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

http://gse.ufsc.br

"EEL7020 – Sistemas Digitais"

Prof. Eduardo Augusto Bezerra

Eduardo.Bezerra@eel.ufsc.br

Florianópolis, agosto de 2012.

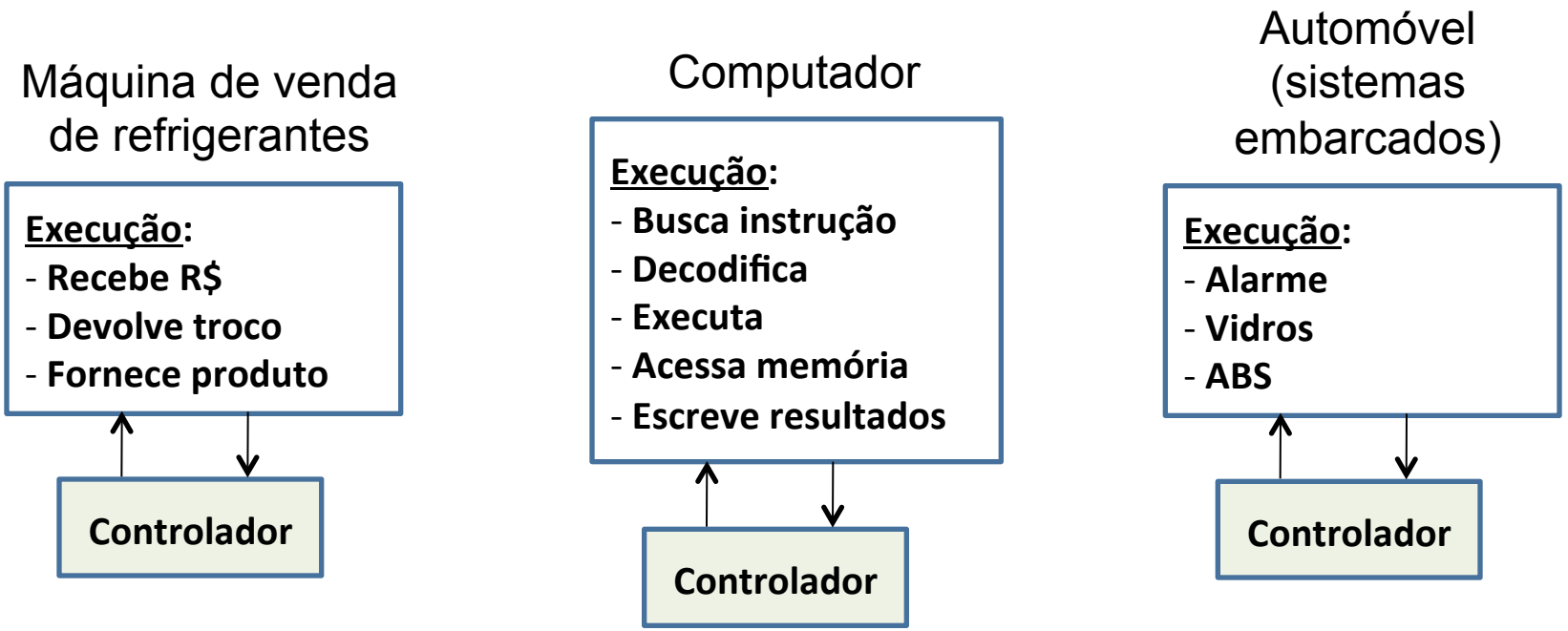
Objetivos do laboratório

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)

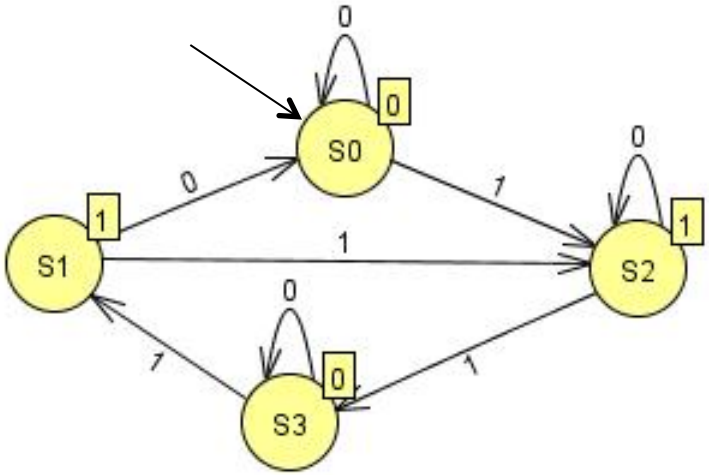
- Sistemas computacionais, normalmente, são compostos por um módulo de "controle" e um módulo para "execução das operações".



