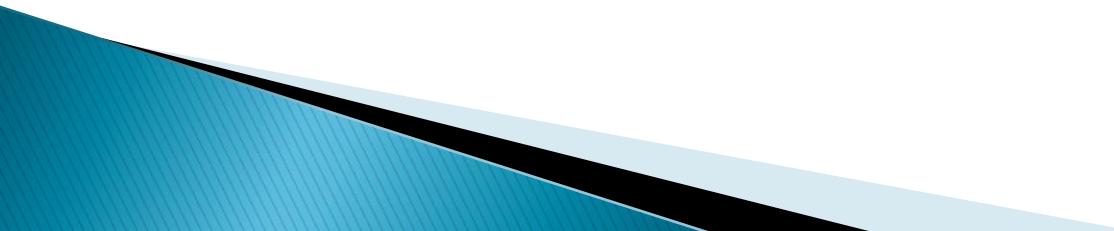
# Codificação de estados

- Ex.: Sequência produzida de forma repetida em duas saídas, x e y: 00, 11, 10 e 01.



**Figure 6.46** FSM for given sequence.

# Codificação de estados

- ▶ Se usamos uma codificação na qual os estados sejam idênticos aos valores de saída de cada estado
  - ▶ Não teremos mais a lógica que gera a saída a partir do estado atual.
  - ▶ As saídas simplesmente estarão ligadas por conexões diretamente aos bits do registrador de estado
  - ▶ Reduzindo assim o número necessário de portas lógicas.
- 

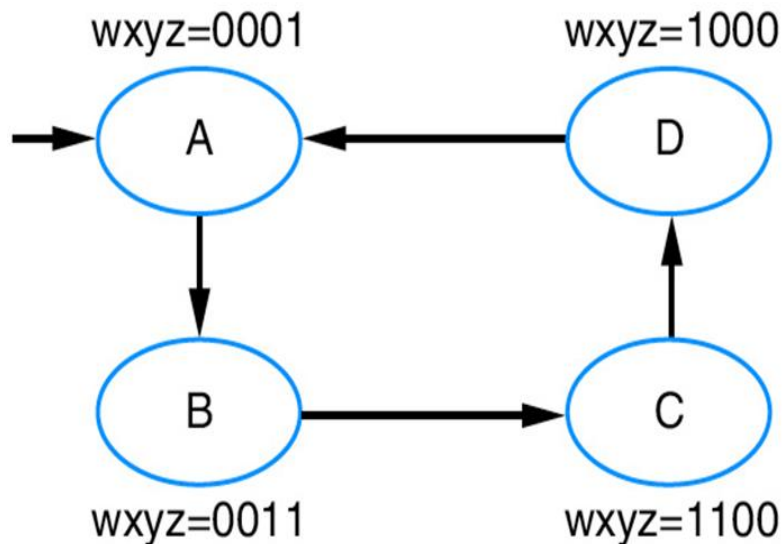
# Codificação de estados

- ▶ Codificação de saída só pode ser usada se:
  - Se uma FSM tiver no mínimo tantas saídas quantas as necessárias para a codificação binária dos estados e
  - Se cada estado tiver uma combinação única de saída
- ▶ Ex.: Para gerar de forma repetida a sequência 00, 11, 01 e 11, não se pode usar a codificação de saída

# Codificação de estados

- ▶ Exemplo: Motor de passo – gerar as saídas: 0001, 0011, 1100, 1000 repetidamente.

