



Universidade Federal de Santa Catarina
Centro Tecnológico – CTC
Departamento de Engenharia Elétrica



“Circuitos e Técnicas Digitais”

Baseado nos Slides do Prof. Eduardo Bezerra e do Prof. Hector

Florianópolis, agosto de 2015.

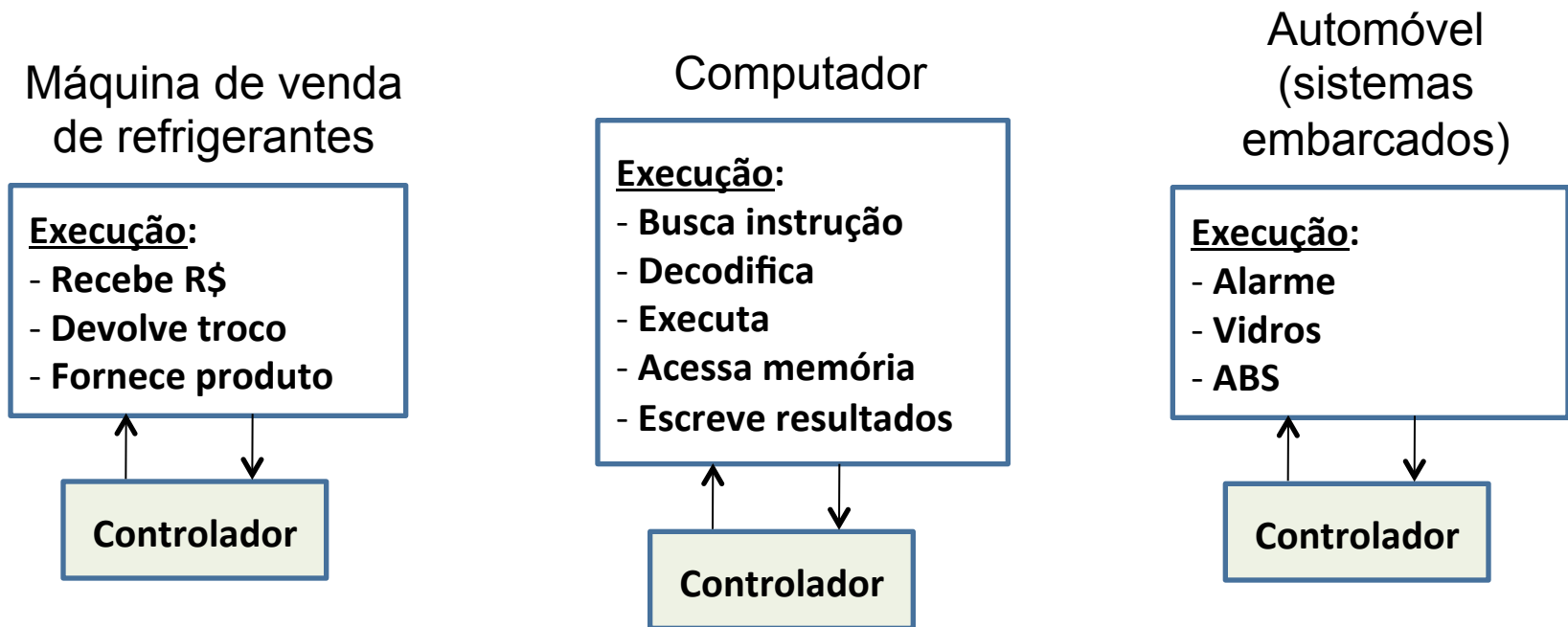
Objetivos do laboratório

1. Entender o conceito de máquinas de estados (FSM).
2. Entender o conceito de circuito sequencial controlando o fluxo de atividades de circuito combinacional.
3. Entender o processo de síntese de FSMs em VHDL.
4. Entender o funcionamento de contadores.
5. Estudo de caso: projeto e implementação em VHDL de um contador baseado em máquinas de estados.

“Síntese de máquinas de estado (FSM)”

Finite State Machine (FSM)

- Sistemas computacionais, normalmente, são compostos por um módulo de “controle” e um módulo para “execução das operações”.



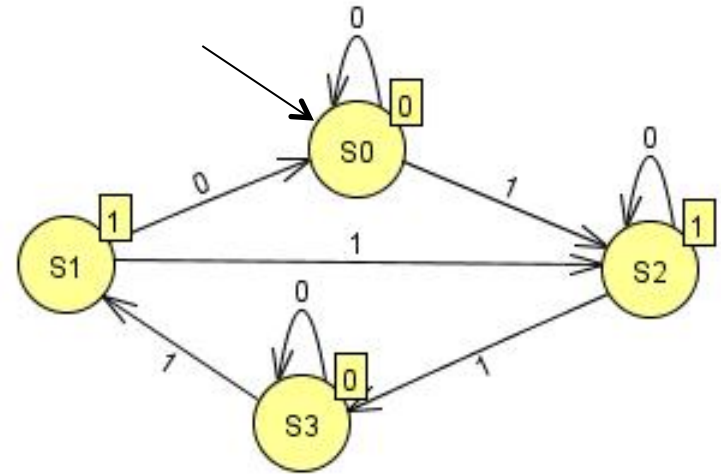
Finite State Machine (FSM)

- O “controlador” é responsável por coordenar a sequência de atividades a ser realizada em um determinado processo (ou sistema)
- Em sistemas digitais são utilizados “circuitos sequenciais” na geração de sinais de controle
- Um circuito sequencial transita por uma série de estados e, a cada estado (a cada momento), poderá fornecer uma determinada saída
- As saídas são utilizadas no controle da execução de atividades em um processo
- A lógica sequencial utilizada na implementação de uma FSM possui um número “finito” de estados.

