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

“Circuitos e Técnicas Digitais”

Prof. Eduardo Augusto Bezerra

Eduardo.Bezerra@ufsc.br

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 FSMs

Uso de VHDL para descrever uma FSM

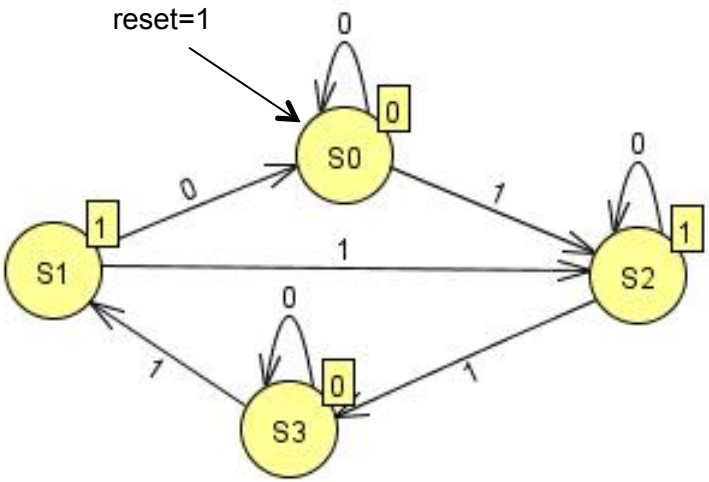
VHDL típico

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(clock, EA, X)
     begin
        if clock'event and clock='1' then
            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 if;
     end process;
end A;
```



Uso de VHDL para descrever uma FSM

VHDL típico

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(clock, EA, X)
     begin
        if clock'event and clock='1' then
            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 if;
     end process;
end A;
```

TIPO ENUMERADO
Sinais EA (estado atual) e PE (próximo estado)

Uso de VHDL para descrever uma FSM

VHDL típico

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(clock, EA, X)
     begin
        if clock'event and clock='1' then
           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 if;
     end process;
end A;
```

Registrador que armazena o EA em função do próximo estado

Uso de VHDL 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(clock, EA, X)
   begin
      if clock'event and clock='1' then
          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 if;
   end process;
end A;
```

Geração do PE e a saída Z
em função do EA e da entrada X
(observar a lista de sensitividade)

Uso de VHDL para descrever uma FSM

VHDL típico

VHDL típico de uma máquina de estados – 2 processos

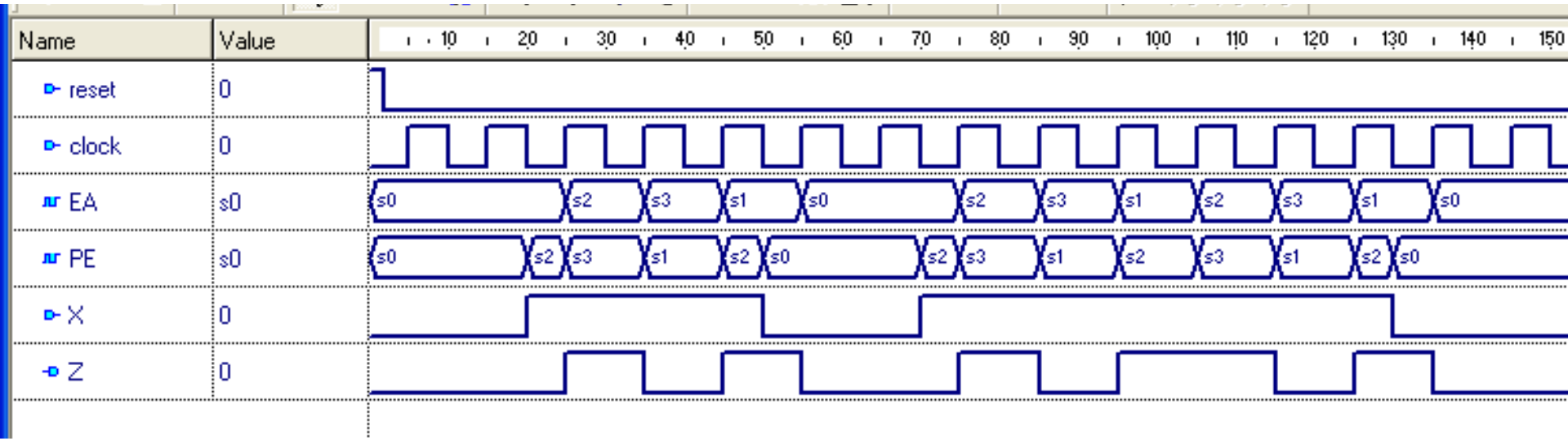
Desenhe a máquina de estados conforme as transições especificadas:

```
process(clock, EA, X)
begin
    if clock'event and clock='1' then
     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 if;
end process;
```

