

Laboratório de Sistemas Digitais**Trabalho Prático nº 8****Modelação, simulação e síntese de Máquinas de Estados Finitos
Aspetos gerais e modelo de Moore****Objetivos**

- Domínio dos procedimentos fundamentais no processo de síntese de Máquinas de Estados Finitos (MEFs) / *Finite State Machines (FSMs)*.
- Utilização de diagramas de estados e da linguagem VHDL para modelação ao nível comportamental de FSMs.
- Desenvolvimento de estratégias de simulação de FSMs.
- Síntese e implementação em FPGA e teste de FSMs.

Sumário

Este trabalho prático é dedicado à modelação comportamental, simulação, implementação em FPGA e teste de FSMs, com enfoque no modelo de *Moore*. Na primeira parte apresenta-se um sistema completo baseado num cronómetro digital, para análise inicial e posterior extensão das funcionalidades implementadas. Na segunda parte é apresentado um problema de modelação, simulação e implementação numa máquina de venda de bebidas com especificações muito restritas e simples. Para esse efeito, a FSM é modelada em VHDL de forma comportamental, recorrendo a dois processos interdependentes. A terceira parte é dedicada ao mesmo problema da parte dois, mas em que a descrição em VHDL é obtida de forma automática a partir do diagrama de estados desenhado numa ferramenta de edição.

Parte I

No *site* de LSD são disponibilizados os ficheiros fonte de um cronómetro digital, cuja arquitetura se encontra na Figura 1. O cronómetro utiliza 4 *displays* de 7 segmentos e apresenta a informação na forma “ss.cc”, em que “ss” corresponde ao contador dos segundos e “cc” ao contador dos centésimos de segundo (ambos os campos compreendidos entre 00 e 99).

Os módulos apresentados na Figura 1 podem ser descritos resumidamente da seguinte forma:

- A unidade de controlo estabelece a operação do cronómetro definindo os sinais de controlo em função de um conjunto de estados e entradas, sendo modelada por uma FSM.
- O contador em BCD, de módulo 10000 (0...9999), é incrementado em cada flanco ativo do sinal de relógio.
- O registo permite “congelar” os *displays*, mesmo que o contador continue a ser incrementado.
- Os descodificadores convertem os dígitos em BCD, de forma a controlar cada um dos segmentos dos *displays*.
- O divisor da frequência do sinal de relógio gera um *clock* com uma frequência de 1MHz a partir do *clock* de 50 MHz do *kit* (todo o circuito opera com o *clock* de 1 MHz).

- O gerador de pulsos produz nas suas saídas dois sinais que funcionam como *enable*, permitindo utilizar um único sinal de *clock* (com a frequência de 1 MHz) em todo o circuito:
 - Um pulso periódico, com a frequência de 100 Hz, ativo durante um período do sinal de relógio de 1MHz (1 micro-segundo), responsável pela ativação do incremento do contador BCD do cronómetro – este pulso está ativo durante 1 período do sinal de relógio e inativo os seguintes 9999 períodos.
 - Um sinal periódico com a frequência de 1 Hz e *duty-cycle* 50% para ativar um LED de forma intermitente.
- Dois módulos de *debouncing* que, além de filtrarem o ruído resultante da comutação de contactos mecânicos (botões de pressão), garantem a duração correta (tempo ativação) das entradas da unidade controlo (neste caso, um período do sinal de relógio, independentemente do tempo que o utilizador estiver a premir o botão de pressão) de forma a realizar a transição correta entre estados.

