SD132 – Circuitos Digitais II
A-205 Máquinas de Estado Moore

Autores	
Gilmar Silva Beserra (INATEL)	Daniel Muñoz Arboleda (INSTITUIÇÃO)
Nome 3 (INSTITUIÇÃO)	Nome 4 (INSTITUIÇÃO)

Histórico de revisões		
10/02/2025	V1.0	Versão inicial

Tópicos

- Definição e Conceitos de Máquinas de Estados Finitos (FSMs)
- Modelo Genérico de Circuitos Síncronos
- Classificação de FSMs
- Representação Matemática de FSMs
- Representação Gráfica de FSMs
- Projeto/Síntese de FSMs
- Análise de FSMs
- Exemplos de Projeto/Síntese e Análise de Máquinas de Moore
- Cuidados Básicos na Implementação de FSMs
- Implementação de Máquinas de Moore em HDL
- Exemplos e Exercícios

Página em branco.

Aula 14
Máquinas de Estados Finitos

Conceitos

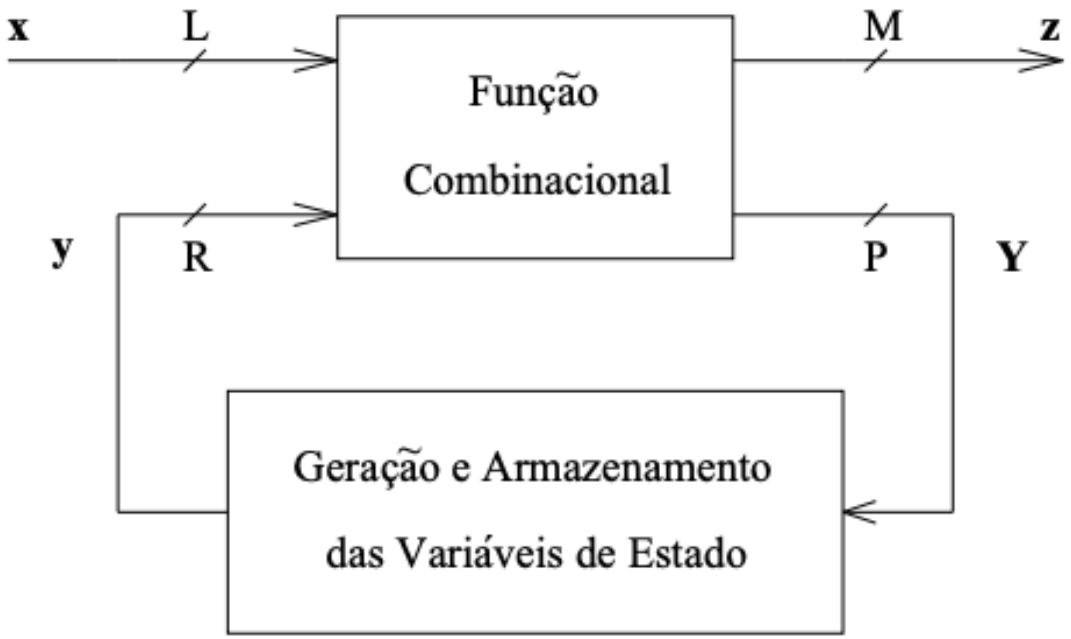
- **Máquinas de Estados Finitos**: modelos computacionais usados para representar sistemas ordenados no tempo.
- **Estado**: termo usado para representar os modos de operação do sistema. O estado geralmente é armazenado em memórias ou registradores.
- **“Finito”**: termo usado para diferenciar as FSMs de modelos matemáticos com número infinito de estados. O fato de ter estados finitos possibilita a sua implementação física.
- **Aplicações**: modelar algoritmos, processos, circuitos, e em geral sistemas com diferentes modos de operação.

Introdução

- **FSMs** também são conhecidas como **autômatos de estados finitos**. Não são máquinas propriamente ditas (*hardware*), apenas são modelos para representar sistemas.
- Em eletrônica digital usamos **FSMs** para **modelar circuitos sequenciais (síncronos e assíncronos)**, os quais possuem **memórias** ou **registradores** para armazenar o estado atual.

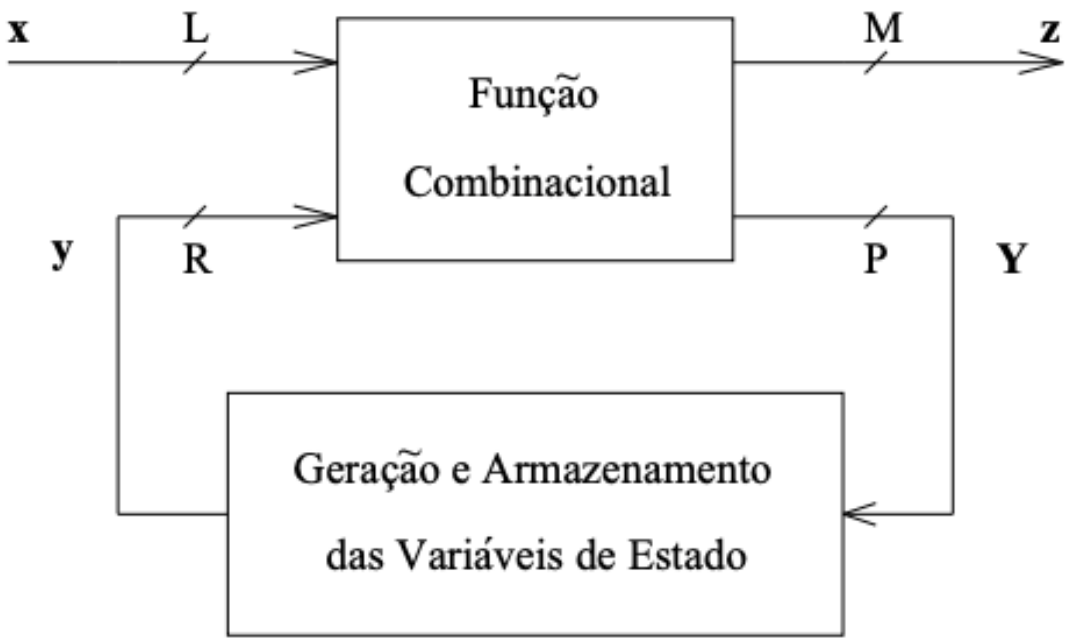
Modelo Genérico para Circuitos Sequenciais

- $x_i \in$ **x**, i = 1, 2, ... , **L** são variáveis de entrada
- $z_i \in$ **z**, i = 1, 2, ... , **M** são variáveis de saída
- $Y_i \in$ **Y**, i = 1, 2, ... , **P** são variáveis de excitação
- $y_i \in$ **y**, i = 1, 2, ... , **R** são variáveis de estado



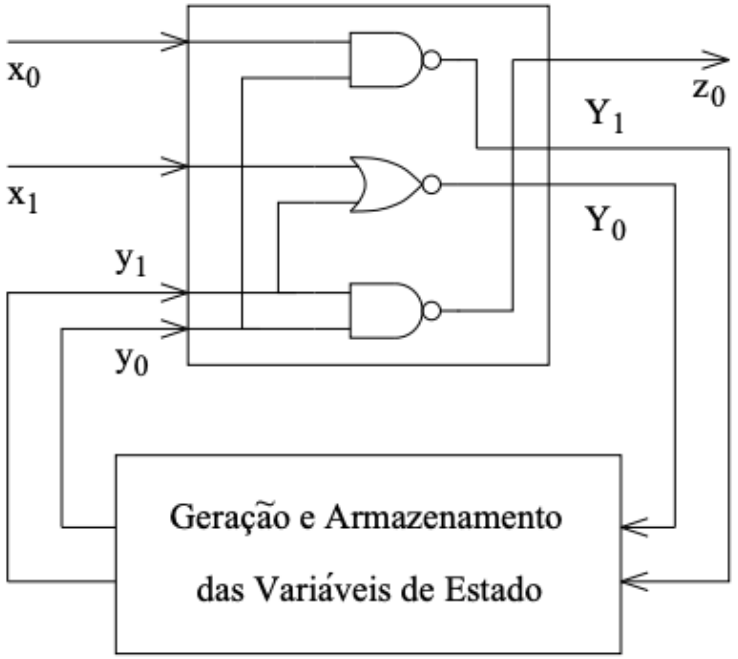
Modelo Genérico para Circuitos Sequenciais

- O circuito combinacional pode ser implementado com portas lógicas, memórias ROM ou circuitos PLA, por exemplo
- A Geração e Armazenamento das Variáveis de Estado representa um dispositivo genérico de memória cujas funções são armazenar o **estado atual** e gerar o **próximo estado**



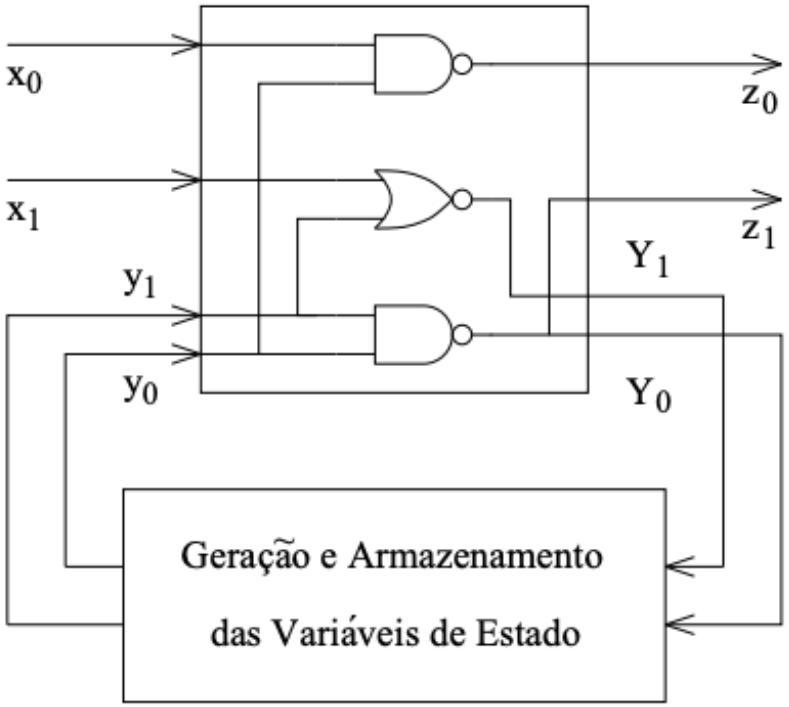
Classificação

- Existem dois modelos para síntese (implementação) de circuitos sequenciais: Moore e Mealy
- **Máquinas de Moore**: as saídas dependem **apenas** do **estado atual**



Classificação

- **Máquinas de Mealy**: as saídas dependem do **estado atual** e das **entradas**



Mealy x Moore

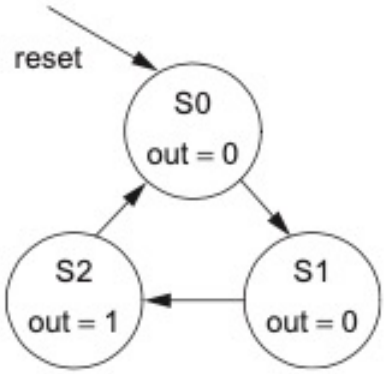
- Geralmente as máquinas de **Mealy** são implementadas por **circuitos mais simples** que as de Moore
- Os valores dos **sinais de saída permanecem constantes entre dois estados consecutivos** nas máquinas de **Moore**, o que simplifica a interação entre os blocos e a depuração de erros

