



FEDERAL UNIVERSITY
OF SANTA CATARINA

EEL5105 – Circuitos e Técnicas Digitais

Aula 7

Prof. Héctor Pettenghi

hector@eel.ufsc.br

<http://hectorpettenghi.paginas.ufsc.br>

Material desenvolvido com apoio de arquivos de apresentação do livro de Frank Vahid

7. Projeto de Circuitos Sequenciais

7.1. Máquinas de Estados Finitos (FSM): Diagrama de estados

7.2. Projeto de FSM

Nesta aula veremos as FSMs, uma técnica muito poderosa para a projeção de circuitos. O conteúdo encontra-se dividido em duas seções:

7.1 - Máquinas de Estados Finitos (FSM): Diagrama de estados

7.2 - Projeto FSM

7.1. Máquinas de Estados Finitos (FSM): Diagrama de estados

7.2. Projeto de FSM

Começemos entendendo o que são as Máquinas de Estados Finitos e como são representadas.

7.1. Máquinas de Estados Finitos (FSM): Diagrama de estados

- Primeiro passo para o projeto de um **círcuito combinacional**:
 - Montagem da **tabela verdade**
 - **Formalismo** que não envolve o tempo e, portanto, é insuficiente para lidar com circuitos sequenciais

Em um círcuito combinacional, o primeiro passo para o seu projeto é a montagem de uma tabela verdade. Apesar de somente a partir disso já ser possível solucionar muitos problemas reais, é uma técnica insuficiente para lidar com circuitos sequenciais, isso porque uma tabela verdade não envolve tempo. Aí que surge a necessidade de uma solução para circuitos sequenciais.

7.1. Máquinas de Estados Finitos (FSM): Diagrama de estados

- Primeiro passo para o projeto de um **círcuito combinacional**:
 - Montagem da **tabela verdade**
 - **Formalismo** que não envolve o tempo e, portanto, é insuficiente para lidar com circuitos sequenciais
- Solução para circuitos sequenciais: **máquina de estados finitos**
 - **Finite State Machine (FSM)**
 - Descreve o comportamento ao longo do tempo, a partir dos possíveis **estados** de um circuito
 - Indica o que pode causar **transições entre estados**
 - Indica como as **saídas do circuito** irão se comportar para cada estado.

A Máquina de Estados Finitos (Finite State Machine - FSM) atende a essa necessidade.

Ela descreve o comportamento ao longo do tempo, a partir dos possíveis estados de um circuito, e indica o que pode causar transições entre estados e como as saídas do circuito irão se comportar para cada estado.

7.1. Máquinas de Estados Finitos (FSM): Diagrama de estados

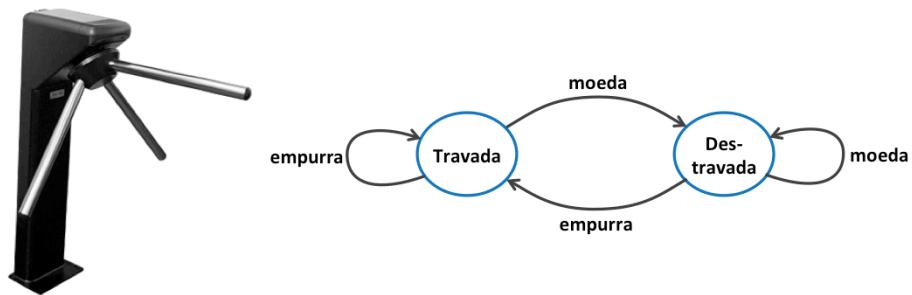
- **Exemplo 1:** **catraca** que abre com moedas



Consideremos uma catraca que abre com moedas.

7.1. Máquinas de Estados Finitos (FSM): Diagrama de estados

- Exemplo 1: **catraca** que abre com moedas

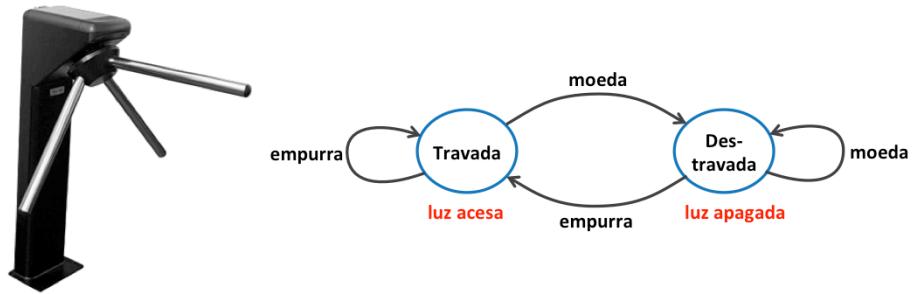


A FSM para o circuito desta catraca seria a seguinte. Veja que, intuitivamente, existem dois estados em que a catraca pode estar, um em que se encontra travada e outro destravada, permitindo a passagem de uma pessoa.

Acompanhe no diagrama de estados: caso a catraca esteja travada, não importa o quanto se empurre, sem colocar uma moeda ela permanecerá travada. A partir do momento em que é depositada uma moeda, a catraca fica destravada para a passagem de apenas uma pessoa, voltando a ficar travada em seguida, mesmo que seja depositada mais de uma moeda de uma vez só.

7.1. Máquinas de Estados Finitos (FSM): Diagrama de estados

- Exemplo 1: **catraca** que abre com moedas
 - Incluindo **indicador luminoso**:



Se incluirmos um indicador luminoso, assim fica o exemplo.

Em uma diagrama de estados funcional "moeda", "empurra", "luz acesa" e "luz apagada" são definidos por entradas (vindas, por exemplo, de sensores ou botões) e saídas (que vão para LEDs ou displays, por exemplo) do circuito.

7.1. Máquinas de Estados Finitos (FSM): Diagrama de estados

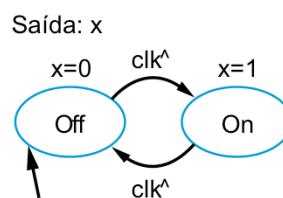
- **Exemplo 2:** Circuito que tem uma saída **x** ligada a um **led**, sendo que tal **led** deve ficar piscando continuamente. Para isso, o **led** deve mudar de estado após cada transição positiva de *clock*.

Neste segundo exemplo, temos um diagrama de estados funcional.

7.1. Máquinas de Estados Finitos (FSM): Diagrama de estados

- **Exemplo 2:** Circuito que tem uma saída x ligada a um led, sendo que tal led deve ficar piscando continuamente. Para isso, o led deve mudar de estado após cada transição positiva de *clock*.