Finite State Machine (FSM)

Modelo de comportamento composto por:

- Estados
- Transições
- Ações



Estado

Armazena informação sobre o passado refletindo as modificações das entradas do início até o presente momento

Transição

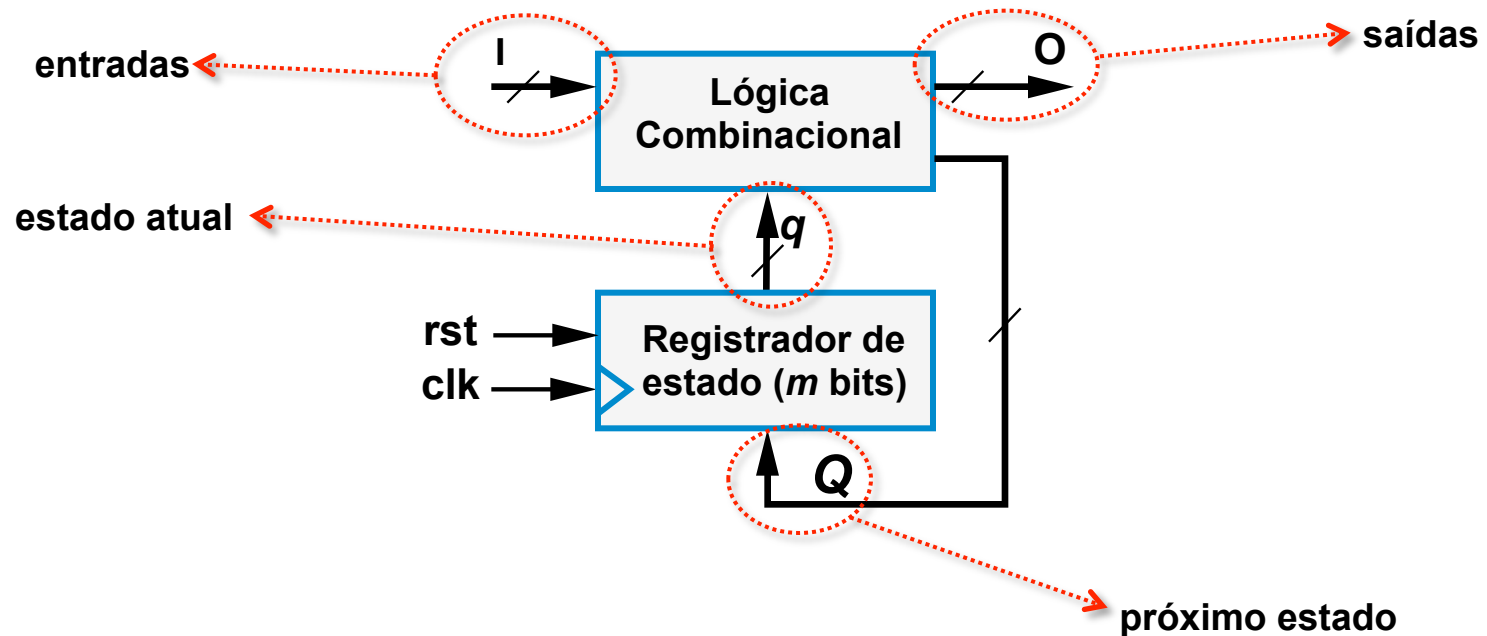
Indica uma troca de estado e é descrita por uma condição que habilita a modificação de estado

Ação

Descrição da atividade que deve ser executada em um determinado instante

Estrutura de uma FSM

- Dois módulos:
 - Armazenamento do “estado atual”; e
 - Cálculo da “saída” e do “próximo estado”



Estrutura de uma FSM

- Armazenamento do “estado atual”
 - Registrador construído a partir de flip-flops
- Cálculo da “saída” e do “próximo estado”
 - Circuito combinacional; ou
 - Tabela verdade da lógica de saída e da lógica de próximo estado armazenada em uma memória (ROM, Flash, RAM, ...)

