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

"EEL7020 – Sistemas Digitais"

Prof. Eduardo Augusto Bezerra

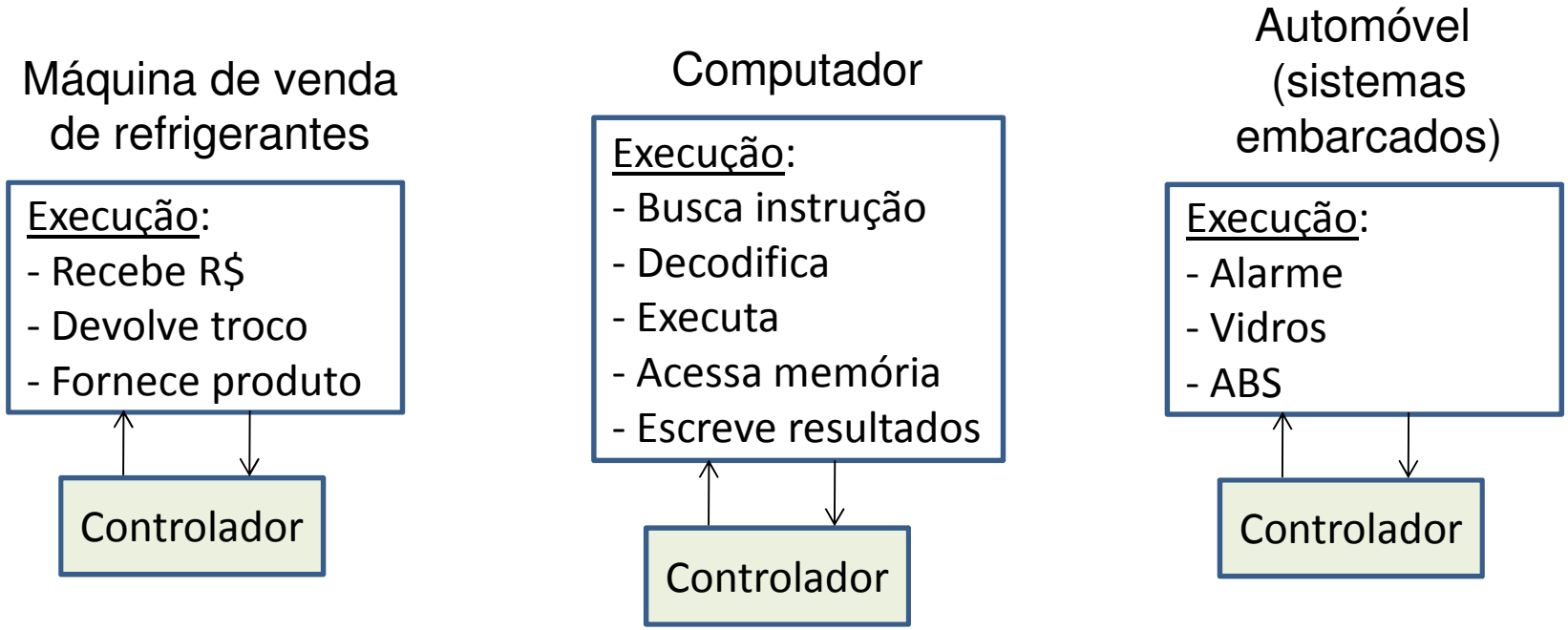
Eduardo.Bezerra@eel.ufsc.br

Florianópolis, agosto de 2011.

“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".



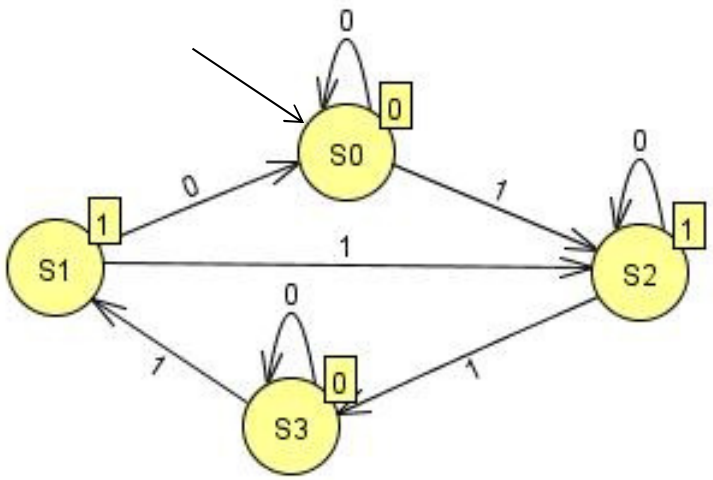
Finite State Machine (FSM)