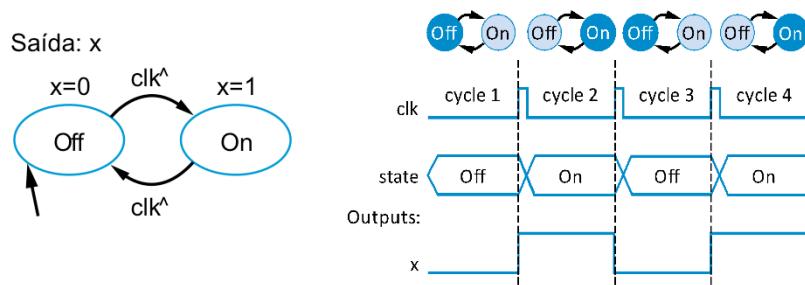
- Estados: **On** e **Off**
- **Diagrama de estados:**



Veja que existem apenas dois estados possíveis para o led, ligado (On) ou desligado (Off). Sendo assim, o diagrama de estado é o deste slide.

7.1. Máquinas de Estados Finitos (FSM): Diagrama de estados

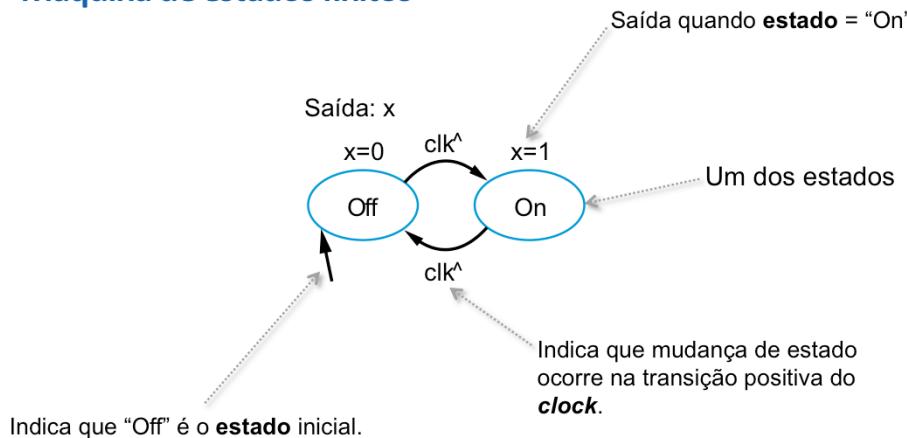
- **Exemplo 2:** Circuito que tem uma saída **x** ligada a um **led**, sendo que tal **led** deve ficar piscando continuamente.
 - Estados: **On** e **Off**
 - Diagrama de estados e comportamento temporal:



O diagrama de tempo deste diagrama de estados é o seguinte. Veja que agora ele contém, além do sinal de clock, as entradas e as saídas, o estado em que se encontra o circuito. Observe que, como descrito no enunciado, a cada transição POSITIVA de clock, o estado muda e, junto com ele, x .

7.1. Máquinas de Estados Finitos (FSM): Diagrama de estados

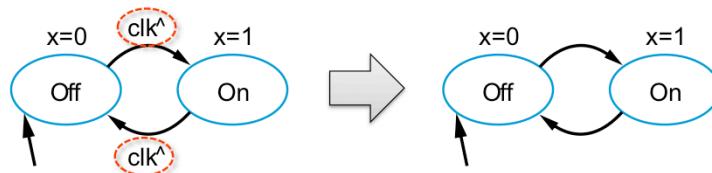
- Máquina de estados finitos



Voltando ao diagrama de estados, vemos que a alteração de estado depende da transição positiva do clock, o que está representado nas flechas indicadas neste slide. "On" e "Off" são os estados, com citado anteriormente. A flecha que aponta para Off sem vir de nenhum estado mostra em qual estado a FSM começa a funcionar, quando é dado reset. Por último vemos x, a saída do circuito ligada a um LED, igual a '0' quando em "Off" e igual a '1' quando em "On".

7.1. Máquinas de Estados Finitos (FSM): Diagrama de estados

- Para **sistemas digitais**, é bastante comum a representação do *clock* de forma **implícita**, considerando que cada seta do diagrama já indica uma transição de *clock*:



Para sistemas digitais, é bastante comum a representação do clock de forma implícita, considerando que cada seta do diagrama já indica uma transição de clock. Isso deixa o diagrama de estados mais limpo, especialmente em projetos maiores e mais complexos.

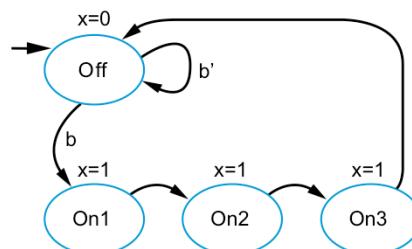
7.1. Máquinas de Estados Finitos (FSM): Diagrama de estados

- **Exemplo 3:** Circuito que mantém sua saída **x** em nível alto por 3 ciclos de *clock* quando um botão **b** for apertado.

Vejamos o próximo exemplo.

7.1. Máquinas de Estados Finitos (FSM): Diagrama de estados

- **Exemplo 3:** Circuito que mantém sua saída **x** em nível alto por 3 ciclos de *clock* quando um botão **b** for apertado.
 - Entrada: **b**
 - Saída: **x**
 - Estados:
Off, On1, On2 e On3
 - Com *clock* implícito:



Perceba que utilizamos três estados On para manter x em nível alto pelo tempo necessário. Agora também temos uma entrada, b , que interfere na mudança de estados.

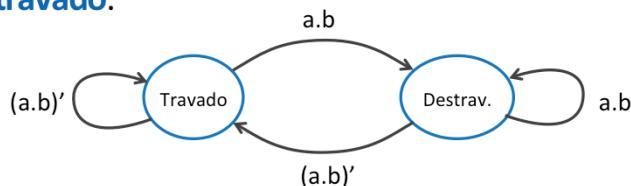
7.1. Máquinas de Estados Finitos (FSM): Diagrama de estados

- **Exemplo 4:** Faça o diagrama de estados com *clock* implícito de um sistema que fica em um estado chamado **travado** até que dois botões **a** e **b** sejam pressionados simultaneamente. Depois que tais botões forem apertados, o sistema deve ir para um estado **destravado** e permanecer nesse estado enquanto ambos os botões estiverem pressionados. Caso pelo menos um botão seja solto, o sistema deve voltar para o estado **travado**.

Neste novo exemplo, temos não uma, mas duas entradas que interferem juntas na passagem de estados.