Tarefa a ser realizada na aula prática

Solução I – FSM estrutural

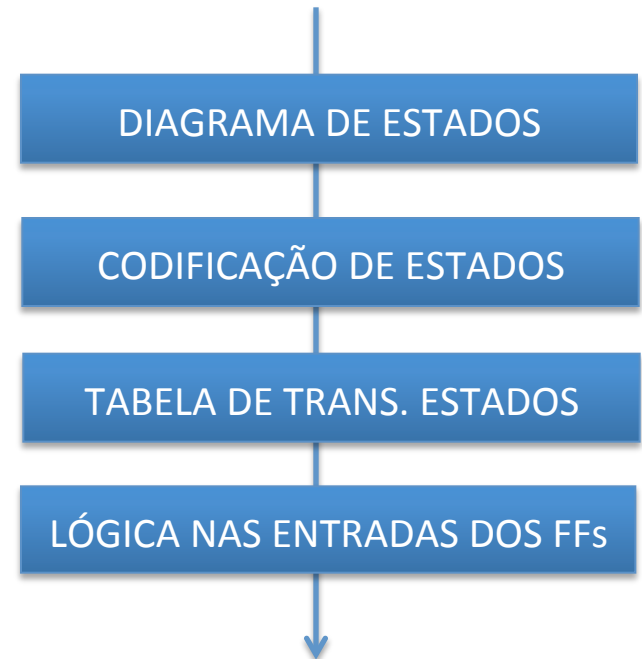
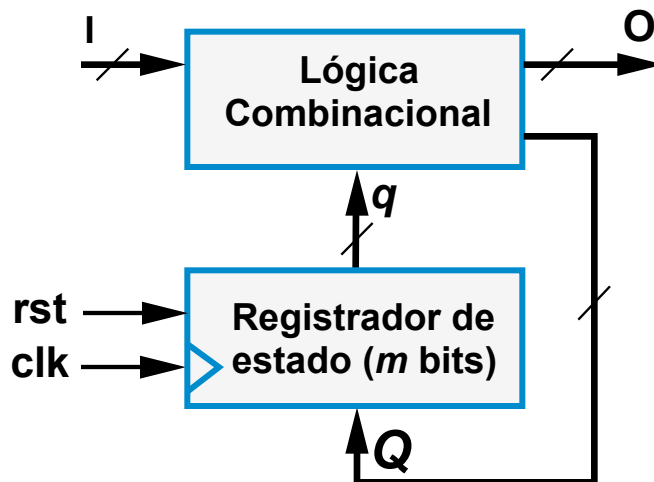
Solução II – FSM comportamental

Primeira tarefa a ser realizada: Solução I

- Implementar uma FSM em VHDL a partir da metodologia teórica para geração da sequencia $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 1$ nos LEDs vermelhos (LEDR).
- FSM com *reset* assíncrono, usando o botão KEY(0) para inicializar um contador com o valor do primeiro caracter a ser gerado “1”.
- Usar KEY(1) como relógio manual, para avançar para o próximo estado, gerando assim o próximo valor da sequencia.

Solução I - Abordagem Estrutural

Determinação “manual” do circuito combinacional (lógica) para geração da saída e próximo estado.



SEQUENCIADOR: 1→ 2→ 3→4→5→1→...

Solução I: Diagrama de estados e codificação de estados

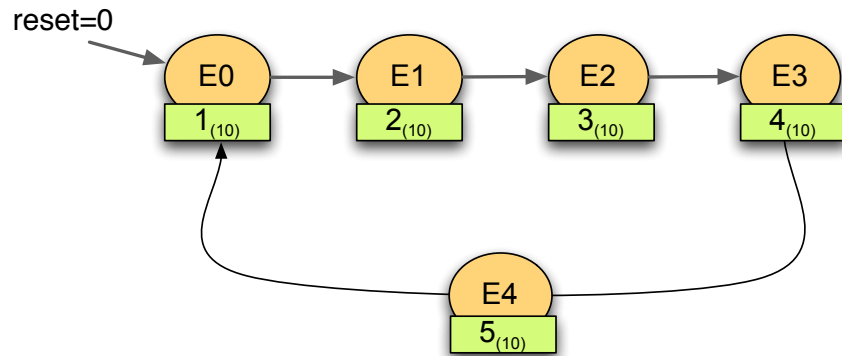


Diagrama de estados

ESTADOS	q_3	q_2	q_1	q_0
E0	0	0	0	1
E1	0	0	1	0
E2	0	0	1	1
E3	0	1	0	0
E4	0	1	0	1

Tabela de codificação de estados

Q ou Q(t+1): Proximo estado

q ou Q(t): Estado atual

Solução I: Tabela de transição de estados e lógica nas entradas dos FFs

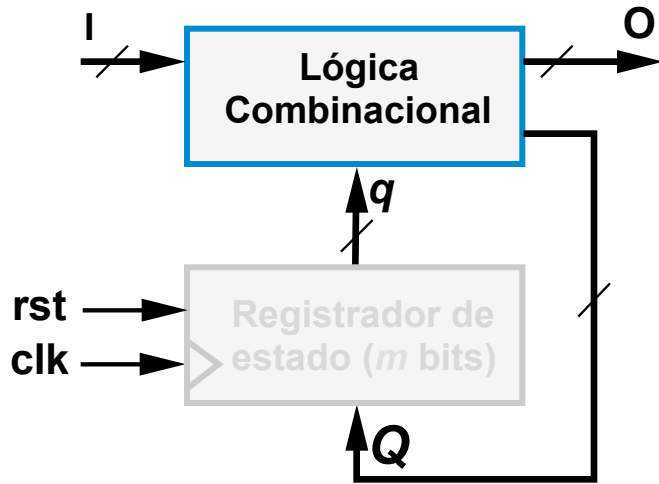


Tabela de transição de estados

Entradas				Saídas							
Estado Atual				Próximo Estado				Saída da Máq.			
q_3	q_2	q_1	q_0	Q_3	Q_2	Q_1	Q_0	O_3	O_2	O_1	O_0
0	0	0	1	0	0	1	0	0	0	0	1
0	0	1	0	0	0	1	1	0	0	1	0
0	0	1	1	0	1	0	0	0	0	1	1
0	1	0	0	0	1	0	1	0	1	0	0
0	1	0	1	0	0	0	1	0	1	0	1