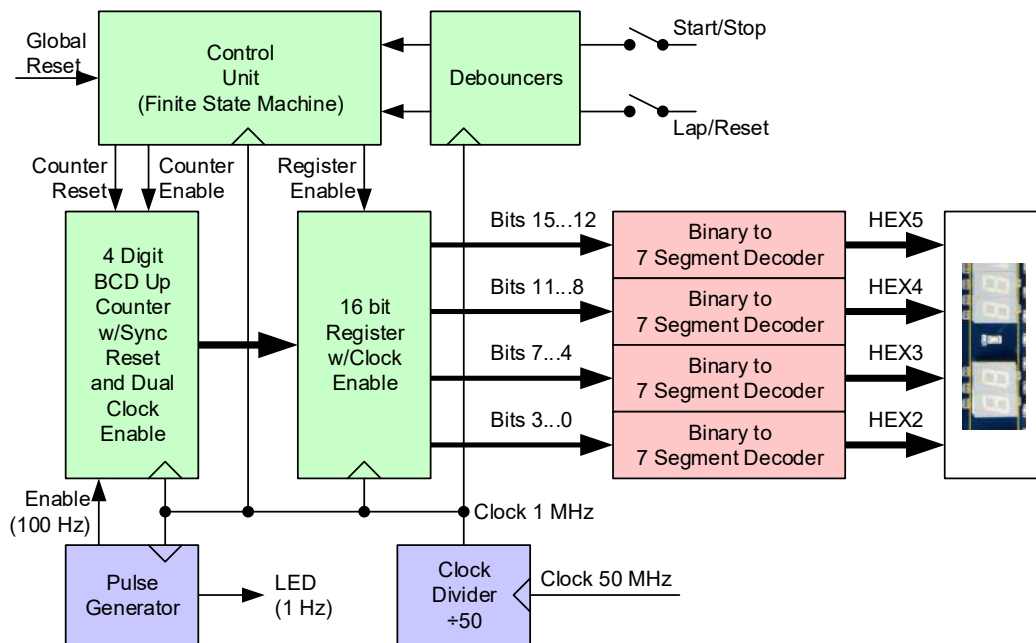


Figura 1 – Arquitetura do cronómetro fornecido.

1. Abra a aplicação “*Quartus Prime*” e crie um novo projeto para a FPGA Cyclone IV EP4CE115F29C7. Designe o projeto e a entidade *top-level* como “Chronometer”.
2. Adicione ao projeto os ficheiros fonte contidos no ficheiro “Chronometer.zip” fornecido juntamente com este trabalho prático, nomeadamente: “Chronometer.bdf”, “ClkDividerN.vhd”, “PulseGeneratorN.vhd”, “DebounceUnit.vhd”, “ControlUnit.vhd”, “CntBCDUp4.vhd”, “RegN.vhd” e “Bin7SegDecoder.vhd”. Não adicione, para já, o ficheiro “ControlUnit.smf”.
3. Selecione o ficheiro “Chronometer.bdf” como o *top-level* do projeto.
4. Compile o projeto e teste-o no *kit* (não se esqueça de importar o ficheiro “master.qsf”).

5. Verifique o funcionamento do cronómetro. As teclas do *kit* para controlo do cronómetro são as seguintes:

- KEY0 – *start/stop*
- KEY1 – *lap/reset*
- KEY3 – *global reset*

O comportamento do cronómetro implementado nos ficheiros fonte fornecidos pode ser resumido da seguinte forma:

- Após o *reset* global, o cronómetro é colocado a “00.00”, assim permanecendo até que seja premido o botão *start/stop*.
- Uma vez premido o botão **start/stop**, o cronómetro começa a contar. Nesta situação, caso seja premido o botão:
 - a) **start/stop** – o cronómetro para (a contagem é suspensa).
 - b) **lap/reset** – o cronómetro continua a contar (em *background*), mas o *display* deixa de ser atualizado (permitindo observar e registar um tempo parcial).
- Na situação a), caso seja premido o botão:
 - **start/stop**, o cronómetro continua a contagem a partir do valor em que foi suspenso.
 - **lap/reset**, o cronómetro é colocado a “00.00”.
- Na situação b), caso seja premido o botão **lap/reset**, o *display* volta a ser atualizado, permitindo visualizar novamente o cronómetro a ser incrementado.