7.1. Máquinas de Estados Finitos (FSM): Diagrama de estados

- **Exemplo 4:** Faça o diagrama de estados com *clock* implícito de um sistema que fica em um estado chamado **travado** até que dois botões **a** e **b** sejam pressionados simultaneamente. Depois que tais botões forem apertados, o sistema deve ir para um estado **destravado** e permanecer nesse estado enquanto ambos os botões estiverem pressionados. Caso pelo menos um botão seja solto, o sistema deve voltar para o estado **travado**.

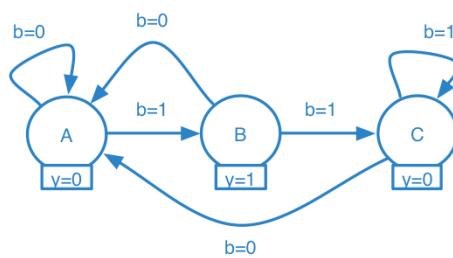


Tudo isso está descrito no seguinte diagrama de estados. Note como a e b são tratados.

PROBLEMAS

Problema 7.1. Faça o diagrama de estados de uma FSM para um circuito que tem um botão **b** ligado em sua entrada e deve deixar a sua saída **y** em nível alto por exatamente um período de *clock* após o aperto do botão. Assim, a saída deve ficar em nível alto por exatamente um ciclo de *clock* mesmo que o botão fique apertado por um tempo superior.

Solução:



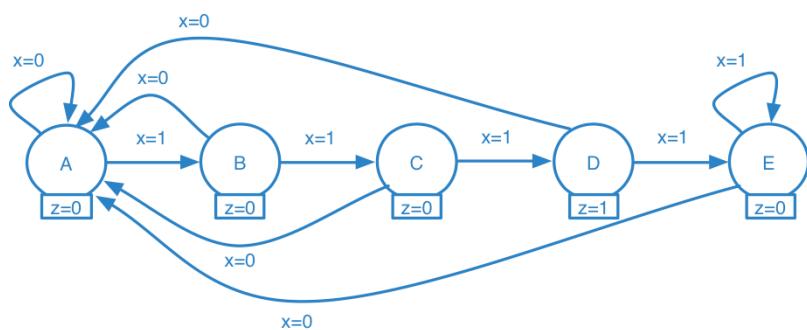
Entradas Saídas

EA	b	PE	y
A	1	B	0
A	0	A	0
B	1	C	1
B	0	A	1
C	1	C	0
C	0	A	0

PROBLEMAS

Problema 7.2. Faça o diagrama de estados de uma FSM para um circuito que gere a saída $Z=1$, durante um ciclo de relógio quando na entrada X for “1” durante os três intervalos precedentes de relógio. Caso quatro ou mais ciclos de relógio X for “1”, a saída será “0”.

Solução:



7.1. Máquinas de Estados Finitos (FSM): Diagrama de estados.

7.2. Projeto FSM

Agora veremos como é feito um projeto com FSMs.

7.2. Projeto FSM

- Circuito sequencial com comportamento descrito por uma diagrama de estados.
- Duas componentes:
 - Bloco de lógica puramente **combinatória** para obter saídas e próximos estados a partir das entradas e estados atuais.
 - Elementos de memória (**registrator**), controlados por um sinal de relógio para armazenar os estados atuais.
- As máquinas de estado síncronas podem ser divididas em:
 - **Máquinas de Moore:** a saída depende *apenas* das variáveis de estado atuais;
 - **Máquinas de Mealy:** a saída é função das variáveis de estado atuais e do valor das entradas presentes no circuito

O resultado de um projeto com FSMs é um circuito sequencial com o comportamento descrito por um diagrama de estados. Ele possui duas componentes: um bloco de lógica puramente combinatória, que é responsável pela obtenção das saídas e próximos estados a partir das entradas e estados atuais; e elementos de memória, ou seja, registradores, os quais são controlados por um sinal de relógio para armazenar os estados atuais.

As FSMs síncronas podem ser divididas em Máquinas de Moore, cujas saídas dependem apenas das variáveis de estado atuais, e Máquinas de Mealy, nas quais as saídas são em função das variáveis de estado atuais E do valor das entradas presentes no circuito.

7.2. Projeto FSM

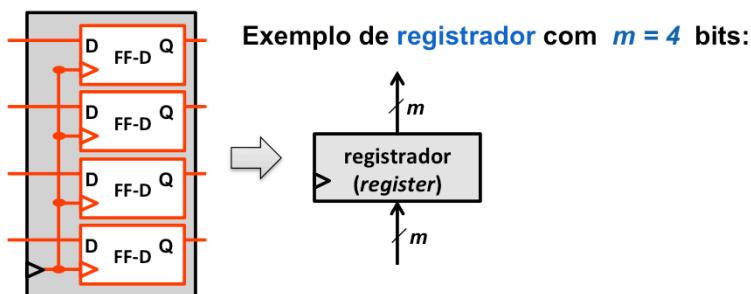
- Duas componentes:

- Bloco de lógica puramente **combinatória** para obter saídas e próximos estados a partir das entradas e estados atuais.



Seguir os passos de projeto de circuitos combinatórios (tabelas de verdade)

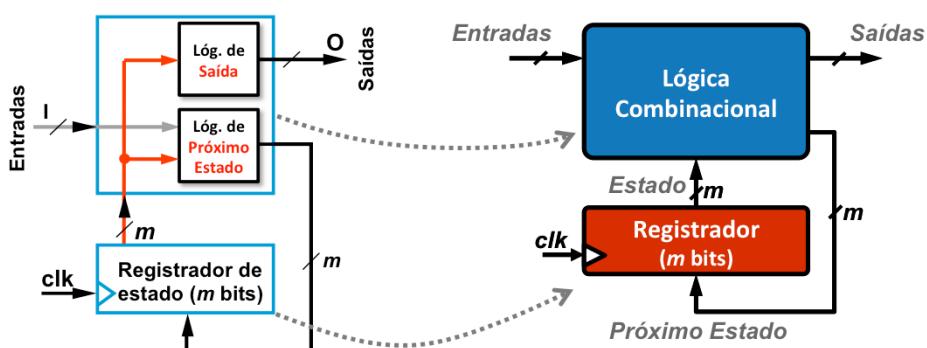
- Elementos de memória (**registrador**), controlados por um sinal de relógio para armazenar os estados atuais.



Para fazer o bloco de logica combinatória, basta seguir os mesmos passos de um projeto de circuito combinatório, que vem de tabelas verdade. Já para a parte de elementos de memória basta adicionar um registrador ao circuito. Veremos tudo isso com mais cuidado a seguir.