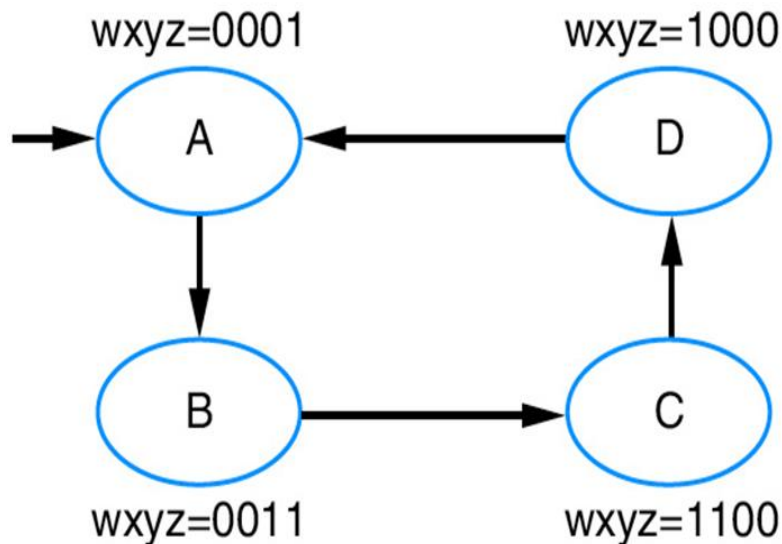
*Inputs: none; Outputs: w, x, y, z*



# Codificação de estados

- Exemplo: Motor de passo – gerar as saídas: 0001, 0011, 1100, 1000 repetidamente.

Inputs: none; Outputs: w, x, y, z



	Inputs				Outputs			
	s3	s2	s1	s0	n3	n2	n1	n0
A	0	0	0	1	0	0	1	1
B	0	0	1	1	1	1	0	0
C	1	1	0	0	1	0	0	0
D	1	0	0	0	0	0	0	1

$$n3 = s1 + s2$$

$$n2 = s1$$

$$n1 = s1'.s0$$

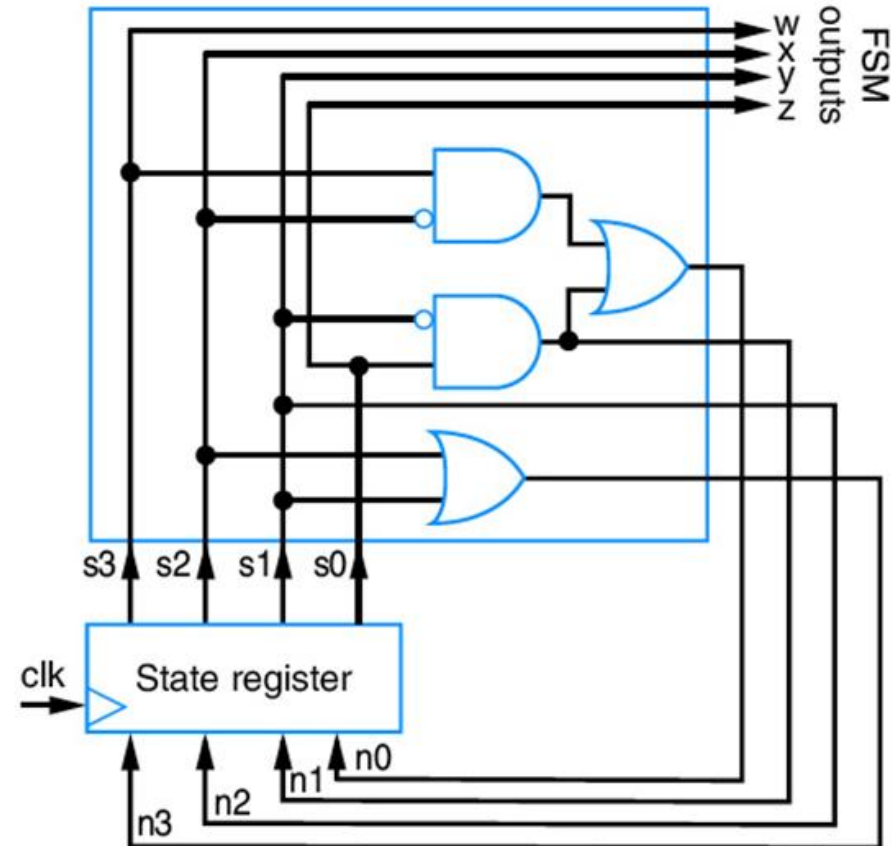
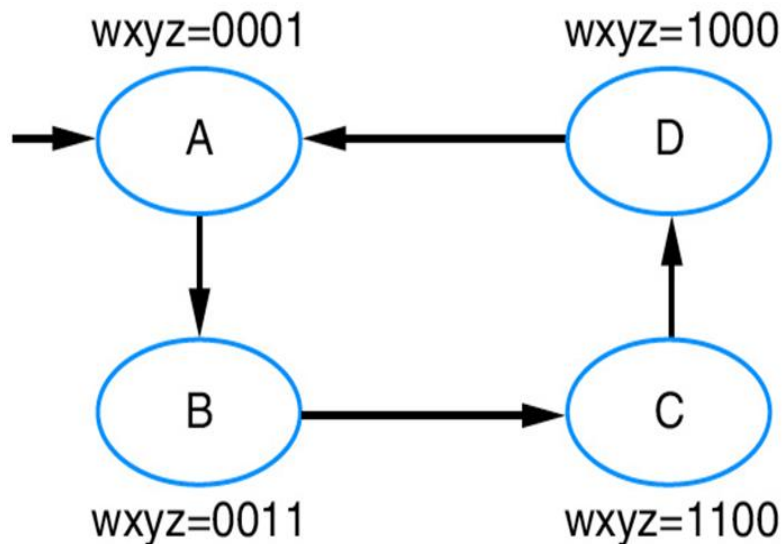
$$n0 = s1'.s0 + s3.s2'$$



