Laboratório de Sistemas Digitais

Aula Teórico-Prática 7

Ano Lectivo 2020/21

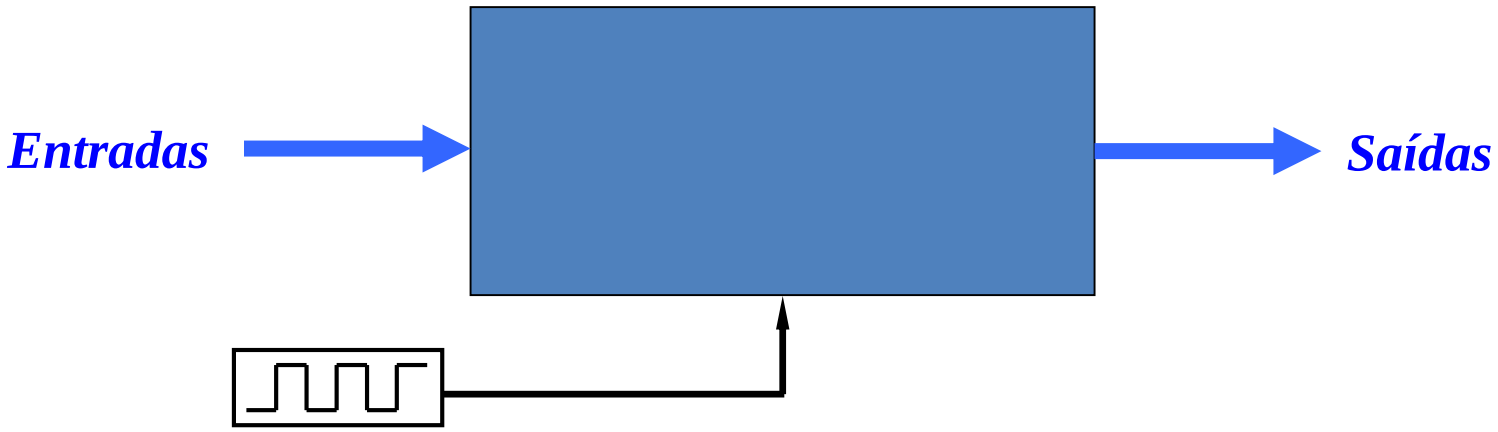
Modelação, simulação e síntese de Máquinas de Estados Finitos

Aspectos gerais e modelo de *Moore*

Conteúdo

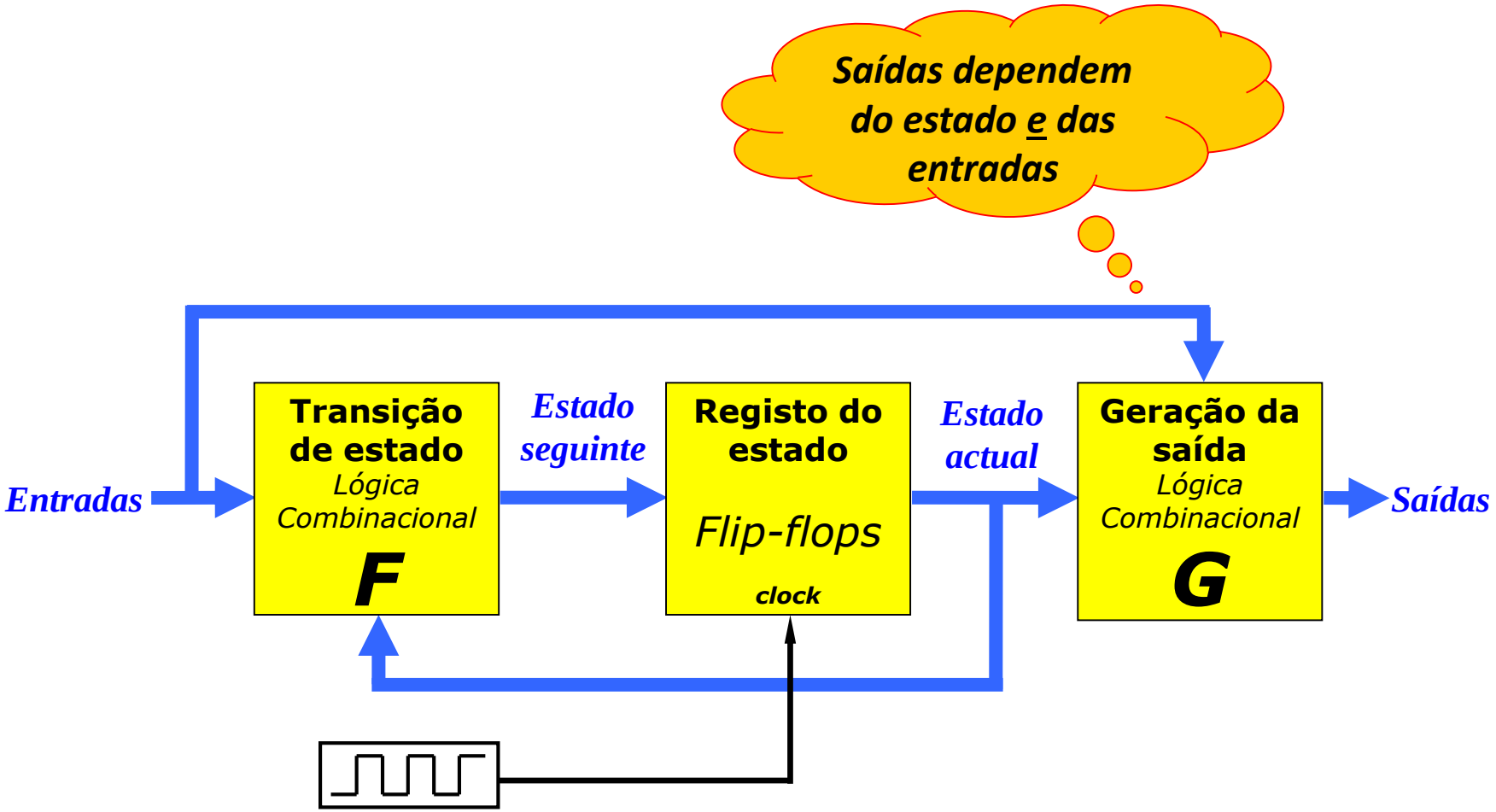
- Revisão sobre Máquinas de Estados Finitos (MEF ou *FSM*)
  - Estrutura-base; modelos de Mealy e **Moore**
- Revisão sobre Diagramas de Estado (DDE)
  - Especificação inicial → DDE → MEF
    - Tradução completa e **rigorosa**
  - Exemplos práticos
  - Métodos de construção
    - Manual
    - Com o editor do *Quartus Prime*
- Procedimentos de síntese e implementação
  - Descrição em VHDL de MEF
    - Estrutural (registo + blocos combinatórios)
    - Comportamental (tradução do DDE): abordagem com 2 processos (Moore)
    - Codificação (e síntese) automática a partir do DDE
  - Simulação de MEF

Circuitos sequenciais síncronos (ou MEF - *Máquinas de Estados Finitos*)