Representação Matemática de uma FSM

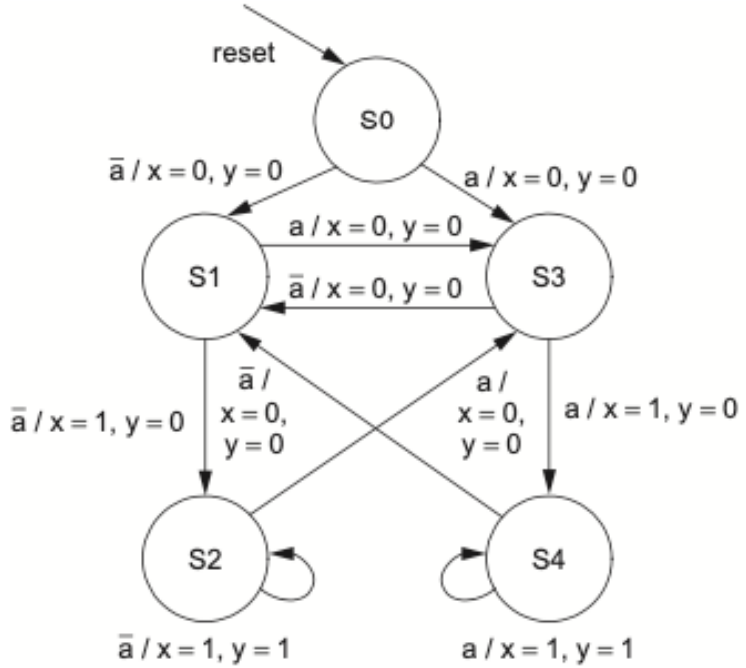
- Uma **FSM** consiste em uma tupla $M = (I,O,S,f,g,\sigma)$
- I: é um conjunto finito de símbolos de entrada;
- O: é um conjunto finito de símbolos de saída;
- S: é um conjunto finito de estados;
- $f: SxI \rightarrow S$ é uma função que define o próximo estado;
- $g: SxI \rightarrow S$ é uma função que define a saída;
- σ: é um estado inicial, onde σ pertence a S.

Representação Gráfica de uma FSM

- Vértices representam estados
- Arestas representam transições de estado
- Saída após uma barra (/), no estado (**Moore**) ou na aresta (**Mealy**)
- Seta representa um estado inicial
- **Exemplos**:



Moore



Mealy

Projeto/Síntese de FSMs

- **Procedimento**:
  1) Descrição funcional detalhada
  2) Diagrama de estados
  3) Tabela de transição de estados e saídas
  4) Tabela de transição reduzida
  5) Codificação de estados
  6) Circuito combinatório para transição de estados
  7) Circuito combinatório para saídas
  8) Diagrama lógico do circuito.

Projeto/Síntese de FSMs

- **Descrição funcional detalhada**
  - Anotar as especificações do sistema (geralmente são variáveis e dados numéricos)
  - Quanto mais detalhado, melhor
  - Geralmente é um processo iterativo para aprimorar o conhecimento do problema, mas deve ser concluído antes de iniciar o desenvolvimento
  - Ao final da descrição é possível prever quais vão ser os estados necessários, o que faz cada estado, quantos bits (quantos flip-flops) serão necessários para registrar o estado e qual a saída em cada estado

Projeto/Síntese de FSMs

- **Diagrama de Estados**
  - Adotar um modelo (**Mealy** ou **Moore**)
  - Separar em estados cada etapa/tarefa/função do sistema, processo ou algoritmo
  - Indicar o estado inicial
  - Indicar condições de transição de entradas
  - Indicar saída(s) em cada transição ou estado
  - Tomar cuidado com condições de transição mal formuladas:
    - A partir de cada estado, uma e apenas uma transição deve ser válida
    - A partir de cada estado, deve ter pelo menos uma transição válida
    - A verificação dessas condições será objeto de estudo futuramente

Projeto/Síntese de FSMs

- **Tabela de Transição de Estados e Saídas**
  - Transformar o diagrama de estados em uma tabela de transição
  - Na coluna esquerda, separar por linhas o estado atual
  - Para cada combinação das entradas, colocar próximo estado
  - Dependendo se o modelo é de Mealy ou Moore, colocar a(s) saída (s) dependendo da combinação de entradas ou apenas do estado atual

Projeto/Síntese de FSMs

- **Tabela de Transição Reduzida**
  - Na tabela de transição, verificar se tem linhas redundantes
  - Quando há duas linhas iguais, ou seja, a partir do mesmo estado atual temos o mesmo próximo estado E a(s) mesma(s) saída(s), então temos estados redundantes
  - Eliminar as linhas que correspondem ao(s) estado(s) redundante(s)

Projeto/Síntese de FSMs

- **Codificação de Estados**
  - Definir número de bits para codificar o estado
  - Se número de estados = $\mathbf{2^n}$ então serão necessários **n** flip-flops
  - Se número de estados < $\mathbf{2^n}$ então serão necessários **n** flip-flops, porém existirão estados não codificados, com comportamento não definido
  - Adotar um código binário (binario, Gray, Johnson, BCD, etc.)
  - Codificar cada estado

Projeto/Síntese de FSMs

- **Circuito Combinacional para Transição de Estados**
  - Para cada bit do estado, obter um mapa de Karnaugh (mapa K)
  - Associar elementos adjacentes do mapa K no intuito de obter as equações booleanas de transição de estado
  - Pode-se usar soma de produtos (associação de **‘1’s**) ou produtos de somas (associação de **‘0’s**)
  - **Dicas**: adotar a associação de **‘1’s**, a não ser que o problema solicite o contrário, e maximizar as associações (por exemplo, é melhor associar uma quadra do que usar duas duplas, pois dessa forma o circuito combinacional terá menos portas lógicas)

Projeto/Síntese de FSMs

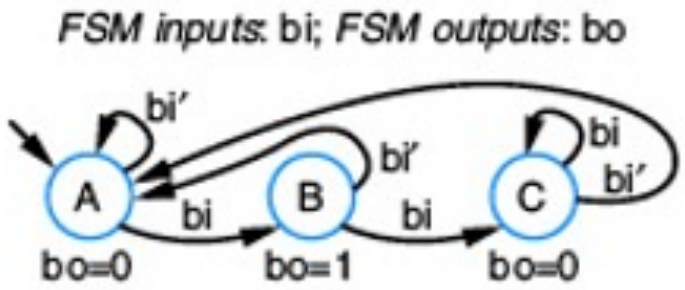
- **Circuito Combinacional para Saídas**
  - Para cada bit da saída, obter um mapa de Karnaugh (mapa K)
  - Associar elementos adjacentes do mapa K no intuito de obter as equações booleanas das saídas
  - Pode-se usar soma de produtos (associação de **‘1’s**) ou produtos de somas (associação de **‘0’s**)
  - **Dicas**: adotar a associação de **‘1’s**, a não ser que o problema solicite o contrário, e maximizar as associações (por exemplo, é melhor associar uma quadra do que usar duas duplas, pois dessa forma o circuito combinacional terá menos portas lógicas)

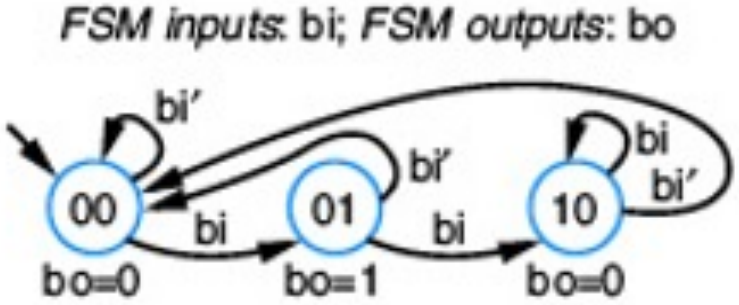
Projeto/Síntese de FSMs

- **Diagrama Lógico do Circuito**
  - A partir das equações booleanas de transição de estado e saída, obter um circuito equivalente
  - Iniciar desenhando os flip-flops. Cada bit de estado precisa de um flip-flop
  - A entrada do flip-flop é a saída de uma equação booleana (um bit), ou seja, o próximo estado
  - A saída do flip-flop é o bit de estado registrado, ou seja, o estado atual
  - Usar portas AND, OR, NAND, NOR ou XOR, dependendo da equação booleana

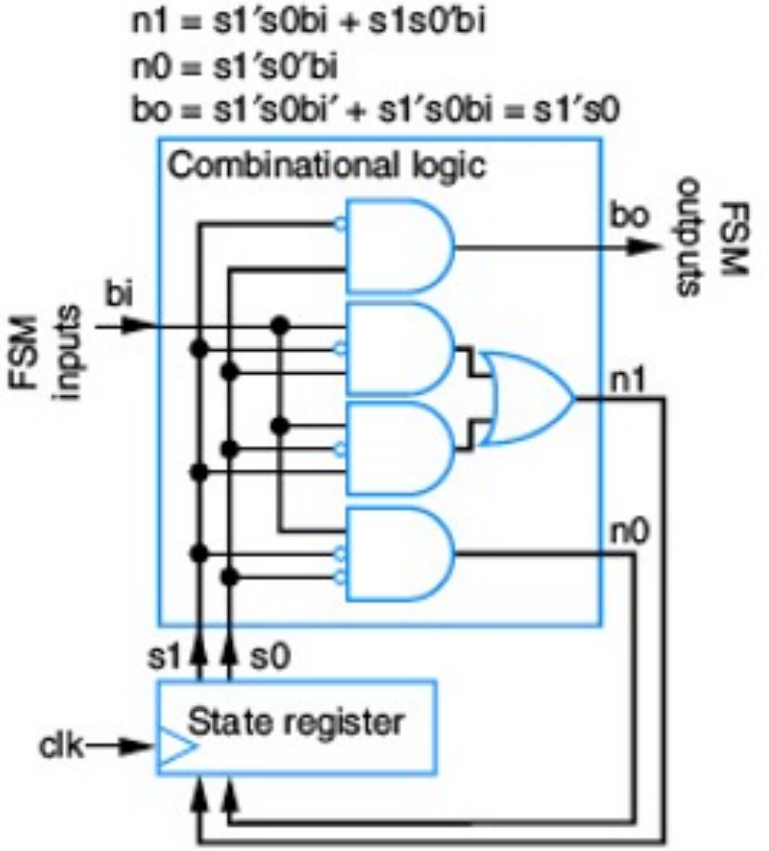
Projeto/Síntese de FSMs

- **Exemplo**: arquitetura, codificação, tabela de estados e saídas, e circuito





	Combinational logic					
	Inputs			Outputs		
	s1	s0	bi	n1	n0	bo
A	0	0	0	0	0	0
	0	0	1	0	1	0
B	0	1	0	0	0	1
	0	1	1	1	0	1
C	1	0	0	0	0	0
	1	0	1	1	0	0
unused	1	1	0	0	0	0
	1	1	1	0	0	0



Análise de FSMs

- **Procedimento**:
  1) Analisar o diagrama lógico do circuito
  2) Obter as equações booleanas
  3) Obter a tabela de transição de estados e saídas
  4) Codificar estados
  5) Obter arquitetura padrão
  6) Obter diagrama de estados

  Finalmente, interpretar funcionamento!