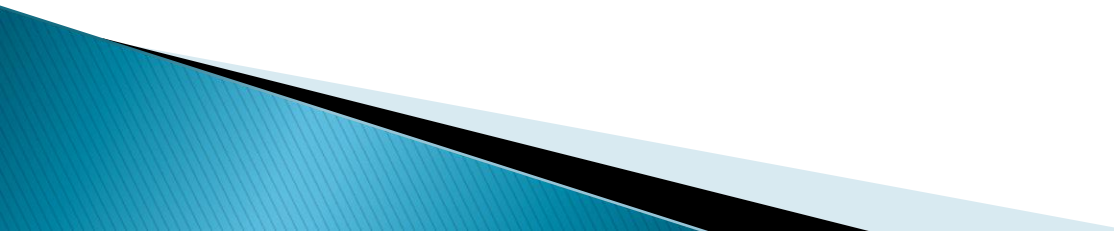
# Codificação de estados

- Exemplo: Motor de passo – gerar as saídas: 0001, 0011, 1100, 1000 repetidamente.

Inputs: none; Outputs: w, x, y, z



# Codificação de estados

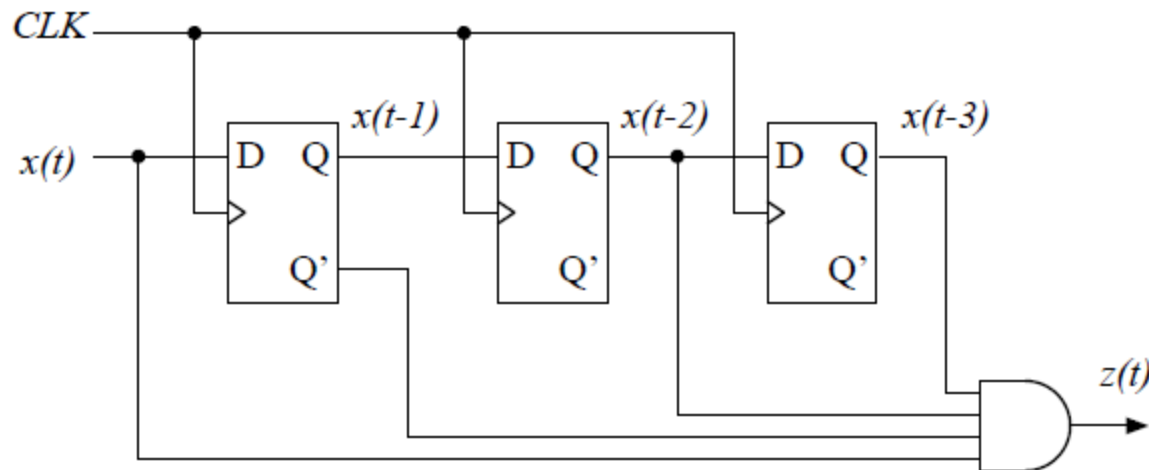
- ▶ Usando registrador de deslocamento (*shift register*)
  - ▶ Pode-se usar um registrador de deslocamento para armazenar os bits de uma entrada e com isto implementar a função usando os bits memorizados
  - ▶ Pode levar a um registrador maior mas a implementação pode ser muito simples
- 

# Codificação de estados

- ▶ Projete um circuito sequencial que gera uma saída  $z = 1$  quando o padrão 1101 aparece na entrada  $x$  usando um registrador de deslocamento

# Codificação de estados

- ▶ Projete um circuito sequencial que gera uma saída  $z = 1$  quando o padrão 1101 aparece na entrada  $x$  usando um registrador de deslocamento