7.2. Projeto FSM

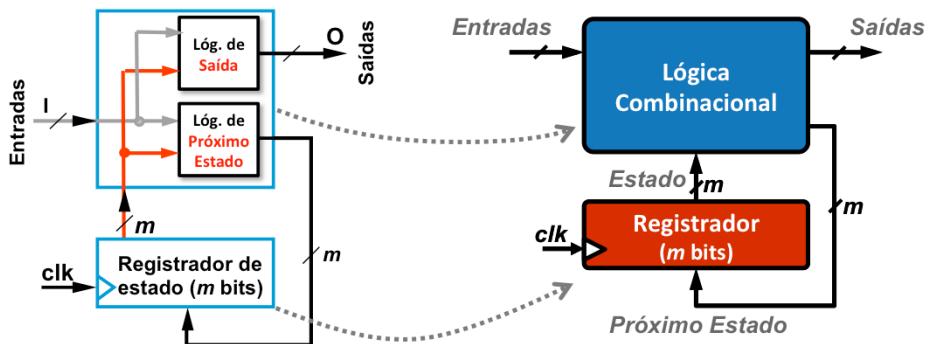
- Duas componentes:
 - Bloco de lógica puramente **combinatória** para obter saídas e próximos estados a partir das entradas e estados atuais.
 - Elementos de memória (**registrator**), controlados por um sinal de relógio para armazenar os estados atuais.
- **Modelo de Moore:**



Falando sobre a lógica combinacional, veja que podemos dividí-la em duas partes: lógica de saída, que define as saídas do circuito, e lógica de próximo estado, que define o próximo estado do circuito. É a partir da análise da lógica combinatória que dizemos se uma FSM é uma Máquina de Moore ou de Mealy. Neste caso, temos uma Máquina de Moore, afinal as entradas interferem apenas na lógica de próximo estado.

7.2. Projeto FSM

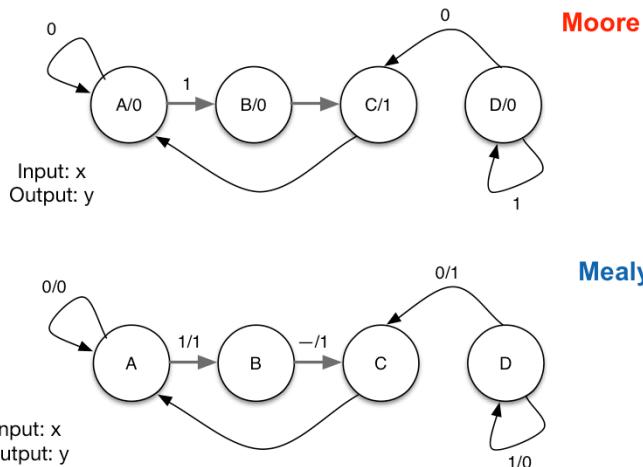
- Duas componentes:
 - Bloco de lógica puramente **combinatória** para obter saídas e próximos estados a partir das entradas e estados atuais.
 - Elementos de memória (**registrator**), controlados por um sinal de relógio para armazenar os estados atuais.
- **Modelo de Mealy:**



Aqui temos uma Máquina de Mealy, já que as entradas interferem também na lógica de saída.

7.2. Projeto FSM

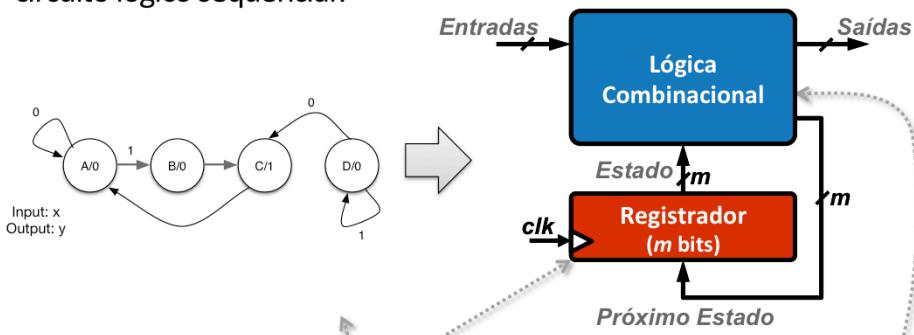
- Exemplos de diagrama de estados:



Aqui temos exemplos de diagramas de estados para os dois casos. Veja que a única diferença é que, na de Moore, temos a saída dentro do estado e, na de Mealy, na transição de estados.

7.2. Projeto FSM

- **Questão:** como transformar um diagrama de estados em um circuito lógico sequencial?



- **Para tal, basta:**
 - Codificar os **estados**
 - Escolher o **registrador** apropriado
 - Projetar a **lógica combinacional**

Ainda assim, como fazemos para transformar um diagrama de estados em um circuito de lógica sequencial? De modo geral, primeiro codificaremos os estados, em seguida escolheremos o registrador apropriado e, por último, projetaremos a lógica combinacional.

7.2. Projeto FSM

- Projeto de Circuitos Sequenciais Síncronos (FSM)
 - **Procedimento:**
 - Especificação formal:
 - Diagrama de estados
 - Fluxograma
 - Projecto:
 1. Codificação dos estados
 2. Tabelas de transição de estados
 3. Determinação das funções lógicas de saída e estado seguinte

Podemos então dividir o procedimento todo em um passo-a-passo de duas partes.

7.2. Projeto FSM

- Projeto de Circuitos Sequenciais Síncronos
 - **Procedimento:**
 - **Especificação formal:**
 - Diagrama de estados ← já visto nos exemplos anteriores...
 - Fluxograma
 - **Projecto:**
 1. Codificação dos estados
 2. Tabelas de transição de estados
 3. Determinação das funções lógicas de saída e estado seguinte

A primeira parte é a especificação formal. Ela inclui a confecção do diagrama de estados, já vista nos exemplos anteriores.

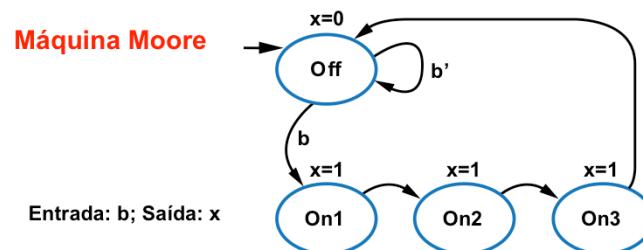
7.2. Projeto FSM

- **Exemplo:** circuito que mantém sua saída **x** em nível alto por 3 ciclos de clock quando um botão **b** for apertado.
 - Entrada: **b**
 - Saída: **x**
 - Estados:
Off, On1, On2 e On3.
- **Especificação formal:** *Obter o diagrama de estados*

Vejamos novamente o exemplo 3.