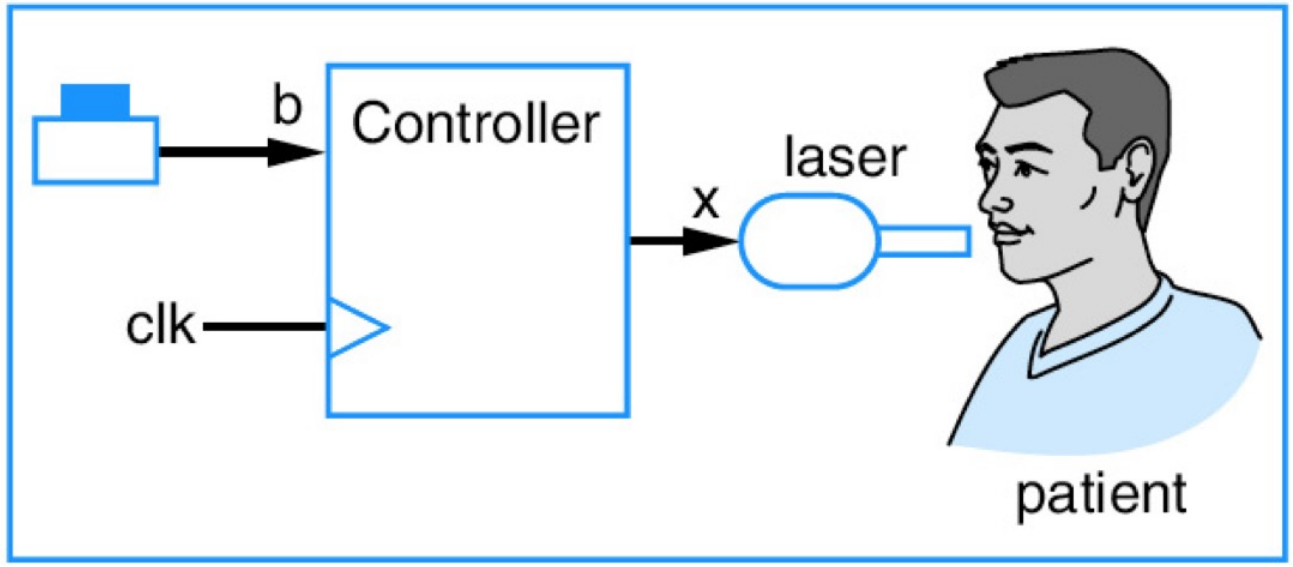
Página em branco.

Aula 15
Projeto/Síntese e Análise de Máquinas de Estado Moore

Execução
Inatel UNIFEI HARDWARE BR Uema cepedi DIGITAL
Realização
CHIP TECH BRASIL Softex MINISTÉRIO DA CIÊNCIA, TECNOLOGIA E INOVAÇÃO GOVERNO FEDERAL BRASIL UNIÃO E RECONSTRUÇÃO

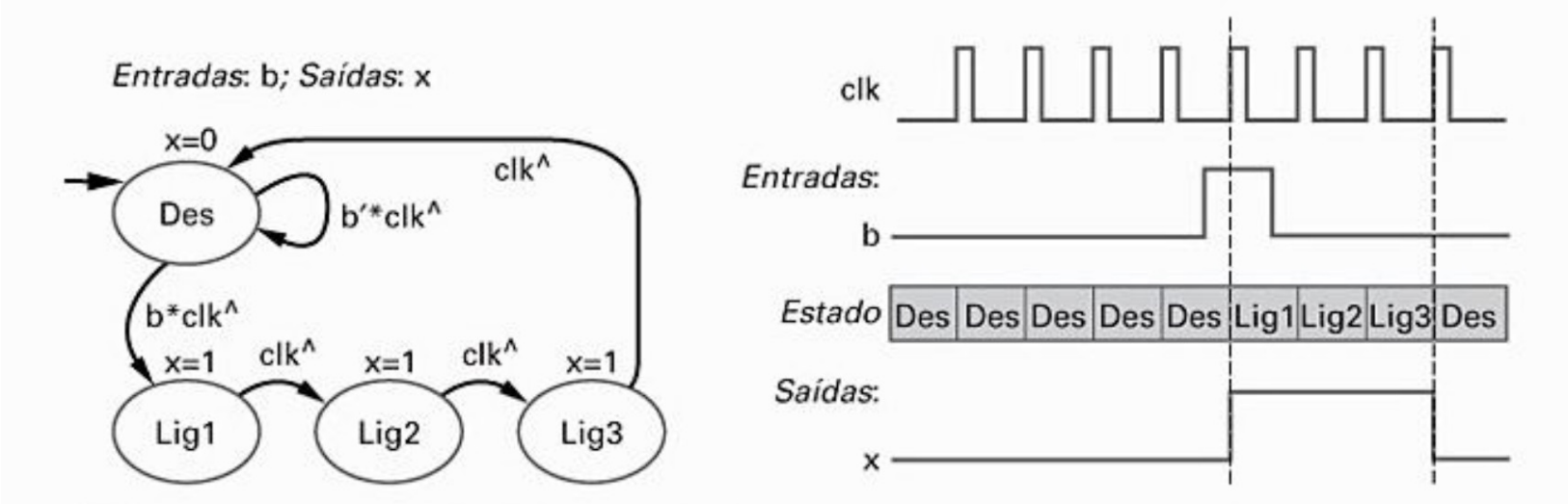
Projeto/Síntese – Exemplo 1

- Projeto de um circuito sequencial que controle um pulso para um sistema de cirurgia a laser. Quando o botão **b = 1**, o laser deve ligar durante três ciclos de clock



Projeto/Síntese – Exemplo 1

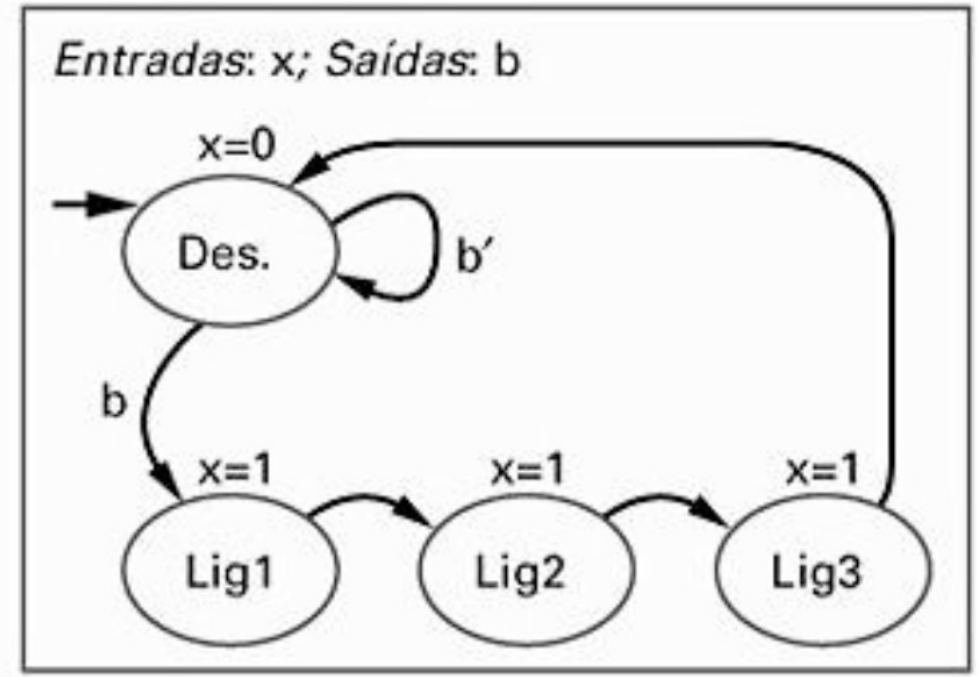
- Descrição detalhada:
  - Se **b = 1**, então **x = 1** por três ciclos de clock
  - Assumindo o período do clock como 10 ns, o laser deve ligar durante 30 ns
  - Solução por *software* é inviável (mais lenta)



  - Des: **x = 0**
  - Lig1, Lig2 e Lig3: **x = 1**

Projeto/Síntese – Exemplo 1

- Diagrama de estados



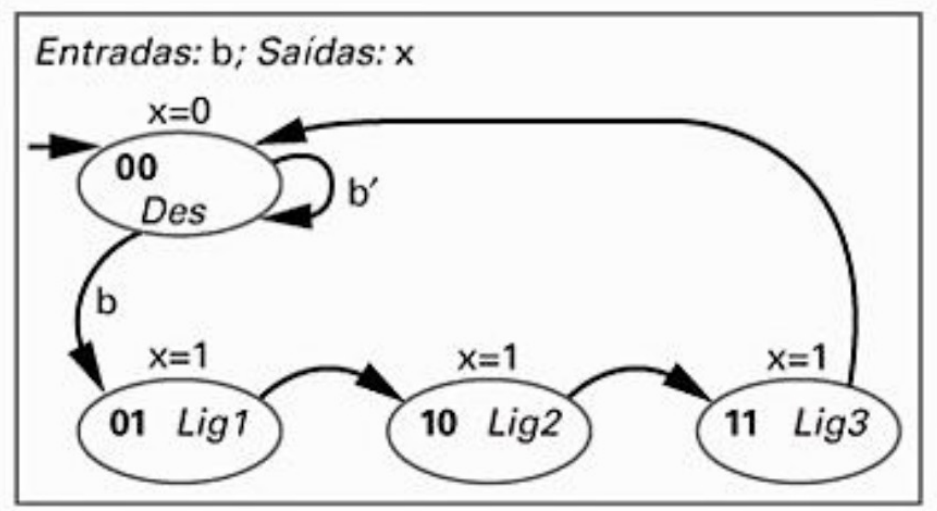
Projeto/Síntese – Exemplo 1

- Tabela de estados
  - Não há duas linhas iguais: tabela já é a reduzida

ESTADO ATUAL	PRÓXIMO ESTADO		SAÍDA
	b = 0	b = 1	x
Des	Des	Lig1	0
Lig1	Lig2	Lig2	1
Lig2	Lig3	Lig3	1
Lig3	Des	Des	1

Projeto/Síntese – Exemplo 1

- Codificação de estados
  - Variáveis de estado atual: s_1 e s_0
  - Próximo estado: n_1 e n_0
  - Usando o código binário
    - Des = 00
    - Lig1 = 01
    - Lig2 = 10
    - Lig3 = 11



ESTADO ATUAL (s_1 s_0)	PRÓXIMO ESTADO (n_1 n_0)		SAÍDA
	b = 0	b = 1	x
0 0	0 0	0 1	0
0 1	1 0	1 0	1
1 0	1 1	1 1	1
1 1	0 0	0 0	1