6. O cronómetro é controlado pelo módulo *ControlUnit*, no qual está implementada uma FSM segundo o modelo de *Moore* e cujo diagrama se encontra na Figura 2. Analise o respetivo código VHDL para verificar se corresponde ao comportamento descrito no diagrama de estados da Figura 2.

7. Estabeleça a relação entre cada um dos estados do cronómetro descrito acima em linguagem natural, o diagrama de estados da Figura 2 e o código VHDL fornecido. Observe a forma como são codificadas em VHDL as transições de cada estado (inclusive para ele próprio, correspondendo às situações de manutenção de estado não representadas na Figura 2) e a atribuição das saídas associadas ao estado (segundo o modelo de *Moore*).

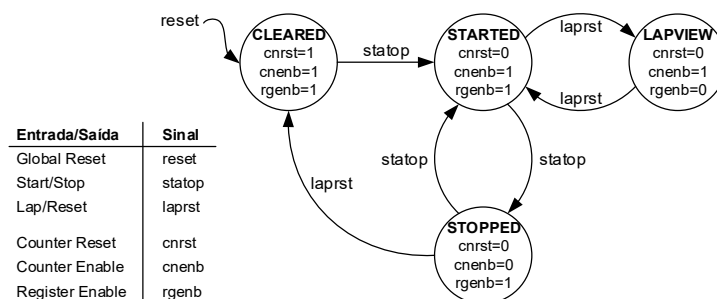


Figura 2 – Diagrama de estados do cronómetro fornecido (por simplicidade da figura apenas são apresentadas as transições relativas às mudanças de estado, i.e. as transições correspondentes à manutenção do estado não estão representadas).

Analise com atenção os restantes módulos do cronómetro, em especial, a forma como é realizada: a divisão do sinal relógio, a geração dos pulsos de *enable*, o *debouncing* dos sinais provenientes dos botões de pressão, a utilização de dois sinais de *enable* para controlar o incremento do contador BCD, a cascata (descrita comportamentalmente) dos vários dígitos do contador BCD e a suspensão da atualização dos *displays*.

8. Apesar do cronómetro fornecido já estar funcional, este não implementa uma funcionalidade comum neste tipo de dispositivos:

Quando o sistema se encontra na situação **b)** do ponto cinco e é premido o botão *start/stop*, o cronómetro deve parar a contagem que está a fazer em *background*, e assim permanecer (sem atualizar os *displays* com o valor do contador nesse instante). Este valor deve ser visualizado nos *displays* após ser premido o botão *lap/reset*.


9. Altere o diagrama de estados fornecido de forma a adicionar a funcionalidade descrita no ponto anterior. Desenhe no seu *log book* o diagrama de estados alterado (completo). **Dica:** terá de adicionar um estado para suportar a funcionalidade descrita no ponto anterior.

10. Edite o ficheiro VHDL do módulo “ControlUnit” de forma a refletir no código as alterações que introduziu no diagrama de estados no ponto anterior.

11. Volte a compilar o projeto e a testá-lo no *kit*, tendo o cuidado de verificar todas as funcionalidades implementadas (as iniciais e a adicionada nos pontos anteriores).

12. Altere o nome do ficheiro “ControlUnit.vhd” para “ControlUnit.backup.vhd”, de forma a ficar com uma cópia de segurança, uma vez que este ficheiro vai ser escrito com outro conteúdo nos pontos seguintes.

13. Adicione ao projeto o ficheiro “ControlUnit.smf” (ficheiro do tipo “*State Machine File*”), contido no ficheiro “Chronometer.zip”, e que possui a descrição da unidade de controlo inicial (correspondente ao código VHDL fornecido), mas na forma de um diagrama de estados. Os ficheiros SMF usam, para especificar as transições de estado, a sintaxe de Verilog (linguagem de descrição de hardware com aspetos sintáticos semelhantes às linguagens de programação de software JAVA e C).