7.2. Projeto FSM

- **Exemplo:** Circuito que mantém sua saída **x** em nível alto por 3 ciclos de *clock* quando um botão **b** for apertado.
 - Entrada: **b**
 - Saída: **x**
 - Estados:
Off, **On1**, **On2** e **On3**.
- **Especificação formal:** *Obter o diagrama de estados*



Como visto anteriormente, este é o seu diagrama de estados. Agora podemos inferir que esta é uma Máquina de Moore.

7.2. Projeto FSM

- Projeto de Circuitos Sequenciais Síncronos
 - **Procedimento:**
 - Especificação formal:
 - Diagrama de estados
 - Fluxograma
 - **Projeto:**
 1. Codificação dos estados
 2. Tabelas de transição de estados
 3. Determinação das funções lógicas de saída e estado seguinte

Começamos a parte de projeto com a codificação dos estados.

7.2. Projeto FSM

Codificação dos estados

- **Método binário:** codificação usando o código binário
 - Método mais eficiente, pois usa o menor número de Flip-Flops possível
 - Menor número de funções para definir o próximo estado → menos portas lógicas (em geral...)
 - Considerando a existência de n estados ($E_0, E_1, E_2, \dots, E_{n-1}$), a codificação usando código binário natural irá usar k Flip-Flops, em que k é o menor inteiro igual ou superior a $\log_2(n)$
 - Exemplo:
 - 6 estados ($E_0, E_1, E_2, E_3, E_4, E_5$)
 - $k = \lceil \log_2(6) \rceil = \lceil 2.584 \rceil = 3$ Flip-Flops

Estado	Codificação		
	s_2	s_1	s_0
E_0	0	0	0
E_1	0	0	1
E_2	0	1	0
E_3	0	1	1
E_4	1	0	0
E_5	1	0	1

Existem várias codificações possíveis!!!

Para fazer isso existem dois métodos. O primeiro é o binário, no qual usa-se o código binário natural, como pode ser visto na tabela do slide. Logo, o número de bits (e, consequentemente, de flip-flops) para codificar todos os estados é o menor inteiro igual ou superior a $\log(n)$ na base 2. Veja, novamente, o exemplo do slide.

Este é o método mais eficiente, pois usa o menor número de flip-flops possível, e ainda normalmente implica em menos portas lógicas, afinal precisa-se de um menor número de funções para definir o próximo estado.

7.2. Projeto FSM

Codificação dos estados

- **Método One-hot:** codificação usando um Flip-Flop por estado
 - Usa tantos Flip-Flops quanto o número de estados
→ maior número de funções combinatórias a sintetizar!
→ mas cada uma destas funções é, em geral, mais simples...
 - Apenas um Flip-Flop tem a saída a '1' em cada instante

- Exemplo:

- 6 estados ($E_0, E_1, E_2, E_3, E_4, E_5$)

- 6 Flip-Flops

Estado	s_5	s_4	s_3	s_2	s_1	s_0
E_0	0	0	0	0	0	1
E_1	0	0	0	0	1	0
E_2	0	0	0	1	0	0
E_3	0	0	1	0	0	0
E_4	0	1	0	0	0	0
E_5	1	0	0	0	0	0

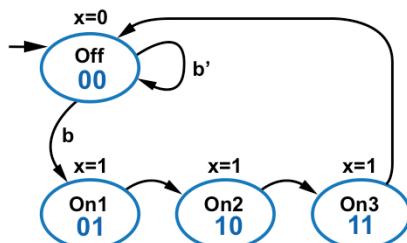
O outro método é o One-hot, que usa um flip-flop por estado. Isso resulta em um maior número de funções combinatórias a sintetizar, mas cada uma destas é, em geral, mais simples.

Como podemos ver na tabela de codificação do slide, apenas um flip-flop tem saída igual a '1' em cada estado.

7.2. Projeto FSM

- **Passo 1:** *Codificação os estados*

- Padrão:



Codificação binária

Estado	Código
Off	00
On1	01
On2	10
On3	11

- Qualquer codificação pode ser usada, desde que o código seja único para cada estado.

Voltando ao exemplo 3, a tabela de codificação de estados, usando a codificação binária, é a apresentada neste slide.

É importante ressaltar que tanto a codificação binária como a One-hot são apenas os métodos mais comuns, sendo que qualquer outra codificação pode ser usada, desde que o código seja único para cada estado.

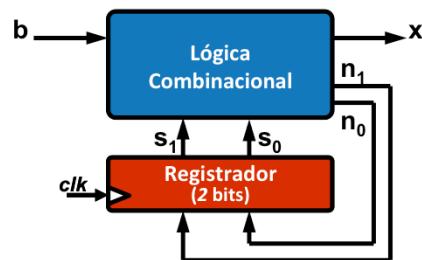
7.2. Projeto FSM

- Projeto de Circuitos Sequenciais Síncronos
 - **Procedimento:**
 - Especificação formal:
 - Diagrama de estados
 - Fluxograma
 - **Projeto:**
 1. Codificação dos estados
 2. Tabelas de transição de estados
 3. Determinação das funções lógicas de saída e estado seguinte

O próximo passo do projeto FSM é a construção da tabela de transição de estados.

7.2. Projeto FSM

- Passo 2: **Tabelas de transição de estados**
 - Tabela verdade para a **parte combinacional**

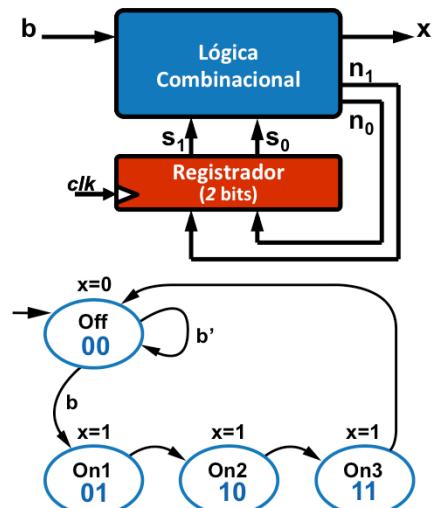


A tebal de transição de estados nada mais é do que uma tabela verdade, para que posteriormente possamos obter as equações lógicas para a parte combinacional.