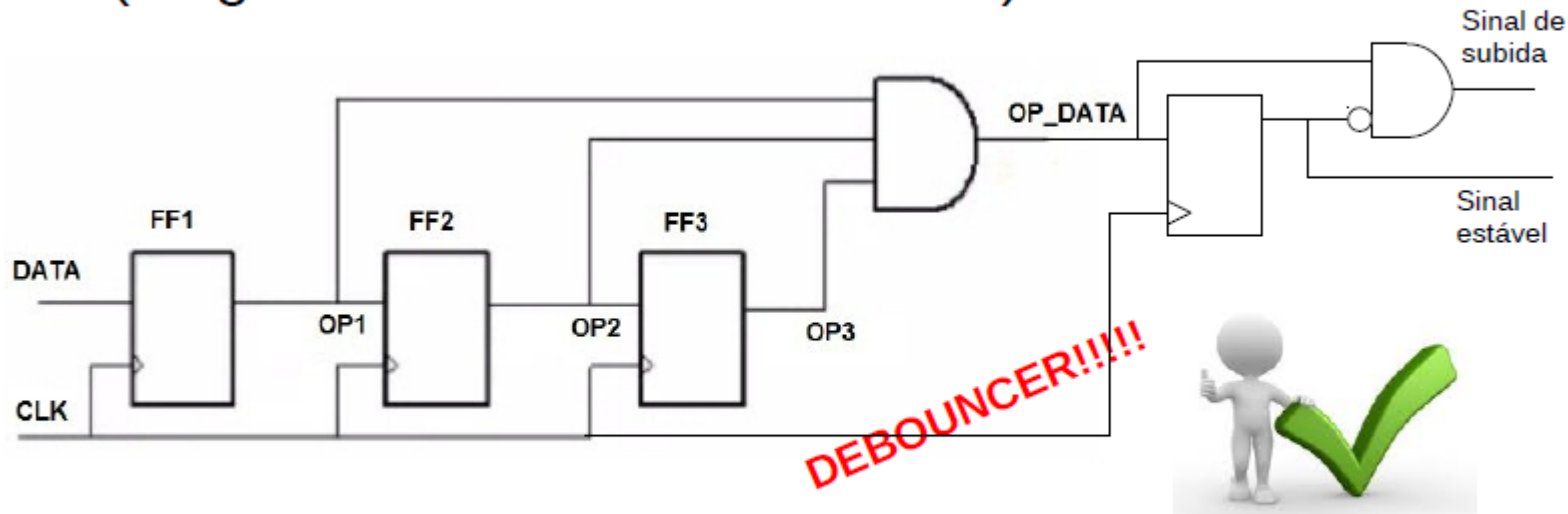
Esta é uma máquina Mealy.

É possível fazer uma máquina Moore?

# Codificação de estados

## *Debouncer* do laboratório

- Sequência de elementos de memória (Registrador de Deslocamento)