Esta é uma máquina Moore. A saída (Z) depende apenas do estado atual (S0, ...). Em uma máquina de Mealy, a saída depende do estado E das entradas.

Uso de VHDL para descrever uma FSM

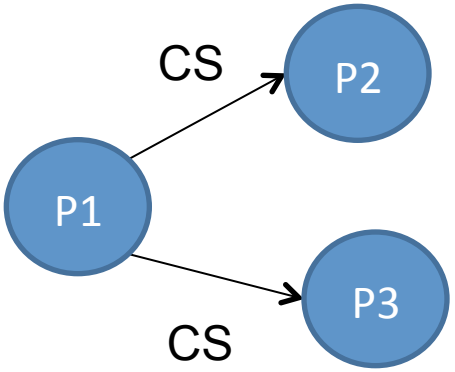
VHDL típico



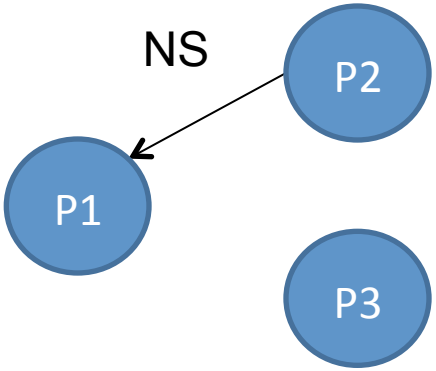

```
process(clock, EA, X)
begin
      if clock'event and clock='1' then
        case EA is
           when S0 => Z <= '0';
                      if X='0' then PE <=S0; else PE <= S2; end if;
           when S1 => Z <= '1';
                      if X='0' then PE <=S0; else PE <= S2; end if;
         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

Uso de VHDL para descrever uma FSM

Máquina de estados – 3 processos



- P1 define o estado atual, atualizando essa informação (CS) para P2 e P3.



- Com base nos valores dos sinais, P2 define o próximo estado, colocando essa informação no sinal NS sem, contudo, realizar a transição (será realizada por P1).
- Com base nos valores dos sinais (status) da FSM, P3 define novos valores para os sinais (do estado atual).

Uso de VHDL para descrever uma FSM

Máquina de estados – 3 processos

P1 - Processo, sensível as transições do clock, que realiza a transição de estados na FSM, fazendo com que o estado atual (CS, Current State) receba o próximo estado (NS, Next State). Essa transição é sensível a borda de descida do clock.

```
P1: process(clk)
begin
   if clk'event and clk = '0' then
      if rst = '0' then
         CS <= S0;
      else
         CS <= NS;
      end if;
   end if;
end process;
```

Uso de VHDL para descrever uma FSM

Máquina de estados – 3 processos

P2 – Realiza as alterações nos estados (define o próximo estado). Sensível a alterações nos sinais definidos na lista de sensitividade. Controla os estados definindo o fluxo, ou seja, define qual será o valor do sinal NS a ser utilizado pelo processo P1 responsável por realizar as transições de estados. Comando "case CS is“ seleciona o estado atual (Current State) e, conforme os sinais da FSM, um próximo estado é definido no sinal NS.

```
process( CS, X )
begin
   case CS is
      when S0 =>
         NS <= S1;
      when S1 =>
         if X = '1' then
            NS <= S2;
         else
            NS <= S1;
         end if;
      when S2 =>
         NS <= S1;
      when others =>
   end case;
end process;
```

Uso de VHDL para descrever uma FSM

Máquina de estados – 3 processos

P3 – Realiza atribuições dos sinais em cada estado. Sinais são alterados na borda de subida, e os estados na borda de descida. São atribuídos todos os sinais, incluindo os sinais de saída e sinais internos do processo.