7.2. Projeto FSM

- Passo 2: Tabelas de transição de estados

Entradas			Saídas		
s_1	s_0	b	x	n_1	n_0
0	0	0			
0	0	1			
0	1	0			
0	1	1			
1	0	0			
1	0	1			
1	1	0			
1	1	1			

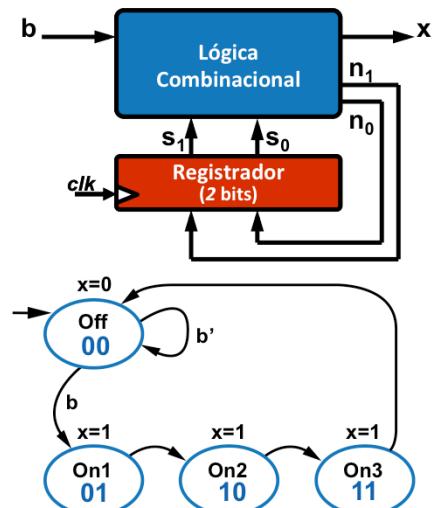


A grande diferença entre as tabelas verdade que vimos até aqui e a tabela de transição de estados é que esta última também tem, nas entradas, as variáveis de estado atuais e, nas saídas, as variáveis de próximo estado.

7.2. Projeto FSM

- Passo 2: Tabelas de transição de estados

Entradas			Saídas		
s_1	s_0	b	x	n_1	n_0
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	1	1	0
0	1	1	1	1	0
1	0	0	1	1	1
1	0	1	1	1	1
1	1	0	1	0	0
1	1	1	1	0	0



Assim, de acordo com a tabela de codificação dos estados, temos que quando o estado atual for "Off", ou seja, "00", e $b=0'$, o próximo estado ainda será "Off", ou "00". Acompanhe na tabela de transição de estados e no diagrama de estados. Na segunda linha, vemos que quando o estado atual for "Off" ("00") e $b=1'$, o próximo estado será "On1" ("01"). Na terceira e quarta percebemos que, não importa o valor de b , se o estado atual for "On1" ("01"), o próximo estado será "On2" ("10"). Tente ler o resto da tabela.

Perceba que o valor da saída x depende apenas do estado atual (Máquina de Moore) e isto também está expresso na tabela.

7.2. Projeto FSM

- Projeto de Circuitos Sequenciais Síncronos
 - **Procedimento:**
 - Especificação formal:
 - Diagrama de estados
 - Fluxograma
 - **Projeto:**
 1. Codificação dos estados
 2. Tabelas de transição de estados
 3. Determinação das funções lógicas de saída e estado seguinte

O terceiro e último passo é a determinação das funções lógicas de saída e estado seguinte.

7.2. Projeto FSM

- Passo 3: *Determinação das funções lógicas de saída e estado seguinte*

Entradas			Saídas		
s_1	s_0	b	x	n_1	n_0
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	1	1	0
0	1	1	1	1	0
1	0	0	1	1	1
1	0	1	1	1	1
1	1	0	1	0	0
1	1	1	1	0	0

7.2. Projeto FSM

- Passo 3: *Determinação das funções lógicas de saída e estado seguinte*

Entradas			Saídas		
s_1	s_0	b	x	n_1	n_0
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	1	1	0
0	1	1	1	1	0
1	0	0	1	1	1
1	0	1	1	1	1
1	1	0	1	0	0
1	1	1	1	0	0

$x = s_1 + s_0$
 $n_1 = s_1' \cdot s_0 + s_1 \cdot s_0'$
 $n_0 = s_1' \cdot s_0' \cdot b + s_1 \cdot s_0'$

Para realizar este passo basta tirar as funções da tabela de transição de estados, da mesma forma que fazemos com a tabela verdade.

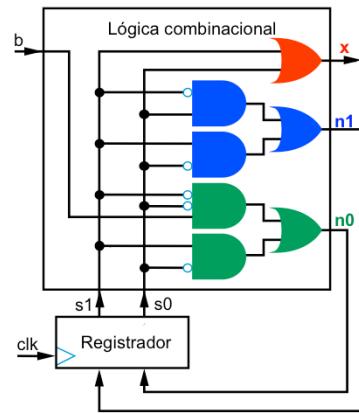
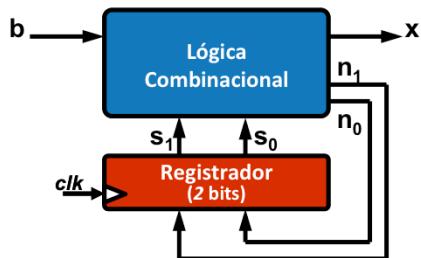
7.2. Projeto FSM

- Passo 3: Determinação das funções lógicas de saída e estado seguinte

$$x = s_1 + s_0$$

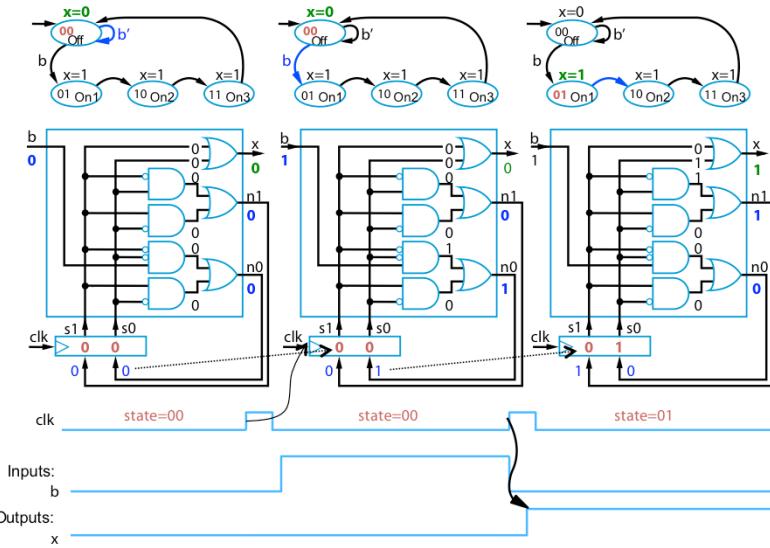
$$n1 = s_1' \cdot s_0 + s_1 \cdot s_0'$$

$$n0 = s_1' \cdot s_0' \cdot b + s_1 \cdot s_0'$$



Assim, ao final, basta transformar as funções booleanas obtidas em portas lógicas e temos o circuito correspondente ao diagrama de estados do início do exemplo. Note que a função correspondente a x pertence à lógica de saída e as funções de $n1$ e $n0$ à lógica de próximo estado.

7.2. Projeto FSM

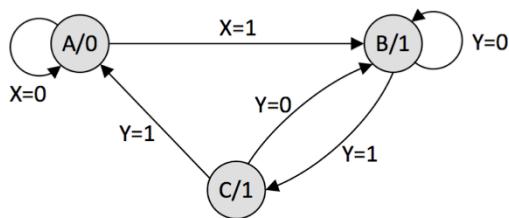


Veja que por causa do registrador, o próximo estado só se torna estado atual após a transição positiva do clock, exatamente como descrito no diagrama de estados.