- 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

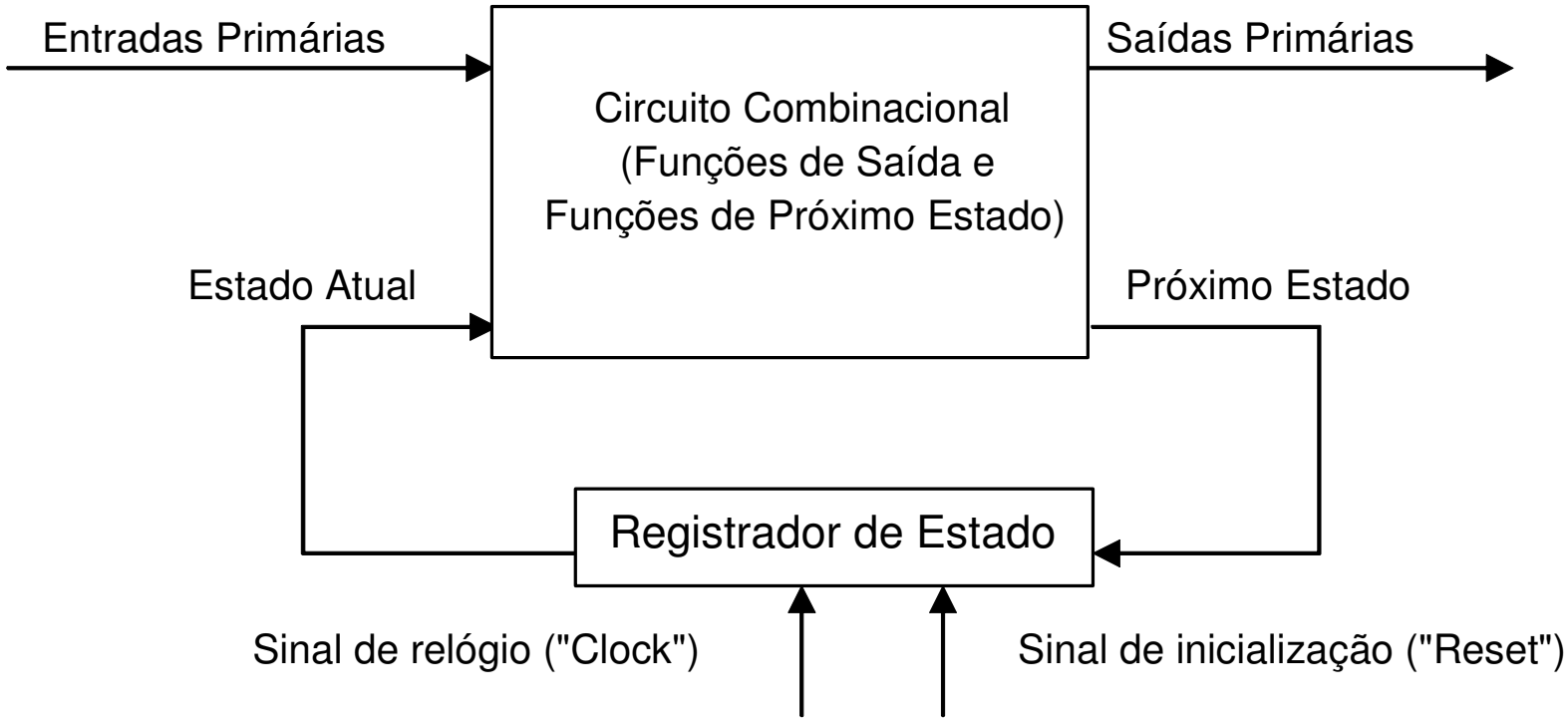
Ação

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, ...)