```
process(clk)
begin
   if clk'event and clk = '1' then
      case CS is
         when S0 =>
            Z <= '0'
         when S1 =>
            Z <= '0'
         when S2 =>
            Z <= '1';
         when others =>
      end case;
   end if;
end process;
```

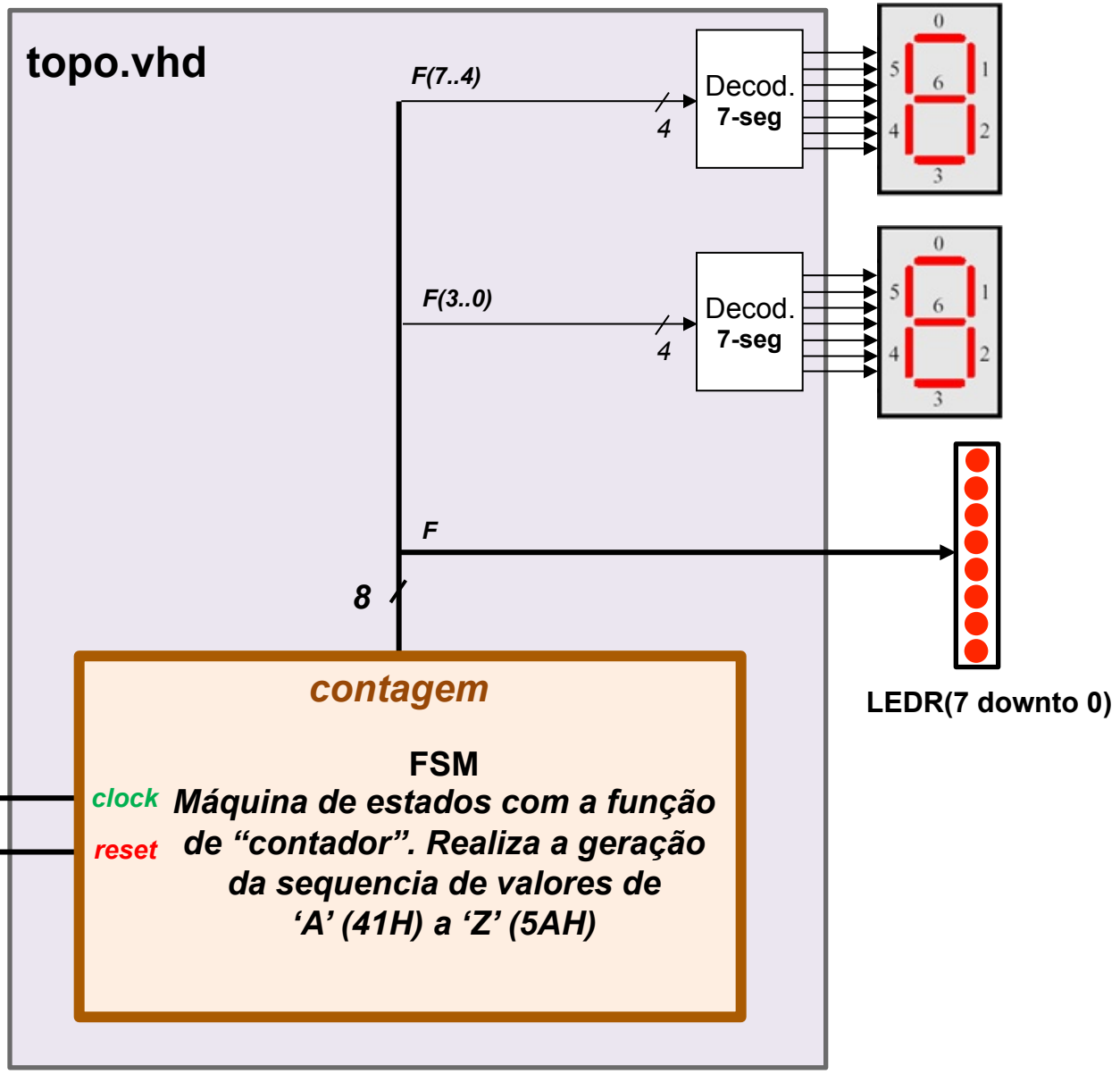
Tarefa a ser realizada na aula prática

Tarefa

- **Implementar uma FSM em VHDL com 2 processos para geração dos caracteres ‘A’ a ‘Z’ da tabela ASCII (http://asciitable.com), apresentando os caracteres (em binário) nos LEDs vermelhos (LEDR).**
- **FSM com *reset* assíncrono (usar botão KEY(0)) para inicializar um contador com o valor do primeiro caractere a ser gerado (‘A’ = 41H).**
- **A cada pulso do relógio de 50 MHz (borda de subida), a FSM deverá avançar para o próximo estado, incrementando assim o contador, e o próximo caractere da tabela ASCII deverá ser apresentado.**
- **A FSM deverá possuir um número reduzido de estados, número esse suficiente para incrementar o contador, e verificar se chegou ao final da contagem (caractere ‘Z’ = 5AH).**
- **Ao atingir o último caractere da tabela ASCII, a FSM deverá voltar ao início da sequencia, gerando novamente o caracter ‘A’.**

Diagrama de blocos do circuito a ser implementado

1. *Incluir o componente Decod7seg dos labs anteriores;*
2. *Criar um signal F no topo;*
3. *Criar duas 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 <= F.*



D I C A S ! ! !

Dica: Os 4 passos para projetar uma FSM comportamental

PASSO 1: Descrever o sistema da forma mais completa possível (requisitos);

PASSO 2: Preparar uma representação gráfica da FSM, refinando/ redesenhando quando necessário;

PASSO 3: Criar uma tabela de transição de estados para a FSM, listando as entradas e as 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;