Projeto/Síntese – Exemplo 1

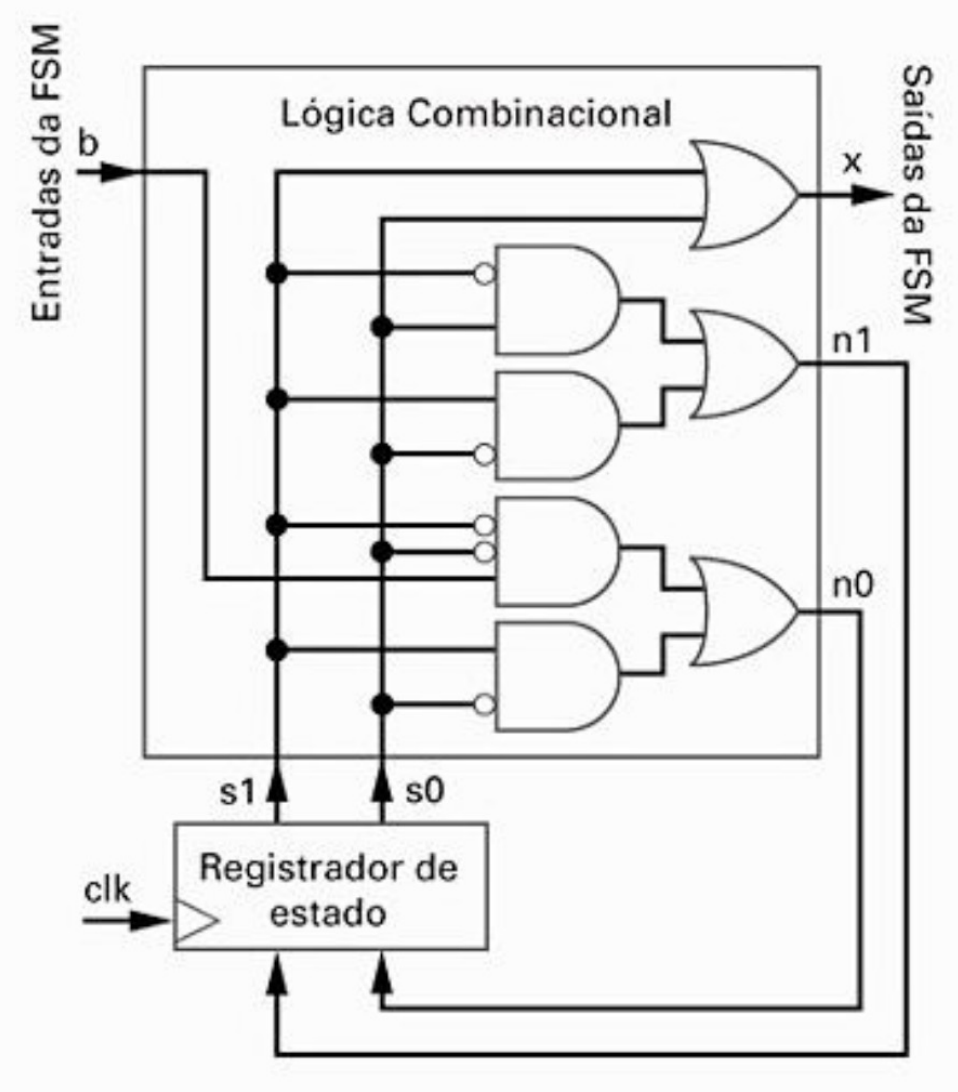
- Equações

$n_1 = s_1'.s_0 + s_1.s_0'$

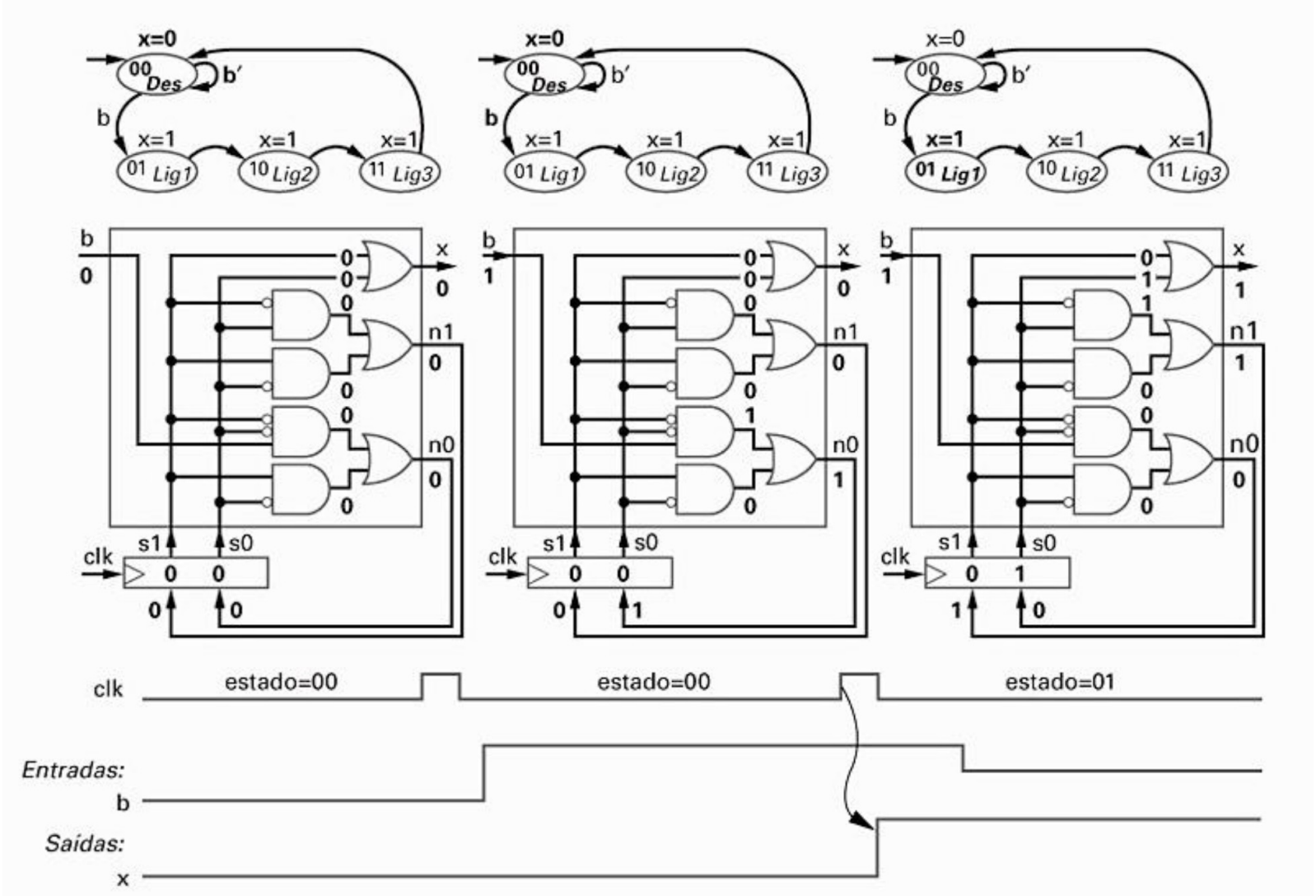
$n_0 = s_1'.s_0'.b + s_1.s_0'$

$x = s_1 + s_0$

- Diagrama do circuito lógico



Projeto/Síntese – Exemplo 1

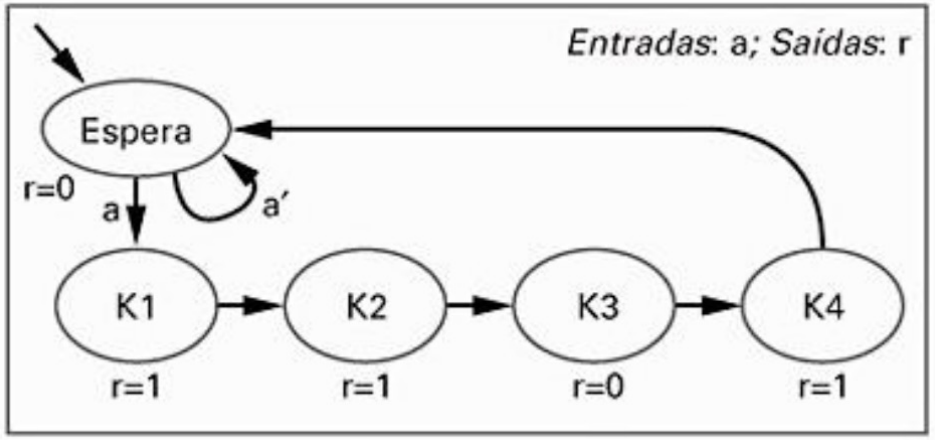


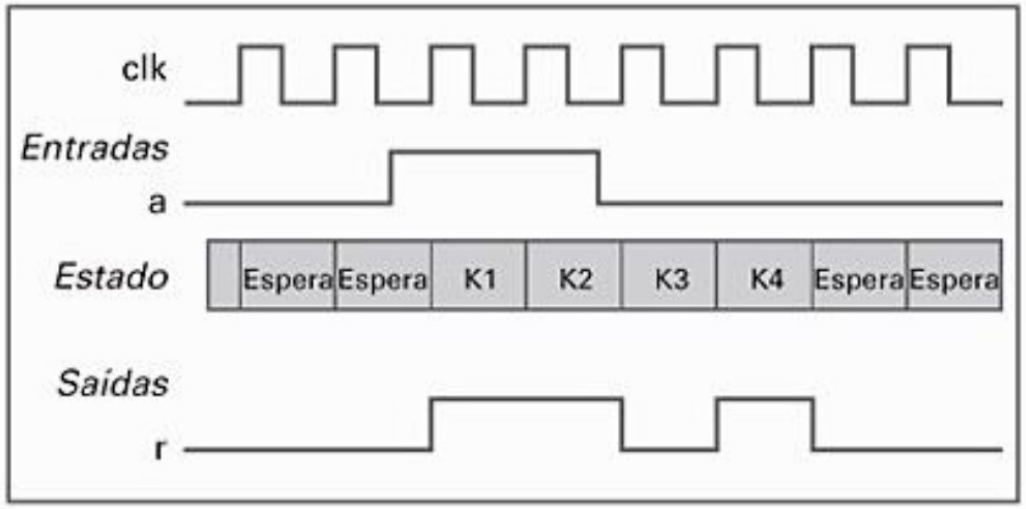
Projeto/Síntese – Exemplo 2

- Projeto do bloco de controle de uma chave de carro segura. Quando a chave é girada na ignição, o carro envia um sinal via rádio solicitando o seu identificador (**ID = 1011**). Caso não receba resposta ou o **ID** seja diferente, o carro não dará a partida.

Projeto/Síntese – Exemplo 2

- Descrição detalhada e diagrama de estados
  - Assumir que o bloco de controle tem uma entrada **a** que será 1 quando o carro solicitar o **ID**. Os 4 bits são enviados pela saída **r** de forma serial, começando pelo bit menos significativo.