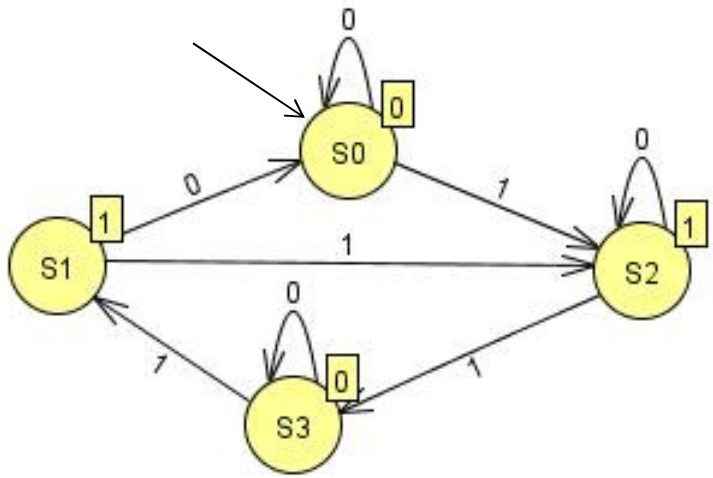
Síntese de FSMs

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

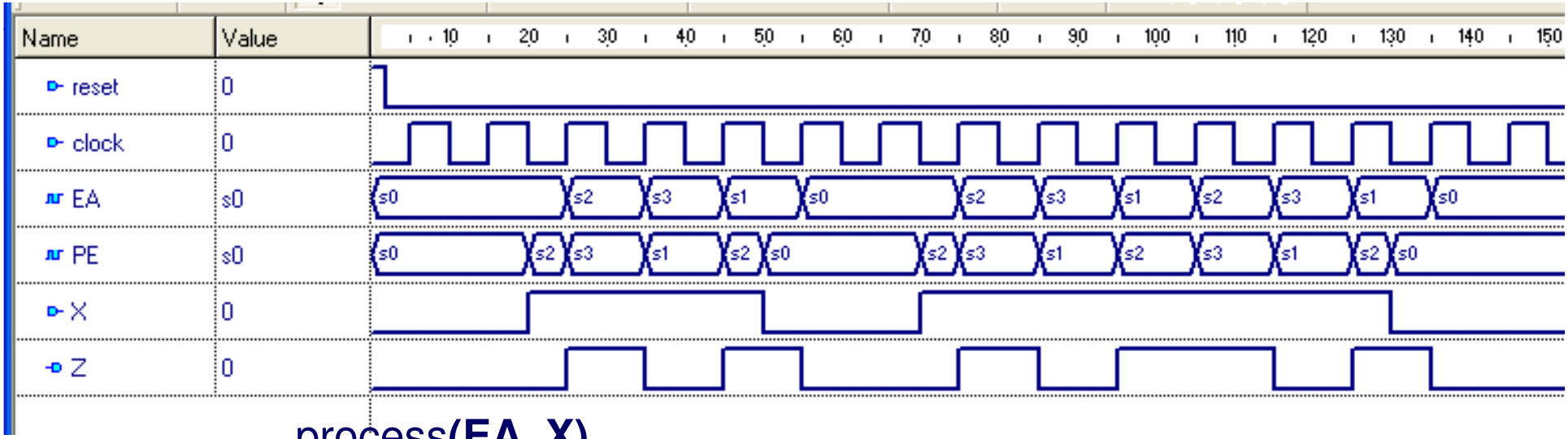
Desenhe a máquina de estados conforme as transições especificadas no processo combinacional:

```
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. Porquê?

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

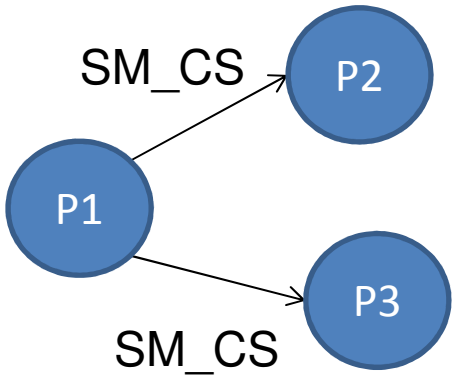
```
entity MOORE is    port(X, clock, reset : in std_logic;     Z: out std_logic);    end;

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

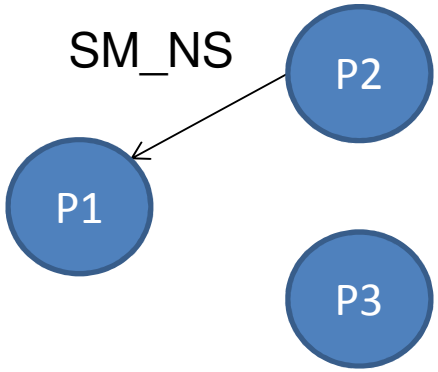
Uso de HDL para descrever uma FSM

VHDL
3 procs

Máquina de estados – 3 processos



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



- Com base nos valores dos sinais, P2 define o próximo estado, colocando essa informação no sinal SM_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