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

Estados

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

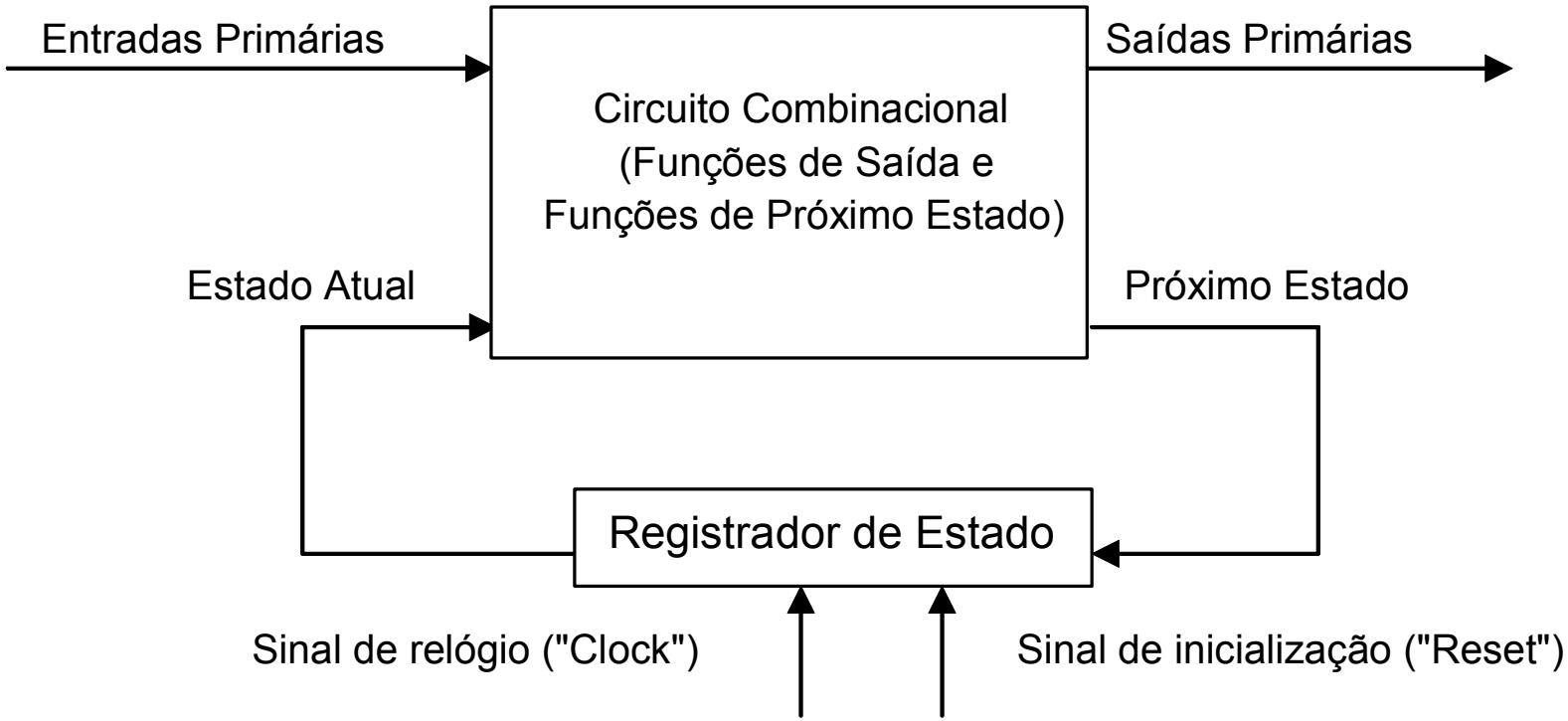
Ação

Descrição da atividade que deve ser executada em um determinado instante

Estrutura de uma FSM

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, ...)

Síntese de FSMs

Uso de HDL para descrever uma FSM

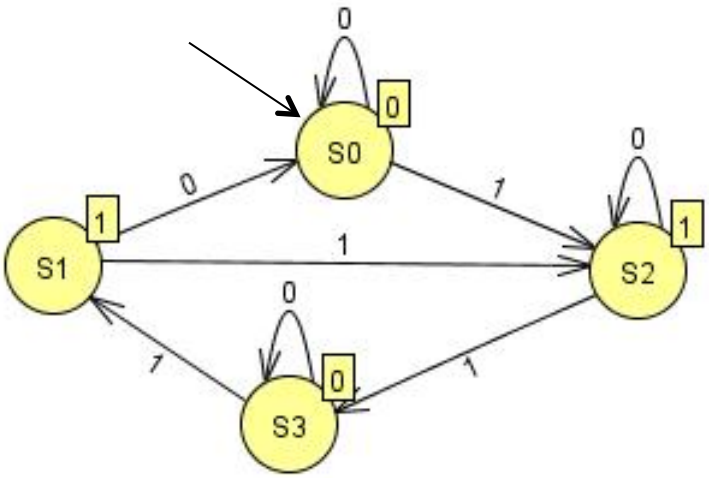
VHDL típico

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(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