14. Gere de forma automática o código VHDL da unidade de controlo, a partir do ficheiro “ControlUnit.smf”, premindo o botão  da barra de ferramentas (disponível quando abrir o ficheiro “ControlUnit.smf”). Abra o ficheiro “ControlUnit.vhd” e analise o seu conteúdo, comparando-o com o “ControlUnit.backup.vhd”.

NOTA: O ficheiro “ControlUnit.smf” permite apenas a modelação gráfica da FSM, não sendo usado na compilação do projeto. Para este efeito é usado um ficheiro VHDL que resulta da sua conversão automática num ficheiro VHDL com o mesmo nome. Por este motivo foi solicitada, no ponto 12, uma cópia de segurança para não perder o ficheiro “ControlUnit.vhd” original. O novo ficheiro “ControlUnit.vhd” será usado no ponto seguinte na compilação com os restantes ficheiros do projeto (se existir no diretório do projeto e mesmo que não faça parte do *workspace* – o que não é recomendável, resultando num *warning* de compilação). O ficheiro “ControlUnit.backup.vhd” não deve fazer parte do *workspace* do projeto.

15. Volte a compilar o projeto e a testá-lo no *kit*, tendo o cuidado de verificar as funcionalidades originais do cronómetro fornecido.

Os pontos seguintes desta parte do trabalho prático destinam-se a trabalho pós-aula.

16. Realize o *bypass* entre o sinal de entrada e de saída de cada *Debouncer*, de forma a aplicar à unidade de controlo as entradas diretas provenientes dos botões de pressão.

17. Compile o projeto e teste-o no *kit*, verificando o seu comportamento incorreto devido ao *bounce* nos sinais de controlo e à ativação incorreta (durante vários ciclos de relógio consecutivos) dos sinais de entrada da unidade de controlo (o tempo que o utilizador carrega num botão de pressão corresponde tipicamente a centenas de milhares de períodos de um sinal de *clock* com a frequência de 1 MHz).

18. Reponha as ligações iniciais (remova o *bypass*) dos *Debounce*s para que estes voltem a estar intercalados entre os sinais dos botões de pressão e as entradas da unidade de controlo.

19. Reflita no ficheiro “ControlUnit.smf” as adições que efetuou (em VHDL) nos pontos 9 e 10.

20. Volte a compilar o projeto e a testá-lo no *kit*, tendo o cuidado de verificar todas as funcionalidades implementadas (as iniciais e a adicionada no ponto anterior).

21. Altere o cronómetro para que passe a usar 6 *displays*, mostrando a informação no formato “mm:ss.cc”, em que “mm” (minutos) e “ss” (segundos) variam entre “00” e “59” e “cc” (centésimos de segundo) varia entre “00” e “99”. Se o cronómetro atingir o valor máximo “59:59.99” deverá parar e apresentar esse valor de forma intermitente até que seja premida a tecla “lap/reset”, passando de seguida a “00:00.00”.

NOTA: Caso altere a interface de um módulo VHDL deverá voltar a criar o seu símbolo e proceder à sua atualização no diagrama lógico (ficheiro BDF).

22. Numa aula teórico-prática posterior a este guião será apresentado um módulo de *Reset*, capaz de produzir um pulso sempre que o sistema arranca (a FPGA é programada), garantindo assim a sua correta inicialização, independentemente da ativação externa no sinal “*Global Reset*”. Adicione esse módulo ao projeto e instancie-o. Compile e teste o sistema resultante.

Parte II

Pretende-se implementar uma FSM que permita controlar uma máquina de venda de bebidas, semelhante à descrita nos slides da aula teórico-prática 7, mas com uma “ligeira” inflação no preço da lata, que passa a custar 0.90€. O diagrama de blocos da máquina de venda é mostrado na Figura 3.