VHDL
3 procs

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 (SM_CS, State Machine Current State) receba o próximo estado (SM_NS, State Machine Next State). Essa transição é sensível a borda de descida do clock.

```
stChange:
process(clk_i)
begin
   if clk_i'event and clk_i='0' then
      if rst_n_i='0' then
         SM_CS <= SM_rst;
      else
         SM_CS <= SM_NS;
      end if;f
   end if;
end process;
```

Uso de HDL para descrever uma FSM

VHDL
3 procs

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 SM_NS a ser utilizado pelo processo P1 responsável por realizar as transições de estados. Comando "case SM_CS is“ seleciona o estado atual (Current State) e, conforme os sinais da FSM, um próximo estado é definido no sinal SM_NS.

```
process( SM_CS, signalA )
begin
   case SM_CS is
      when state_rst =>
         SM_NS <= state_idle
      when state_idle =>
         if signalA = '1' then
            SM_NS <= state_exec;
         else
            SM_NS <= state_idle;
         end if;
      when state_exec =>
         if signalA = '1' then
            SM_NS <= state_exec;
         else
            SM_NS <= state_idle;
         end if;
      when state_exec =>
         SM_NS <= state_idle
      when others =>
         end case;
end process;
```

Uso de HDL para descrever uma FSM

VHDL
3 procs

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_i)
begin
   if clk_i'event and clk_i='1' then
      case SM_CS is
         when state_rst =>
            signal_out <= '0'
         when state_idle =>
            signal_out <= '0'
         when state_exec =>
            signal_out <= '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 para geração dos caracteres ‘A’ a ‘Z’ da tabela ASCII, apresentando esses caracteres nos LEDs verdes.**
- **A FSM deverá possuir *reset* assíncrono, acionado pelo botão KEY(0), para inicializar um contador com o valor do primeiro caracter da tabela ASCII a ser gerado (‘A’ = 41H).**
- **A cada pulso do relógio de 27 kHz (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’.**

Simulação com *ModelSim*

Simulação do projeto com ModelSim

1. Criar uma nova pasta dentro da pasta do projeto.
2. Copiar os *scripts* de simulação disponíveis na página da disciplina para dentro da nova pasta.
3. Entrar na nova pasta, editar o arquivo "*compila.do*", e alterar *ascii.vhd* para o nome do seu arquivo VHDL a ser simulado.
4. Copiar APENAS o seu arquivo VHDL (*contador*) a ser simulado para a nova pasta, que já deve possuir os *scripts* de simulação copiados da página da disciplina.
5. Executar o *ModelSim-Altera*, que se encontra no menu *Iniciar* do Windows, pasta "*Altera*".
6. No menu "*File*" do *ModelSim*, definir a pasta do projeto (opção "*Change Directory*"), selecionando a nova pasta.

Simulação do projeto com ModelSim (cont.)

7. **Execução da simulação (arquivo *compila.do*)**
   a) **No menu "Tools" do ModelSim, selecionar "*Tcl*" -> "*Execute Macro*".**
   b) **Selecionar o arquivo "*compila.do*", e "*Open*".**

8. **O *ModelSim* irá compilar os arquivos VHDL e iniciar a simulação.**
9. **A janela com as formas de onda irá abrir, apresentando o resultado da simulação.**

***Obs.* Se desejar, editar o arquivo tb.vhd para alterar a simulação a ser realizada, e repetir o passo 7.**

Simulação do projeto com ModelSim (cont.)

Obs. **Se o resultado da simulação não estiver de acordo com o esperado, alterar o seu VHDL, salvar, e executar novamente a simulação (arquivo *compila.do*).**

Obs. **A simulação só irá funcionar se o seu projeto possuir EXATAMENTE a seguinte *entity*:**

```
entity ascii IS
    PORT (LEDG: OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
          KEY: IN STD_LOGIC_VECTOR(3 DOWNTO 0);
          CLOCK_50    : IN    STD_LOGIC
    );
end ascii;
```

Simulação do projeto com ModelSim (cont.)

Obs. O arquivo "*compila.do*" contém os comandos do *ModelSim* necessários para realizar a simulação, incluindo:

a) Criação da biblioteca de trabalho - comando *vlib.*

b) Compilação dos arquivos VHDL para a biblioteca de trabalho - comando *vcom*.

c) Inicialização do simulador com o arquivo *testbench* - comando *vsim*.

d) Execução da janela de formas de onda (*waveform*) - comando *wave*.

e) Adição dos sinais na janela de formas de onda - comando *wave*.

f) Execução da simulação - comando *run*.

