

Projeto de Sistemas Digitais 2018.2

Máquinas de Estados

Relatório: diferenças entre os tipos de implementações em HDL

ALUNOS:

João Santos de Sousa Filho

Matrícula: 201621250018

Jardel Brandon de Araujo Regis

Matrícula: 201621250014

DOCENTE:

Henrique do Nascimento Cunha

Máquinas de Estados

1. Introdução

Máquinas de estados finito (FSM) são amplamente utilizadas em projetos de circuitos digitais. Geralmente, ao projetar uma máquina de estados usando uma Linguagem de Descrição de Hardware (HDL), as ferramentas de síntese irão otimizar o circuito gerado para atender a descrição da FSM em HDL da forma mais eficiente. No entanto, em alguns casos a otimização não é aceitável. Por exemplo, se o circuito é ligado em um estado inválido, ou se o circuito está em um ambiente de trabalho extremo e uma falha o envia para um estado indesejado, o circuito pode nunca voltar à sua condição normal de operação.

Nesse relatório será abordado quais são os tipos e as classificações das máquinas de estados presentes no ambiente de desenvolvimento integrado (IDE) quartus da empresa Intel, além de explicar o porquê que ocorrem as falhas expostas acima e como realizar uma especificação de implementação para que as ferramentas de síntese não otimizem os estados não utilizados, assim, uma máquina de estado “segura” pode ser gerada.

2. Tipos

2.1 Mealy

Uma Mealy Machine é uma FSM cuja saída depende do estado atual e da entrada atual. Na máquina de mealy o bloco always referente a saída é sensível a mudança de entrada ou estado e em cada “caso” existe uma estrutura condicional para analisar a entrada atual e determinar a saída. Podemos ver a tabela de transição referente a FSM de Mealy:

Present State	Next State			
	a = 0		a = 1	
	State	Output	State	Output
-> q0	q3	0	q1	1
q1	q0	1	q3	0
q2	q2	1	q2	0
q3	q1	0	q0	1

2.2 Moore

A máquina Moore é uma FSM cujas saídas dependem apenas do estado atual. Na máquina de moore o bloco always referente a saída, é sensível apenas a mudança de estado, tendo em vista que a saída muda levando em consideração o estado atual, além disso cada “caso” remete diretamente a uma saída. Podemos ver a tabela de transição referente a FSM de Mealy:

Present State	Next State		Output
	a = 0	a = 1	
-> q0	q3	q1	1
q1	q0	q3	0
q2	q2	q2	0
q3	q1	q0	1

2.3 Diferenças: Moore e Mealy

A diferença crucial já foi citada a função de saída da máquina de Mealy pode ser representada da seguinte forma:

$$F(\text{estado_atual}, \text{entrada_atual}) \rightarrow \text{saída}$$

Enquanto a função de saída da máquina de Moore pode ser desta forma representada:

$$F(\text{estado_atual}) \rightarrow \text{saída}$$

A máquina de Mealy, geralmente tem menos estados que a máquina de Moore e muda as saídas de acordo com clock, enquanto a máquina de moore pode mudar a saída apenas com a mudança na entrada atual.

3. Classificação

A opção lógica **State Processing Machine** especifica o estilo de processamento para sintetizar uma máquina de estado. A codificação de máquina de estado padrão, Auto , usa uma codificação hot-hot para dispositivos do tipo Matriz de Portas Programáveis em Campo (FPGA) e codificação de bits mínimos para os Dispositivo Lógico Complexo Programável (CPLDs). Essas configurações

alcançam os melhores resultados em média, contudo outro estilo de codificação pode ser mais apropriado para o design, então essa opção permite controlar a codificação da máquina de estado.

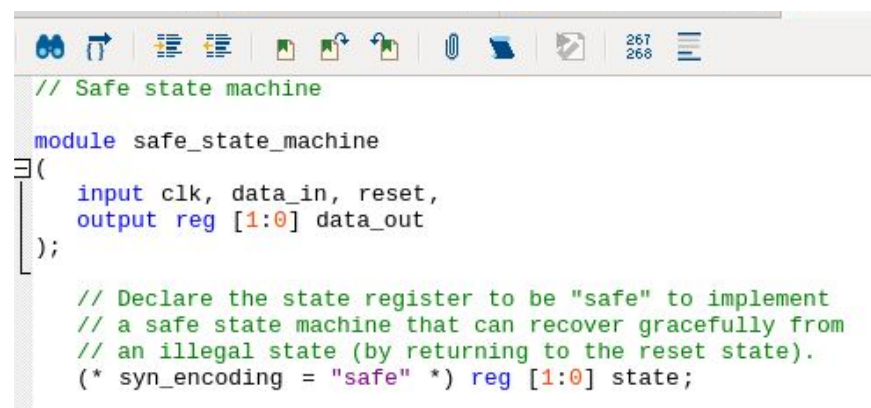
Todavia este relatório é focado na codificação da máquina de estado manualmente, com isso segue os tipos de classificações e atributos possíveis nas FSM do software Quartus, assim serão tratadas as principais classificações das máquinas de estados, que podem ser atribuídas independentemente dos tipos acima descritos.