PROBLEMAS

Problema 7.3. Considere o seguinte diagrama de estados de um circuito sequencial síncrono, caracterizado por duas entradas (X, Y) e uma saída (Z):

- Apresente, a tabela de transição de estados deste circuito. Considere a utilização de uma codificação binária e de flip-flops do tipo D.
- Sintetize as funções lógicas correspondentes às entradas dos flip-flops e à saída do circuito.



PROBLEMAS

Problema 7.3. Considere o seguinte diagrama de estados de um circuito sequencial síncrono, caracterizado por duas entradas (X,Y) e uma saída (Z):

- a) Apresente, a tabela de transição de estados deste circuito. Considere a utilização de uma codificação binária e de flip-flops do tipo D.

Solução: a)

EA	q1	q0
A	0	0
B	0	1
C	1	0
D	1	1

Estado atual (EA)=q1q0

Proximo Estado (PE)=Q1Q0

Entradas			Saídas			
q1	q0	X	Y	Q1	Q0	Z
0	0	0	—	0	0	0
0	0	1	—	0	1	0
0	1	—	0	0	1	1
0	1	—	1	1	0	1
1	0	—	0	0	1	1
1	0	—	1	0	0	1
1	1	—	—	—	—	—

PROBLEMAS

Problema 7.3. Considere o seguinte diagrama de estados de um circuito sequencial síncrono, caracterizado por duas entradas (X,Y) e uma saída (Z):

- b) Sintetize as funções lógicas correspondentes às entradas dos flip-flops e à saída do circuito.

b)

Q1

XY		00	01	11	10
q1q0		0	0	0	0
00		0	0	0	0
01		0	1	1	0
11		—	—	—	—
10		0	0	0	0

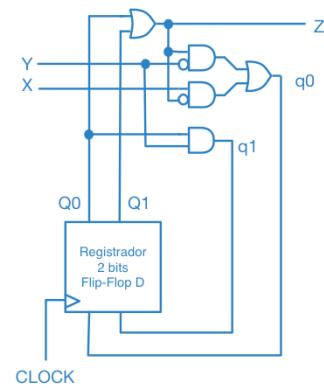
Z

XY		00	01	11	10
q1q0		0	0	0	0
00		0	0	0	0
01		1	1	1	1
11		—	—	—	—
10		1	1	1	1

Q0

XY		00	01	11	10
q1q0		0	0	1	1
00		0	0	0	1
01		1	0	0	1
11		—	—	—	—
10		1	0	0	1

$$\begin{aligned} Q1 &= q0Y \\ Q0 &= q1Y' + q0Y' + q1'q0'X = Y'(q1+q0) + X(q1+q0)' \\ Z &= q1 + q0 \end{aligned}$$

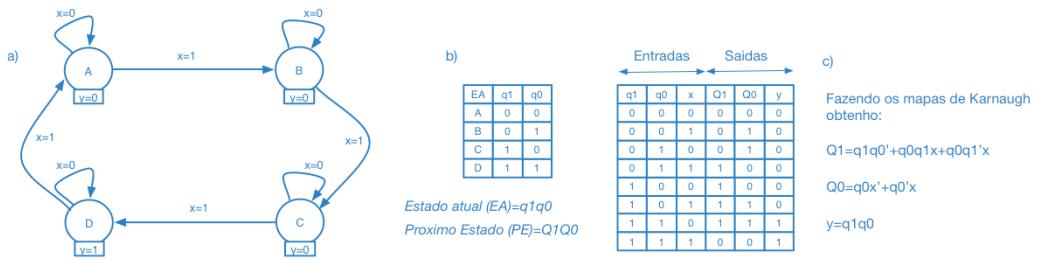


PROBLEMAS

Problema 7.9. Considere um contador de 2 bits com contagem ascendente, pausa e indicação de máximo da contagem.

- Faça o diagrama de estados
- Apresente a tabela de transição de estados deste circuito. Considere a utilização de uma codificação binária e de flip-flops do tipo D.
- Sintetize as funções lógicas correspondentes às entradas dos flip-flops e à saída do circuito.

Solução:





FEDERAL UNIVERSITY
OF SANTA CATARINA

EEL5105 – Circuitos e Técnicas Digitais

Aula 7

Prof. Héctor Pettenghi

hector@eel.ufsc.br

<http://hectorpettenghi.paginas.ufsc.br>

Material desenvolvido com apoio de arquivos de apresentação do livro de Frank Vahid

Exercícios

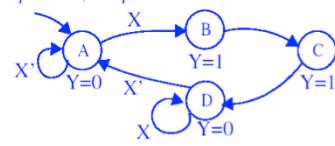
- Exercícios do Livro do **Frank Vahid**
 - **3.23** até **3.27**
 - **3.29** e **3.30**
 - **3.32** e **3.33**
 - **3.38** até **3.42**
- **A versão digital do livro do Frank Vahid está disponível no site da BU**
 - Mais especificamente em:
http://150.162.4.10/pergamon/biblioteca_s/php/login_pearson.php