Tarefa Adicional

Controlador de uma máquina de venda de refrigerantes

Estudo de caso: Controlador de uma máquina de venda de refrigerantes

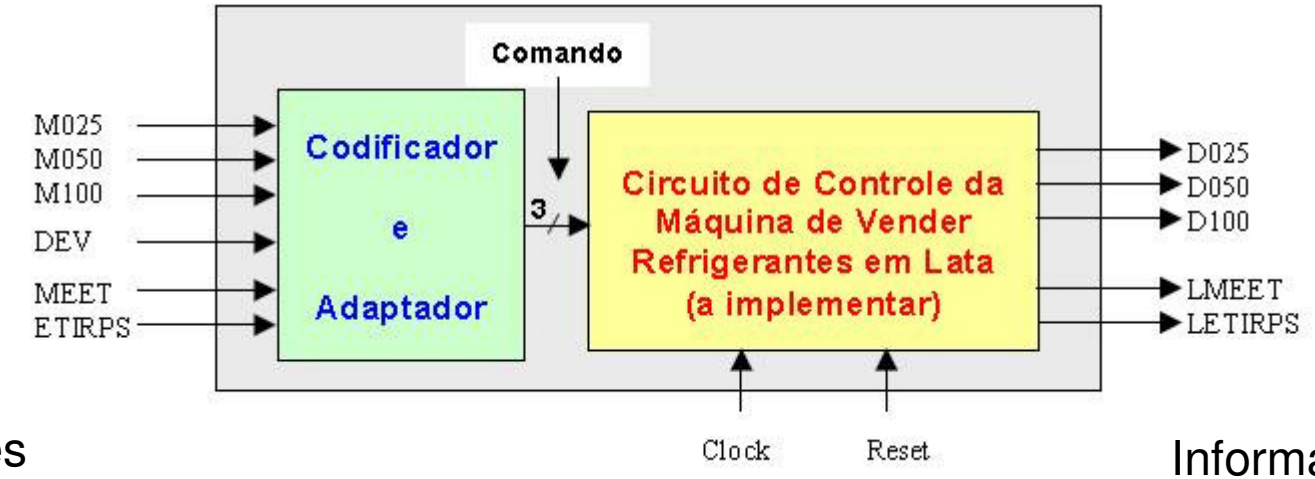
Projetar o circuito de controle (FSM) para gerência das operações de uma máquina de venda de refrigerantes.

Especificação:

A máquina fornece dois tipos de refrigerantes, denominados MEET e ETIRPS. Estes estão disponíveis para escolha pelo usuário a partir de duas teclas no painel com o nome dos refrigerantes. Ambos refrigerantes custam R$1,50 e existe na máquina uma fenda para inserir moedas com um sistema eletromecânico capaz de reconhecer moedas de R$1,00, R$0,50 e R$0,25, e capaz de devolver automaticamente qualquer outro tipo de moeda ou objeto não reconhecido. Além disso, durante a compra, o usuário pode desistir da transação e apertar a tecla DEV que devolve as moedas inseridas até o momento. Somente após acumular um crédito mínimo de R$1,50 o usuário pode obter um refrigerante. A devolução de excesso de moedas é automática sempre que o valor inserido antes de retirar um refrigerante ultrapassar R$1,50. Uma terceira simplificadora consiste em ignorar a composição exata das moedas inseridas na máquina, atendo-se apenas ao montante total inserido.

Estudo de caso: Controlador de uma máquina de venda de refrigerantes

Solução: Diagrama de blocos



Informações fornecidas pelos sensores

Informações enviadas para os atuadores (eletro-mecânicos

Estudo de caso: Controlador de uma máquina de venda de refrigerantes

Solução: Tabela de estados

Estado Atual	Comando de Entrada						
	Nada	M025	M050	M100	DEV	MEET	ETIRPS
S000		S025			S000	S000	
S025		S050			S000, D025		
S050						S050	
S075							
S100	S100			S150, D050			
S125							
S150							

Estudo de caso: Controlador de uma máquina de venda de refrigerantes

Solução: As cinco etapas usadas para o projeto de controladores (Frank Vahid)

1. **Captura da FSM** – definir uma FSM que descreva o comportamento desejado do controlador.
2. **Definição da arquitetura** – criar a arquitetura padrão de uma FSM, utilizando um registrador para armazenar os estados, e a lógica combinacional para, a partir das entradas e do estado atual, definir as saídas e o próximo estado.
3. **Codificação dos estados** – Definir uma identificação única para os estados.
4. **Criação da tabela de estados** – Criar a tabela verdade para a lógica combinacional, de forma a gerar os valores apropriados de saídas e próximo estado. As se ordenar as entradas e a codificação de estados, faz com que essa tabela verdade venha a representar a “tabela de estados”.
5. **Implementação da lógica combinacional**.