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;
```

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

Uso de HDL 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(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;
```

Registrador que armazena o EA em função do próximo estado

Uso de HDL 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(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;
```

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

Uso de HDL para descrever uma FSM

VHDL típico

VHDL típico de uma máquina de estados – 2 processos

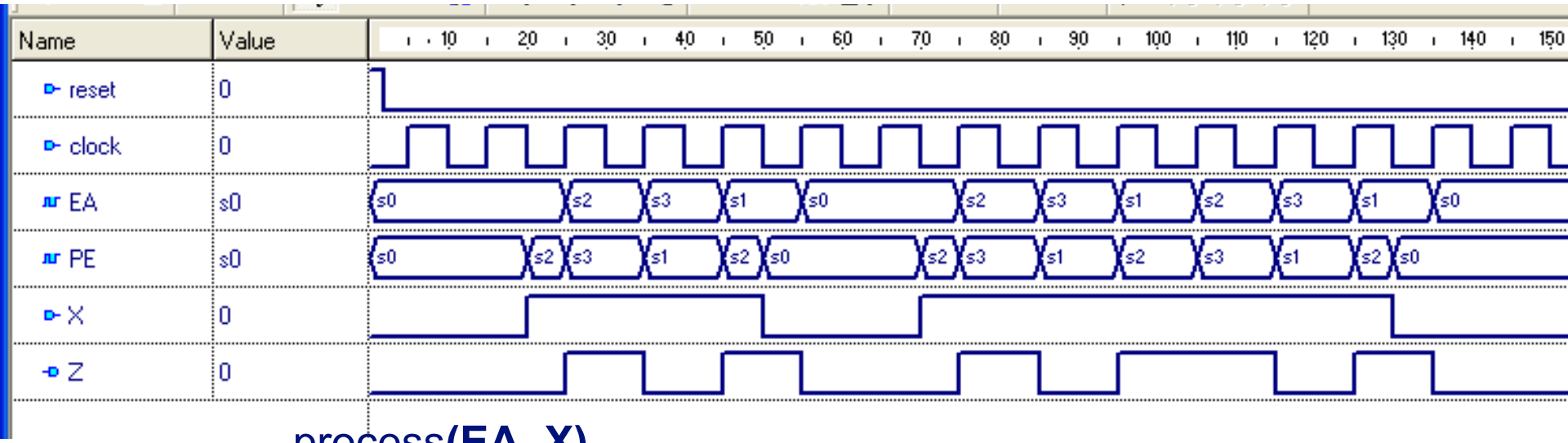
Desenhe a máquina de estados conforme as transições especificadas no processo combinacional:

```
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;
```

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 HDL para descrever uma FSM

VHDL típico




```
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;
```

Uso de HDL para descrever uma FSM

VHDL
1 proc

Máquina de estados – 1 processo

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

architecture B of MOORE is
      type STATES is (S0, S1, S2, S3);
      signal EA: STATES;
begin
      process(clock, reset)
      begin
            if reset= '1' then
                  EA  <= S0;
            elsif clock'event and clock='1' then
                  case EA is
                        when S0 => Z <= '0';
                                    if X='0' then EA <=S0; else EA <= S2; end if;
                        when S1 => Z <= '1';
                                    if X='0' then EA <=S0; else EA <= S2; end if;
                        when S2 =>  Z <= '1';
                                     if X='0' then EA <=S2; else EA <= S3; end if;
                        when S3 => Z <= '0';
                                    if X='0' then EA <=S3; else EA <= S1; end if;
                  end case;
            end if;
      end process;
end B;
```