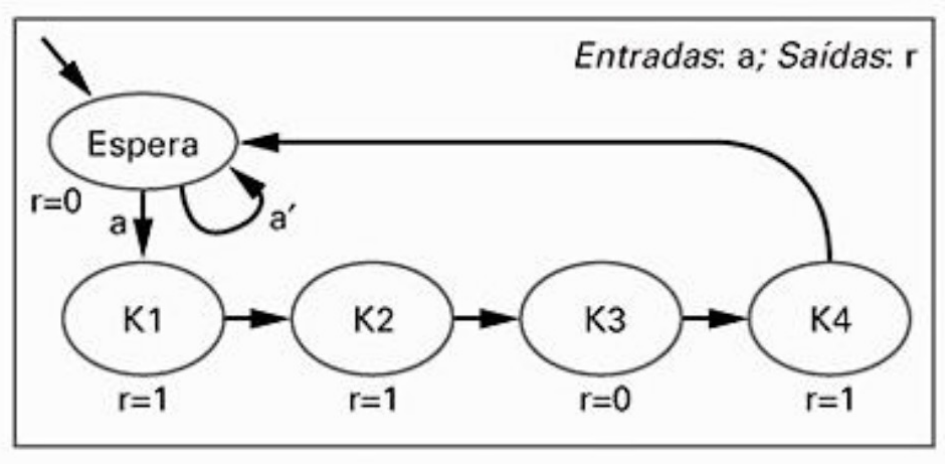




Projeto/Síntese – Exemplo 2

- Tabela de estados
  - Não há duas linhas iguais: tabela já é a reduzida

ESTADO ATUAL	PRÓXIMO ESTADO		SAÍDA
	a = 0	a = 1	r
Espera	Espera	K1	0
K1	K2	K2	1
K2	K3	K3	1
K3	K4	K4	0
K4	Espera	Espera	1



Projeto/Síntese – Exemplo 2

- Codificação de estados
  - Variáveis de estado atual: s_2, s_1 e s_0
  - Próximo estado: n_2 , n_1 e n_0
  - Usando o código binário
    - Espera = 000
    - K1 = 001
    - K2 = 010
    - K3 = 011
    - K4 = 100

ESTADO ATUAL (s_2 s_1 s_0)	PRÓXIMO ESTADO (n_2 n_1 n_0)		SAÍDA
	a = 0	a = 1	r
0 0 0	0 0 0	0 0 1	0
0 0 1	0 1 0	0 1 0	1
0 1 0	0 1 1	0 1 1	1
0 1 1	1 0 0	1 0 0	0
1 0 0	0 0 0	0 0 0	1
1 0 1	0 0 0	0 0 0	0
1 1 0	0 0 0	0 0 0	0
1 1 1	0 0 0	0 0 0	0

  - Estados não utilizados: 101, 110 e 111
    - **Custo mínimo**: usar ***don't care*** para os próximos estados
    - **Risco mínimo**: usar ***0's*** para os próximos estados (FSM vai para um estado conhecido, geralmente o inicial)

Projeto/Síntese – Exemplo 2

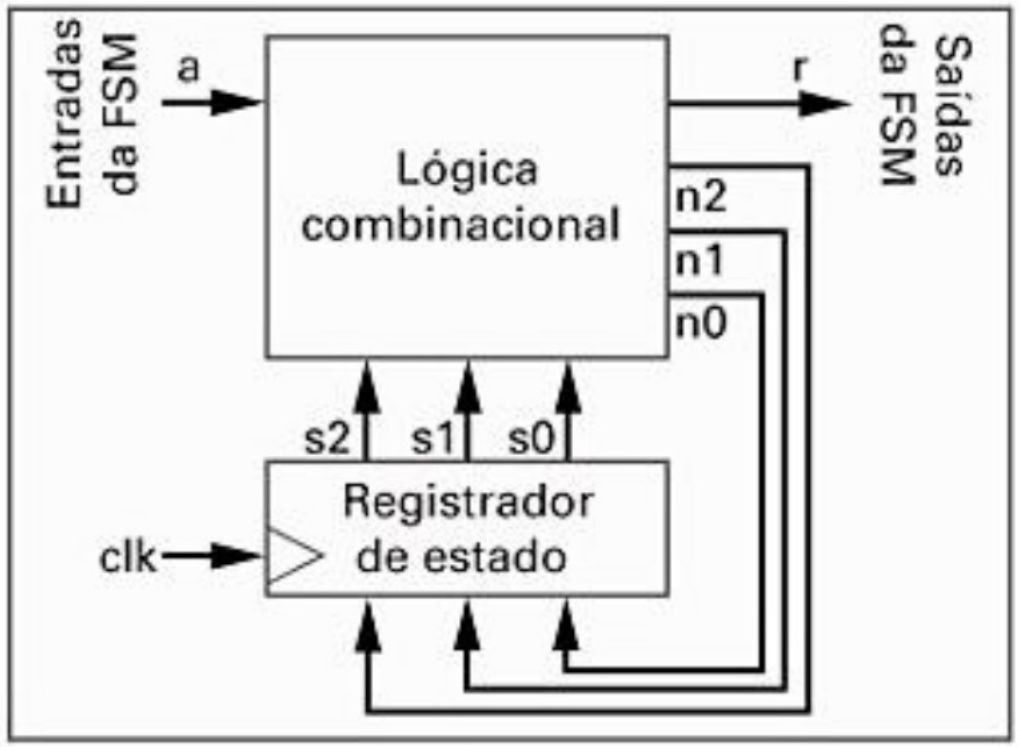
- Equações

$n_2 = s_1'.s_0 + s_1.s_0'$

$n_1 = s_1'.s_0 + s_1.s_0'$

$n_0 = s_1'.s_0'.a + s_1.s_0'$

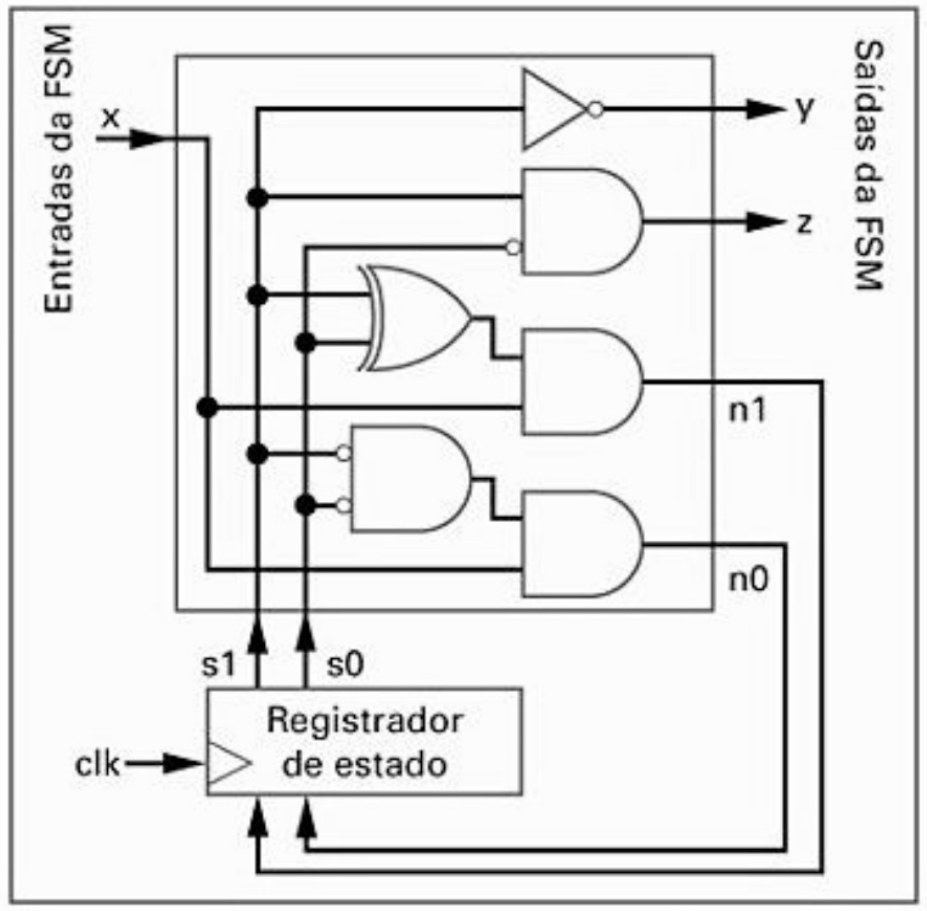
$r = s_1 + s_0$



- Diagrama do circuito lógico

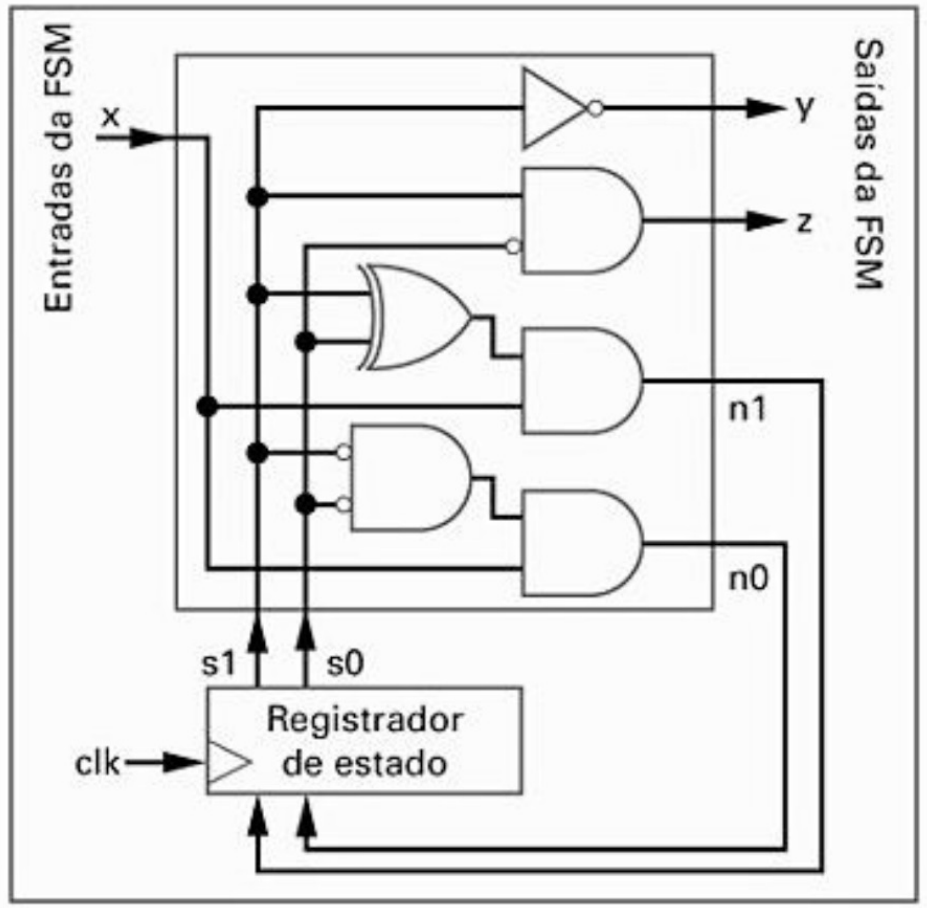
Análise – Exemplo

- Analisar o diagrama do circuito ao lado
- Identificar
  - O registrador de estados
  - A lógica combinacional de transição de estados
  - A lógica combinacional para as saídas