Exercícios

- Respostas Livro do Frank Vahid

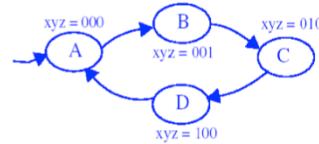
3.23

Inputs: X, Outputs: Y



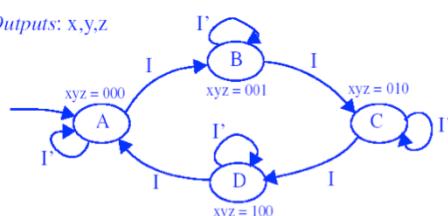
3.24

Inputs: None, Outputs: x,y,z



3.25

Inputs: I, Outputs: x,y,z

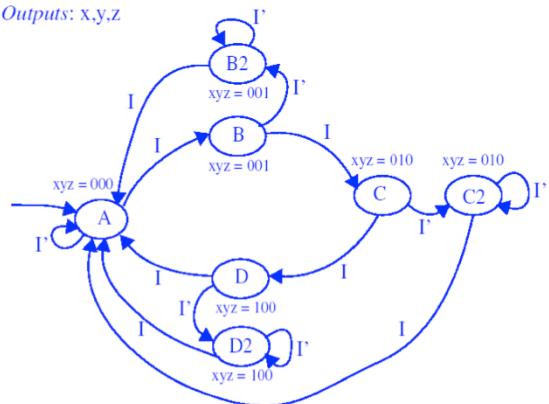


Exercícios

- Respostas Livro do Frank Vahid

3.26

Inputs: I, Outputs: x,y,z

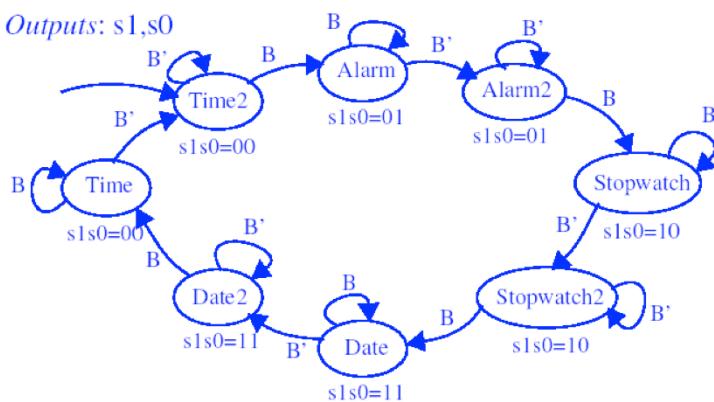


Exercícios

- Respostas Livro do Frank Vahid

3.27

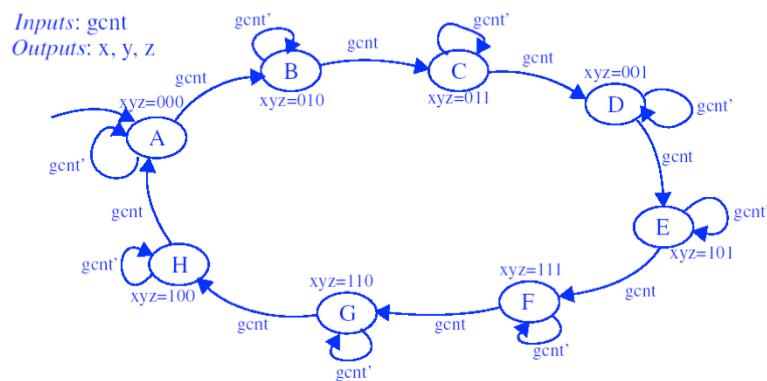
Inputs: B, Outputs: s1,s0



Exercícios

- Respostas Livro do Frank Vahid

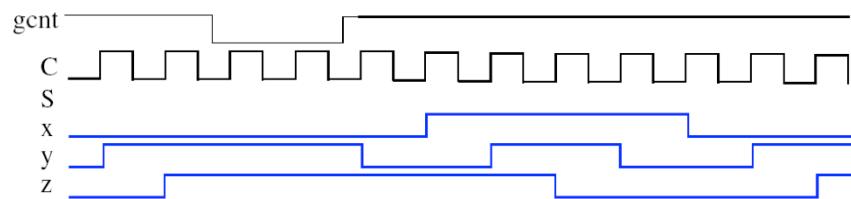
3.29



Exercícios

- Respostas Livro do Frank Vahid

3.30



Exercícios

- Respostas Livro do **Frank Vahid**

3.32 a) 2 bits

b) 3 bits

c) 4 bits

d) 5 bits

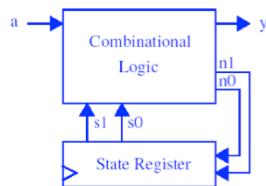
e) 10 bits

3.33

$2^{16} = 65,536$ possible states

Exercícios

- Resposta do 3.38



A straightforward encoding is A=00, B=01, C=10, D=11.

Inputs			Outputs		
s1	s0	a	n1	n0	y
0	0	0	0	1	0
0	0	1	0	0	0
0	1	0	0	1	1
0	1	1	1	0	1
1	0	0	1	1	1
1	0	1	1	1	1
1	1	0	0	0	0
1	1	1	0	0	0

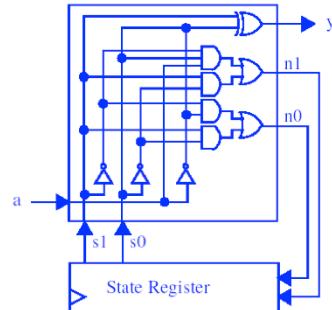
Exercícios

- Resposta do 3.38

$$n1 = s1's0a + s1s0'a' + s1s0'a = s1's0a + s1s0'$$

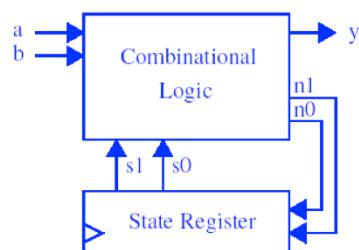
$$n0 = s1's0'a' + s1's0a' + s1s0'a' + s1s0'a = s1'a' + s1s0'$$

$$y = s1's0a' + s1's0a + s1s0'a' + s1s0'a = s1's0 + s1s0' = s1 \text{ xor } s0$$



Exercícios

- Resposta do 3.39



A straightforward encoding is A=00, B=01, C=10, D=11.

Inputs				Outputs		
s1	s0	a	b	n1	n0	y
0	0	0	0	1	0	0
0	0	0	1	0	1	0
0	0	1	0	0	0	0
0	0	1	1	0	0	0
0	1	0	0	0	1	1
0	1	0	1	0	1	1
0	1	1	0	1	0	1
0	1	1	1	1	0	1
1	0	0	0	1	0	1
1	0	0	1	1	1	1
1	0	1	0	1	0	1
1	0	1	1	1	1	1
1	1	0	0	0	0	0
1	1	0	1	0	0	0
1	1	1	0	0	0	0
1	1	1	1	0	0	0

Exercícios

- Resposta do 3.39

Step 5 - Implement the combinational logic

$$n1 = s1's0'a'b' + s1's0a + s1s0'$$

$$n0 = s1's0'a'b + s1's0a' + s1s0'b$$

$$y = s1's0 + s1s0'$$

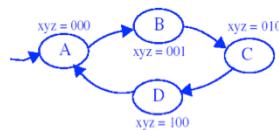
Note: The above equations can be minimized further.

Exercícios

- Resposta do 3.40

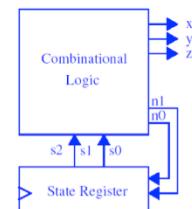
Step 1 - Capture the FSM

Inputs: None, Outputs: x,y,z



The FSM was created during Exercise 3.24.

Step 2 - Create the architecture



Step 3 - Encode the states

A straightforward encoding is A=00, B=01, C=10, D=11.

Step 4 - Create the state table

Inputs		Outputs				
s1	s0	n1	n0	x	y	z
0	0	0	1	0	0	0
0	1	1	0	0	0	1
1	0	1	1	0	1	0
1	1	0	0	1	0	0

Step 5 - Implement the combinational logic

$$n1 = s1's0 + s1s0' = s1 \text{ XOR } s0$$

$$n0 = s1's0' + s1s0' = s0'$$

$$x = s1s0$$

$$y = s1s0'$$

$$z = s1's0$$