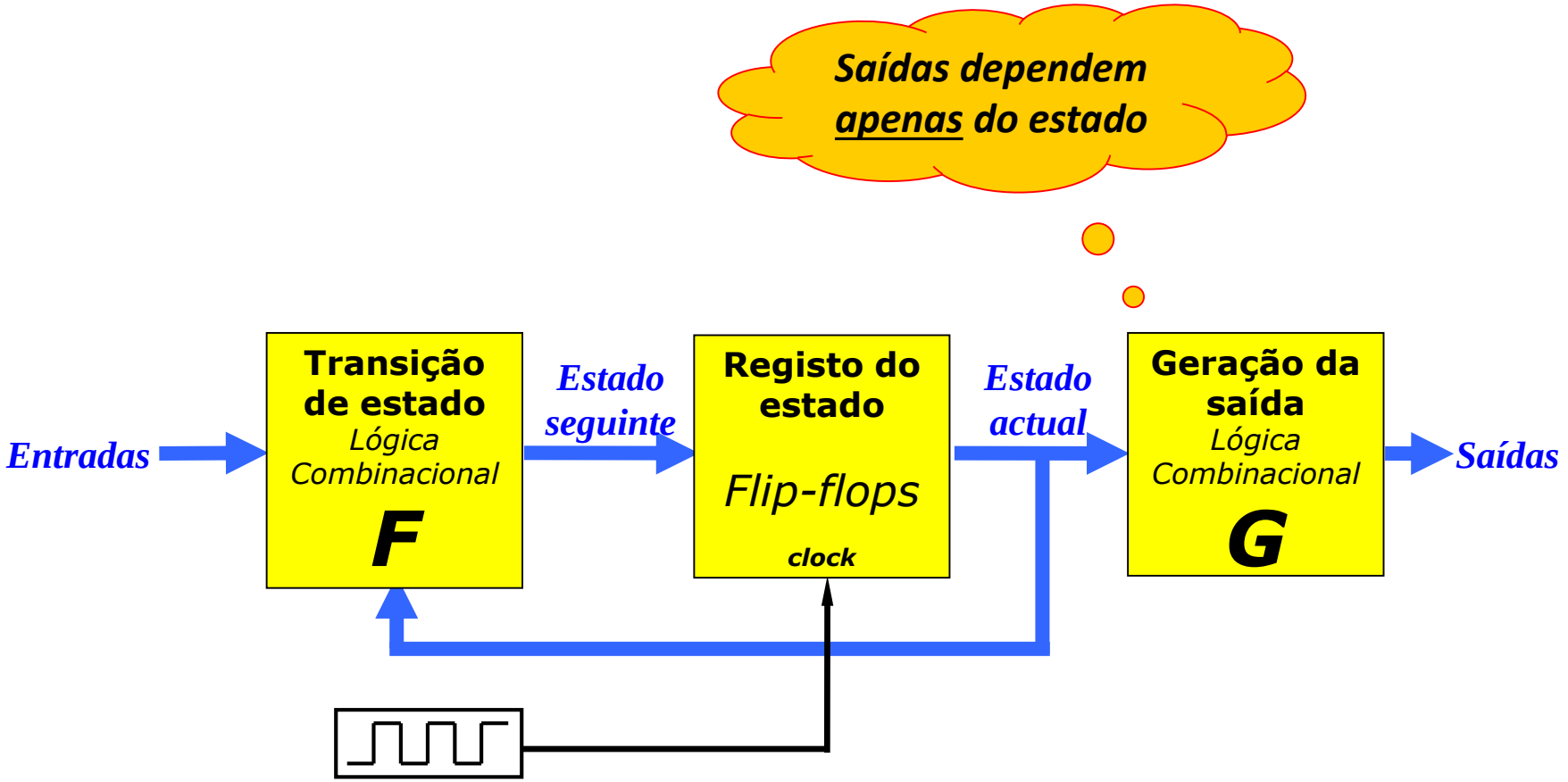


- Modelos estruturais: *Mealy*, ***Moore***

MEF: modelo de *Mealy*

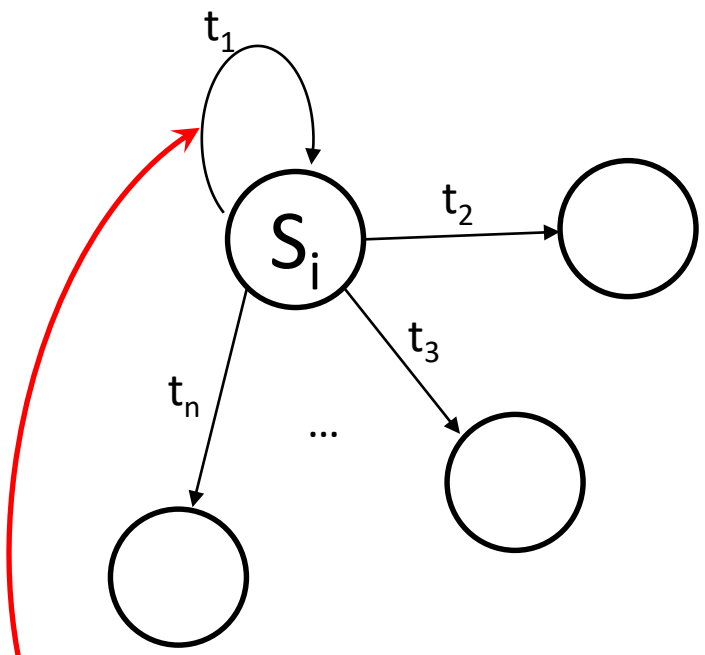


MEF: modelo de *Moore*



Diagramas de Estados

Estado genérico S_i



Transição => condição

Função lógica das entradas

$t_i = t_i(X)$

Transições partindo de S_i:

Sempre uma activa

$t_1 + t_2 + \ldots + t_n = 1$

(e **uma só**)

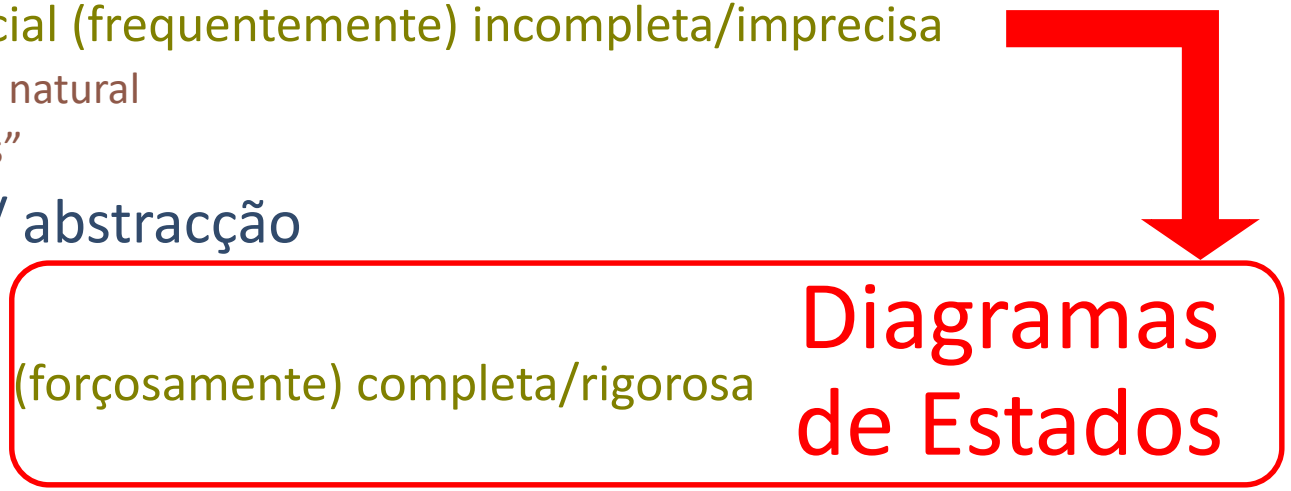
$t_i \cdot t_k = 0 \qquad \forall\, i, k$

Flanco de CLK => transição (sempre!)