Análise – Exemplo

- Obter as equações booleanas a partir dos circuitos combinacionais



$y = s1'$

$z = s1.s0'$

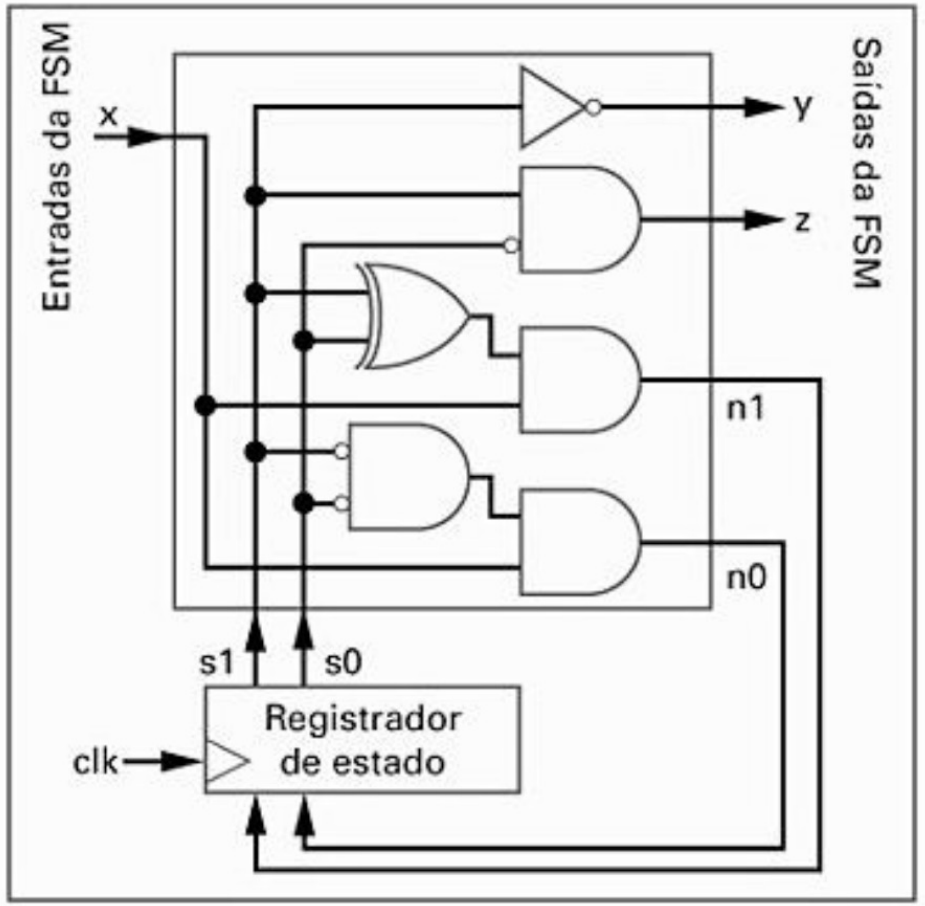
$n1 = (s1 \oplus s0).x$

$n0 = (s1'.s0').x$

Análise – Exemplo

- Obter a tabela de transição de estados e saídas

ESTADO ATUAL (s_1 s_0)	PRÓXIMO ESTADO (n_1 n_0)		SAÍDAS	
	x = 0	x = 1	y	z
0 0	0 0	0 1	1	0
0 1	0 0	1 0	1	0
1 0	0 0	1 0	0	1
1 1	0 0	0 0	0	0

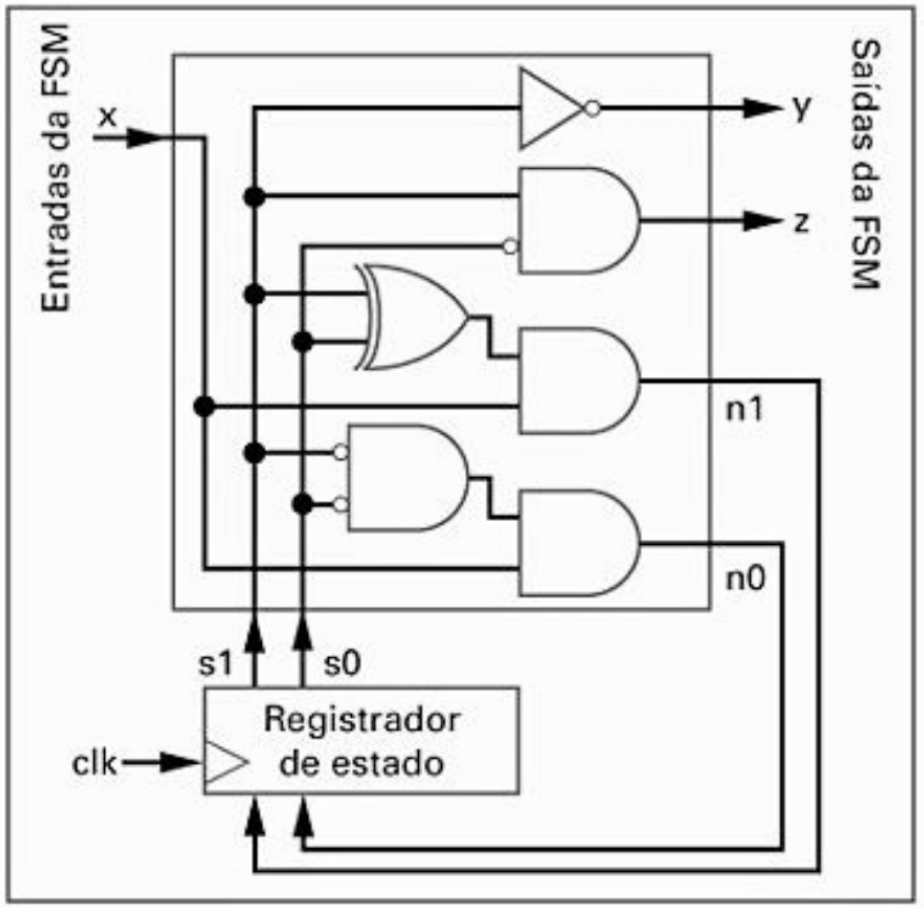


Análise – Exemplo

- Codificar os estados, usando nomes simbólicos para os estados enumerados

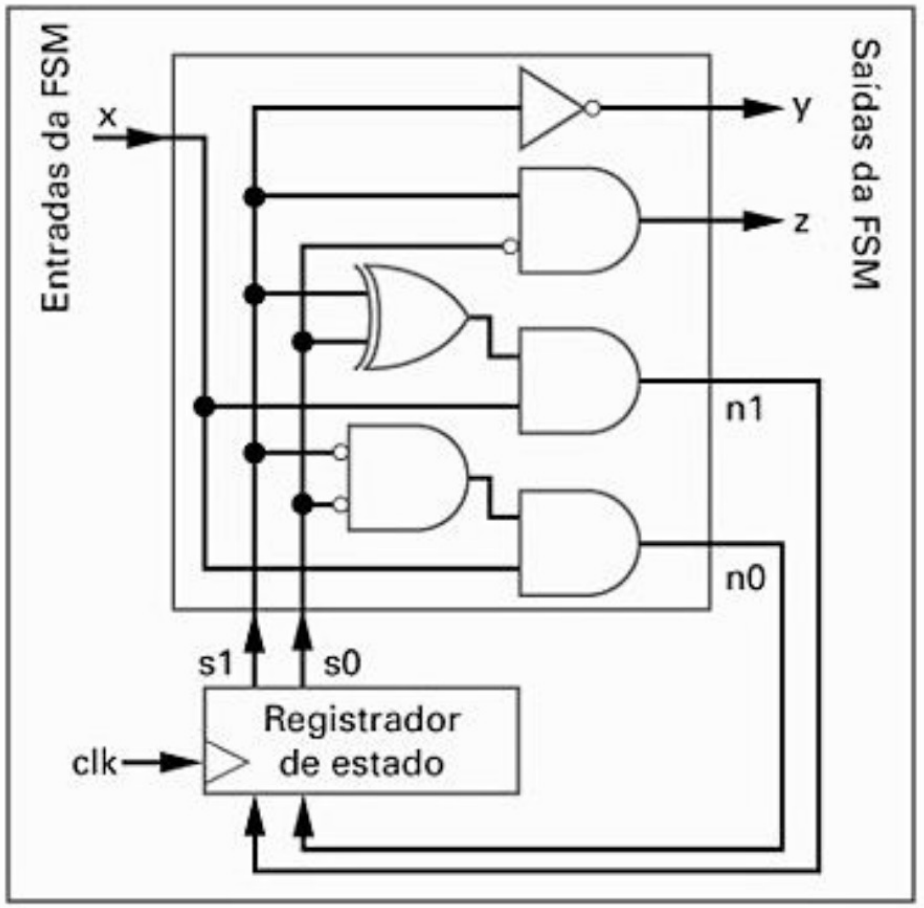
A = 00; B = 01; C = 10; D = 11

ESTADO ATUAL	PRÓXIMO ESTADO		SAÍDAS	
	x = 0	x = 1	y	z
A	A	B	1	0
B	A	C	1	0
C	A	C	0	1
D	A	A	0	0



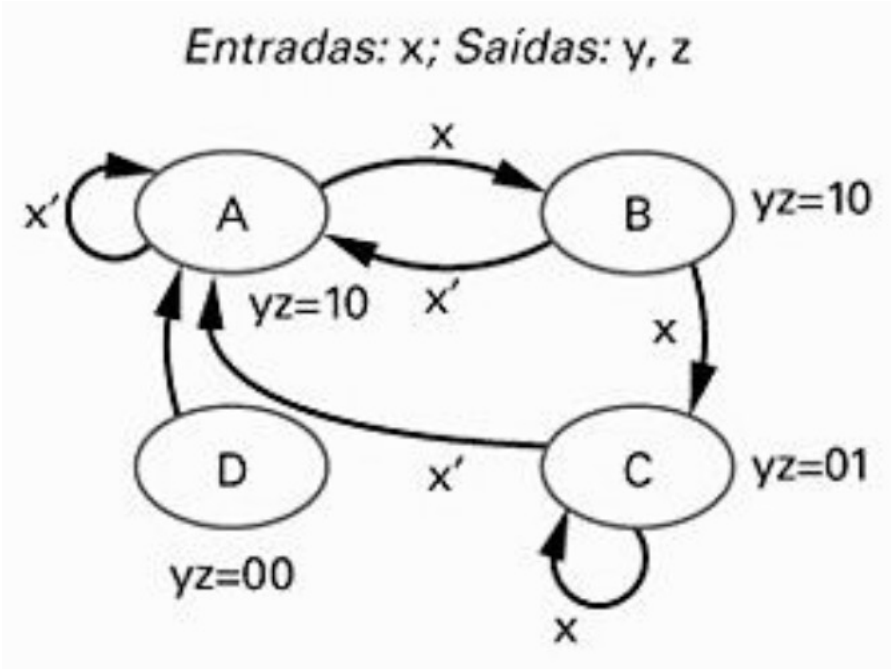
Análise – Exemplo

- **Obter arquitetura padrão**: esboço do bloco de controle em nível de sistema (diagrama de blocos), colocando nome e tamanho em bits da(s) entrada(s) no bloco de transição de estados, indicando bits de estado (atual e próximo), e nome e tamanho em bits da(s) saída(s)



Análise – Exemplo

- **Obter diagrama de estados**: a partir da tabela codificada, obter o diagrama indicando as transições de estado, as saídas em cada estado (Moore) ou em cada transição (Mealy), e o estado inicial



Cuidados Básicos na Implementação de FSMs

- Deve-se verificar que **uma e apenas uma** das condições de transição de cada estado deve ser verdadeira a qualquer borda de subida do clock
- **Duas propriedades** devem ser satisfeitas
  1. Apenas uma condição deve ser verdadeira **E**
  2. Uma das condições deve ser verdadeira