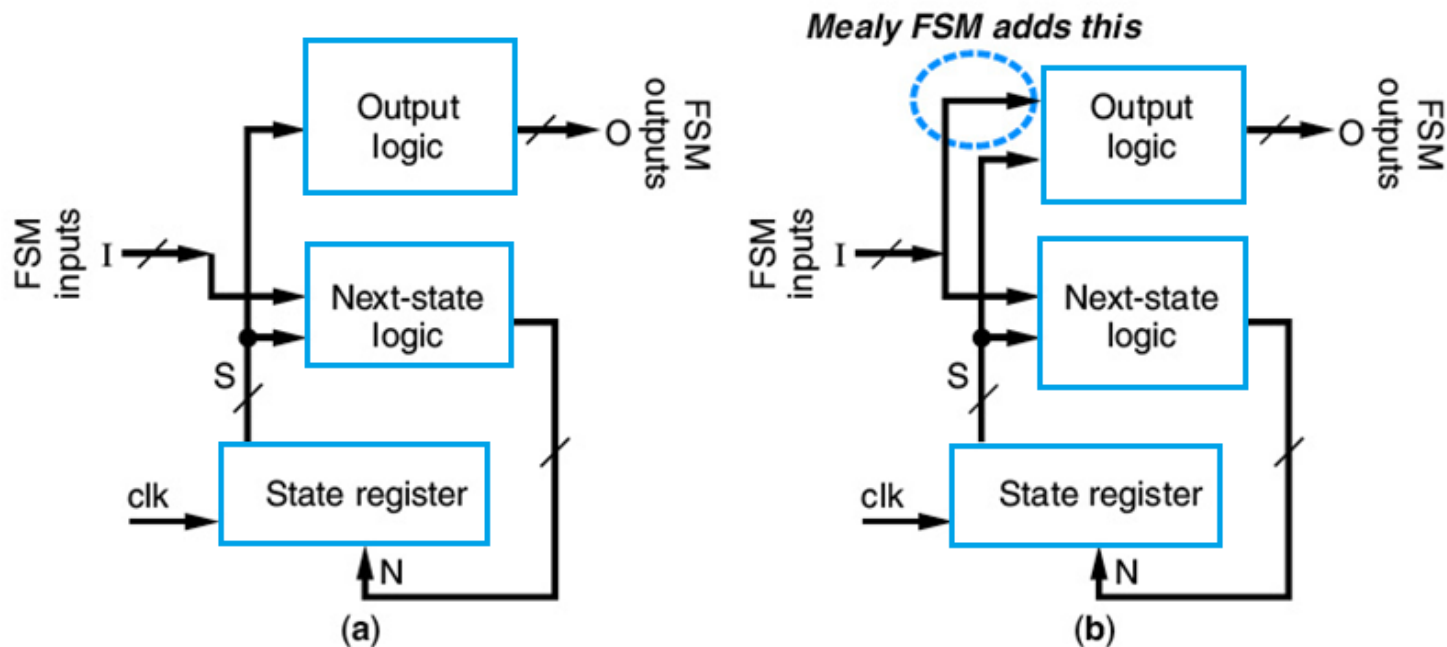
- O sinal "OP\_DATA" na saída só será "1" quando o valor "1" se repetir 3 vezes no sinal "DATA"
- A seguir ele é armazenado como estável.
- O sinal de subida será "1" quando o sinal estável for "0" e OP\_DATA acabou de virar "1"

# Otimização e *Tradeoffs*

- ▶ FSM Moore: Saídas são uma função do estado da FSM
- ▶ FSM Mealy: Saídas são uma função dos estados da FSM e das entradas
- ▶ Otimização:
  - Algumas vezes, uma FSM Mealy produz menos estados do que uma FSM Moore
- ▶ Tradeoff:
  - Ocasionalmente, esses estados em menor número são obtidos às custas de complexidades de tempo que devem ser tratadas

# Otimização e *Tradeoffs*

## ► Moore versus Mealy



**Figure 6.50** Controller architectures for: (a) a Moore FSM, (b) a Mealy FSM.



# Otimização e *Tradeoffs*

## ► Exemplo: Máquina de refrigerante

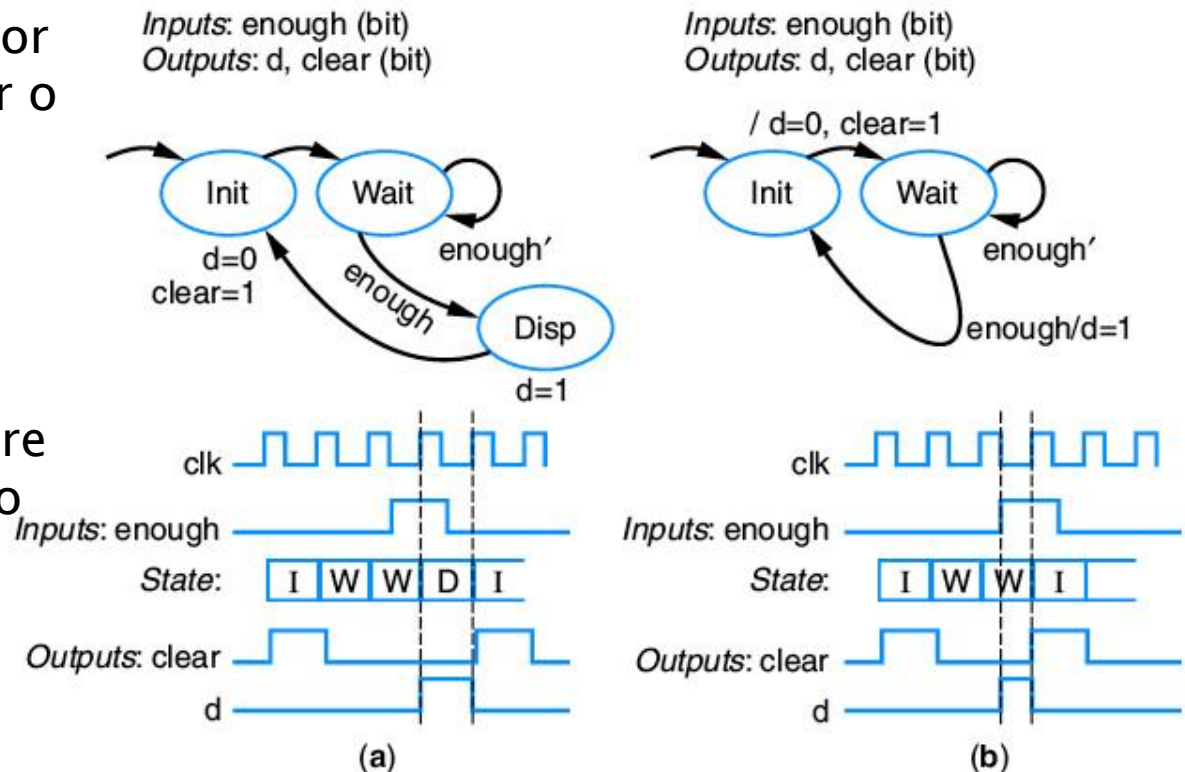
Iniciar, esperar que o valor seja suficiente e entregar o refrigerante