3.1 Segurança

Uma máquina de estado seguro no Quartus é uma máquina de estado que, caso venha a atingir um estado ilegal (por qualquer motivo excepcional, como falha, metaestabilidade, etc.), será setado o estado de reinicialização após um ciclo de clock, e presumivelmente para se obter essa classificação será consumido mais recursos do FPGA para realizar o projeto da máquina de estado. Na implementação da FSM no Quartus, a opção lógica **Safe State Machine** e o valor do atributo `syn_encoding` correspondente ao `safe` especifica que o software deve inserir uma lógica extra para detectar um estado ilegal, e forçar a transição da máquina de estado para o estado de reinicialização caso a “ilegalidade” aconteça.

3.1.1 Safe FSM

Se o número de estados (N) for uma potência de 2 e for usado um algoritmo de codificação binário ou código cinza, o estado da máquina é “safe”. Isso garante que você tenha o número M de registros onde $N = 2^M$. Porque todos os possíveis valores de estado (ou status de registro) são alcançáveis, assim o design é considerado “seguro”. É possível definir uma máquina de estados segura inserindo o valor “safe” no atributo `syn_encoding` como segue em imagem abaixo.

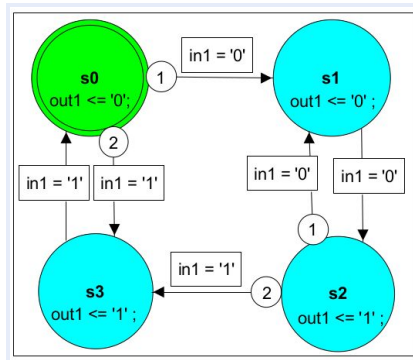


```
// Safe state machine
module safe_state_machine
(
    input clk, data_in, reset,
    output reg [1:0] data_out
);

    // Declare the state register to be "safe" to implement
    // a safe state machine that can recover gracefully from
    // an illegal state (by returning to the reset state).
    (* syn_encoding = "safe" *) reg [1:0] state;
```

3.1.2 Insegura

Se o número de estados não for uma potência de 2, ou se não for usado o algoritmo de codificação binária ou código cinza, por exemplo “one-hot”, a máquina de estado é “inseguro”. Por exemplo, a figura abaixo mostra um design que contém 4 estados:



O número de estados é 4, se for usado o algoritmo de codificação de one-hot. Por exemplo:

s0 => 0001

s1 => 0010

s2 => 0100

s3 => 1000

Assim, existem mais 12 estados que não estão definidos. Geralmente, esses estados são cobertos pelos “others” (em VHDL) ou “default” (para Verilog) da instrução case. A operação padrão da ferramenta de síntese irá otimizar estados inacessíveis para obter um circuito de alto desempenho. Mas a otimização cria um circuito “inseguro”.

3.2 User encoded

User encoded, ou em uma tradução livre codificação do usuário, trata-se de um método de implementação em que o usuário atribui o valor user para o atributo de síntese syn_encoding, dessa maneira é possível instruir o software e codificar cada estado com o valor definido no código-fonte Verilog HDL. Alterando os valores das constantes de estado, sendo possível alterar a codificação da máquina de estados, na figura abaixo é possível visualizar esta declaração.

```
// User-encoded state machine
module user_encoded
(
    input clk, data_in, reset,
    output reg [1:0] data_out
);

// Declare state register
(* syn_encoding = "user" *) reg [1:0] state;
```

3.3 All synchronous safe state machine

Uma máquina de estado síncrona é projetada para transição determinística através de um padrão de estados definidos. Um estado é representado por um registro (conjunto de DFFs) e é referido como "estado atual".