Em uma metodologia clássica de projeto de FSM em sistemas digitais, mais passos são necessários, como apresentado na aula anterior.

Dica: Os 4 passos para projetar uma FSM comportamental

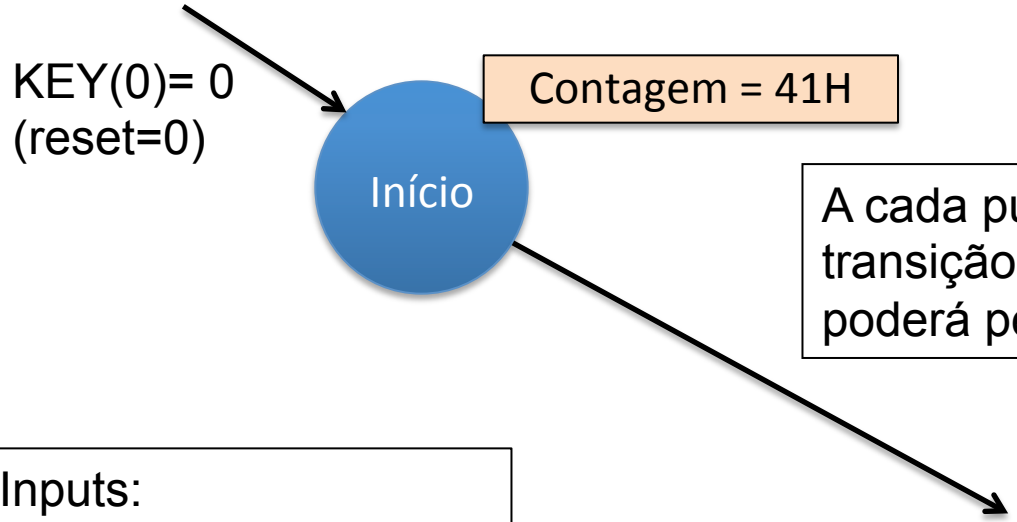
PASSO 1: Descrever o sistema da forma mais completa possível (requisitos).

Exemplo de descrição da FSM:

- No estado inicial, a FSM fornecerá a saída “01000001” (ou 41 em hexadecimal = “A” em ASCII)
- Haverá uma mudança de estado quando o pino clock sofrer uma transição
- Uma evento de transição do sinal de relógio (CLOCK_50), causará uma alteração (incremento) do valor de saída
- O último estado da FSM fornecerá a saída com o caractere “Z” em ASCII (5A em hexadecimal)
- Após o último estado, a FSM deve retornar ao primeiro estado (apresentar saída “A”)
- A qualquer instante, quando o botão de reset for pressionado, a FSM também deverá retornar para o estado inicial, apresentando a saída “A” em ASCII.

Dica: Os 4 passos para projetar uma FSM comportamental

Passo 2: Preparar uma representação gráfica da FSM, refinando/redesenhando quando necessário.



A cada pulso de clock ocorre uma transição para o próximo estado (ou poderá permanecer no mesmo estado)

?

Quem é o próximo estado?
Qual a saída do próximo estado?
Quantos estados serão necessários?
Dica: São necessários menos de 5 estados!
(menos de 10 estados, se considerar o atraso)

Inputs:
- Reset – KEY(0)
- Clock – CLOCK_50

Output:
- Contagem

Completar o diagrama de estados!

Dica: Os 4 passos para projetar uma FSM comportamental

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

ENTRADAS		SAÍDAS	
Reset	**EA**	**PE**	**Contagem**
0	Inicio	Inicio	41H
1	Inicio		42H
1			
1			
1			
1			
1			
1			
1			
1			
1			

EA – Estado Atual
PE – Próximo Estado

Dica: Os 4 passos para projetar uma FSM comportamental

PASSO 4: Descrever em VHDL o comportamento da FSM a partir do diagrama e da tabela de estados.