Moore – 3 estados

Mealy – 2 estados

A saída da máquina Moore fica em 1 por um ciclo do clock

Na máquina Mealy o tempo é menor



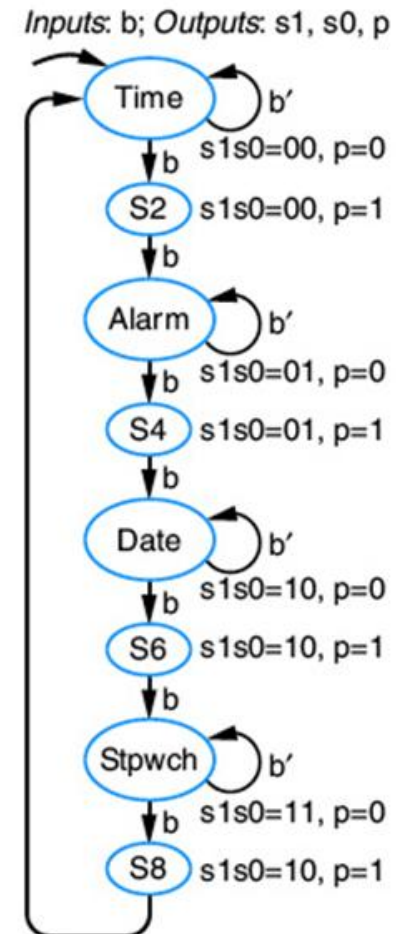
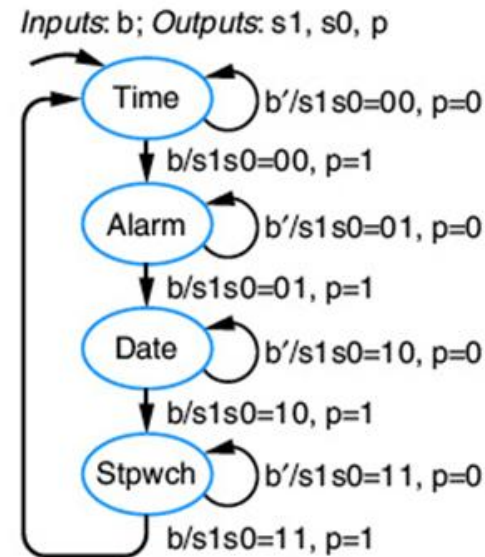
**Figure 6.52** FSMs for soda dispenser controller: (a) Moore FSM has actions in states, (b) Mealy FSM has actions on transitions, resulting in this case in fewer states.