- EA: mesmo comportamento
- Saída Z ficará defasada 1 ciclo de clock

Uso de HDL para descrever uma FSM

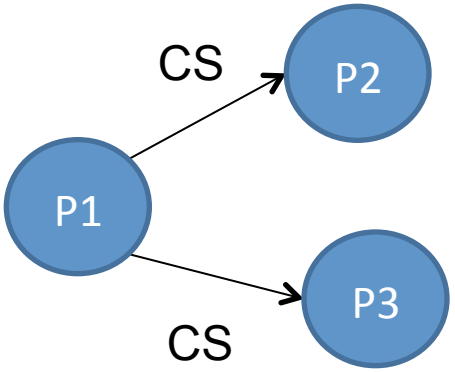
Máquina de estados – 1 processo

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

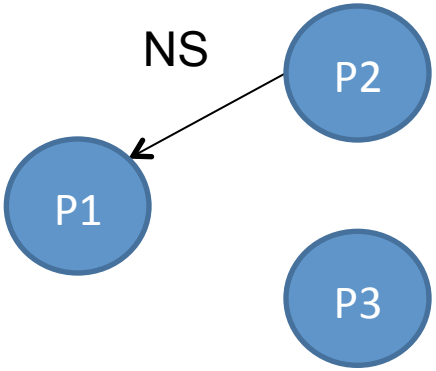
Não tem o PE da FSM com 2 processos!

Uso de HDL 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 HDL 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 HDL 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 HDL 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;
```

Uso de sinais (pinos) disponíveis na DE2 para Clock e Reset

```
library  ieee;
use  ieee.std_logic_1164.all;
entity FSM is
port (
 LEDR: out std_logic_vector(7 downto 0);
 KEY: in std_logic_vector(3 downto 0);
 CLOCK_50: in std_logic
);
end FSM;
architecture FSM_beh of FSM is
 type states is (S0, S1, S2, S3);
 signal EA, PE: states;
 signal clock: std_logic;
 signal reset: std_logic;
begin
 clock <= CLOCK_50;
 reset <= KEY(3);
```

```
process (clock, reset)
 begin
  if reset = '0' then
        EA <= S0;
  elsif clock'event and
                  clock = '1' then
        EA <= PE;
  end if;
 end process;
```