Cuidados Básicos na Implementação de FSMs

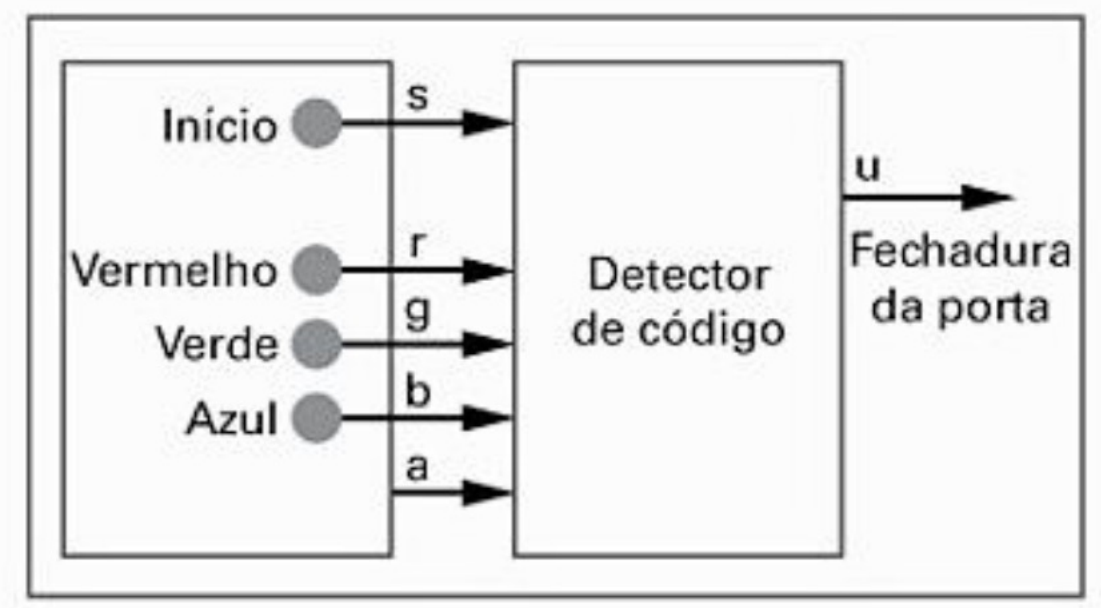
- **Apenas uma condição deve ser verdadeira**: o projetista deve assegurar que as condições de transição em cada estado são exclusivas
  - Em cada estado, verificar se o resultado da AND de cada par de condições de transição resulta ‘0’
  - Se para um par de condições não se verifica a propriedade, então é não determinística, ou seja, a FSM está mal projetada

Cuidados Básicos na Implementação de FSMs

- **Uma das condições deve ser verdadeira:** o projetista deve garantir que, em cada estado, pelo menos uma condição é verdadeira. Todas as combinações possíveis de transição devem ser consideradas
  - Verificar em cada estado se a OR de todas as condições de transição resulta em ‘1’
  - Se a propriedade anterior não é satisfeita, então é não determinística, ou seja, a FSM está mal projetada

Exemplo

- Verificar as propriedades de transição de estados para um detector de sequência de cores que identifica a sequência *red*, *blue*, *green* e *red* (**r**, **b**, **g**, **r**). Para iniciar a sequência, pressiona-se *start* (**s**). A saída **a = 1** quando qualquer botão de cor é pressionado, e a saída **u = 1** quando a sequência é detectada.

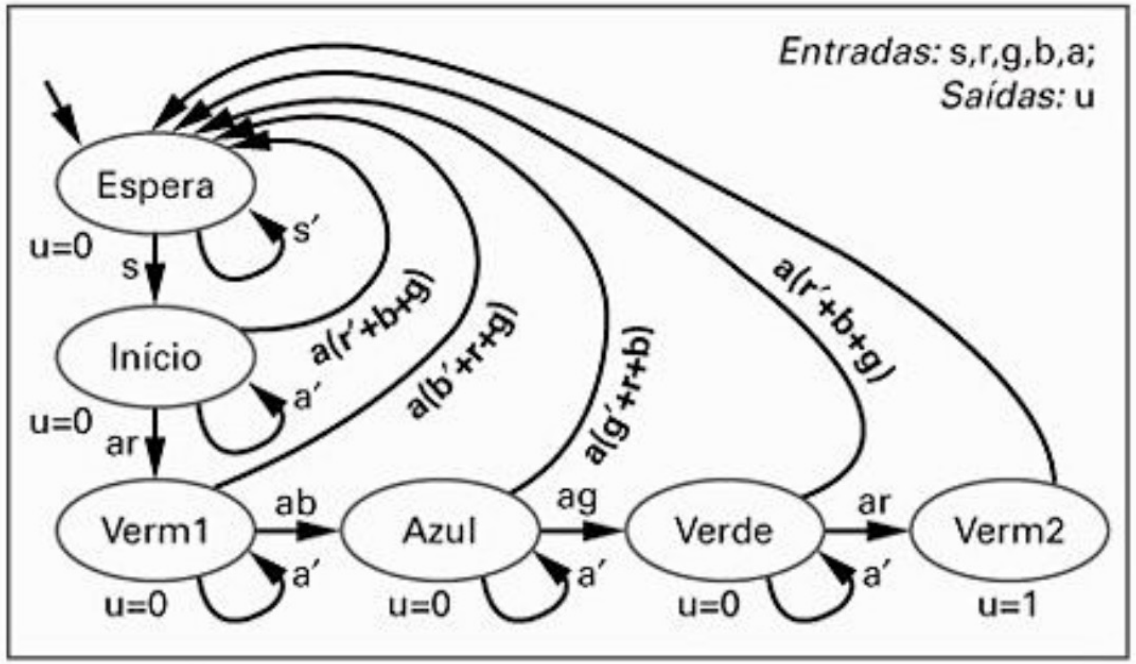


Exemplo

- Apenas uma condição deve ser verdadeira (operador **AND**). No estado *Início*:

```
ar * a'          a' * a(r'+b+g)        ar * a(r'+b+g)
= (a*a')r        = (a'*a)*(r'+b+g)     = (a*a)*r*(r'+b+g)
= 0*r            = 0*(r'+b+g)          = a*r*(r'+b+g)
= 0              = 0                   = arr'+arb+arg
                                       = 0 + arb+arg
                                       = arb + arg
                                       = ar(b+g)
```

- Correção: para a transição entre *Início* e *Verm1*, a condição correta é **arb'g'** (um botão foi pressionado, é o **r**, e os botões **b** e **g** não foram pressionados)



Exemplo

- Substituindo a condição **ar** por **arb'g'**:

```
arb'g' * a'          a'*a(rb'g')'        arb'g' * a(rb'g')'
= aa'*rb'g'          = 0*(rb'g')'        = a*a*(rb'g')*(rb'g')'
= 0*rb'g'            = 0                 escreva rb'g' como Y para ficar mais claro...
= 0                                      = a*a*Y*Y'
                                         = a*a*0
                                         = 0
```

- Alterar nos demais estados e verificar os pares de condições nas transições

Exemplo

- Uma condição deve ser verdadeira. Aplicando-se a operação OR nas três condições do estado *Start*:

```
arb'g' + a' + a(rb'g')'
= a' + arb'g' + a(rb'g')' (escreva rb'g' como Y para ficar mais claro)
= a' + aY + aY'
= a' + a(Y+Y') = a' + a(1)
= a' + a
= 1
```

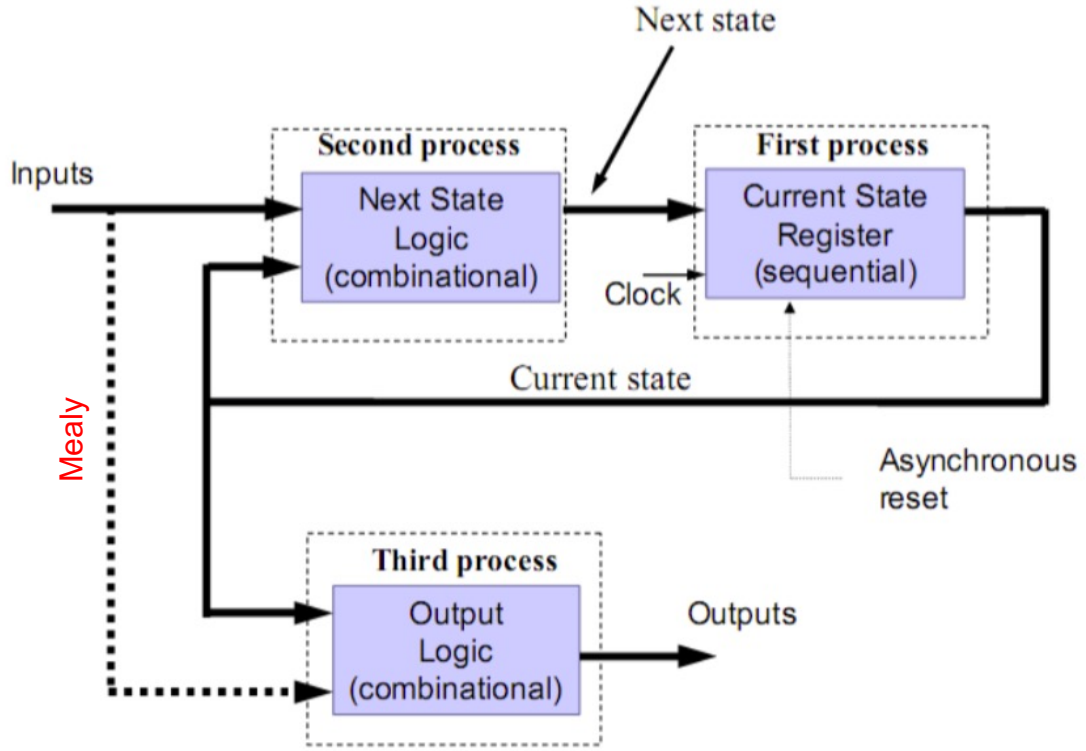
- Aplicar a operação nos outros estados

Página em branco.

Aula 16
Implementação de Máquinas de Estado de Moore em HDL

Modelagem

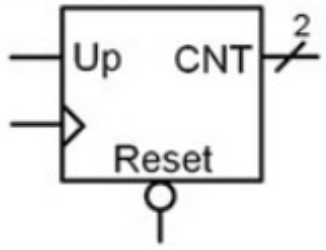
- As FSMs possuem **registradores** para armazenar o **estado atual** e uma **combinação lógica** para funções de **próximo estado** e **saída**
- Isso pode ser modelado em **3 processos**



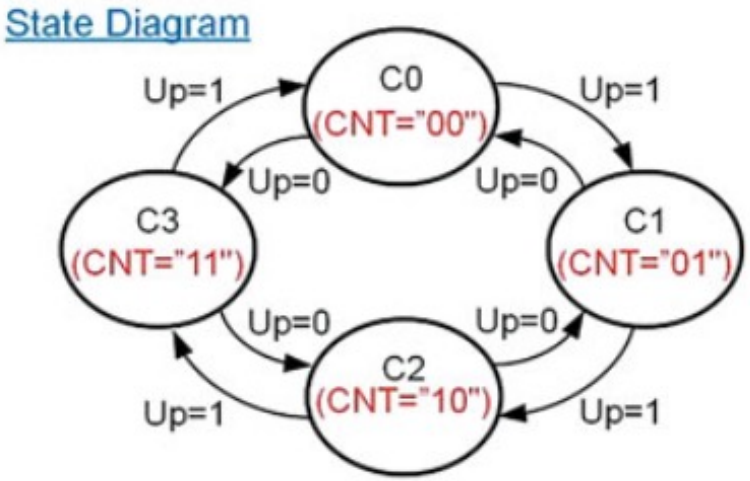
Exemplo (Verilog)

- **Contador binário *up/down* de 2 bits**
- **Entradas**: Up (habilita contagem crescente quando igual a 1, e decrescente, quando igual a 0), Clock e Reset (ativo em nível baixo)
- **CNT**: saída do contador de 2 bits

Counter_2bit_UpDown.v




```
module Counter_2bit_UpDown
    (output reg [1:0] CNT,
     input  wire      Clock, Reset, Up);
                :
                :
```



Exemplo (Verilog)

- **Modelagem dos estados**
  - Criar dois sinais que serão usados como variáveis de estado: ***current_state*** e ***next_state***
  - Os sinais devem ser declarados como **reg**. O tamanho do vetor depende do número de estados e da codificação dos mesmos
  - Declarar parâmetros para os nomes dos estados e a codificação dos mesmos

```
reg [1:0] current_state, next_state;
parameter C0 = 2'b00,
          C1 = 2'b01,
          C2 = 2'b10,
          C3 = 2'b11;
```