1. Elabore no seu *log book* o diagrama de estados/saídas segundo o modelo de *Moore*. Evite estados redundantes.

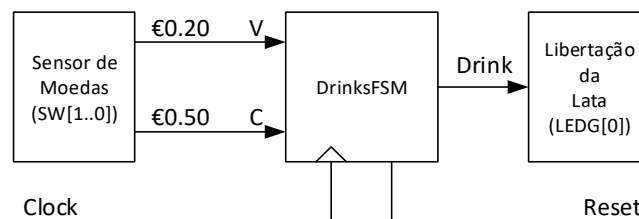


Figura 3 - Diagrama de blocos inicial da máquina de venda de bebidas.

2. Crie um novo projeto no IDE “*Quartus Prime*” para a FPGA Cyclone IV EP4CE115F29C7. Designe o projeto e a entidade *top-level* como “*DrinksMachine*”.

3. Crie um ficheiro VHDL, chamado “DrinksFSM.vhd”, para modelar o funcionamento da FSM, traduzindo diretamente em VHDL o comportamento descrito pelo diagrama de estados/saídas elaborado no ponto 1. Adote um estilo de codificação em VHDL baseado numa descrição comportamental com dois processos interdependentes:

- Um processo que atualiza o estado atual da FSM (correspondendo ao registo de estado).
- Um processo onde se definem as atribuições do estado seguinte e das saídas (correspondendo a um circuito combinatório).

4. Simule funcionalmente a FSM com a ajuda duma *testbench*. Defina criteriosamente os vetores de simulação de forma a validar adequadamente por simulação a descrição VHDL da FSM.

5. Crie o ficheiro *top-level* para associar as entradas e saídas da FSM a pinos da FPGA. Sugere-se o seguinte mapeamento das entradas da FSM com as seguintes interfaces do *kit*:

- | | | | |
|---------------|-----------|---------------|------------|
| ▪ “Reset” | => KEY[0] | ▪ “C” (€0.50) | => SW[1] |
| ▪ “V” (€0.20) | => SW[0] | ▪ “Drink” | => LEDG[0] |

Utilize para sinal de *clock* da FSM um sinal periódico de 1 Hz obtido a partir do sinal CLOCK_50. Use LEDs adicionais para visualizar o estado atual da FSM e o sinal de *clock* da FSM.

6. Compile o projeto, programe a FPGA e teste-o no *kit*. A introdução duma moeda deve ser concretizada (emulada) do seguinte modo: “SW[0] on” → “SW[0] off” ou “SW[1] on” → “SW[1] off”. Como é evidente apenas um SW deverá estar ativo de cada vez.

7. O sistema testado no ponto anterior possui um problema grave: o funcionamento correto e a reatividade dependem do tempo de ativação dos sensores, da ausência de *bouncing* e da frequência do sinal de *clock* da FSM. Para resolver este problema adicione ao sistema os módulos de *debounce* tal como ilustrado na Figura 4. Estes módulos efetuam o *debounce* dos sinais provenientes do “sensor de moedas”, emulado pelos interruptores do *kit*. O componente a usar para este efeito deverá ser a “DebounceUnit” usada na parte I deste guião e disponibilizada no *site* de LSD no documento “Bounce na comutação de contactos mecânicos – problema e possíveis soluções”. O módulo “DebounceUnit” garante a ativação da saída apenas durante um ciclo de relógio (neste caso por cada “moeda inserida”). De notar que o *debouncing* é fundamental para evitar múltiplas transições incorretas da máquina de estados (quer provocadas por *glitches* dos contactos mecânicos, quer por uma duração incorreta dos sinais de entrada da FSM). Neste caso a frequência do sinal de *clock* para a FSM e para os debouncers poderá ser 50 MHz.

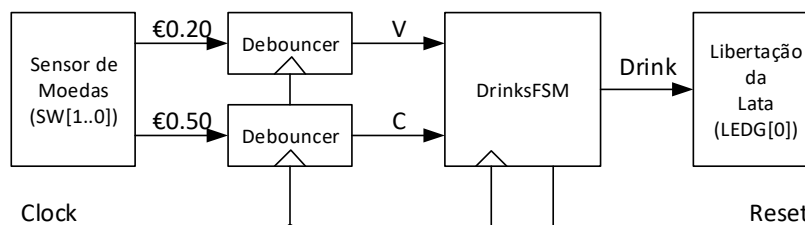



Figura 4 - Diagrama de blocos da máquina de venda de bebidas com os módulos de *debounce*.