```
process (EA, KEY(0), KEY(1))
begin
 case EA is
  when S0 => if KEY(0) = '0' then
        PE <= S3; else PE <= S0;
        end if;
  when S1 =>
        LEDR <= "01010101";
        PE <= S0;
  when S2 =>
   case KEY(1) is
     when '0' => LEDR <= "10101010";
     when '1' => LEDR <= "00000000";
     when others => LEDR <= "11111111";
   end case;
   PE <= S1;
  when S3 =>
         PE <= S2;
   end case;
end process;
end FSM_beh;
```

Tarefa a ser realizada na aula prática

Tarefa

- **Implementar uma FSM em VHDL com 1 processo para geração dos caracteres ‘A’ a ‘Z’ da tabela ASCII, apresentando os caracteres nos LEDs verdes (LEDG).**
- **FSM com *reset* assíncrono (usar botão KEY(0)) para inicializar um contador com o valor do primeiro caracter a ser gerado (‘A’ = 41H).**
- **A cada pulso do relógio de 27 MHz (borda de subida), o contador deverá ser incrementado, gerando o próximo caracter da tabela ASCII.**
- **A FSM deverá possuir um número reduzido de estados, número esse suficiente para incrementar o contador, e verificar se chegou ao final da contagem (caracter ‘Z’ = 5AH).**
- **Ao atingir o último caracter da tabela ASCII, a FSM deverá voltar ao início da sequencia, gerando novamente o caracter ‘A’.**

Obs. Alternativamente, no lugar do relógio de 27MHz o sinal de clock poderá ser gerado por um botão (KEY), porém é preciso cuidar o problema do *debounce*.

Diagrama de blocos do circuito a ser implementado (FSM utilizada como contador)

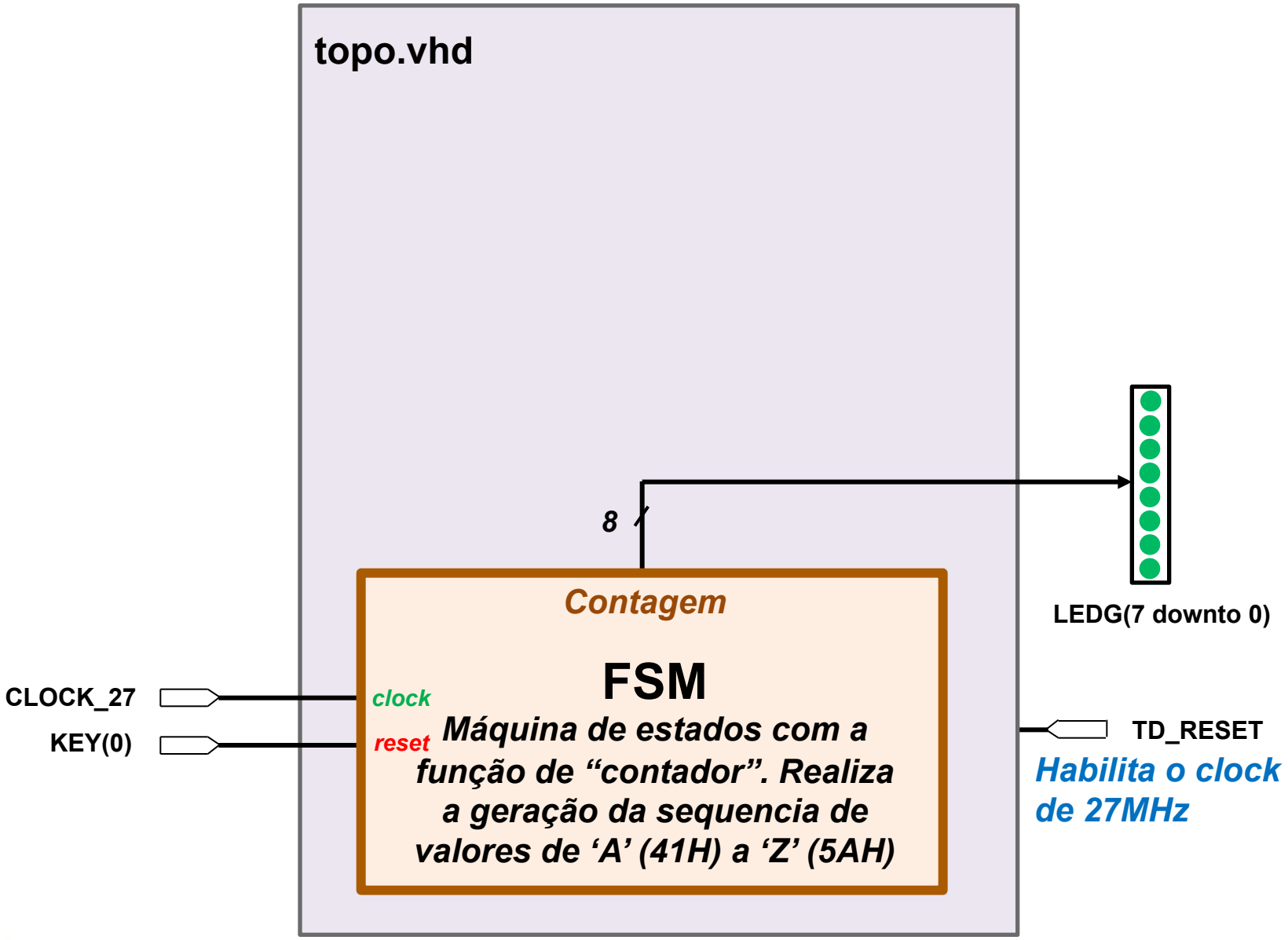
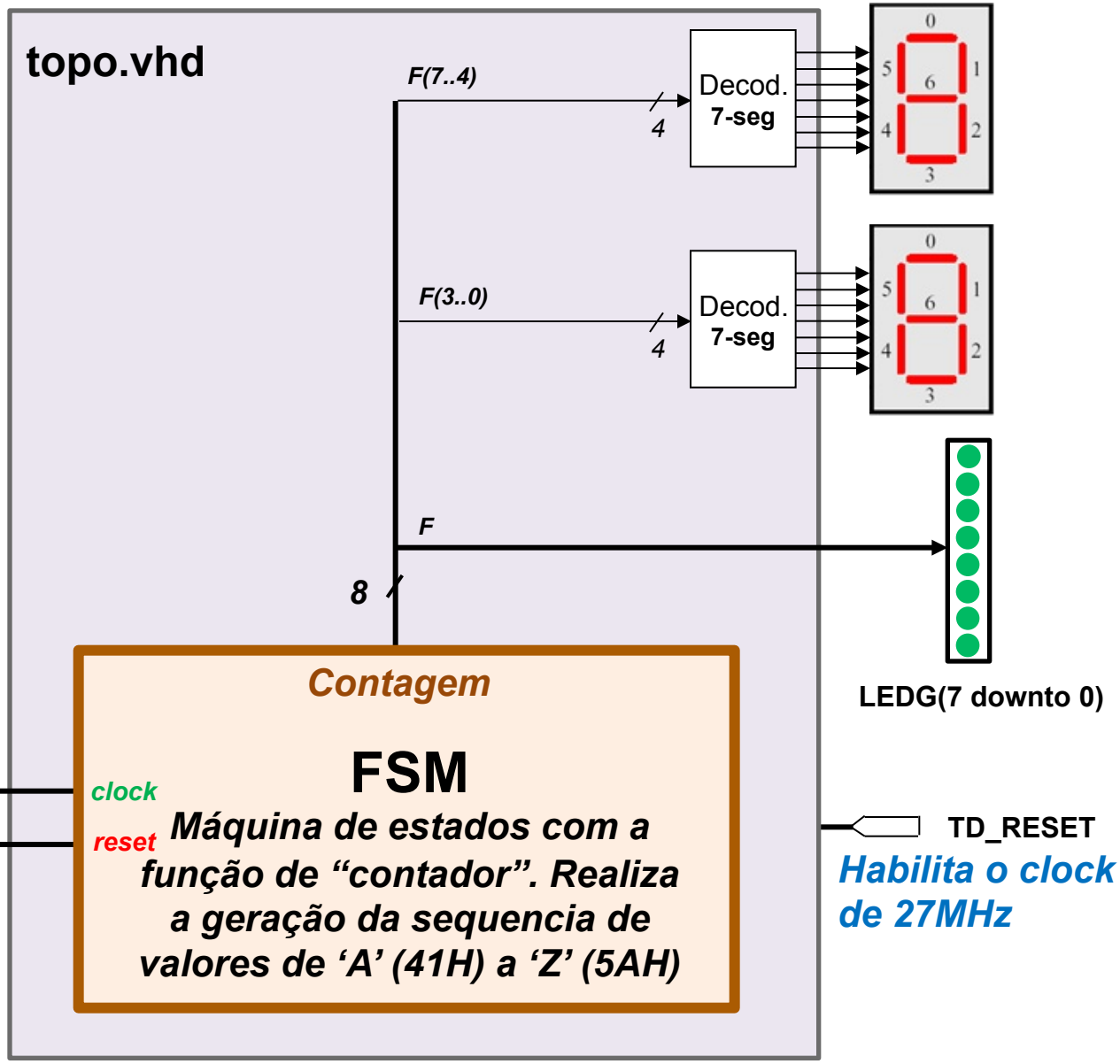


Diagrama de blocos do circuito a ser implementado
"Uso dos displays de 7-segmentos como saída"

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



Dica: Topo.vhd com componente FSM e clock de 27MHz (sem utilizar os displays de 7-segmentos)