Tarefa a ser realizada:

- 1.- Preencher a tabela de transição de estados
- 2.- Obter a lógica para o cálculo do próximo estado e das saídas.

$$O_3 = q_3$$

$$Q_3 = 0$$

$$O_2 = q_2$$

$$Q_2 = \overline{q_3} \overline{q_2} q_1 q_0 + \overline{q_3} q_2 \overline{q_1} \overline{q_0}$$

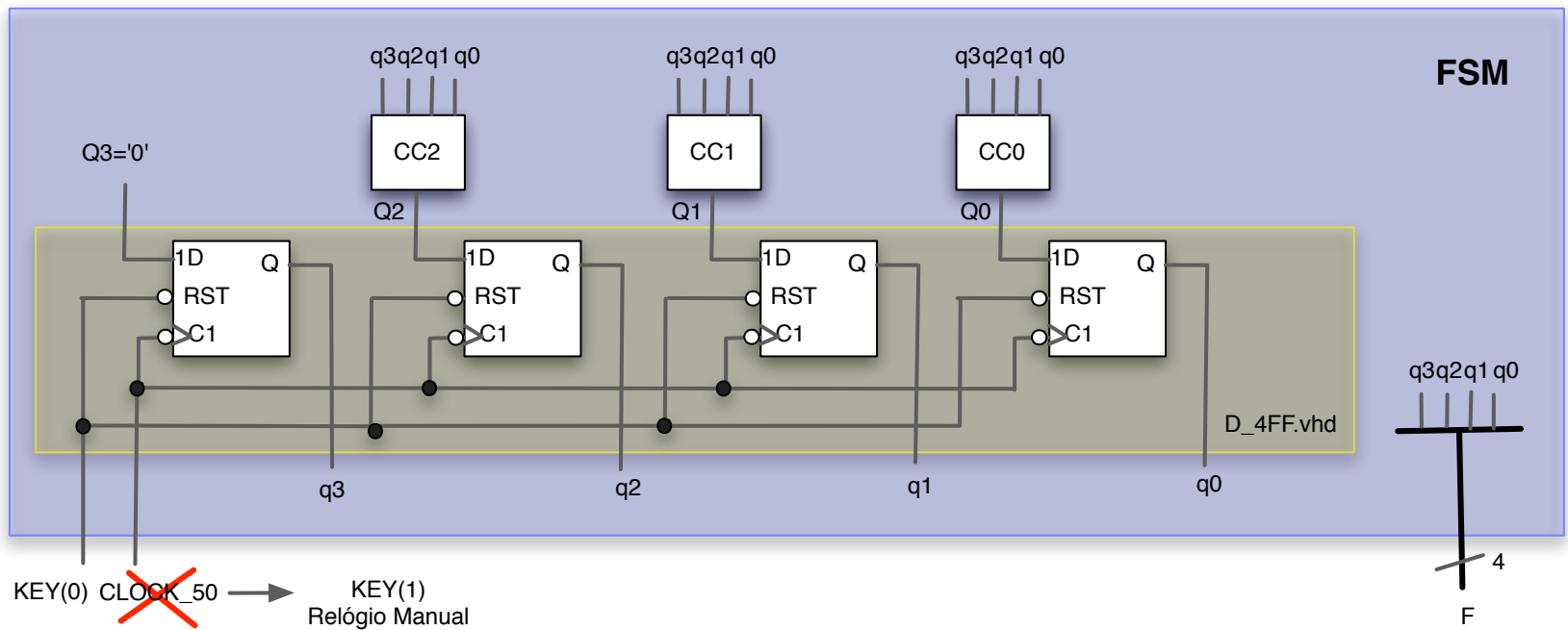
$$O_1 = q_1$$

$$Q_1 = \overline{q_3} \overline{q_2} (q_1 \oplus q_0)$$

$$O_0 = q_0$$

$$Q_0 = \overline{q_3} q_2 \overline{q_1} + \overline{q_3} \overline{q_2} q_1 \overline{q_0}$$

Solução I: Diagrama de blocos do circuito a ser implementado



Tarefa a ser realizada:

- 1.- Copiar o código VHDL do slide 16.
- 2.- Criar uma componente D_4FF.vhd com VHDL do registrador de 4 bits da aula 6 (slide 14), sensível à borda de descida.

Solução I: Codigo VHDL da hierarquia superior topo

```
library ieee;
use ieee.std_logic_1164.all;

entity topo is
port (
  KEY : IN std_logic_vector(3 downto 0);
  HEX0: out std_logic_vector(6 downto 0);
  HEX1: out std_logic_vector(6 downto 0);
  LEDR : out std_logic_vector(9 downto 0)
);
end topo;

architecture topo_estru of topo is
signal QQ, q: std_logic_vector(3 downto 0);
signal F: std_logic_vector (3 downto 0);

component D_4FF
port (
  CLK, RST: in std_logic;
  D: in std_logic_vector(3 downto 0);
  Q: out std_logic_vector(3 downto 0)
);
end component;

component decod7seg
port (
  C: in std_logic_vector(3 downto 0);
  F: out std_logic_vector(6 downto 0)
);
end component;
```

```
begin
```

```
-- Inicio da FSM --
```

```
QQ(3) <= '0';
QQ(2) <= (not(q(3)) and not(q(2)) and q(1) and q(0)) or (not(q(3)) and
q(2) and not(q(1)) and not(q(0)));
QQ(1) <= (not(q(3)) and not(q(2))) and (q(1) xor q(0));
QQ(0) <= (not(q(3)) and q(2) and not(q(1))) or (not(q(3)) and not(q(2))
and q(1) and not(q(0)));
```

```
L0: D_4FF port map (KEY(1), KEY(0), QQ(3 downto 0), q(3 downto 0));
F <= q;
```



```
-- Fim da FSM --
```

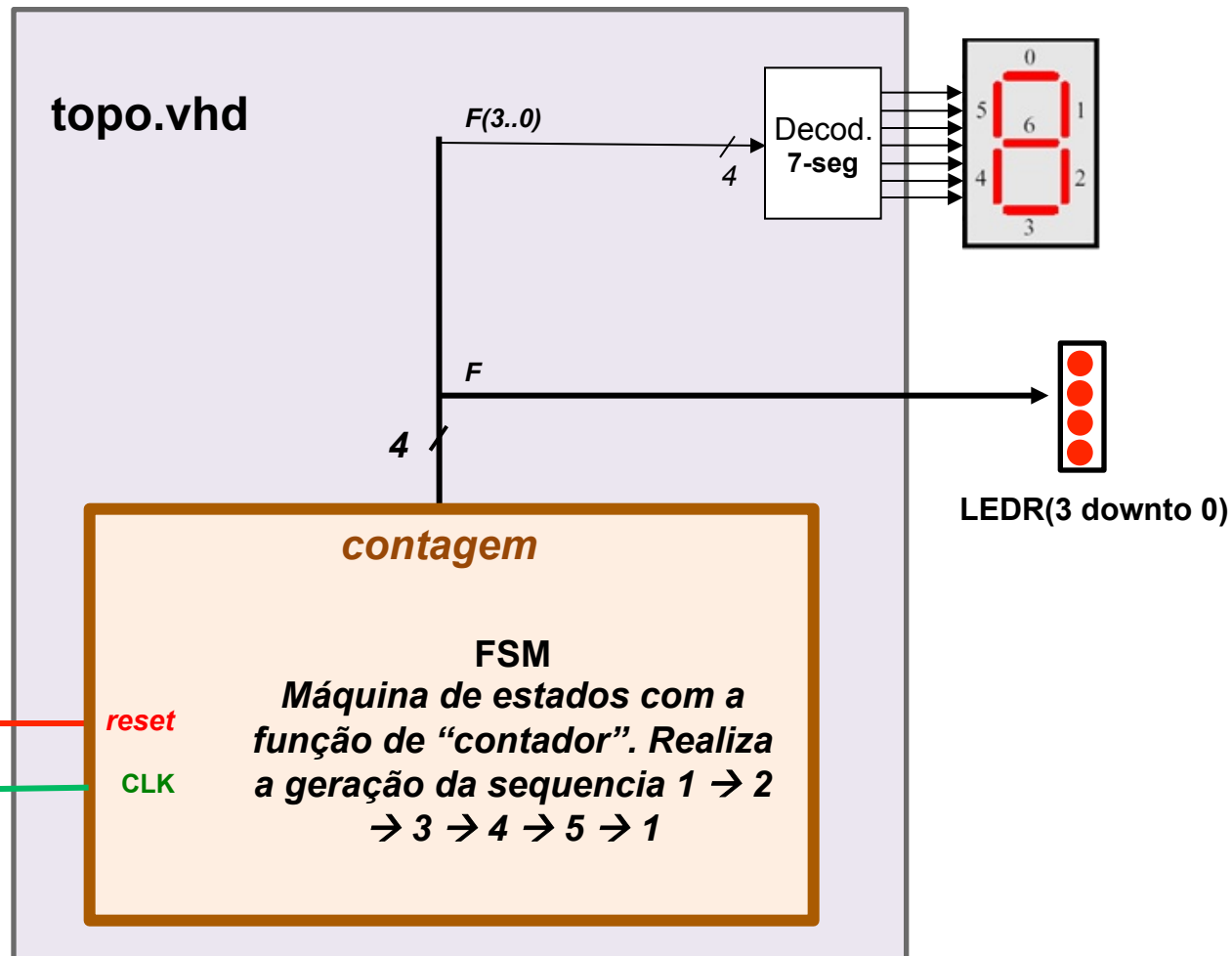
```
LEDR <= "000000" & F;
L2: decod7seg port map (F(3 downto 0), HEX0);
end topo_estru;
```

Solução I: Diagrama de blocos do circuito a ser implementado

“Uso dos displays de 7-segmentos como saída”

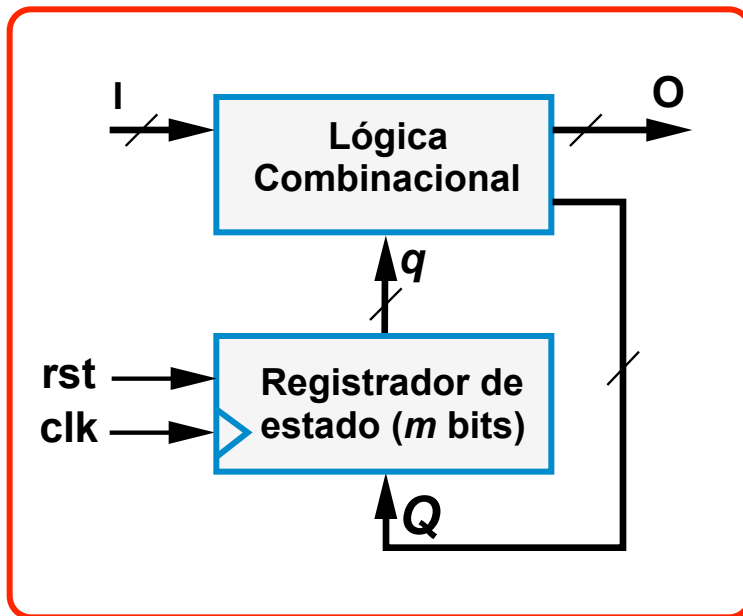
1. Incluir o componente *Decod7seg* dos labs anteriores;
2. Criar um signal *F* no topo;
3. Criar uma instâncias do decodificador (com port map);
4. Usar o novo sinal interno *F* para conectar a saída da FSM (Contagem) com as entradas dos decodificadores (usando port map);
5. Conectar o sinal *F* ao LEDR:
 $LEDR \leq F$.

KEY(0) 
KEY(1)
(Manual clock) 

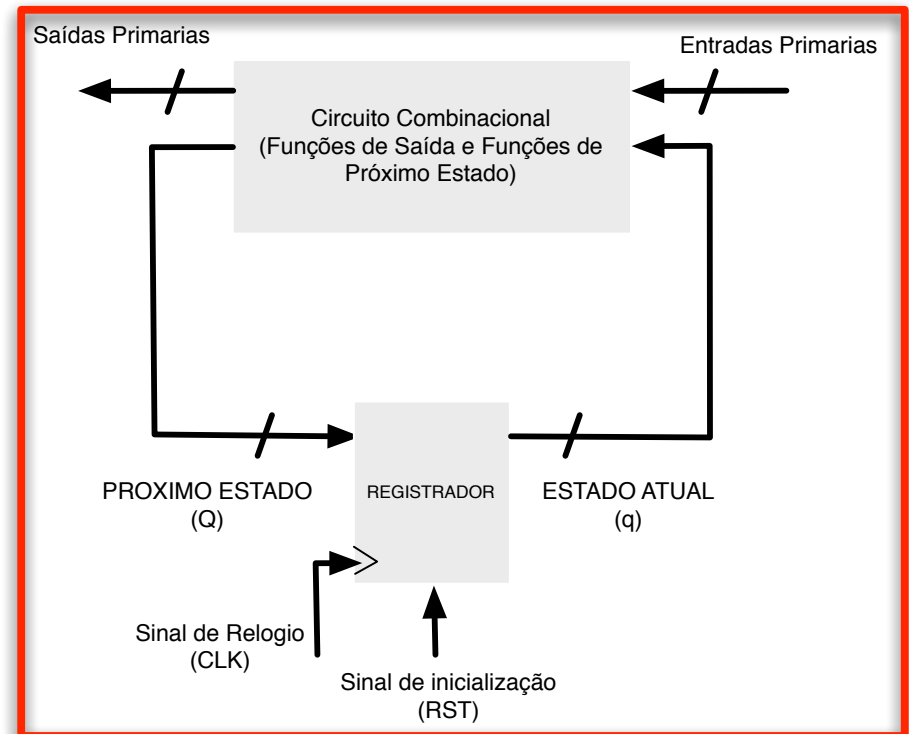


Solução II - Abordagem Comportamental

A ferramenta de síntese (do Quartus II) realiza a geração automática de toda lógica necessária.



Nomenclatura 1



Nomenclatura 2

Segunda tarefa a ser realizada: Solução II

- Implementar uma FSM em VHDL a partir da metodologia teórica para geração da sequencia $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 0$ nos LEDs vermelhos (LEDR).
- FSM com *reset* assíncrono, usando o botão KEY(0) para inicializar um contador com o valor do primeiro caractere a ser gerado “1”.
- Usar KEY(1) como relógio manual, para avançar para o próximo estado, gerando assim o próximo valor da sequencia.

Uso de HDL para descrever uma FSM

VHDL típico de uma máquina de estados – 2 processos

```
entity MOORE is  port(X, clock, reset : in std_logic;  Z: out std_logic);  end;
```

```
architecture A of MOORE is
```

```
  type STATES is (S0, S1, S2, S3);
```

```
  signal EA, PE : STATES;
```

```
begin
```

```
  process (clock, reset)
```

```
  begin
```

```
    if reset= '1' then
```

```
      EA <= S0;
```

```
    elsif clock'event and clock='1' then
```

```
      EA <= PE ;
```

```
    end if;
```

```
  end process;
```

```
  process(EA, X)
```

```
  begin
```

```
    case EA is
```

```
      when S0 =>
```

```
        Z <= '0';
```

```
        if X='0' then PE <=S0; else PE <= S2; end if;
```

```
      when S1 =>
```

```
        Z <= '1';
```

```
        if X='0' then PE <=S0; else PE <= S2; end if;
```

```
      when S2 =>
```

```
        Z <= '1';
```

```
        if X='0' then PE <=S2; else PE <= S3; end if;
```

```
      when S3 =>
```

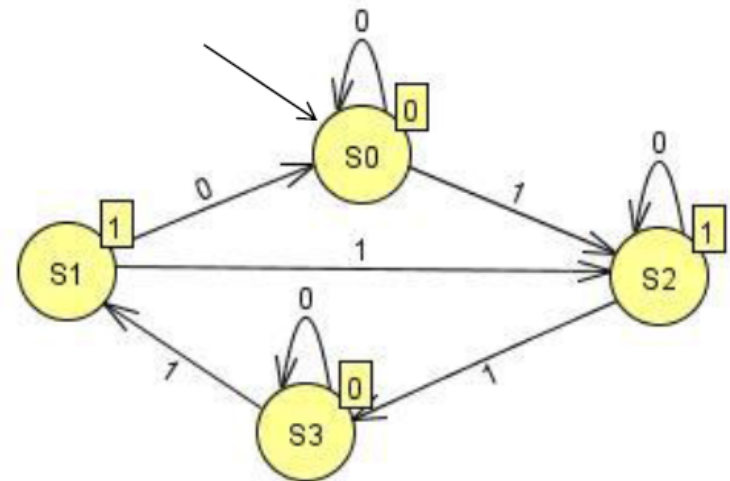
```
        Z <= '0';
```

```
        if X='0' then PE <=S3; else PE <= S1; end if;
```

```
    end case;
```

```
  end process;
```

```
end A;
```



Uso de HDL para descrever uma FSM

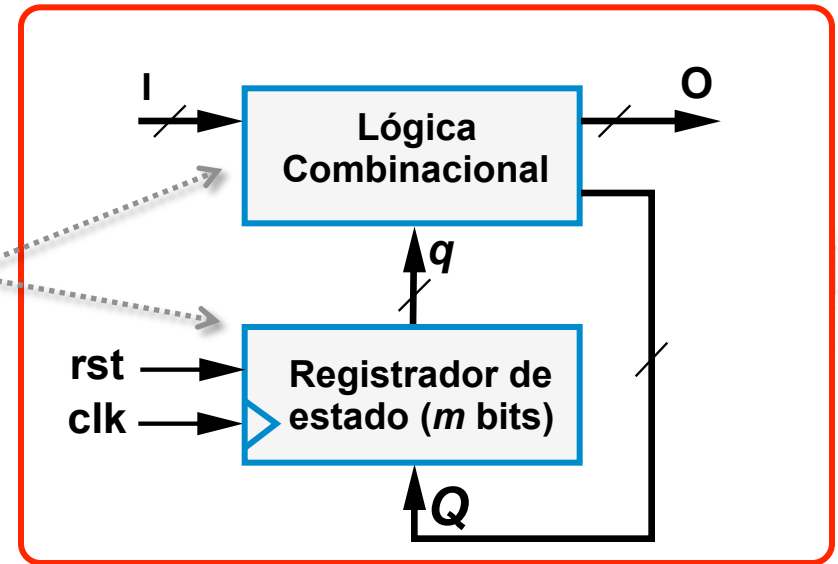
VHDL típico de uma máquina de estados – 2 processos

```
entity MOORE is  port(X, clock, reset : in std_logic;  Z: out std_logic);  end;
```

```
architecture A of MOORE is
  type STATES is (S0, S1, S2, S3);
  signal EA, PE : STATES;
```

```
begin
  process (clock, reset)
  begin
    if reset= '1' then
      EA <= S0;
    elsif clock'event and clock='1' then
      EA <= PE ;
    end if;
  end process;
```

```
  process(EA, X)
  begin
    case EA is
      when S0 =>    Z <= '0';
                    if X='0' then PE <=S0; else PE <= S2; end if;
      when S1 =>    Z <= '1';
                    if X='0' then PE <=S0; else PE <= S2; end if;
      when S2 =>    Z <= '1';
                    if X='0' then PE <=S2; else PE <= S3; end if;
      when S3 =>    Z <= '0';
                    if X='0' then PE <=S3; else PE <= S1; end if;
    end case;
  end process;
end A;
```



Solução II

Os 4 passos para projetar uma FSM comportamental

PASSO 1: Descrição funcional do sistema (requisitos);

PASSO 2: Representação gráfica da FSM (diagrama de estados);

PASSO 3: Escrever tabela de transição de estados para a FSM, listando entradas e saídas, incluindo todos os estados (atual e próximo);

PASSO 4: Descrever em VHDL o comportamento da FSM a partir do diagrama e da tabela de estados, conforme slides 23 a 24;

Em uma metodologia clássica de projeto de FSM em sistemas digitais, mais passos são necessários, como apresentado na solução 1 (Slide 11).

Solução II

Os 4 passos para projetar uma FSM comportamental

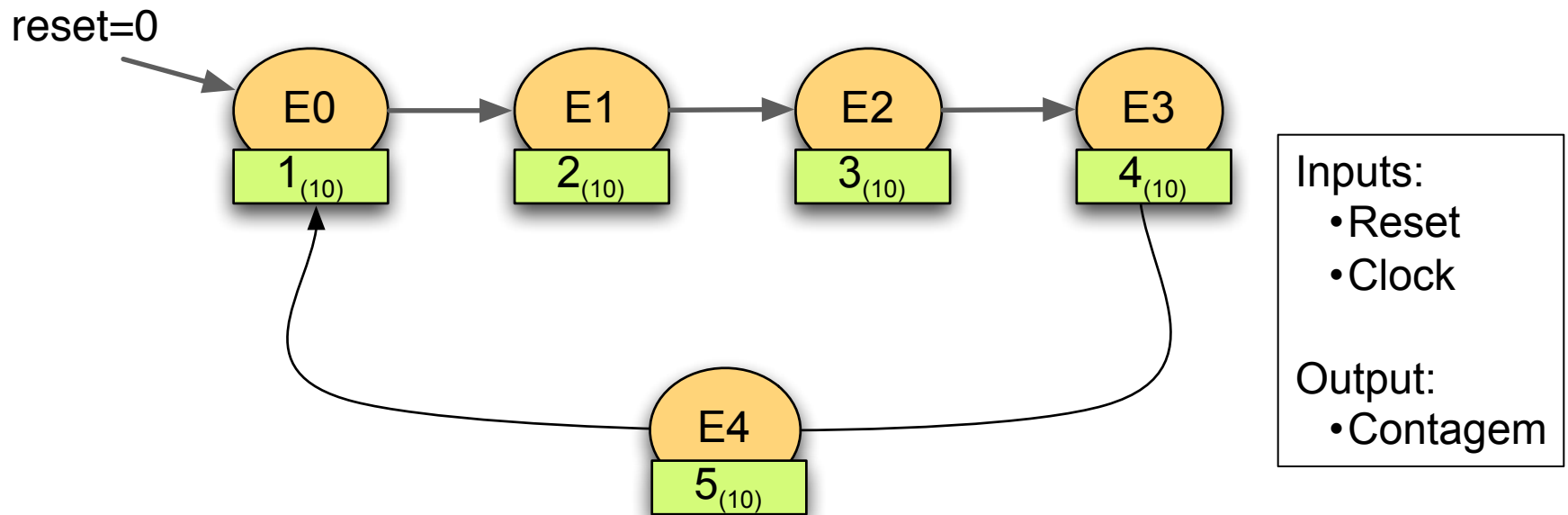
PASSO 1: Descrição funcional do sistema

- Em estado inicial a FSM fornecerá a saída “0001” (ou 1 em decimal)
- A transição de 1 para 0 no clock (KEY(1) pressionada), causará o incremento do valor de saída
- O último estado da FSM fornecerá a saída com o caractere “5” em decimal. Após o último estado, a FSM deve retornar ao primeiro estado (apresentar saída “1”)
- A qualquer instante, quando o botão de RESET for pressionado, a FSM também deverá retornar para o estado inicial, apresentando a saída “1”.

Solução II

Os 4 passos para projetar uma FSM comportamental

Passo 2: Representação gráfica da FSM.



Solução II

Os 4 passos para projetar uma FSM comportamental

PASSO 3: Escrever tabela de transição de estados para a FSM, listando entradas e saídas, incluindo todos os estados (atual e próximo).

ENTRADAS			SAÍDAS	
Reset	Clock	EA	PE	Contagem
0	X	X	E0	1
1	0	E0	E1	1
1	1	E0	E0	1
1	0	E1	E2	2
1	1	E1	E1	2
1	0	E2		
1				
1				
1				
1		E4		
1		E4		

EA – Estado Atual

PE – Próximo Estado

Solução II

Os 4 passos para projetar uma FSM comportamental

P4: Descrever em VHDL o comportamento da FSM a partir do diagrama e da tabela de estados, conforme slides 23 a 24.

```
library ieee;
use ieee.std_logic_1164.all;
entity FSM_Conta is
    contagem: out std_logic_vector(3 downto 0);
    clock: in std_logic;
    reset: in std_logic;
end;
architecture bhv of FSM_Conta is
    type STATES is (E0, !!!!!completar com estados );
    signal EA, PE: STATES;
begin
    P1: process(clock, reset)
    begin
        if reset= '0' then
            EA <= E0;
        elsif clock'event and clock= '0' then
            EA <= PE;
        end if;
    end process;
    P2: process(EA)
    begin
        case EA is
            when E0 =>
                contagem <= "0001";
                PE <= E1;
            when ???? =>
                contagem <= ????;
                PE <= ????;
            when ???? =>
                ???? ;
        end case;
    end process;
end bhv;
```

Entradas já estão
inseridas no registrador

Dica: **Topo.vhd** com componente FSM (sem utilizar os displays de 7-segmentos)

```
library ieee;
use ieee.std_logic_1164.all;
entity Topo is
    port (
        LEDR: out std_logic_vector(3 downto 0);
        KEY: in std_logic_vector(1 downto 0)
    );
end Topo;
architecture topo_beh of Topo is
    component FSM_Conta -- Esse e' o componente FSM
    port (
        contagem: out std_logic_vector(3 downto 0);
        clock: in std_logic;
        reset: in std_logic
    );
Begin
    L0: FSM_Conta port map ( LEDR, KEY(1), KEY(0) );
end topo_beh;
-- O aluno deve incluir o display no topo.vhd
```

Diagrama de blocos do circuito a ser implementado

“Uso dos displays de 7-segmentos como saída”

1. Incluir o componente *Decod7seg* dos labs anteriores;
2. Criar um signal *F* no topo;
3. Criar uma instância do decodificador (com port map);
4. Usar o novo sinal interno *F* para conectar a saída da FSM (Contagem) com as entradas dos decodificadores (usando port map);
5. Conectar o sinal *F* ao LEDR:
LEDR <= F.