8. Compile o projeto, programe a FPGA e teste-o no *kit*. Defina uma abordagem para conseguir visualizar a ativação do LED que emula a libertação da lata (sugestão: adicione um *flip-flop* à saída da FSM que faça o *toggle* da ativação do LED cada vez que é libertada uma lata).

Parte III

Nesta parte pretende-se refazer o projeto anterior, mas agora com a especificação da FSM realizada no “*State Machine Editor*” integrado no IDE “*Quartus Prime*”. Neste caso deve também ter sempre presente a especificação prévia da FSM, quer através do respetivo diagrama de estados/saídas, quer através da correspondente tabela de estados.


1. Crie um novo projeto no IDE “*Quartus Prime*” para a FPGA Cyclone IV EP4CE115F29C7. Designe o projeto e a entidade *top-level* como “*DrinksMachineV2*”.
2. Crie um novo ficheiro do tipo “*State Machine File*”, chamado “*DrinksFSM.smf*”. Após a sua criação entra no “*State Machine Editor*” do “*Quartus Prime*”.
3. A edição da FSM pode ser feita de vários modos. Podemos graficamente desenhar o diagrama de estados e as respetivas transições, mas sugere-se inicialmente a utilização do “*State Machine Wizard*” (botão  disponível na barra de ferramentas). A partir da tabela de estados/saídas, a FSM pode ser completamente descrita tendo em conta que o *wizard* permite definir (Figura 5):

- Portos de entrada e saída
- Estados
- Transições de saída de cada estado
- Condições lógicas associadas a cada transição (usando sintaxe de Verilog)
- Ações (saídas) associadas a cada estado (modelo de *Moore*)
- Ações (saídas) associadas a cada estado e, adicionalmente, eventuais transições (modelo de *Mealy*)



Figura 5 – “*State Machine Wizard*” como ponto de partida para especificação de uma MEF.

NOTA: Consulte o ficheiro “*ControlUnit.smf*” da parte I deste trabalho prático para obter dicas sobre como preencher os vários passos do “*State Machine Wizard*”. Além disso, está disponível no *site* de LSD um documento intitulado “*Quartus State Machine Editor*” com informação sobre a utilização desta ferramenta.

4. Completada a descrição da FSM com o *wizard* pode imediatamente ver-se o diagrama de estados correspondente. O passo seguinte consiste em traduzir automaticamente a especificação da FSM guardada no ficheiro “*DrinksFSM.smf*”, num módulo VHDL que possa ser compilado no projeto. Grave o ficheiro “*DrinksFSM.smf*” e seguidamente pressione o botão  do editor de máquinas de estado e escolha “VHDL”. Se tudo estiver corretamente especificado será criado um ficheiro “*DrinksFSM.vhd*”. Adicione este ficheiro ao projeto. Verifique a designação dos sinais associados às variáveis de estado. Note que se o diagrama de estados estiver incompleto ou incorretamente especificado a criação automática do correspondente ficheiro VHDL não é efetuada.
5. Sintetize o ficheiro “*DrinksFSM.vhd*”. Observe os resultados dos “*Netlist Viewers*” do “*Quartus Prime*”: “RTL”, “Post-mapping”, “State Machine” (verifique a consistência com a especificação e a codificação atribuída aos estados).
6. Crie um ficheiro “*VWF*” e simule a FSM. Defina temporalmente as entradas V e C de acordo com os fluxos possíveis de entrada de moedas. Verifique o comportamento de *Moore* da FSM.

7. Por vezes é importante seguirmos através de simulação, não só o comportamento entradas/saídas da FSM, mas também as transições de estado. Para tal é necessário adicionar aos sinais e portos apresentados na simulação, os sinais da FSM que internamente associámos às variáveis de estado. Reveja o código VHDL gerado automaticamente identifique os referidos sinais e adicione-os ao ficheiro VWF. Para tal deve aceder à lista dos sinais visíveis em “Post-synthesis”. Volte a simular e verifique agora, para além do comportamento entrada/saída, as correspondentes transições de estado.

8. Durante o processo de síntese, a codificação dos estados é, por omissão, feita automaticamente e normalmente baseada na abordagem “One-Hot” (um *flip-flop* por cada estado). O número de *flip-flops* usados é, em geral, superior ao estritamente necessário, sendo que a lógica de estado seguinte é também em geral mais simples. Poderá eventualmente ser interessante adotarmos outro tipo de codificação de estados. Para tal deve escolher uma das opções de compilação relacionadas com a síntese das FSM, como indicado na Figura 6 (menu “Assignments→Settings→Compiler Settings→Advanced Settings (Synthesis)”). Experimente as opções “Sequential” e “One-Hot”. Após recompilação do ficheiro VHDL volte a analisar os resultados da síntese com os vários “Netlist Viewers”. Repita a simulação da FSM para cada opção de codificação. Constate a consistência das transições de estado com as novas codificações.

9. Repita os pontos 5 e 6 da parte II de forma a testar no *kit* a nova versão da máquina de venda, com a FSM descrita num ficheiro SMF.

[TPC] Altere o projeto e adicione os componentes necessários de forma a que seja possível visualizar em 3 *displays* de 7 segmentos a quantia acumulada no ato de compra de cada bebida. Recomendação fundamental: elabore um diagrama de blocos completo do sistema antes de escrever qualquer linha de código.

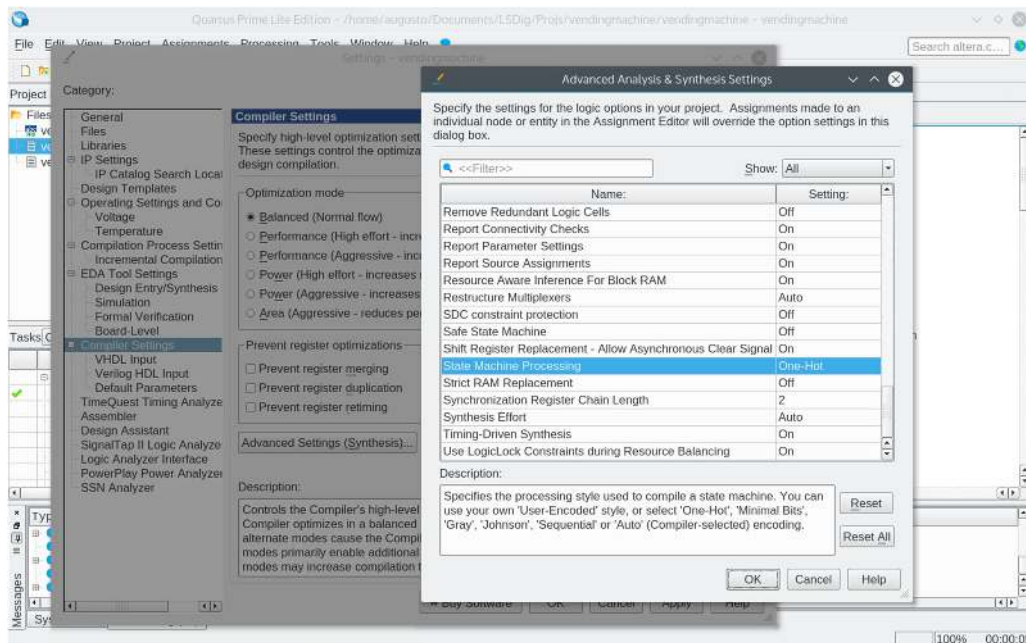


Figura 6 – Alteração do estilo de codificação de estados para a compilação duma FSM.