# Otimização e *Tradeoffs*

## ► Exemplo: FSM para relógio com bipe

A cada vez que o botão é pressionado,  $p = 1$ , faz com que um bipe seja ouvido

No caso da máquina Mealy não há garantia de que o bipe vai durar um ciclo do clock no mínimo



# Otimização e *Tradeoffs*

- ▶ Questões envolvendo o tempo:
- ▶ Saídas do tipo Moore:
  - Sincronizadas com as bordas de relógio
  - Somente se alteram quando se entra em um novo estado.
- ▶ Saídas do tipo Mealy:
  - Não estão sincronizadas com as bordas de relógio
  - Poderão se modificar não apenas quando se entra em um novo estado, mas a qualquer instante em que as entradas apresentarem alterações

# Otimização e *Tradeoffs*

- ▶ Propriedade indesejável:
  - *Glitches* nas entradas durante os ciclos de relógio, podem causar *glitches* nas saídas.
- ▶ Solução:
  - Inserir flip-flops entre as entradas assíncronas e a lógica de uma FSM Mealy
  - Inserir flip-flops entre a lógica e as saídas da FSM
- ▶ Flip-flops:
  - Tornarão síncrona a FSM Mealy
  - Introduzem um atraso correspondente a um ciclo de relógio

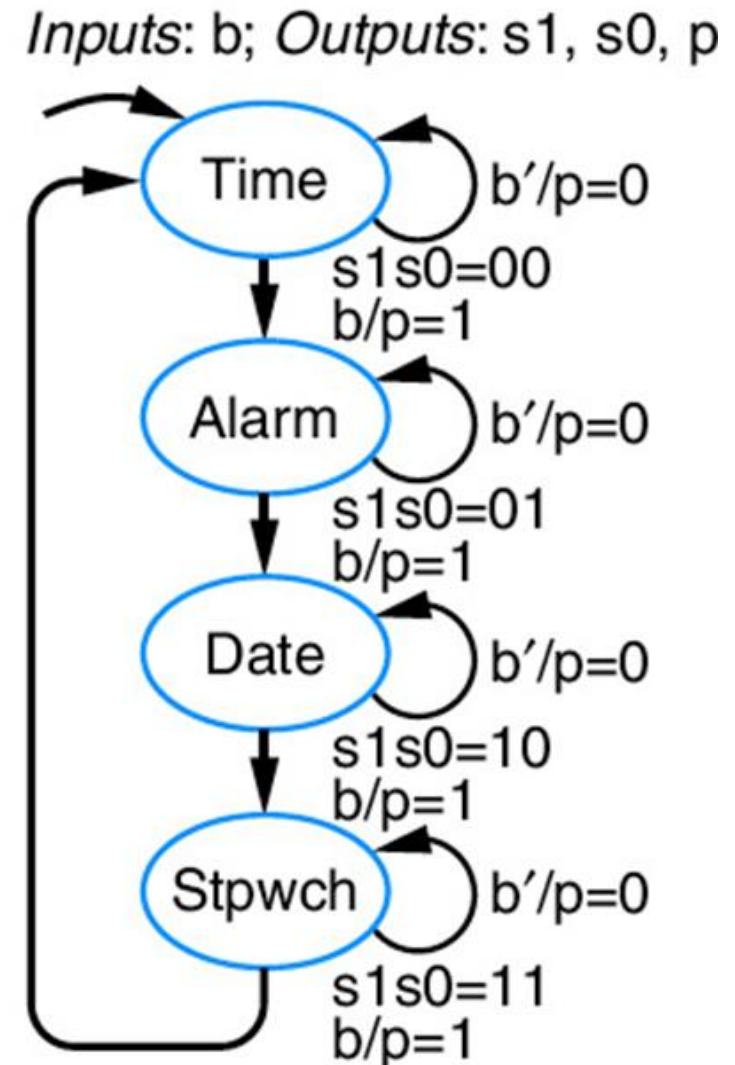
# Otimização e *Tradeoffs*

- ▶ Combinação dos tipos Moore e Mealy
  - Frequentemente, os projetistas utilizam FSMs que são uma combinação dos tipos Moore e Mealy
  - Permite ações nos estados e outras nas transições
  - Vantagem do número reduzido de estados de uma FSM Mealy
  - Evita que as ações de um estado apareçam repetidas em todas as transições que saem desse estado.
  - A implementação será igual à de uma FSM Mealy que tem as ações repetidas nas transições que saem de um estado

# Otimização e *Tradeoffs*

- ▶ Exemplo relógio de pulso com bipe

As saídas  $s1$  e  $s0$  são Moore e a saída  $p$  é Mealy



Para ser continuado...