Ao se utilizar VHDL para a síntese de uma FSM, as abordagens com um, dois ou três processos, irão modelar o comportamento do hardware projetado a partir dessas cinco etapas de projeto. A ferramenta de síntese irá então gerar o circuito digital para a FSM modelada.

Codificações de estados para FSMs

Codificação de estados para FSMs

• Definição dos estados da FSM da máquina de venda de refrigerantes:

```
package refri is
        type cmd is (Nada, M025, M050, M100, DEV, MEET, ETIRPS);
end package;
```

• Ferramentas de síntese geram a codificação mais adequada para os estados simbólicos:

Codificação	Exemplo – FSM com 4 estados	Otimização
Sequencial (binária)	00, 01, 10, 11	Área
One hot	0001, 0010, 0100, 1000	Velocidade
Grey	00, 01, 11, 10	Ruído

• Escolha do estilo de codificação visa redução de custos e atrasos

Codificação de estados para FSMs

Codificação Binária

Vantagens:

- Mais simples e direta, visto que segue a numeração binária padrão
- Número de FFs também é menor, quando comparado a outros métodos tais como one-hot
- Custo de área menor

Desvantagens

- Mais de um bit pode mudar de uma transição para outra
- Maior consumo (maior atividade de chaveamento nos FFs)
- Lógica mais complexa para determinar o estado atual

Gray

Codificação de estados para FSMs

Codificação Gray

- Estados são identificados através de números binários, mas na codificação *Gray*

- Para transformar um número binário na codificação *Gray:*
  "Seja o número: $b_1b_2..b_{n-1}b_n$. Se b_{n-1} é 1, trocar b_n para 1-b_n, caso contrário manter como está. Repetir o procedimento até atingir o bit mais significativo."

Ex.:

```
000 → 000
001 → 001
010 → 011
011 → 010
100 → 110
101 → 111
```

One hot

Codificação de estados para FSMs

Codificação one hot

- Lógica de decodificação é mais simplificada, mas utiliza um número maior de FFs

- Cada estado possui seu próprio FF com o valor 1 (um), enquanto os demais ficam em 0 (zero).

Ex.:

```
000 → 000001
001 → 000010
010 → 000100
011 → 001000
100 → 010000
101 → 100000
```

Codificação de estados para FSMs

- Codificação VHDL da FSM do exemplo da *vending machine*:

```
package refri is
        type cmd is (Nada, M025, M050, M100, DEV, MEET, ETIRPS);
end package;
```

- Synplify gera, automaticamente, codificação one hot:

@N:CD231 : VendingMachine.vhd(29) | Using onehot encoding for type cmd (nada="1000000")

Codificação de estados para FSMs

Justificativa para one hot ser *default* em FPGAs:

- Abundância de flip-flops em FPGAs facilita o mapeamento.
- FSMs one hot são mais rápidas. Uma FSM mais codificada pode ficar mais lenta com a inclusão de novos estados.
- Facilidade para projetar FSMs one hot. VHDL pode ser escrito a partir de um diagrama de estados, sem necessidade de criação de uma tabela de estados.
- Facilidade para realizar modificações.
- Facilidade para realizar a síntese em VHDL ou Verilog.

Codificação de estados para FSMs

É possível forçar a utilização de outras codificações no processo de síntese. Incluir no VHDL:

```
package refri is
   type cmd is (Nada, M025, M050, M100, DEV, MEET, ETIRPS);

   attribute syn_enum_encoding : string;
   attribute syn_enum_encoding of cmd : type is “gray";
end package;

-- opções disponíveis: onehot, gray, sequential
```

Log da síntese do Synplify mostrará:

@N:CD232 : VendingMachine.vhd(29) | Using gray code encoding for type cmd

Codificação de estados para FSMs

Outra opção, selecionar o tipo de codificação diretamente na ferramenta de síntese, antes de iniciar o processo: