Laboratório de Sistemas Digitais

Aula Teórico-Prática 8

Ano Letivo 2022/23

Modelação, simulação e síntese de Máquinas de Estados Finitos

Aspetos gerais e

Modelo de *Moore*

Conteúdo

- Processo de síntese de Máquinas de Estados Finitos (MEFs) / *Finite State Machines* (FSMs)
- *Workflow* da síntese e implementação
- Construção do diagrama de estados a partir da especificação
  - Manualmente ou com o editor do Quartus Prime
- Modelação de FSMs baseada em VHDL
  - Modelo de *Moore*
  - Abordagem baseada em 2 processos
- Simulação de FSMs
- Síntese de FSMs
  - Codificação de estados

Síntese de FSMs (unidades de controlo e circuitos sequenciais em geral)

- Passagem da especificação à implementação
  - Conceção de uma solução para um problema concreto (muitas vezes) inicialmente descrito em linguagem natural ou na forma de “use-cases”
  - Formalização/abstração segundo o modelo de Máquina de Estados Finitos (MEF/FSM)
  - Processo composto por diferentes passos de modelação, otimização e geração de hardware
  - Solução frequentemente
    - Não única
    - Sub-ótima

Instrumentos/Ferramentas de Modelação e Síntese de FSMs

- **Diagramas de Estado**
- Cartas ASM: *Algorithmic State Machines*
- Tabelas de Estado/Saídas
- Tabelas de Transição/Saídas
- Tabelas de Excitação
- Diagramas temporais
- **Linguagens de Descrição de Hardware**
  - **VHDL**, Verilog, ...
- ...

Workflow de Síntese e Implementação de FSMs

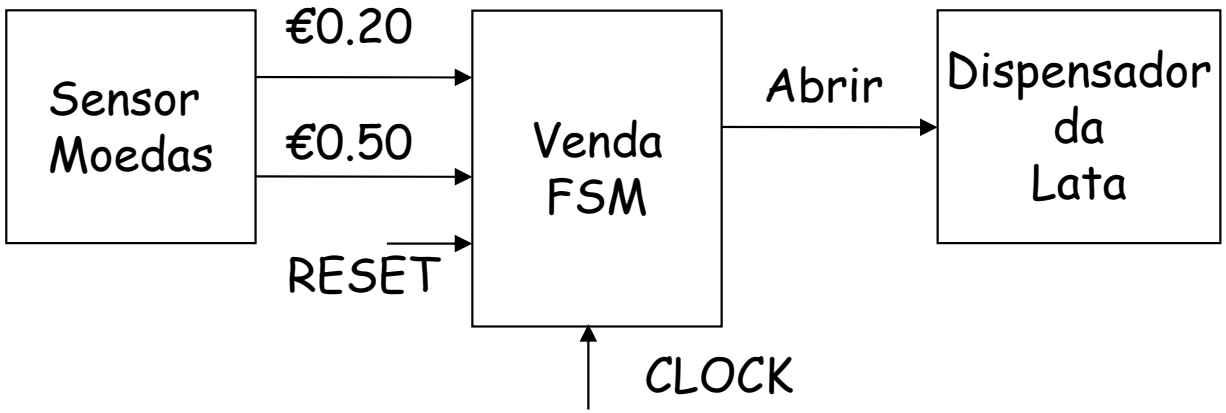
- Entender a especificação inicial
- Obter uma representação abstrata da FSM
  - Diagrama de Estados/Saídas, Tabelas de Estado/Saídas, …
- Modelação em VHDL a partir do diagrama de estados
  - Manualmente ou automaticamente
- Simulação funcional da FSM
  - Com uma *testbench*
- Síntese lógica e implementação em hardware (em FPGA no contexto de LSD)
- Verificação do comportamento da FSM em hardware

FSMs: Especificação → Formalização

- Problema principal (aspeto mais importante)
  - Como passar da especificação ao diagrama de estados?
    - Qual do significado de cada estado?
- Especificação deve
  - ser tão precisa quanto possível …
  - identificar as entradas e as saídas (E/S)
  - identificar o comportamento E/S
  - incluir “use cases”
- Vamos analisar alguns exemplos...

Exemplo 1 - Especificação

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



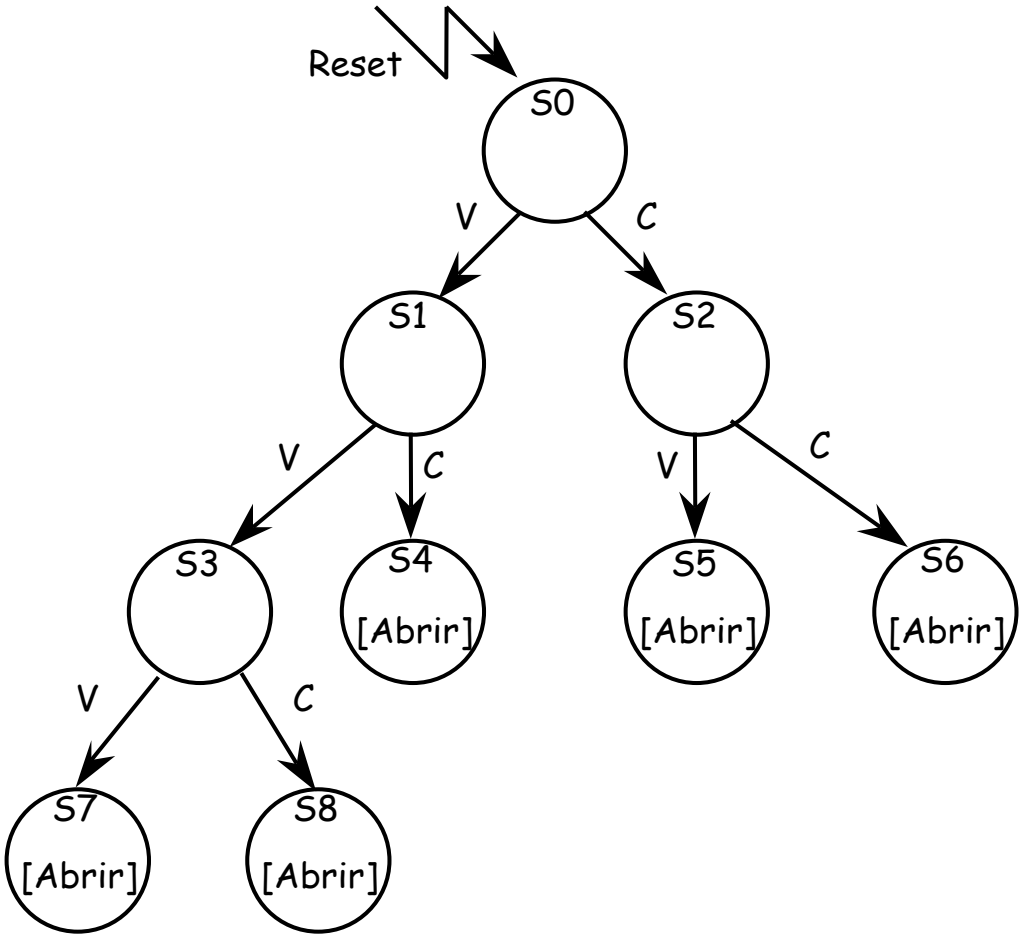
Exemplo 1 – Análise de Requisitos

- Começar por 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

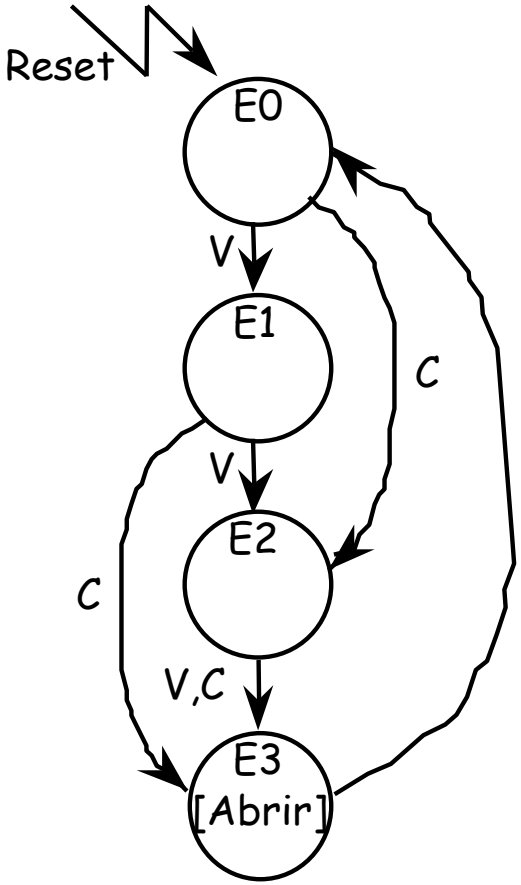
- Identificar entradas de saídas:
  - Entradas:
    - V (sensor ativo para €0.20)
    - C (sensor ativo para €0.50)
  - Saída
    - Abrir

Exemplo 1 – Diagrama de Estados

- Diagrama de estados primário (inicial)

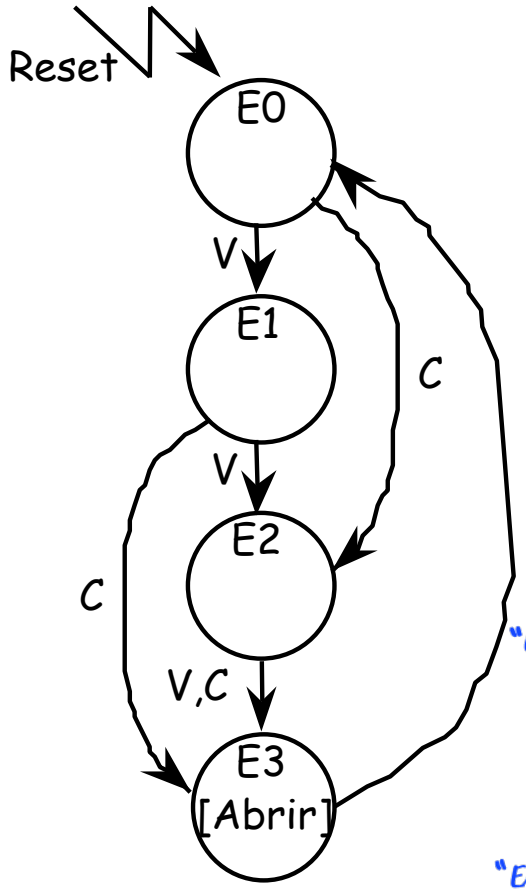


- Diagrama de estados correto com reutilização de estados



Exemplo 1 – Tabela de Estados

- Tabela de Estados/Saídas decorre diretamente do DE

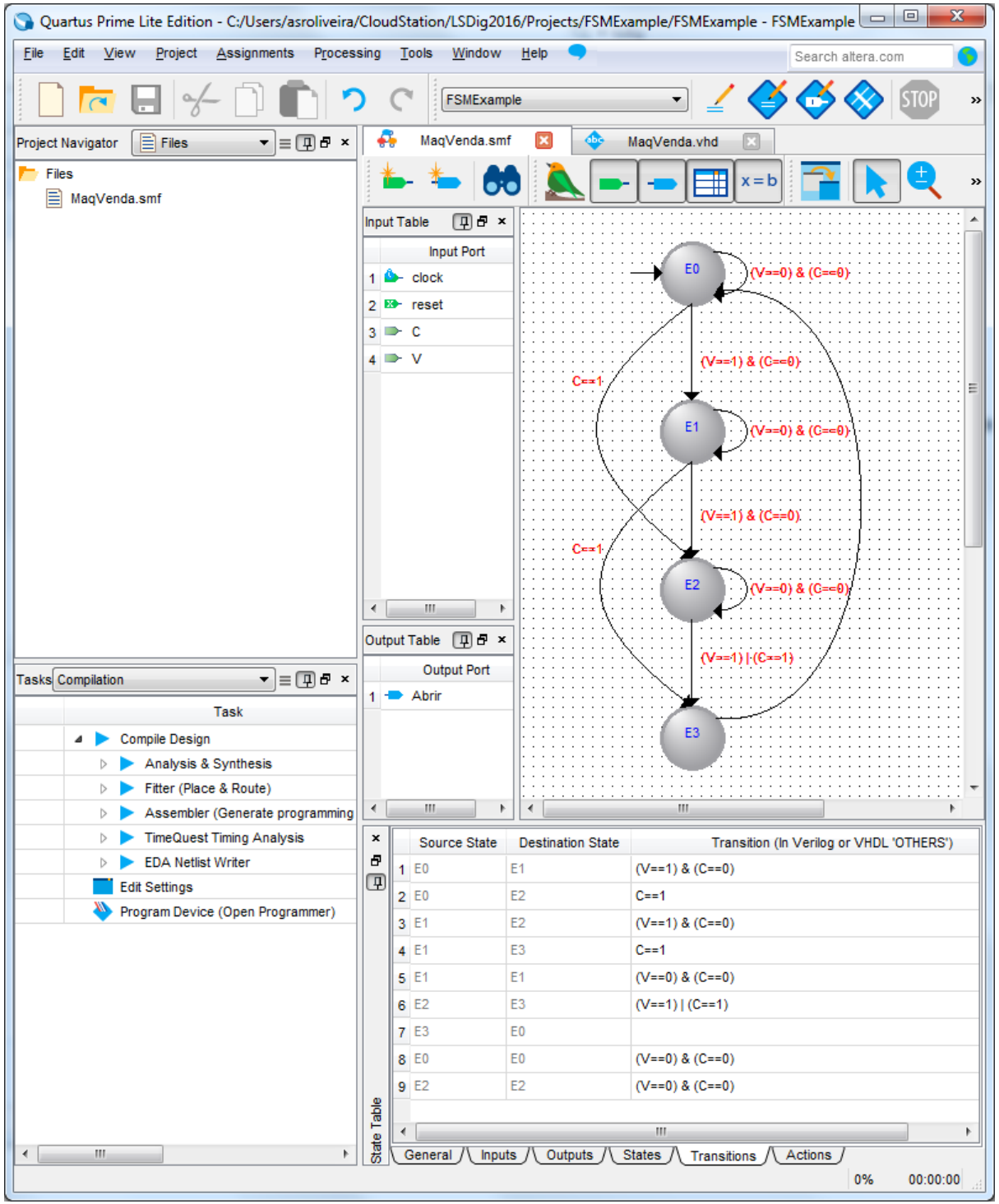


Não podemos utilizar "open" pois é reservada em VHDL

PState	Inputs		NState	Output
	V (0,2€)	C (0,5€)		Abrir
E0	0	0	E0	0
"Estado inicial"	0	1	E2	0
	1	0	E1	0
	1	1	X (Inválido)	X
E1	0	0	E1	0
"Estado de 0,2€"	0	1	E3	0
	1	0	E2	0
	1	1	X	X
E2	0	0	E2	0
"Estado de 0,4 ou 0,5€" Falta 1 moeda	0	1	E3	0
	1	0	E3	0
	1	1	X	X
E3	X	X	E0	1
"Estado de entrega"				

Não coloca moeda

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

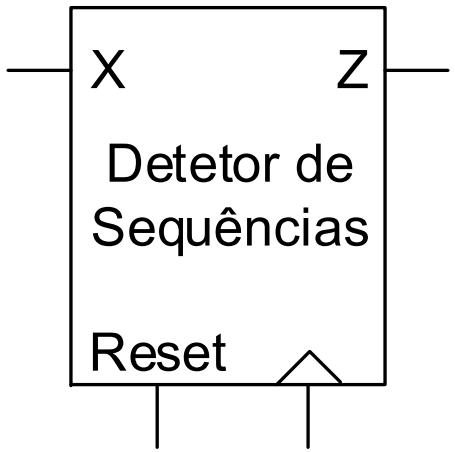
TPC: analise o código VHDL gerado automaticamente (disponibilizado no site)

Aqui o reset é síncrono

```
PROCESS (fstate,reset,C,V)
BEGIN
    IF (reset='1') THEN
        reg_fstate <= E0;
        Abrir <= '0';
    ELSE
        Abrir <= '0';
        CASE fstate IS
            WHEN E0 =>
                IF (((V = '1') AND (C = '0'))) THEN
                    reg_fstate <= E1;
                ELSIF ((C = '1')) THEN
                    reg_fstate <= E2;
                ELSIF (((V = '0') AND (C = '0'))) THEN
                    reg_fstate <= E0;
                -- Inserting 'else' block to prevent latch inference
                ELSE
                    reg_fstate <= E0;
                END IF;

                Abrir <= '0';
            WHEN E1 =>
                IF (((V = '1') AND (C = '0'))) THEN
                    reg_fstate <= E2;
                ELSIF ((C = '1')) THEN
                    reg_fstate <= E3;
                ELSIF (((V = '0') AND (C = '0'))) THEN
                    reg_fstate <= E1;
                -- Inserting 'else' block to prevent latch inference
                ELSE
                    reg_fstate <= E1;
                END IF;

                Abrir <= '0';
            WHEN E2 =>
                IF (((V = '1') OR (C = '1'))) THEN
                    reg_fstate <= E3;
                ELSIF (((V = '0') AND (C = '0'))) THEN
                    reg_fstate <= E2;
                -- Inserting 'else' block to prevent latch inference
                ELSE
                    reg_fstate <= E2;
                END IF;

                Abrir <= '0';
            WHEN E3 =>
                reg_fstate <= E0;

                Abrir <= '1';
            WHEN OTHERS =>
                Abrir <= 'X';
                report "Reach undefined state";
        END CASE;
    END IF;
END PROCESS;
```

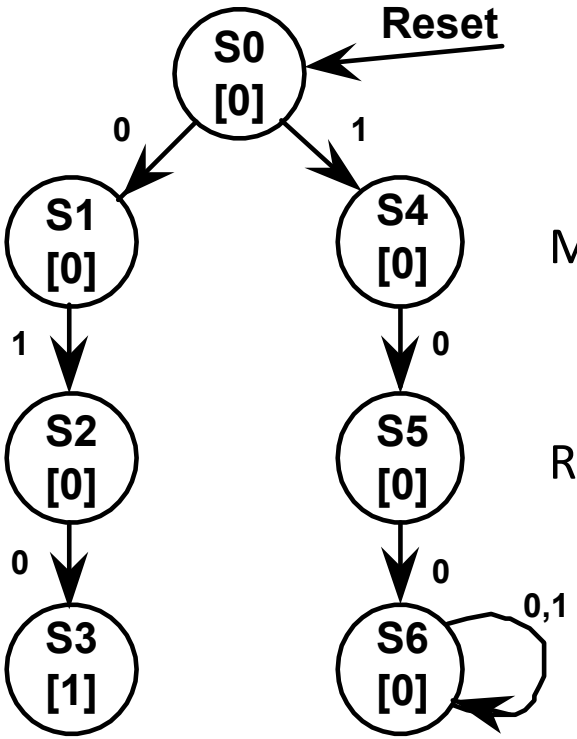
Exemplo 2: Detetor de Sequências

- 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 ativada 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**00010…
    - Z: 00000**1**00000…
    - X: 0**01010100**10…
    - Z: 000**1**0**1**0**1**000…



Detetor de Sequências: Diagrama de Estados

– Desenhar o diagrama de estados para os padrões que devem ser reconhecidos i.e., 010 e 100.

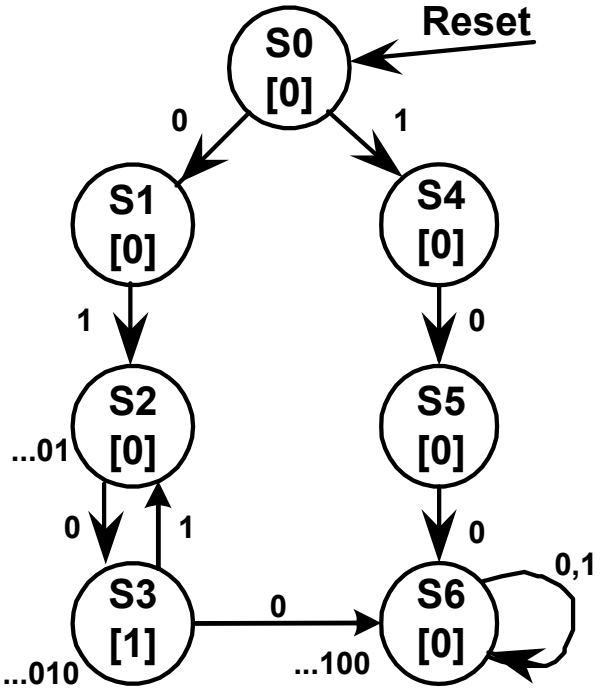


Modelo de Moore

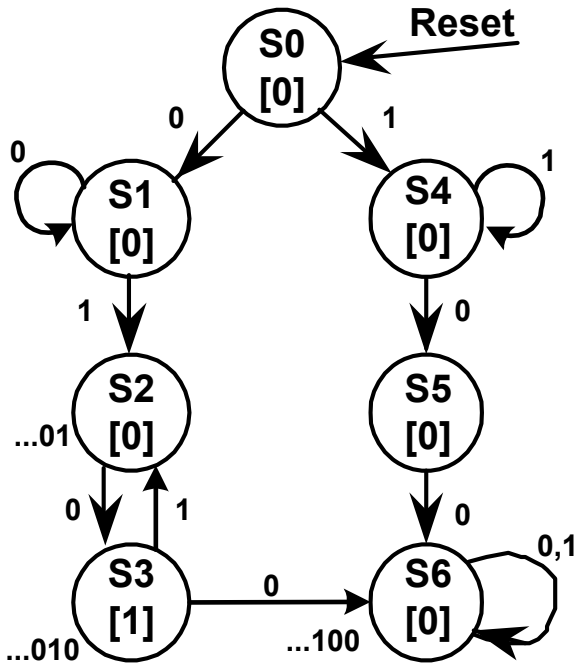
Reset “leva” a máquina para o estado S0

Detetor de Sequências: Diagrama de Estados

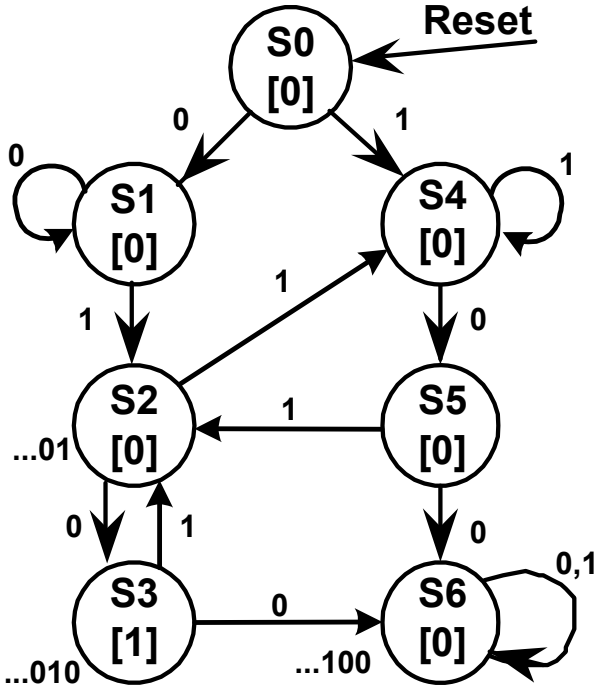
– Completar o diagrama analisando as condições de transição de cada estado



Transições em S3



Transições em S1 e S4



Transições em S2 e S5

Detetor de Sequências: Revisão do Procedimento

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

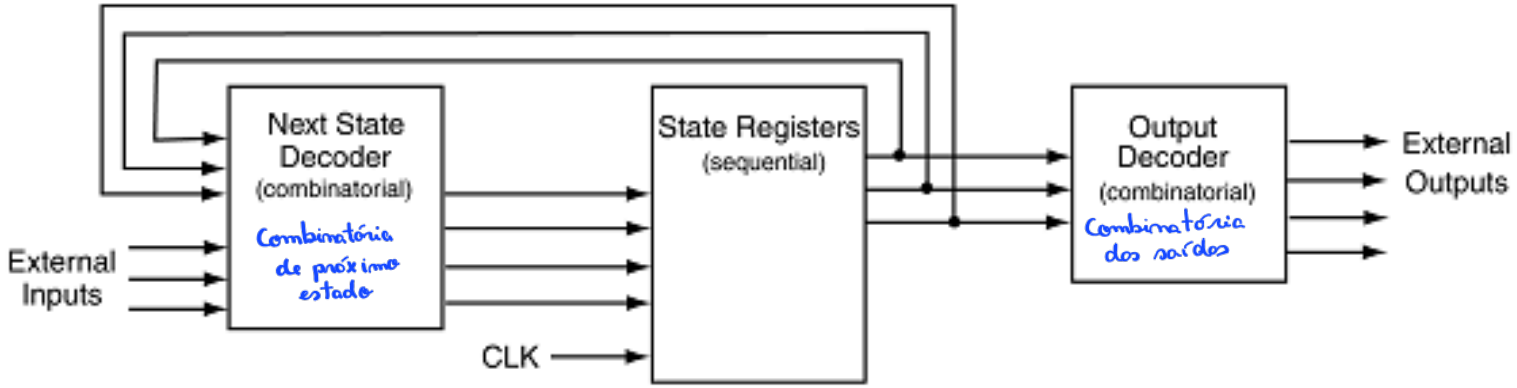
Agora que vimos alguns exemplos de passagem da especificação para o diagrama de estados, vamos ver como descrever eficientemente uma FSM em VHDL…

Modelos Comportamentais Textuais de FSMs

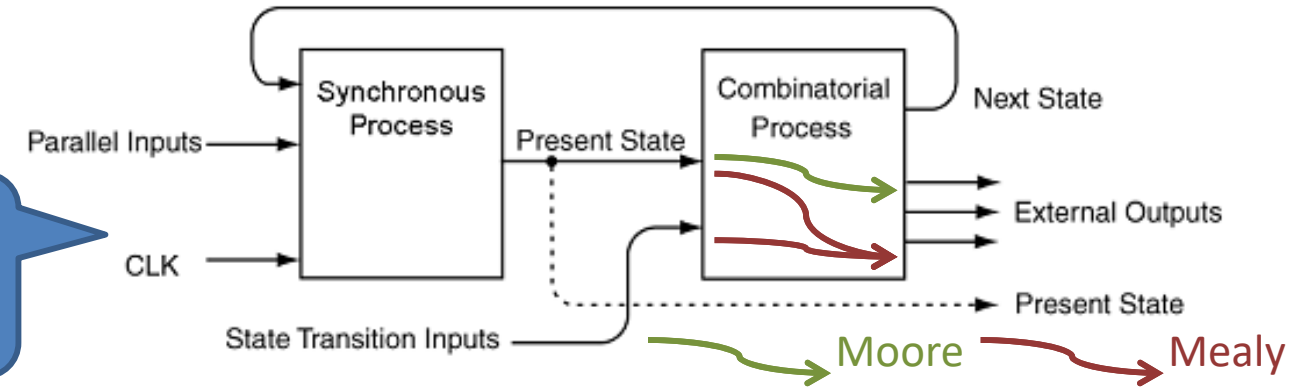
- A linguagem VHDL proporciona descrições de alto nível (comportamentais) de FSMs
  - Muito próximas dos diagrama de estados
  - Vários estilos de escrita possíveis em VHDL (mais frequentemente com 2 ou 3 processos)
  - Tradução direta do diagrama de estados para VHDL no editor de texto do IDE (Quartus Prime em LSD)
  - Por omissão é a ferramenta de síntese (compilador) que determina a codificação dos estados com base em estados simbólicos
  - Não são necessárias equações lógicas para explicitar saídas e o "próximo estado" (equações de excitação)
    - Minimização lógica realizada pelo compilador (ferramenta de síntese)

FSMs em VHDL - O Estilo de Codificação Baseado em "2" Processos

- Modelo de *Moore* revisitado



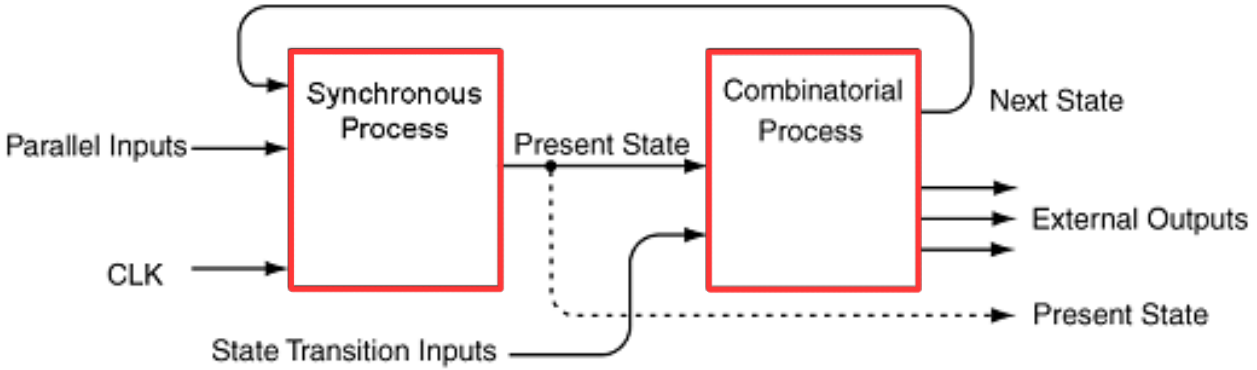
- Podemos reorganizar sinais e portos segundo dois processos (compatível com *Moore* e *Mealy*)



Entradas responsáveis pela inicialização e transição de estado: RESET, CLK, ENABLE, etc…

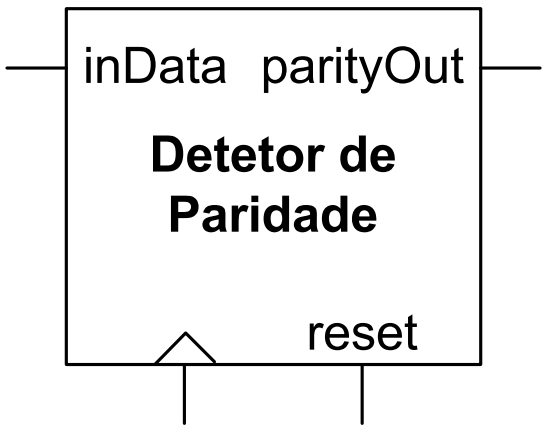
FSMs em VHDL - O Estilo de Codificação Baseado em "2" Processos

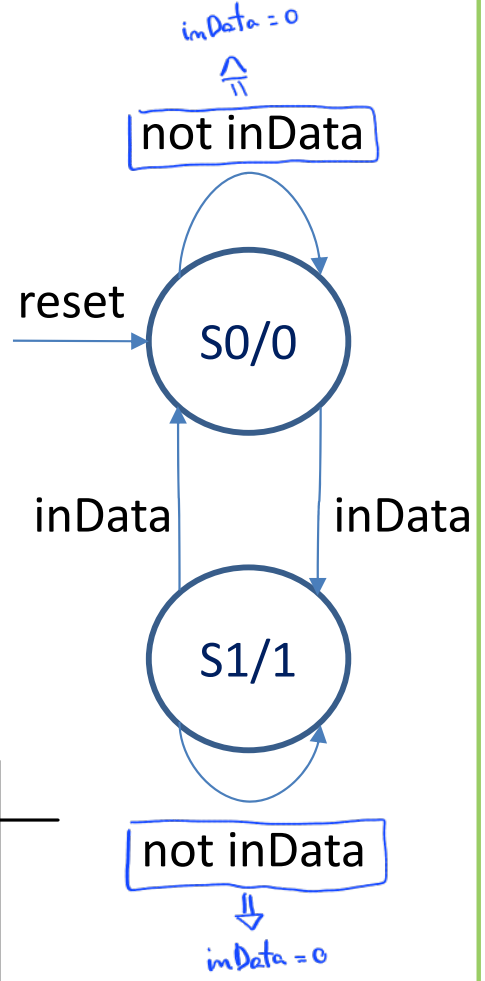
- Processo "*synchronous*" (apenas uma designação)
  - Atribuições dependentes dum evento de relógio e/ou de atribuição/inicialização assíncrona dos elementos de memória
- Processo "*combinatorial*" (apenas uma designação)
  - Atribuições relacionadas com a determinação de
    - Saídas
    - Estado seguinte
- Os 2 processos são interdependentes



Exemplo Trivial (segundo *Moore*) – Detetor de Paridade

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

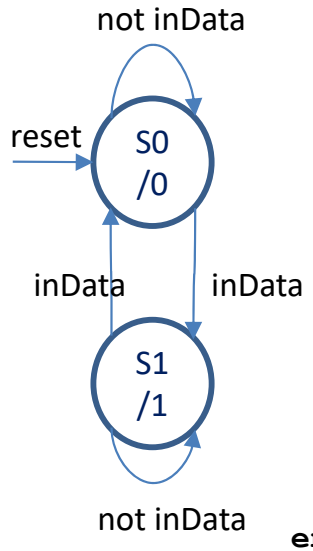




Exemplo Trivial (segundo *Moore*) – Detetor de Paridade

```
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;
          estado atual   próximo estado

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"

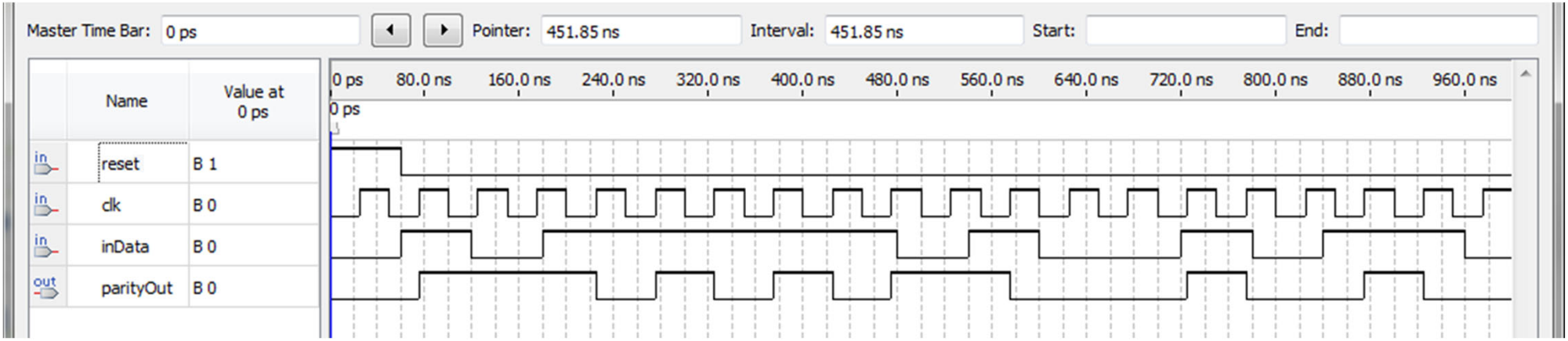



```
comb_proc : process(pState, inData)
begin
   case pState is
      when S0 =>
         parityOut <= '0'; -- Moore output
         if (inData = '1') then
            nState <= S1;
         else
            nState <= S0;
         end if;
      when S1 =>
         parityOut <= '1'; -- Moore output
 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

Apenas alteramos o prox. estado, porc quando vier o rising_edge(clk) ele altera corretamente

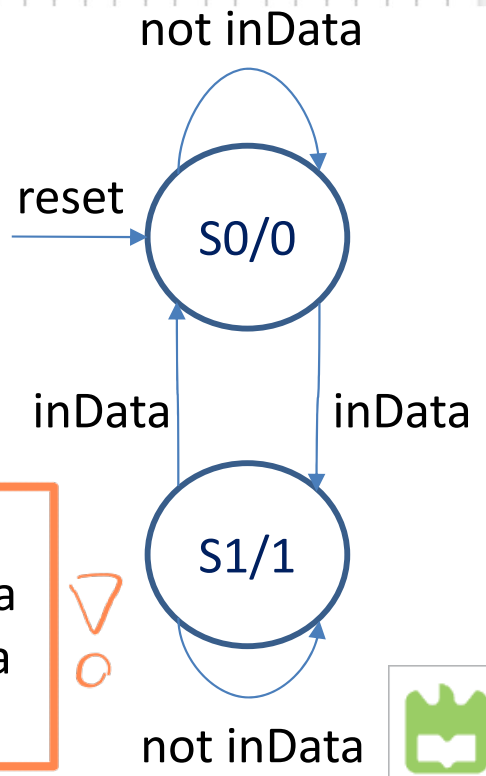
Tradução direta do Diagrama de Estados

Simulação de FSMs – Exemplo com Detetor de Paridade



Simulação com uma *testbench* onde é instanciada a UUT (FSM)

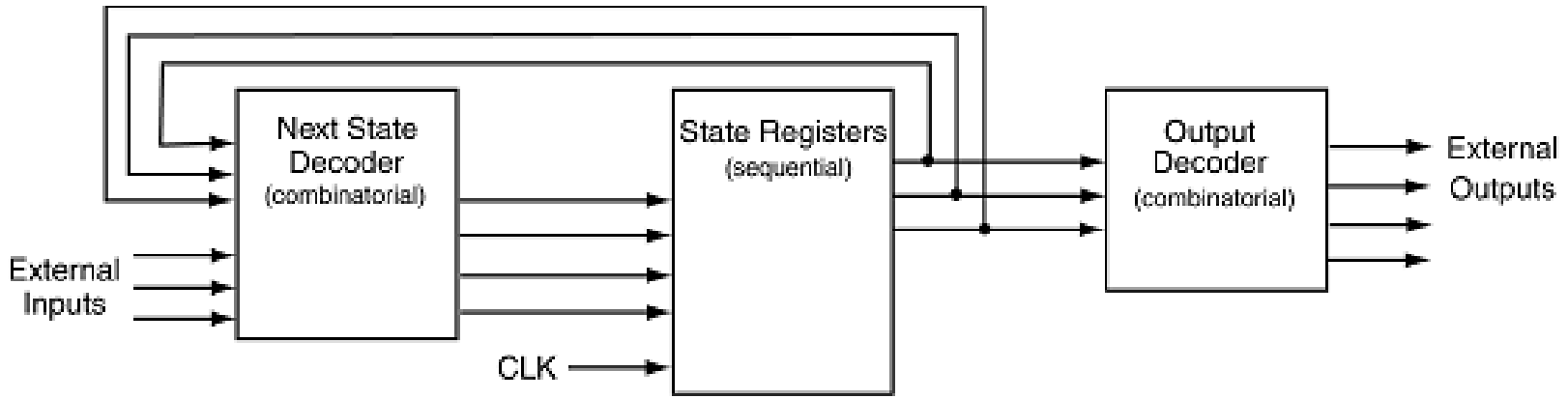
- Testbench gerada a partir do ficheiro VWF
- Testbench gerada diretamente em VHDL de acordo com o *template* para componentes sequenciais
  - 1 processo para geração do sinal de relógio
  - 1 processo para inicialização e geração da entrada



- **<u>Muito importante:</u>**
  Não comutar as entradas (variáveis independentes e reset) na vizinhança das transições ativas do sinal de relógio (refletir na simulação o cumprimento dos tempos de *setup* e de *hold*)

Síntese de FSMs

- 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 otimização dos circuitos combinatórios de estado seguinte e das saídas



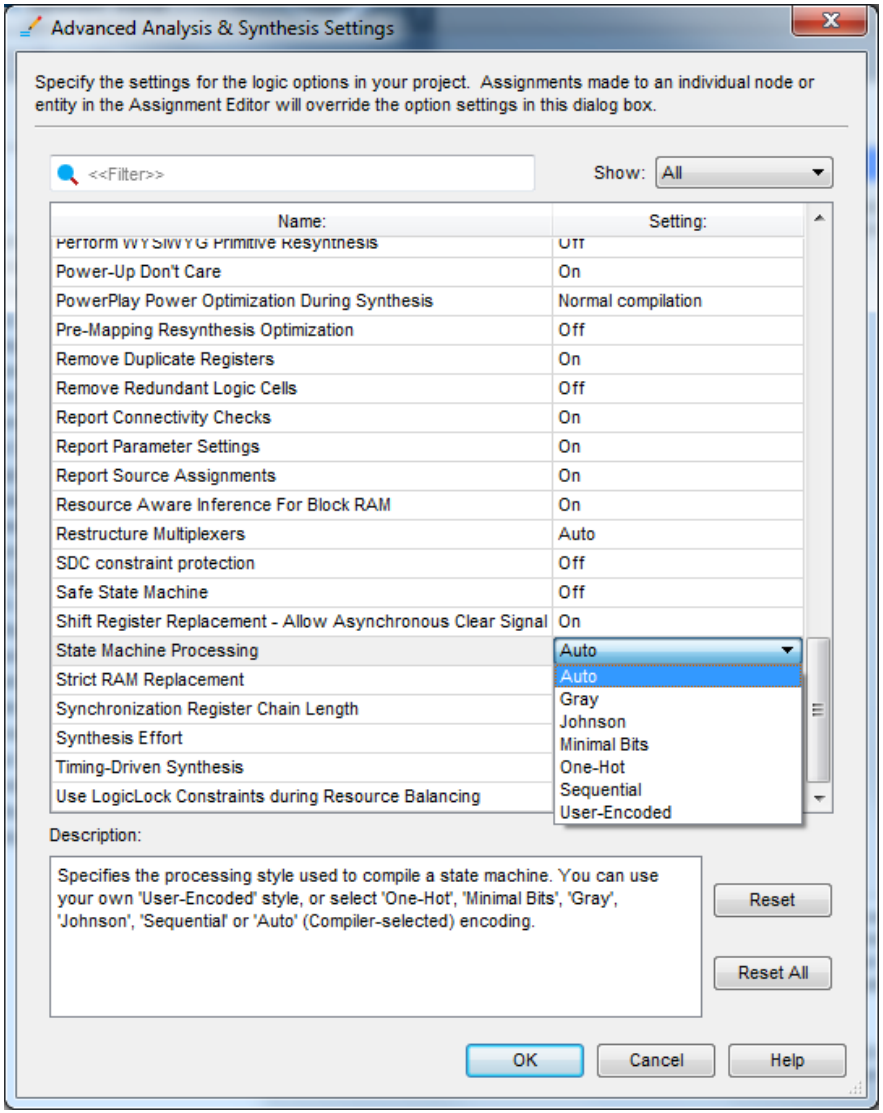
Codificação dos Estados

- Em VHDL o recurso a um tipo enumerado para codificar simbolicamente os estados
  - Permite uma descrição de alto-nível muito próxima do diagrama de estados
  - Confere ao sintetizador a tarefa da codificação binária dos estados
  - O hardware sintetizado inclui frequente mais *flip-flops* que o nº mínimo (quando a codificação é binária ou *gray*)

- Codificação frequente – *"One-hot"*
  - Exemplo: 4 estados => 4 flip-flops (000**1**, 00**1**0, 0**1**00, **1**000)
  - Adequada à implementação em FPGA (elevado número de *flip-flops* disponíveis mas LUTS 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

- No "Quartus Prime" é também possível "forçar" a técnica de codificação de estado em *"Assignments → Settings → Compiler Settings → Advanced Settings (Synthesis)"*



Comentários Finais

- No final desta aula e do trabalho prático 8 de LSD, deverá ser capaz de:
  - Reconhecer o modelo de FSM como uma ferramenta adequada para formalizar e projetar sistemas sequenciais
  - Construir diagramas de estados com base na especificação de uma FSM
  - Conhecer os passos necessários à síntese de máquinas de estados finitos
  - Usar descrições comportamentais em VHDL próximas do diagrama de estados e de saídas
    - Modelo de *Moore*
  - Conceber *testbenches* para a simulação funcional das FSMs
  - Sintetizar, implementar em FPGA e testar FSMs
- … bom trabalho prático 8, disponível no site da UC
  - elearning.ua.pt