```
entity Topo is
    port (          LEDG: out std_logic_vector(7 downto 0);
                    KEY: in std_logic_vector(3 downto 0);
                    TD_RESET: out std_logic;
                    CLOCK_27: in std_logic
         );
end Topo;
architecture topo_beh of Topo is
    component ContaASCII   -- Esse e' o componente FSM
        port (
                valorASCII: out std_logic_vector(7 downto 0);
                clock: in std_logic;
                reset: in std_logic
        );
begin
    TD_RESET <= '1';

    L0: ContaASCII port map ( LEDG, CLOCK_27, KEY(0) );

end topo_beh;
```

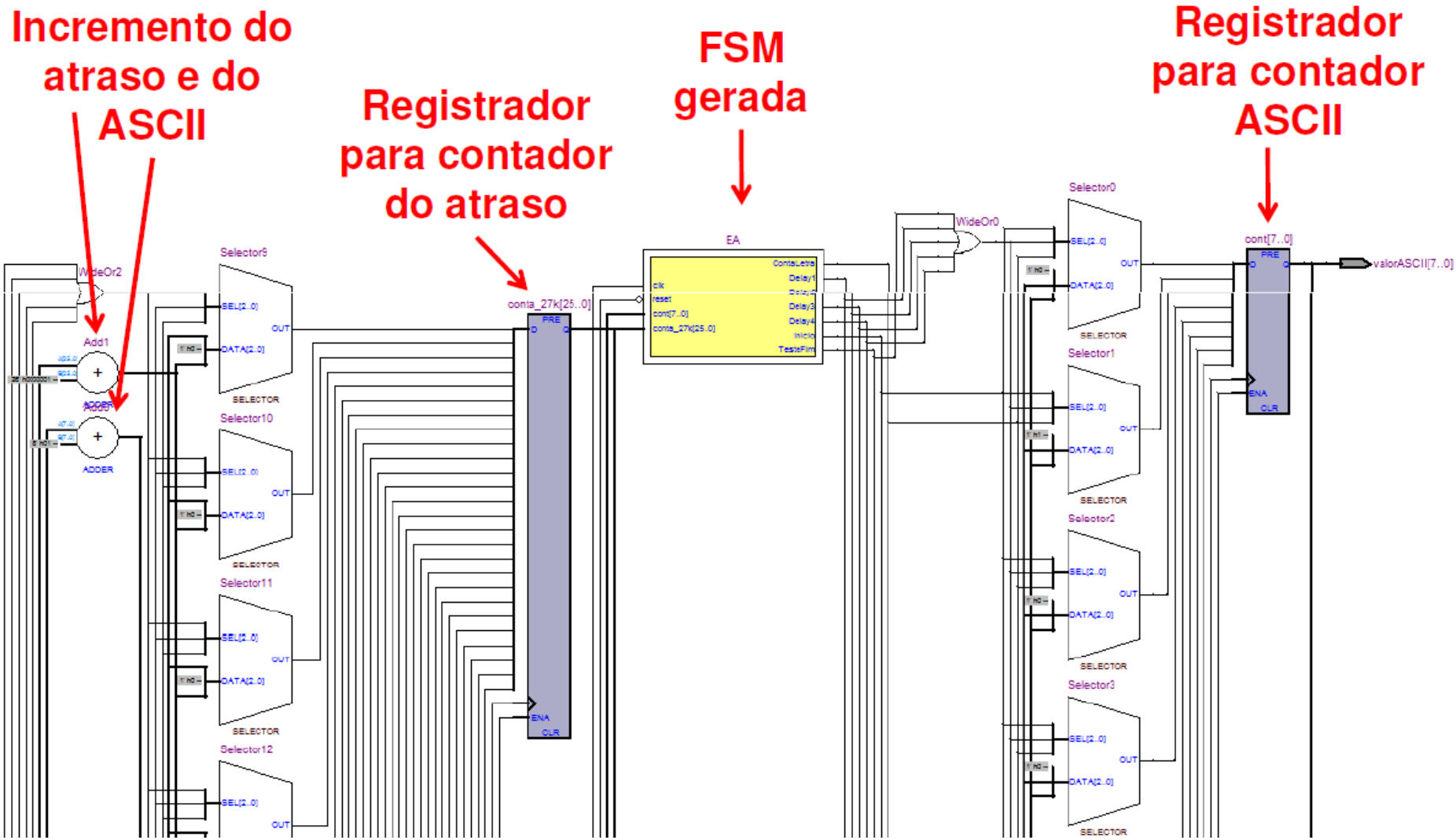
Colocar TD_RESET em '1' para "ligar" o sinal de CLOCK_27 da DE2

Dica: ContaASCII.vhd – trecho para geração de atraso (delay) com clock de 27MHz

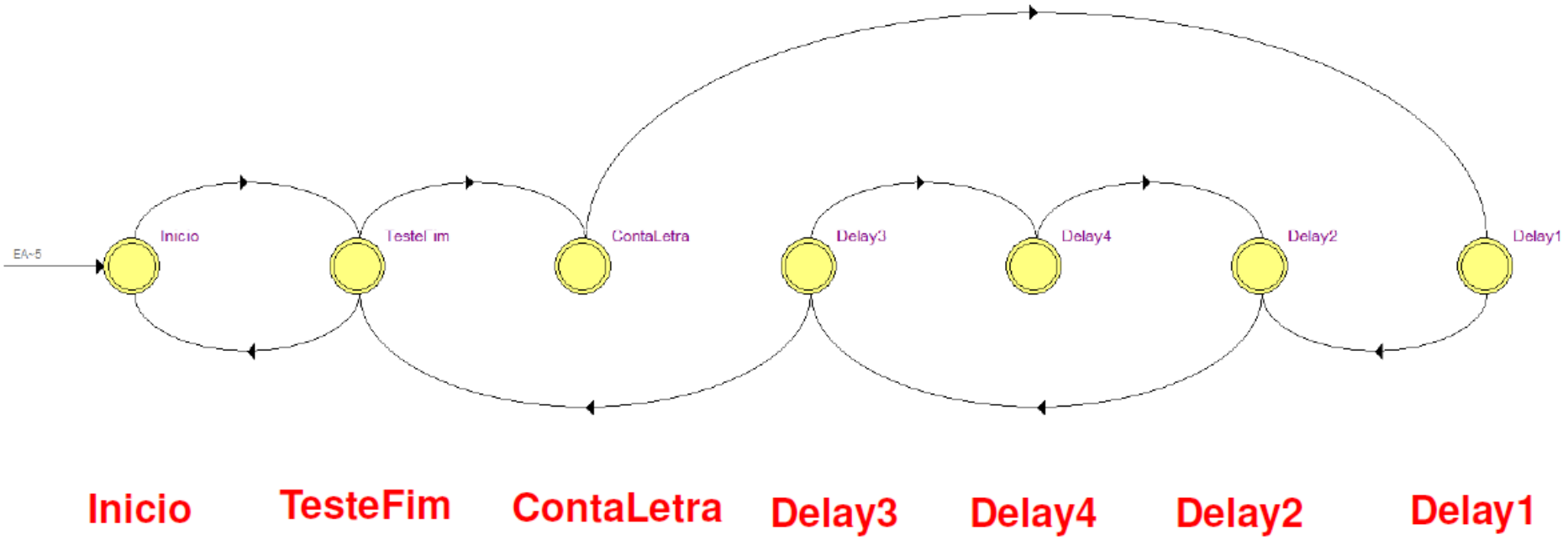
```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;     -- Para usar o '+' nos incrementos.