```
entity FSM_Conta is  port (
                                contagem: out std_logic_vector(7 downto 0);
                                clock, reset : in std_logic);
end;

architecture comportamental of FSM_Conta is
     type STATES is (Inicio, !!!!!completar com os estados que achar necessario!!!!);
      signal EA, PE : STATES;
      signal contador : std_logic_vector(7 downto 0);
begin
      contagem <= contador;              -- contagem e' atualizada fora do process
     process (clock, reset)
     begin
            if reset= '0' then
                  EA  <= Inicio;
            elsif clock'event and clock='1' then      -- clock borda de subida ('1')
                   EA <= PE ;
            end if;
     end process;

     process(clock, EA, X)
     begin
         if clock'event and clock='0' then             -- clock borda de descida ('0')
             case EA is
                 when Inicio => contador <= 0x"41";      -- inicializa o contador (dentro do process)
                                PE <= !!!!!nome do proximo estado a ser definido na lista acima!!!!
                  when …    !!!!!completar com os demais estados, incluindo estados de "atraso"!!!!!

             end case;
        end if;
     end process;
end comportamental;
```

Dica: trecho para geração de atraso (delay) com clock de 50MHz

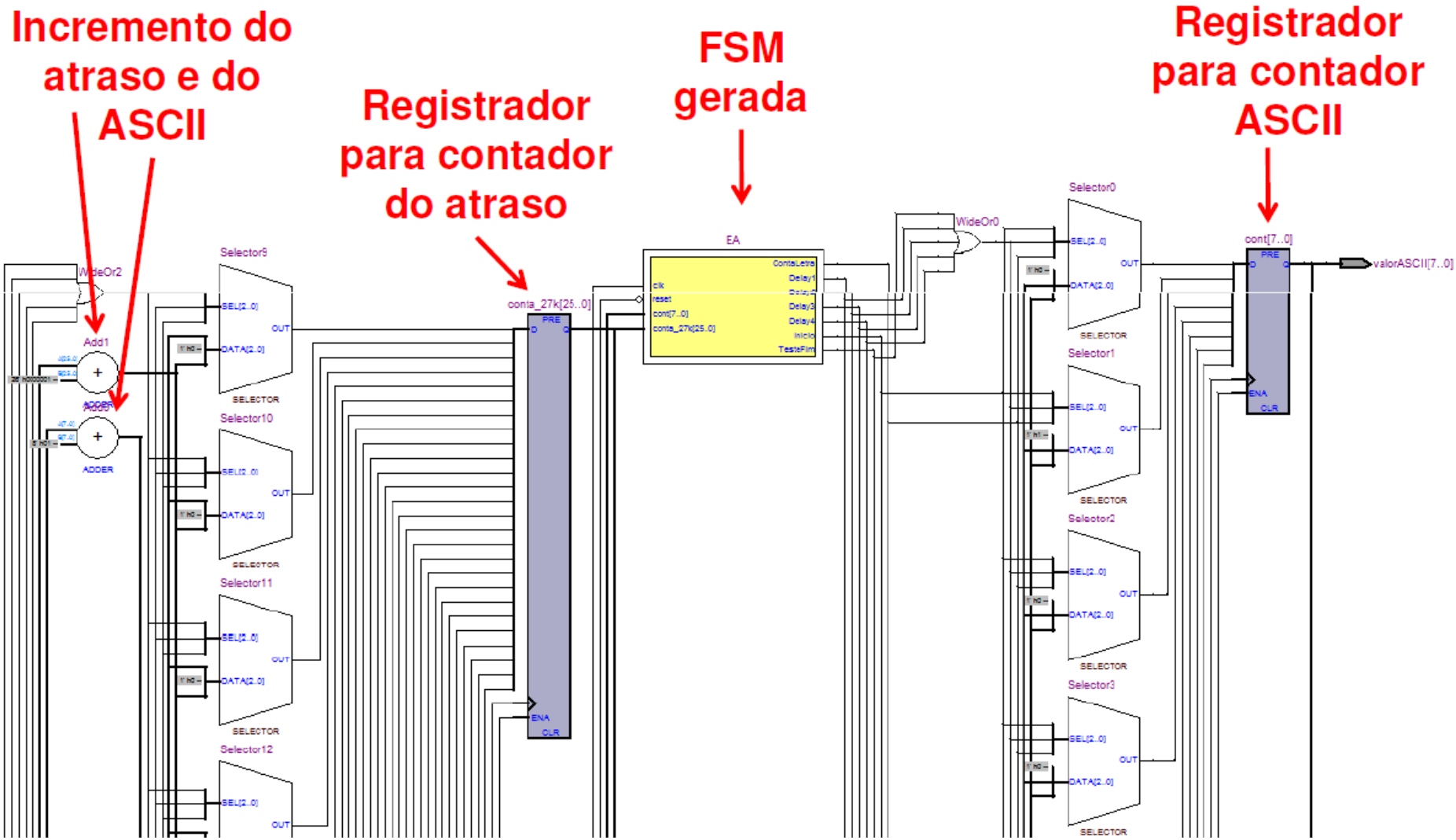
```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;     -- Para usar o '+' nos incrementos.