Exemplo (Verilog)

- **Processo 1**: estado atual
  - Modela o comportamento dos flip-flops, que armazenam o estado atual em suas saídas Q
  - O estado atual é atualizado com os valores presentes nas entradas D dos flip-flops a cada borda de subida do **Clock**
  - Deve incluir a condição para o **Reset**, indicando o estado inicial
  - A lista de sensibilidade contém apenas os sinais de **Clock** e **Reset**

```
always @ (posedge Clock or negedge Reset)
  begin: STATE_MEMORY
    if (!Reset)
      current_state <= C0;
    else
      current_state <= next_state;
  end
```

Exemplo (Verilog)

- **Processo 2**: lógica combinacional de próximo estado
  - Incluir todos os sinais de entrada e o sinal ***current_state*** na lista de sensibilidade
  - Usar uma combinação de case e if-else para modelar as atribuições para diferentes condições nas entradas
  - Incluir uma cláusula **default** para assegurar que a FSM voltará para o estado inicial em caso de falhas inesperadas

```
always @ (current_state or Up)
  begin: NEXT_STATE_LOGIC
    case (current_state)
      C0      : if (Up == 1'b1) next_state = C1; else next_state = C3;
      C1      : if (Up == 1'b1) next_state = C2; else next_state = C0;
      C2      : if (Up == 1'b1) next_state = C3; else next_state = C1;
      C3      : if (Up == 1'b1) next_state = C0; else next_state = C2;
      default : next_state = C0;
    endcase
  end
```

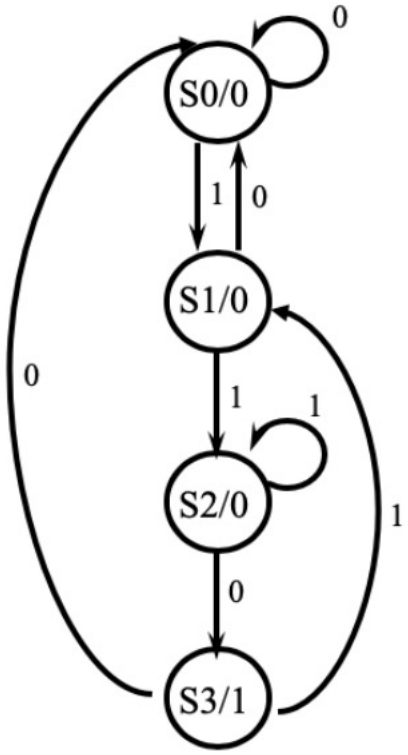
Exemplo (Verilog)

- **Processo 3**: lógica combinacional de saída
  - Incluir somente o sinal ***current_state*** na lista de sensibilidade. Como se trata de uma máquina de Moore, as saídas dependem apenas do estado atual
  - Incluir uma cláusula **default** para assegurar uma saída conhecida em caso de falhas inesperadas

```
always @ (current_state)
  begin: OUTPUT_LOGIC
   case (current_state)
     C0      : CNT = 2'b00;
     C1      : CNT = 2'b01;
     C2      : CNT = 2'b10;
     C3      : CNT = 2'b11;
     default : CNT = 2'b00;
   endcase
end
```

Exemplo (VHDL)

- **Detector de Sequência**: implementar uma máquina de Moore que detecte (saída **Z = 1**) a sequência "**110**" na entrada **A**

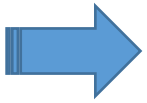



```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity detector_sequencia_110 is
    Port ( clk : in  STD_LOGIC;
           reset : in  STD_LOGIC;
           A : in  STD_LOGIC;
           saida : out  STD_LOGIC);
end detector_sequencia_110;

architecture Behavioral of detector_sequencia_110 is

type state is (e0,e1,e2,e3);
signal current_state, next_state : state := e0;

begin
```

Exemplo (VHDL)

- **Processo 1**: estado atual

```
state_reg: process(clk,reset)
begin
    if rising_edge(clk) then
        if reset='1' then
            current_state <= initial_state;
        else
            current_state <= next_state;
        end if;
    end if
end process state_reg;
```



```
armazena_estado: process(clk,reset)
begin
    if rising_edge(clk) then
        if reset='1' then
            current_state <= e0;
        else
            current_state <= next_state;
        end if;
    end if;
end process;
```

Exemplo (VHDL)

- **Processo 2**: lógica combinacional do próximo estado

```
next_state_logic: process(current_state,input1,input2,...)
begin
    case current_state is
        when state1 =>
            if condition1 then
                next_state <= state_value;
            elsif condition2 then
                next_state <= state_value;
            ...
            else
                next_state <= state_value;
            end if;
        when state2=>
            ...
    end case;
end process next_state_logic;
```

Exemplo (VHDL)

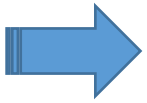
- **Processo 3**: lógica combinacional de saída

```
output_logic: process(current_state,input1,input2,...)
begin
    case current_state is
        when state1 =>
            moore_output1 <= value;
            moore_output2 <= value;
            ...
            if condition1 then
                mealy_output1 <= value;
                mealy_output2 <= value;
                ...
            elsif condition2 then
                mealy_output1 <= value;
                ...
            else
                mealy_output1 <= value;
                ...
            end if;
        when state2=>
            ...
    end case;
end process output_logic;
```

Exemplo (VHDL)

- Com frequência, os processos 2 e 3 podem ser combinados em um único processo

```
fsm_logic: process(current_state, input1, input2,...)
begin
   case current_state is
      when state1 =>
         if condition1 then
            next_state <= state_value;
            mealy_output1 <= value;
            mealy_output2 <= value;
            ...
         elsif condition2 then
            next_state <= state_value;
            mealy_output1 <= value;
            ...
         ...
         else
            next_state <= state_value;
            mealy_output1 <= value;
            ...
         end if;
         moore_output1 <= value;
         ...
      when state2 =>
         ...
   end case;
end process fsm_logic;
```



```
transicao_estado: process(current_state,A)
begin
    case current_state is
        when e0 =>
            saida <= '0';
            if A='0' then next_state <= e0;
            else next_state <= e1;
            end if;
        when e1 =>
            saida <= '0';
            if A='0' then next_state <= e0;
            else next_state <= e2;
            end if;
        when e2 =>
            saida <= '0';
            if A='0' then next_state <= e3;
            else next_state <= e2;
            end if;
        when e3 =>
            saida <= '1';
            if A='0' then next_state <= e0;
            else next_state <= e1;
            end if;
    end case;
end process;
```

Exercício

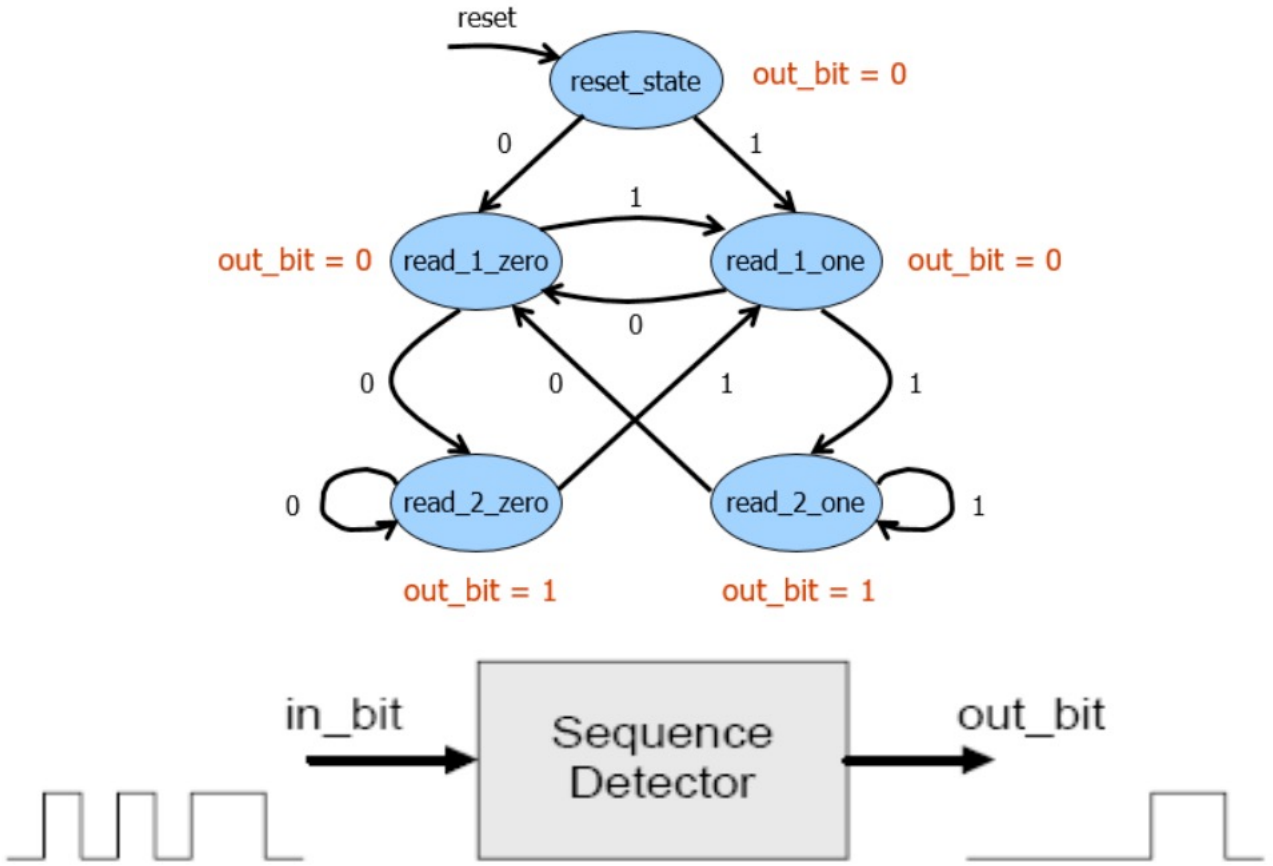
- Implemente em Verilog o detector de sequência do exemplo anterior. Elabore um testbench e faça a simulação comportamental para validar o funcionamento. Por fim, implemente fisicamente na placa de desenvolvimento.

Página em branco.

Aula 17
Implementação de Máquinas de Estado de Moore em HDL

Exercício 1 (Verilog)

- Implementar um detector de sequência de dois bits consecutivos iguais a 0 ou 1 em um sinal serial



Exercício 2 (Verilog)

- Projetar e implementar uma máquina de Moore que controla um semáforo em um cruzamento entre uma rodovia movimentada e uma estrada secundária pouco utilizada.
- Em condições normais, a rodovia permanece com o semáforo verde. A estrada secundária possui um detector de veículos que indica a chegada de um carro ativando um sinal chamado **CAR**. Quando **CAR** for ativado, o semáforo da rodovia mudará de verde para amarelo e, em seguida, de amarelo para vermelho. Uma vez na posição vermelha, um temporizador interno iniciará uma contagem regressiva e fornecerá ao controlador um sinal chamado **TIMEOUT** após 15 segundos. Quando **TIMEOUT** for ativado, o semáforo da rodovia voltará ao verde.
- Portanto, o sistema possui três saídas: **GRN, YLW** e **RED**, que controlam o estado dos semáforos voltados para a rodovia (1 = ligado, 0 = desligado). Esboce o diagrama de estados dessa máquina e escolha qualquer método de codificação de estados que desejar.

Página em branco.