process(clock, … )          -- Ao usar o clock de 27MHz, esse process será
    begin                   -- executado 27 milhões de vezes por segundo.
        …    -- 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 LEDG
          atraso <= atraso + 1;     -- "atraso" foi inicializado com zero em D1.
          EA <= D3;
        when D3 =>      -- Estado para testar se atingiu o valor máximo.
          if atraso >= x"800000" then   -- 8.388.608 / 27.000.000 = 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 LEDG.
```

Dica: Componente ContaASCII 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

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.

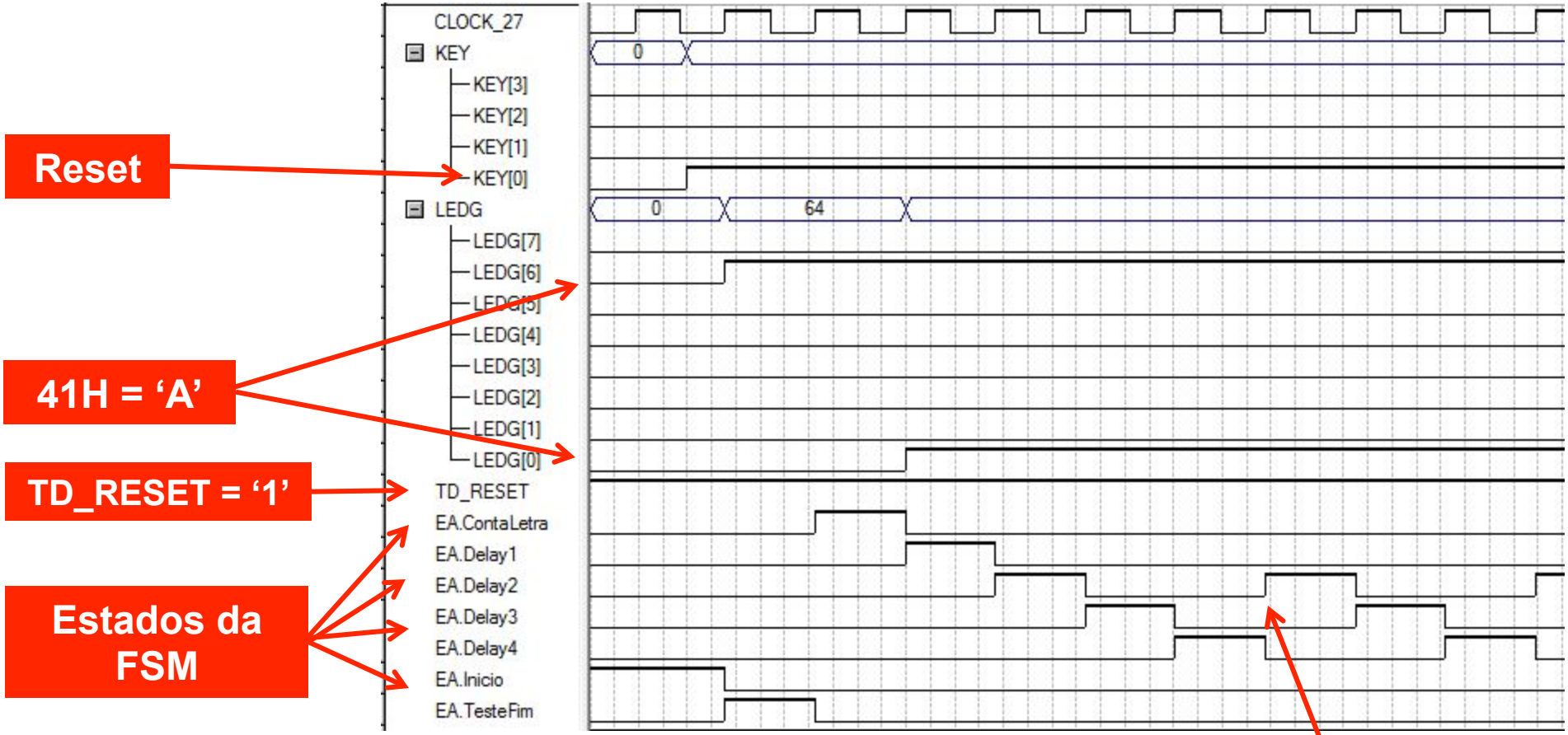
Sistema

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

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



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 LEDG, a cada pulso no estado EA.ContaLetra