Mas transição pode ligar ao mesmo estado!
(por vezes omitida nesse caso)

Especificação inicial → MEF

- Problema concreto
  - Descrição inicial (frequentemente) incompleta/imprecisa
    - linguagem natural
    - "use-cases"
- Formalização / abstracção
  - Especificação (forçosamente) completa/rigorosa



- Modelação
  - Manual
  - Automática

em VHDL

- Simulação (testbench)
  - Revisão / correcção / optimização
- Implementação (FPGA)
  - Verificação
  - Solução pode ser não única / sub-óptima

Formalização (esp. inicial → DDE)

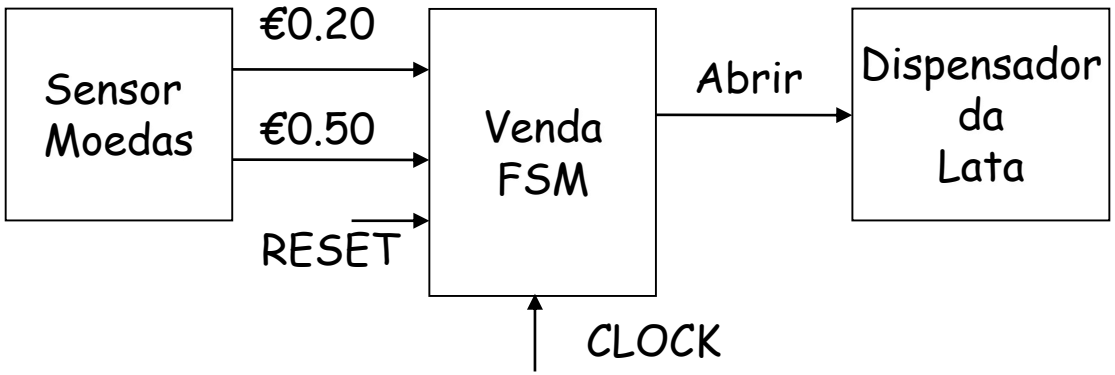
Problema principal (aspecto mais importante)

- Esclarecer ao máximo a especificação inicial
- Identificar as entradas e as saídas (E/S)
- Identificar o comportamento E/S
- Definir estados
  - Estabelecer o significado de cada estado
- Analisar "use cases"

Seguem-se exemplos...

Exemplo 1 - Especificação

- Máquina de venda de bebidas
  - Requisitos gerais:
    - entrega lata de cerveja (sem álcool ☺) após receber € 0.60
    - uma única entrada para moedas de € 0.20 e € 0.50
    - não dá troco
  - Passo 1: perceber o problema (fazer um desenho / diagrama de blocos!...)

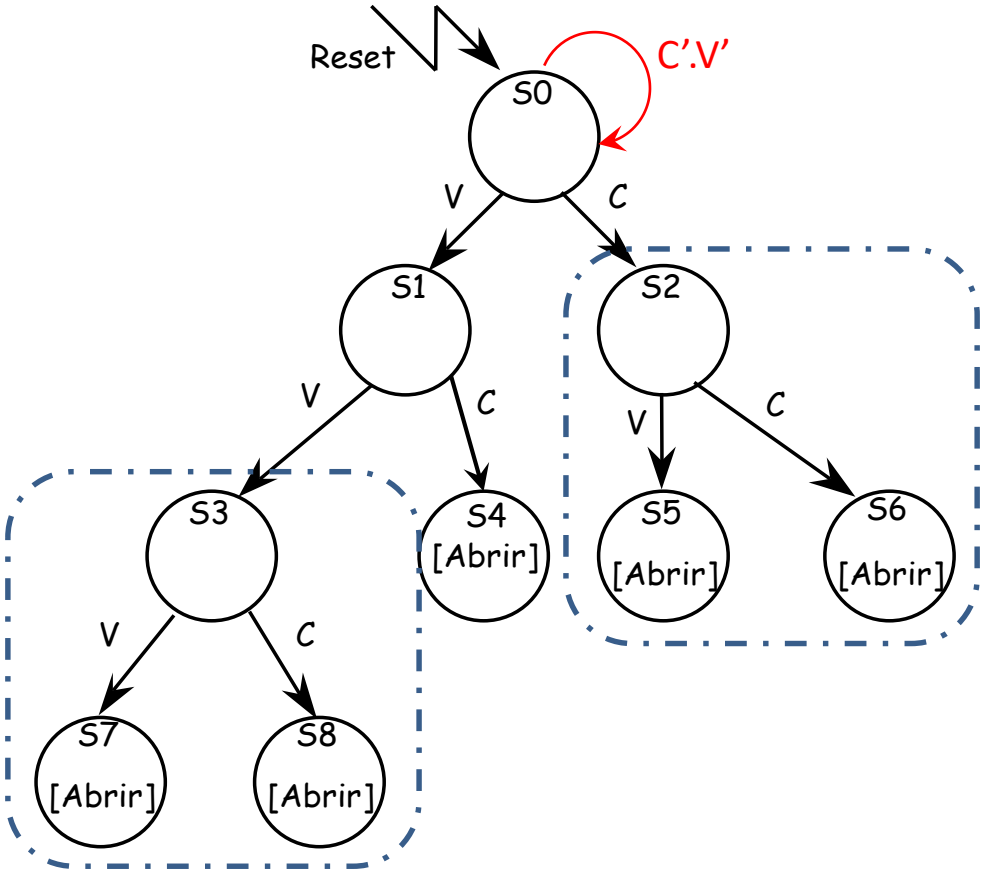


Exemplo 1 – Análise de Requisitos

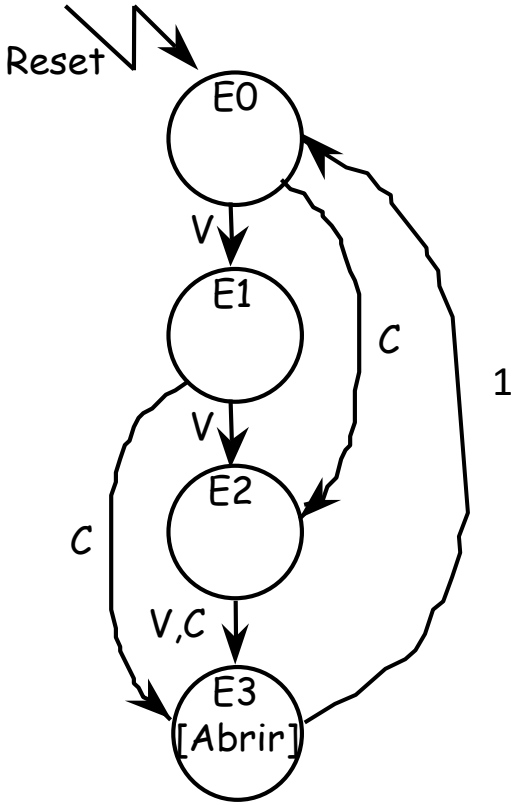
- Identificar as sequências de entradas que levam diretamente à abertura
  - 3 moedas de €0.20
  - 1 moeda de €0.20 + 1 moeda de €0.50
  - 1 moeda de €0.50 + 1 moeda de €0.20
  - 2 moedas de €0.50
  - 2 moedas de €0.20 + 1 moeda de €0.50
- Identificar entradas e saídas:
  - Entradas:
    - V (sensor ativo para €0.20)
    - C (sensor ativo para €0.50)
  - Saída
    - Abrir

Exemplo 1 – Diagrama de Estados

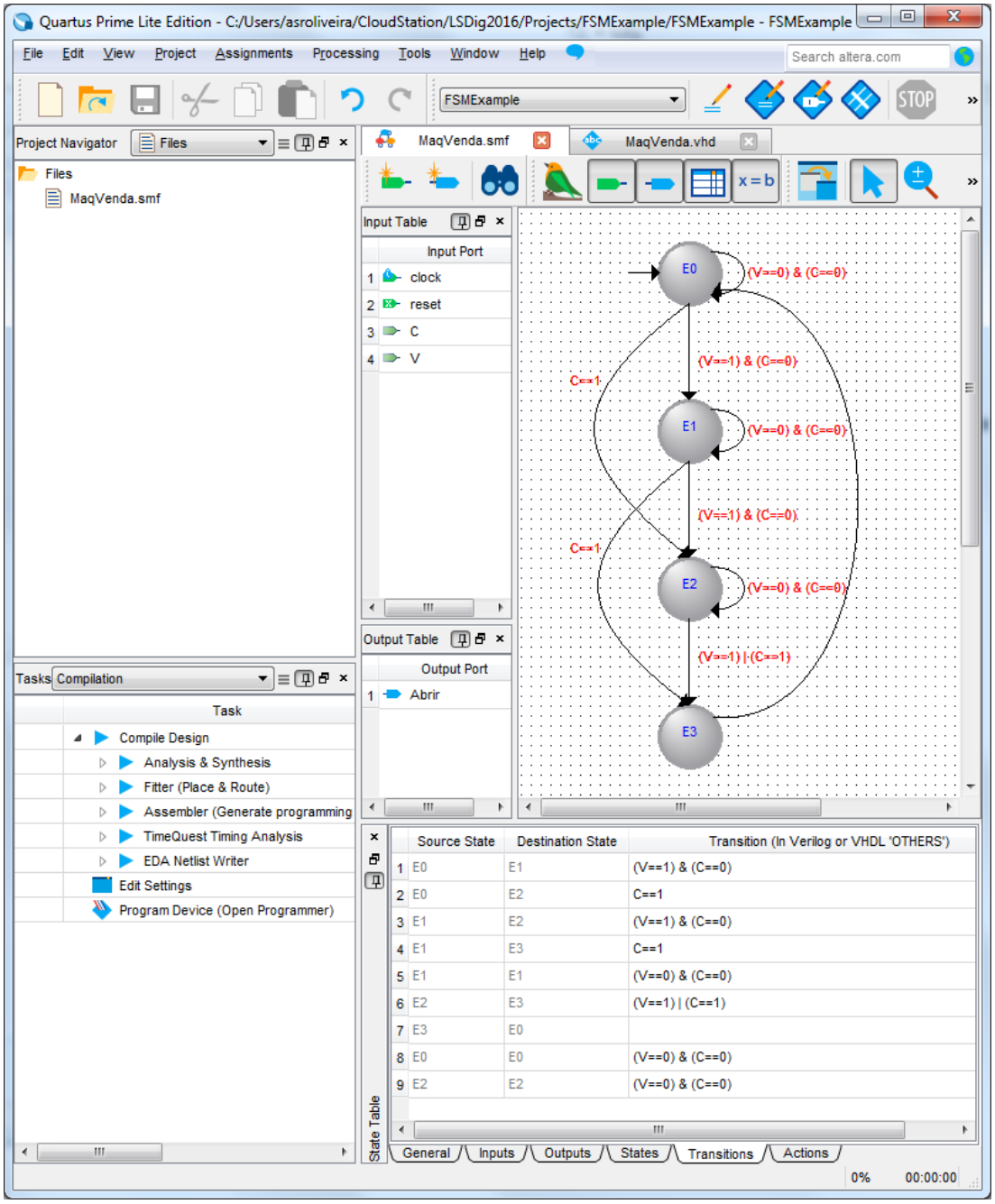
- Diagrama de estados primário (inicial)



- Diagrama de estados correto com reutilização de estados



Modelação com o Editor de FSMs do Quartus Prime

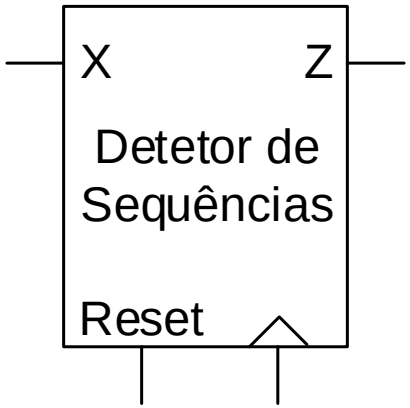


- As **condições** das transições **<u>têm de ser mutuamente exclusivas</u>** (boa prática mesmo nos diagramas construídos com “papel e lápis”)
- As **condições não especificadas** correspondem à **manutenção do estado**
- Geração de VHDL a partir do ficheiro SMF

TPC: analise o código VHDL gerado automaticamente (disponibilizado no site)

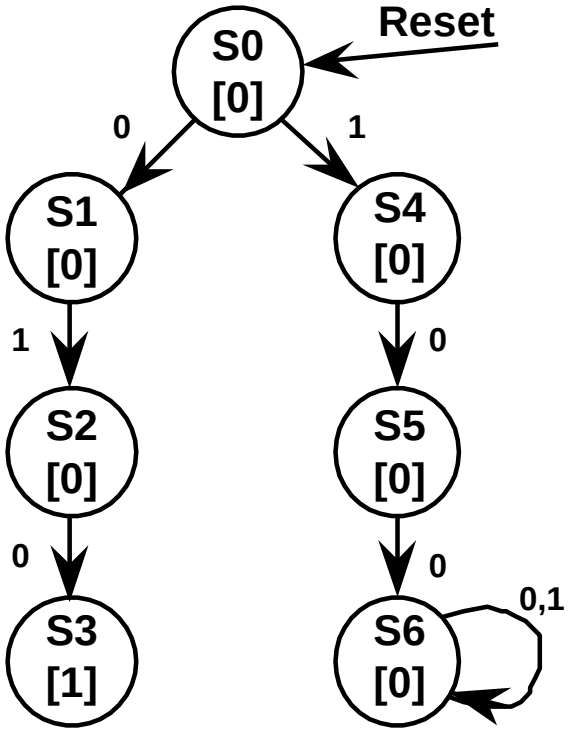
Exemplo 2: Detector de Sequências

- Reconhecimento de padrões em frases de comprimento finito. Exemplo:
  - Um reconhecedor de frases finitas tem uma entrada (X) e uma saída (Z). A saída é activada sempre que a sequência de entrada …010…é observada, desde que a sequência 100 nunca tenha surgido.
  - Exemplo do comportamento entrada/saída:
    - X: 011**010**00~~010~~…
    - Z: 00000**1**00000…
    - X: 0**0101010**~~010~~…
    - Z: 000**1**0**1**0**1**000…



Detector de Sequências: abordagem à construção do DDE

Começar pelos padrões que devem ser reconhecidos: 010 e 100.

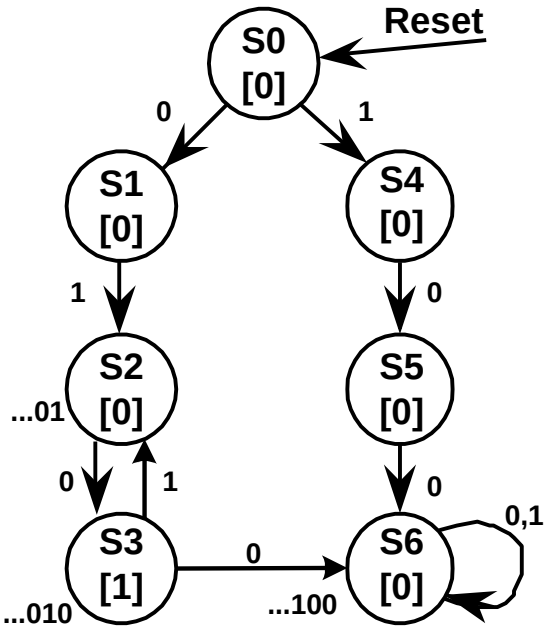


Modelo de Moore

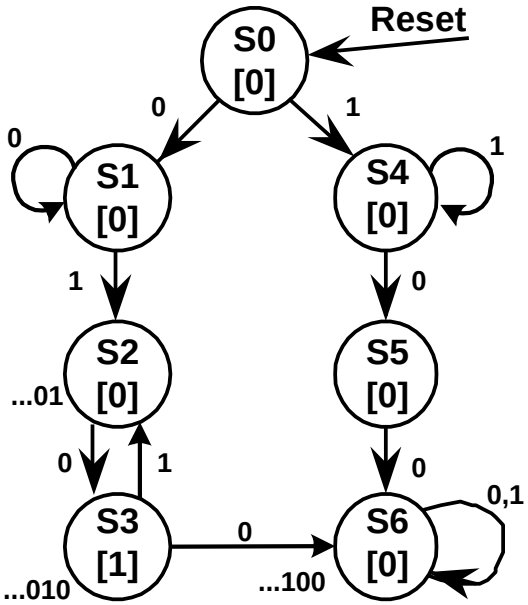
Reset conduz ao estado S0

Detector de Sequências: finalização do DDE

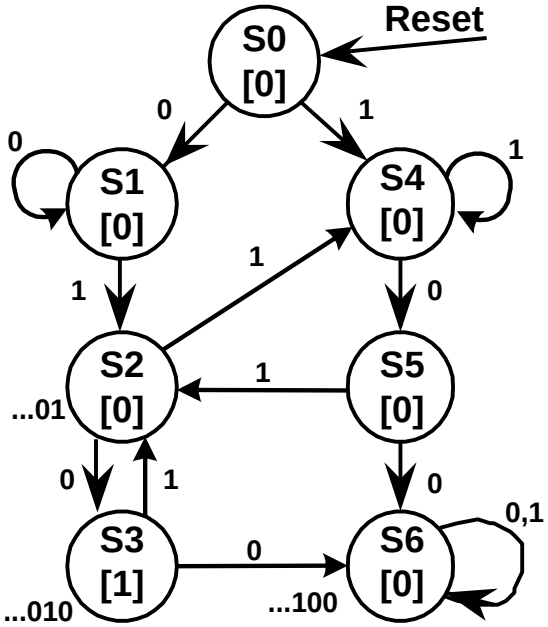
Completar analisando as condições de transição de cada estado



Transições em S3



Transições em S1 e S4



Transições em S2 e S5

Detector de Sequências

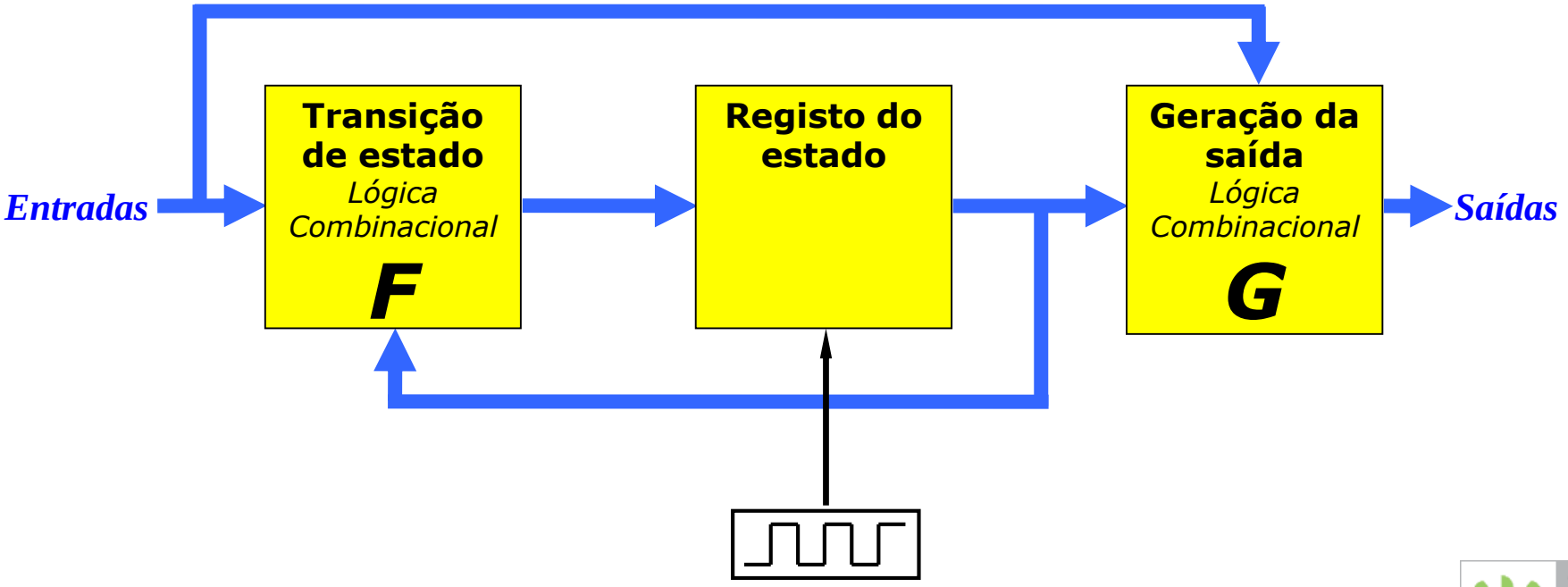
Revisão do procedimento

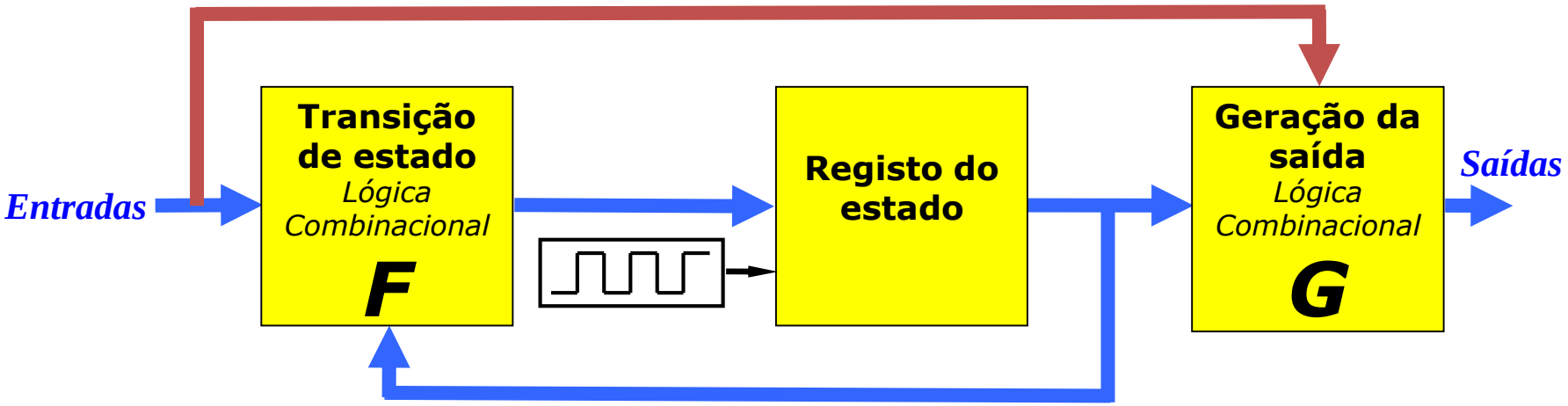
1. Escrever sequências de teste com as entradas/saídas para perceber a especificação
2. Criar uma sequência de estados e transições para as sequências que se pretende ver reconhecidas
3. Acrescentar transições em falta; reutilizar o mais possível os estados existentes
4. Verificar o comportamento E/S do diagrama de estados para assegurar que funciona como pretendido

Modelação de MEF em VHDL

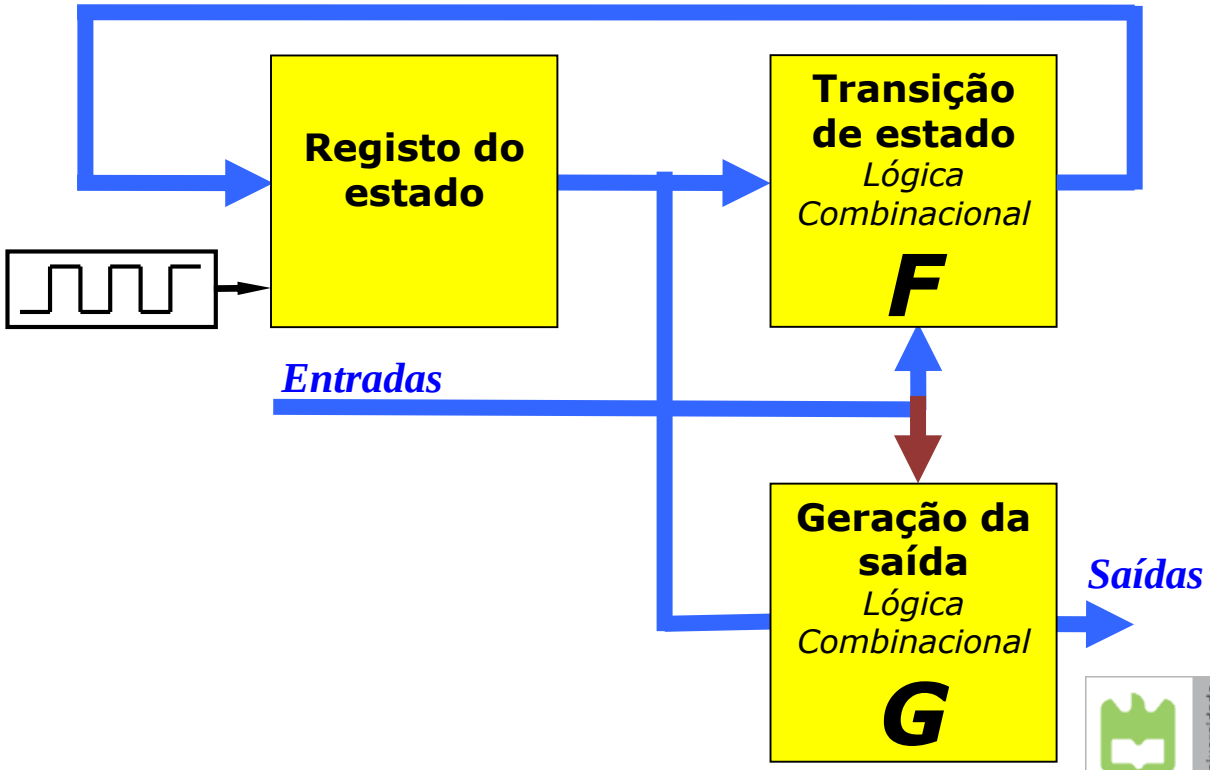
Abordagem estrutural

- Registo
  - Codificação de estados
- Bloco combinacional de transição de estado
  - Equações lógicas
- Bloco combinacional de geração das saídas
  - Equações lógicas



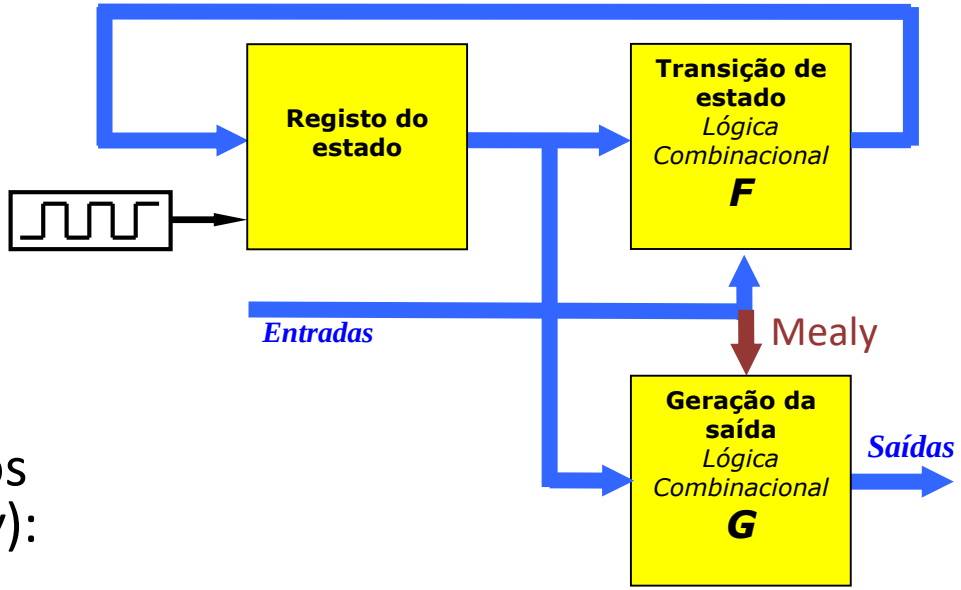


Descubra as diferenças…

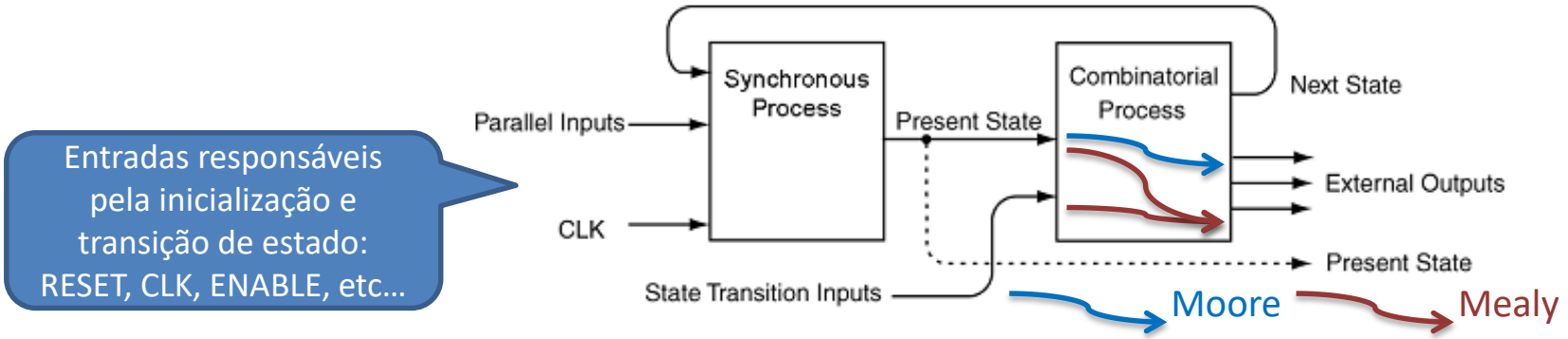


MEF em VHDL comportamental: método '2 processos'

Modelo de MEF revisitado:

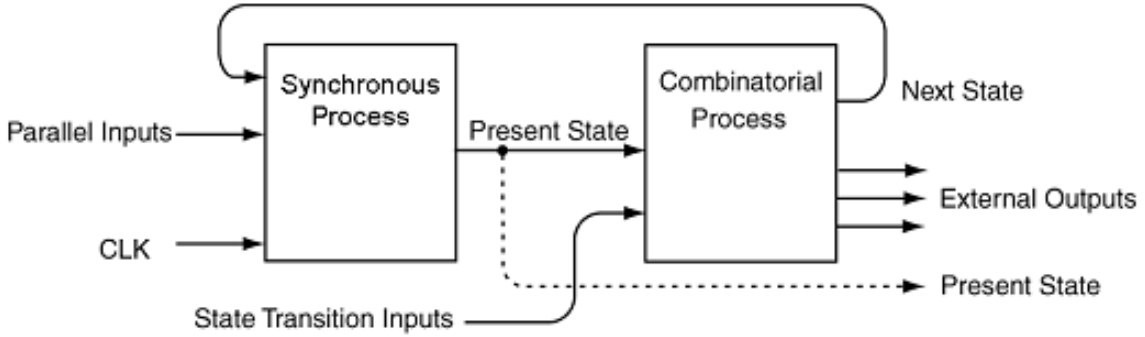


Reorganização em dois processos (compatível com *Moore* e *Mealy*):



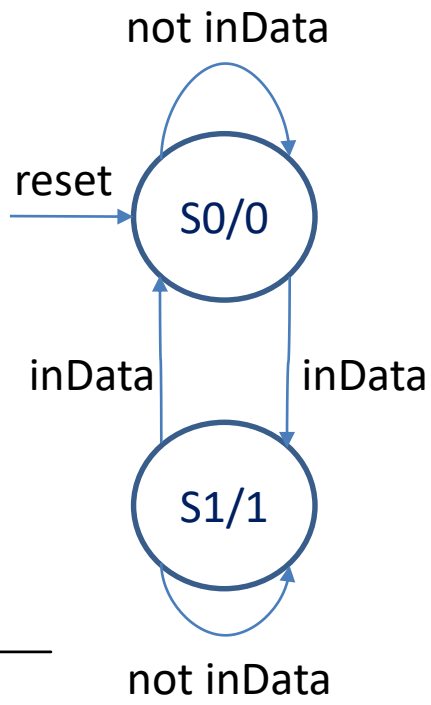
MEF em VHDL comportamental: método '2 processos'

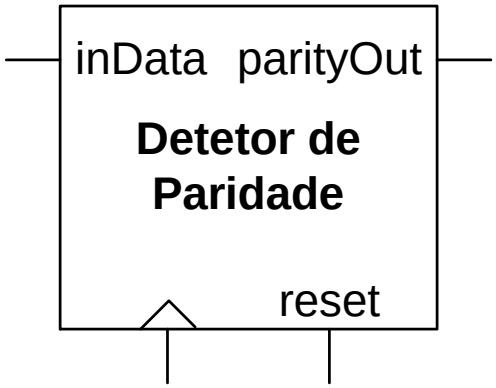
- Processo *síncrono*
  - Atribuições dependentes dum evento de relógio e/ou de atribuição/inicialização assíncrona do registo
- Processo *combinacional*
  - Atribuições relacionadas com a determinação de
    - Saídas
    - Estado seguinte
- Os 2 processos são interdependentes



Exemplo simples (segundo *Moore*): Detector de Paridade

- Detetor de paridade
  - Detecta se o número de '1' ocorridos numa série de *bits* é ímpar (ou par)
  - Entradas
    - **`clk`**
    - **`reset (síncrono)`**
    - **`inData`**
  - Saída
    - **`parityOut`**

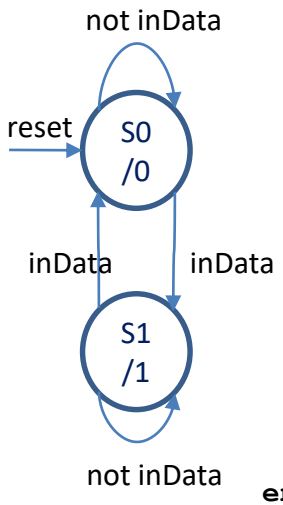




Detector de paridade: código ‘2 processos’

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity ParityDetector is
   port(reset     : in  std_logic;
        clk       : in  std_logic;
        inData    : in  std_logic;
        parityOut : out std_logic);
end ParityDetector;

architecture Behavioral of ParityDetector is

   type TState is (S0,S1);
   signal pState, nState: TState;

begin
   sync_proc : process(clk)
   begin
      if (rising_edge(clk)) then
         if (reset = '1') then
            pState <= S0;
         else
            pState <= nState;
         end if;
      end if;
   end process;
```

Criação dum novo tipo de dados “enumerado”

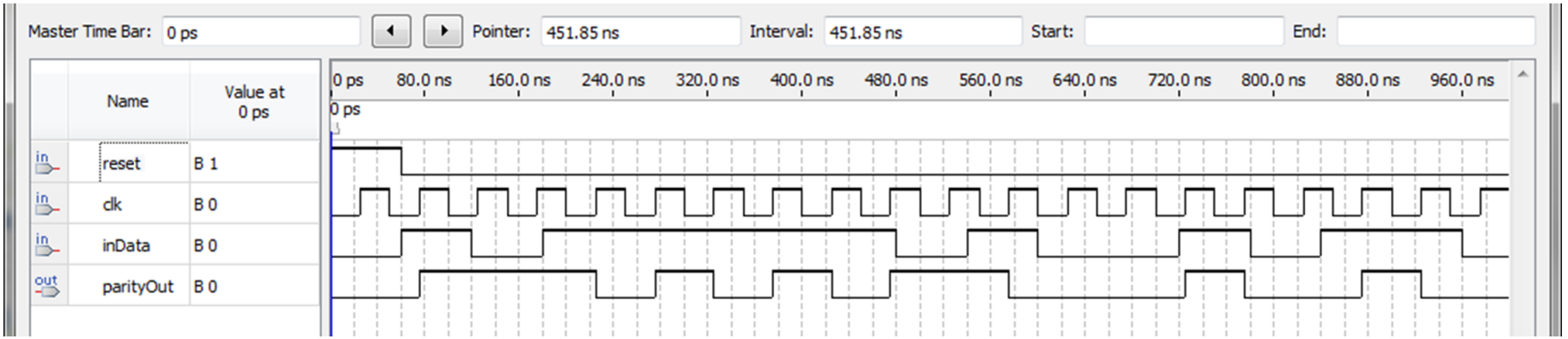



```
comb_proc : process(pState, inData)
begin
   case pState is
      when S0 =>
         parityOut <= '0'; -- Moore output
         if (inData = '1') then
            nState <= S1;
         else
            nState <= S0;
         end if;
      when S1 =>
         parityOut <= '1'; -- Moore output
         if (inData = '1') then
            nState <= S0;
         else
            nState <= S1;
         end if;
      when others => -- “Catch all” condition
         nState    <= S0;
         parityOut <= '0';
   end case;
end process;
end Behavioral;
```

Tradução direta do Diagrama de Estados

- **Qual é a variável de estado?**

Detector de paridade: simulação



MEF é a UUT

Testbench gerada:

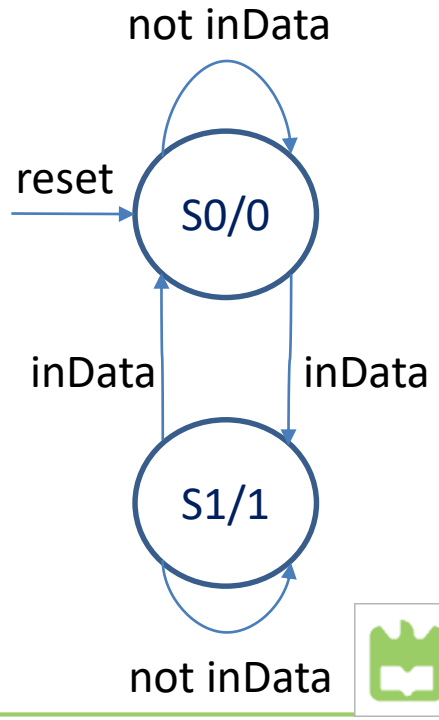
-a partir do ficheiro VWF

ou

-em VHDL segundo o *template* para UUT sequencial:

1 processo para geração do sinal de relógio

1 processo para inicialização e geração da entrada



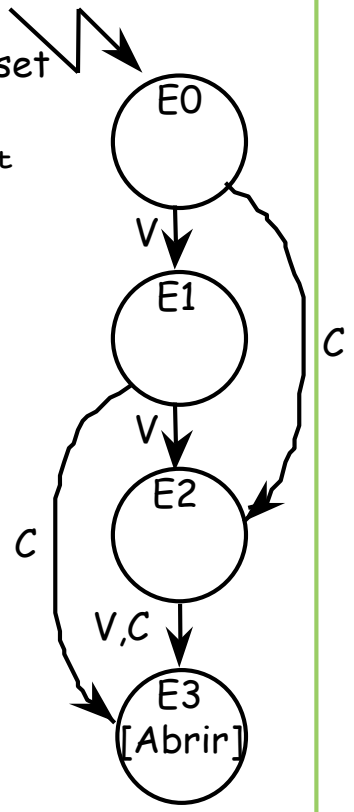
Muito importante:

Não comutar as entradas (incluindo *reset*) na vizinhança das transições activas do sinal de relógio (reflectir na simulação o cumprimento dos tempos de *setup* e de *hold*)

Máquina de Vendas

```
entity DrinksFSM is
          port(reset : in  std_logic;
          clk        : in std_logic;
          v          : in  std_logic;
          c          : in  std_logic;
          abrir      : out std_logic);
end DrinksFSM;

architecture Behavioral of DrinksFSM is

type TState is (E0, E1, E2, E3);
signal s_currentState, s_nextState : TState;
begin
sync_proc : process(clk)
begin
if (rising_edge(clk)) then
    if (reset = '1') then
          s_currentState <=E0;
    else
          s_currentState <= s_nextState;
     end if;
end if;
end process;
```

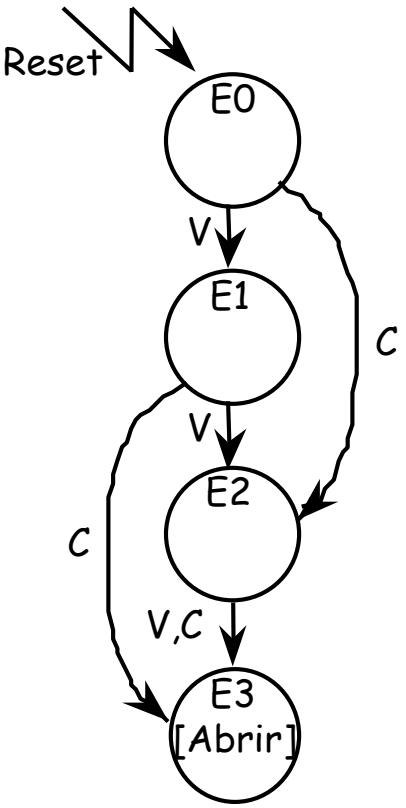
```
comb_proc : process(s_currentState, v, c)
begin
 case (s_currentState) is

when E0 =>
     abrir <= '0';  --Moore Output
     if (v = '1') then
        s_nextState <= E1;
     elsif (c = '1') then
       s_nextState <= E2;
     else
       s_nextState <= E0;
     end if;

  when E1 =>
     abrir <= '0';
      if (v = '1') then
        s_nextState <= E2;
      elsif (c = '1') then
        s_nextState <= E3;
      else
        s_nextState <= E1;
end if;
```



Tradução direta do Diagrama de Estados

Máquina de Vendas

```
  when E2 =>
    abrir <= '0';

    if (v = '1') or (c = '1')  then

           s_nextState <= E3;
    else
           s_nextState <= E2;
    end if;

   when E3 =>

     drink <= '1';

     s_nextState <= E0;

   when others =>
       abrir <= '0';
       report "Reach undefined state";

   end case;

end process;

end Behavioral;
```



Síntese de MEF

- Realizada pelo compilador / ferramenta de síntese
  - Codificação de estados (com base nos estados simbólicos)
  - Geração do registo de estado (com o número de *bits* necessários em função da codificação de estados)
  - Determinação e optimização dos circuitos combinacionais de estado seguinte e das saídas

Codificação dos Estados

Recurso a tipo enumerado:

- Contribui para descrição de alto nível, próxima do diagrama de estados
- Deixa ao sintetizador a tarefa de codificar os estados
  - Conduz frequentemente a *flip-flops* em número superior ao mínimo (codificação binária ou *gray*)

Codificação *"one-hot" (*frequente)

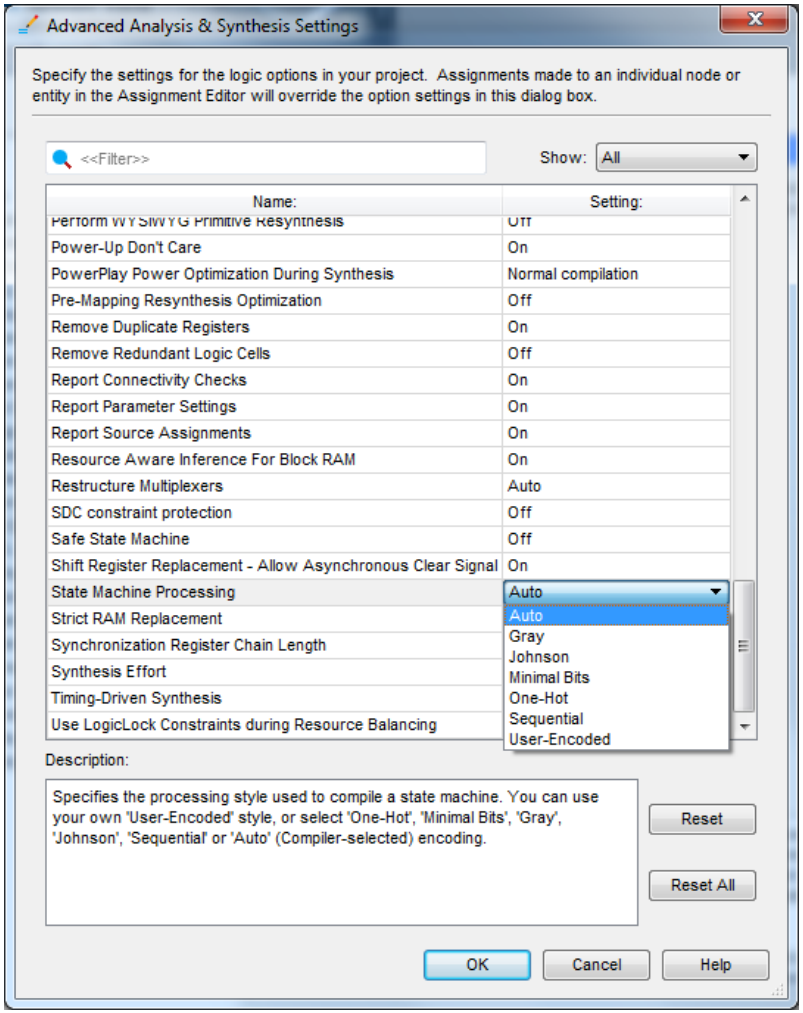
- Exemplo: 4 estados => 4 FF (000**1**, 00**1**0, 0**1**00, **1**000)
- Adequada à implementação em FPGA
  - Muitos *flip-flops* disponíveis
  - Mas LUT com poucas entradas (tipicamente 4 a 6)
- Lógica de estado seguinte e de saída tendencialmente mais simples (com menos níveis e mais rápida)

Codificação dos Estados

Quartus Prime

permite "forçar"

técnica de codificação



"Assignments → Settings → Compiler Settings → Advanced Settings (Synthesis)"

Comentários Finais

- No final desta aula e do trabalho prático 8, deverá ser capaz de:
  - Construir diagramas de estados com base na especificação de uma MEF
  - Conhecer os passos necessários à síntese de MEF
  - Usar descrições comportamentais em VHDL 'próximas' da do diagrama de estados
    - Modelo de *Moore*
  - Conceber *testbenches* para a simulação funcional de MEF
  - Sintetizar, implementar em FPGA e testar MEF
- … bom trabalho prático 8, disponível no site da UC
  - elearning.ua.pt