process(clock, … )   -- Ao usar o clock de 50MHz, esse process será executado
    begin            -- 100 milhões de vezes por segundo (subida e descida).
        …    -- Colocar aqui os estados do contador ASCII (ex. Inicio, inc, fim).
        when D1 =>      -- Estado para iniciar contador do atraso
          atraso <= ( others => '0' );
          EA <= D2;
        when D2 =>      -- Estado para gerar atraso ao mostrar dado no LEDR
          atraso <= atraso + 1;      -- "atraso" foi inicializado com zero em D1.
          EA <= D3;
        when D3 =>      -- Estado para testar se atingiu o valor máximo (15.000.000).
          if atraso >= x"E4E1C0" then   -- 15.000.000 / 50.000.000 = 0,3 * 3 = 1 s.
            EA <= S1;   -- Ao atingir o valor máximo, sai do laço de atraso
          else          -- e volta para o processamento do contador ASCII.
            EA <= D4;   -- Permanece no laço de contagem para gerar atraso.
          end if;
        when D4 =>      -- Estado para continuar contagem do atraso.
          EA <= D2;     -- Essa repetição irá gerar um atraso para
                        -- possibilitar a visualização do dado no LEDR.
```

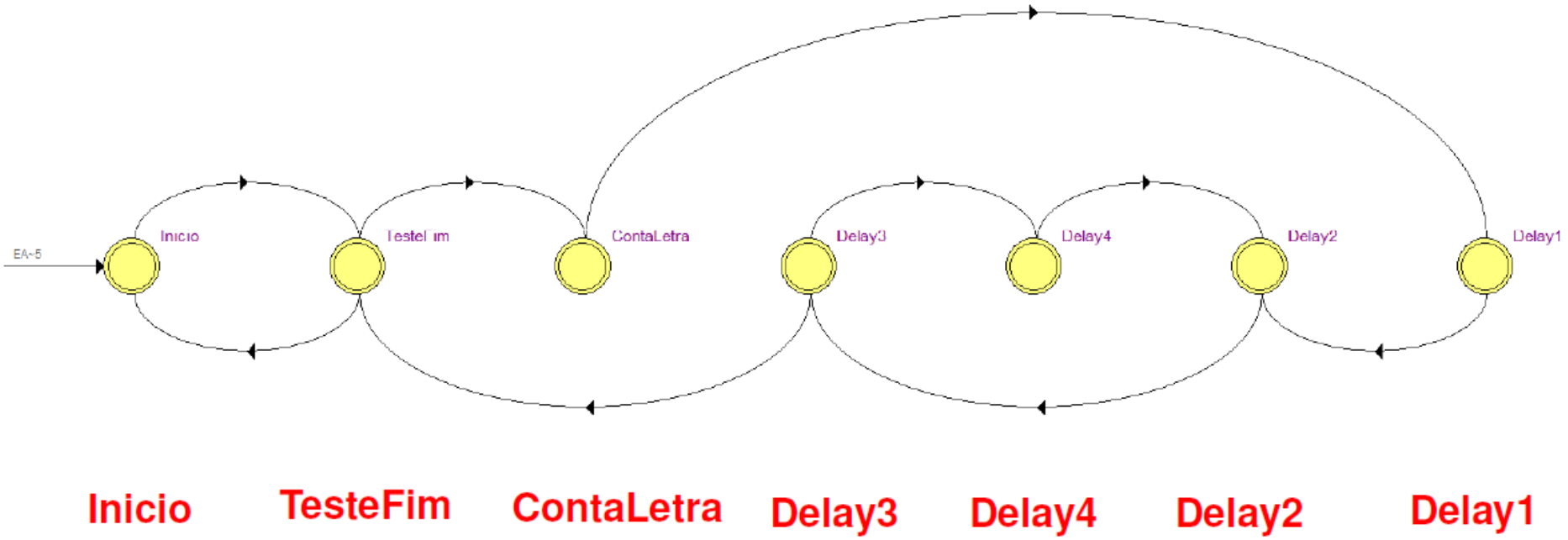
Dica: Topo.vhd com componente FSM e clock de 50MHz
(falta incluir o decodificador BCD/HEX e os displays de 7-segmentos)

```
entity Topo is
     port (          LEDR: out std_logic_vector(7 downto 0);
                     KEY: in std_logic_vector(1 downto 0);
                     CLOCK_50: in std_logic
          );
end Topo;
architecture topo_beh of Topo is
    signal F: std_logic_vector(7 downto 0);
    component FSM_Conta   -- Esse e' o componente FSM
        port (
                contagem: out std_logic_vector(7 downto 0);
                clock: in std_logic;
                reset: in std_logic
        );
    end component;
    -- incluir component decodificador
begin
     L0: FSM_Conta port map (F, CLOCK_50, KEY(0) );
     LEDR <= F;
     --incluir decodificadores
end topo_beh;
```

Dica: Componente FSM_Conta gerado pelo Quartus II



Dica: FSM gerada pelo Quartus II (componente EA do slide anterior)



Simulação no Quartus II (com atraso de +/- 1s)

Type	Message
	Info: Using vector source file "C:/tmp/ContaASCII/ContaASCII.vwf"
	Info: Option to preserve fewer signal transitions to reduce memory requirements is enabled
	Info: Simulation has been partitioned into sub-simulations according to the maximum transition count determined by the eng
	Info: Simulation partitioned into 131 sub-simulations
	Info: Simulation coverage is 67.89 %
	Info: Number of transitions in simulation is 22484275007
	Info: Quartus II Simulator was successful. 0 errors, 0 warnings
	Info: Peak virtual memory: 152 megabytes
	Info: Processing ended: Sun May 20 05:54:13 2012
	Info: Elapsed time: 19:26:26
	Info: Total CPU time (on all processors): 19:28:42

Sistema

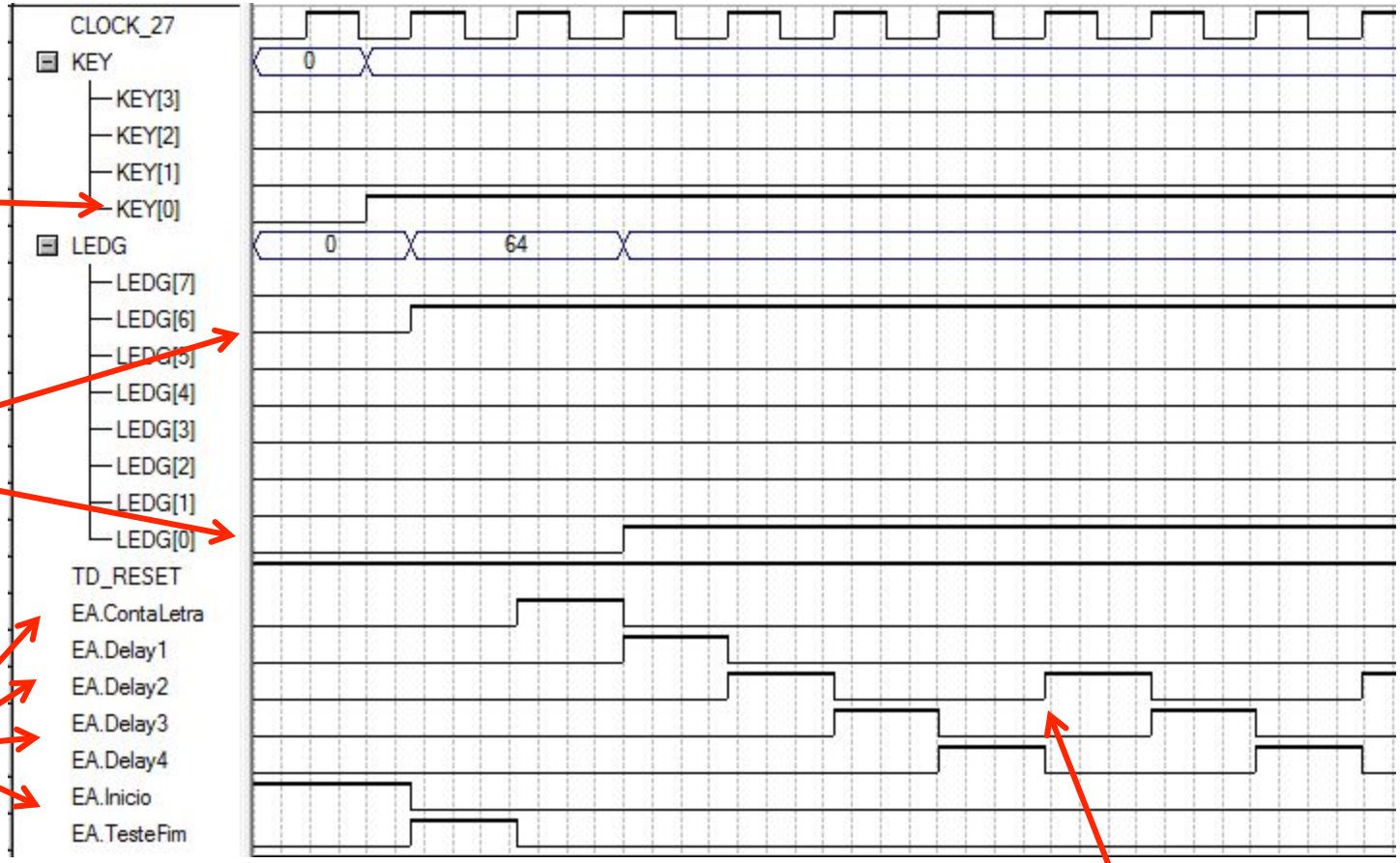
Classificação:	4,4 Índice de Experiência do Windows
Processador:	Intel(R) Core(TM) i7 CPU 870 @ 2.93GHz 2.93 GHz
Memória instalada (RAM):	8,00 GB
Tipo de sistema:	Sistema Operacional de 64 Bits
Caneta e Toque:	Nenhuma Entrada à Caneta ou por Toque está disponível para este vídeo

Nome do computador, domínio e configurações de grupo de trabalho

Nome do computador:	Bezerra
Nome completo do computador:	Bezerra
Descrição do computador:	Bezerra Desktop
Grupo de trabalho:	GRUPO

O tempo total para realizar uma simulação de 20 segundos em um i7 quad core (*hyper threading*, logo "8 cores"), 2,93GHz e 8 G RAM foi de 19 horas e 28 minutos.

Simulação no Quartus II (com atraso de +/- 1s)



Reset

41H = 'A'

Estados da FSM

A cada 3 pulsos de clock, repete os estados do atraso (D2 no slide das dicas de atraso).

Simulação no Quartus II (com atraso de +/- 1s)

Apresentação da contagem em LEDR, a cada pulso no estado EA.ContaLetra