Diferentemente das classificações antigas em que as atualizações das saídas eram realizadas nas transições dos estados e os estados eram atualizados na borda de subida ou descida do clock e reset, uma FSM classificada como all synchronous safe, realiza todas essas atualizações sincronamente com o clock e reset, além de possuir o atributo syn_encoding com o valor "safe", o que garante que a máquina de estados em questão seja segura como foi discutido em tópicos acima.

4. Codificando os estados de uma FSM em HDL

Existem diferentes métodos de codificação que podem ser usados para implementar os estados de um FSM. A seguir esses métodos de codificação discutidos anteriormente serão apresentados, como algoritmo binário, cinza ou "one-hot", existem algumas outras opções de codificação de estado, mas, na prática, geralmente esses são os mais utilizados. Sendo possível verificar que, para um dado diagrama de estado, o método de codificação de estado pode reduzir o consumo de energia do FSM ou aumentar sua frequência de clock. A classificação de uma FSM segura, pode ser atribuída caso seja implementado algum desses algoritmos em sua construção.

4.1 Codificação Binária

Em um esquema de codificação binária, a relação entre o número de variáveis de estado (q) e o número de estados (n) é dado pela equação: $q = \log_2(n)$. Com esta fórmula, pode-se facilmente determinar o número mínimo de variáveis de estado necessárias para uma máquina de estado codificada. O número de flip-flops usados é igual ao número de variáveis de estado (q). Nesta técnica, os estados são atribuídos em sequência binária, onde os estados são numerados a partir de binário 0. Por exemplo, em uma máquina de 8 estados, a atribuição de estado é feita como mostrado abaixo;

State	Value
Idle	000
r1	001
r2	010
r3	011
r4	100
c	101
p1	110
p2	111

4.2 Codificação cinza

A atribuição de código cinza é uma atribuição de estado na qual códigos de estados consecutivos diferem apenas por um bit (adjacente). Em outras palavras, apenas um flip-flop muda ao mesmo tempo ao alterar estados consecutivos. Nesta codificação, o número de flip-flops (registrador) usado também é determinado por: Número de variáveis de estado = número de flip-flops = \log_2 (número de estados). A seguir a implementação do mesmo exemplo anterior utilizando código cinza:

State	Value
Idle	000
r1	001
r2	011
r3	010
r4	110
c	111
p1	101
p2	100

4.3 Codificação one-hot

Na codificação one-hot, apenas um bit da variável de estado é “1” ou “hot” para qualquer estado dado. Todos outros bits de estado são zero. Portanto, um flip-flop (registrador) é usado para todos os estados máquina ou seja, n estados usa n flip-flops. Usando uma codificação quente, as equações do próximo estado podem ser derivadas facilmente a partir de diagramas de estado. A decodificação de estado é simplificada, pois os próprios bits de estado podem ser usados diretamente para indicar se a máquina está em um estado particular. A seguir a implementação do mesmo exemplo anterior utilizando código one-hot:

State	Value
Idle	00000001
r1	00000010
r2	00000100
r3	00001000
r4	00010000
c	00100000
p1	01000000
p2	10000000

4.4 Resumo

- Codificação binária, talvez dentre os métodos apresentados a implementação binária possa ser considerada como a mais intuitiva entre eles, porém esse método apresenta um nível maior concentrado de capacitância parasita que estão presentes no circuito. Esta capacitância parasita é introduzida pelas interconexões dos circuitos e pelos estágios de entrada dos circuitos combinacionais que geram o próximo estado e as saídas, sendo possível a diminuição desse efeito implementando o código cinza.
- Com o código cinza, apenas um bit é alterado ao se mover entre estados adjacentes. Como resultado, esta técnica de codificação pode reduzir o consumo de energia de um FSM. Além disso, a codificação cinza torna as saídas assíncronas de um FSM tornando resiliente a falhas.
- A codificação one-hot simplifica os blocos lógicos para gerar as saídas e blocos lógicos para gerar o próximo estado. Com estes dois blocos simplificados, podemos gerar as saídas FSM e o próximo estado mais rapidamente.
- Obter a atribuição de estado ideal para um FSM é um problema difícil, mas, na prática, é possível usar ferramentas de síntese de FPGA com algoritmos de otimização proprietários para chegar a uma implementação eficiente de um FSM automaticamente.

Table 16–3. syn_encoding Attribute Values

Attribute Value	Enumeration Types
"default"	Use an encoding based on the number of enumeration literals in the Enumeration Type. If the number of literals is less than five, use the "sequential" encoding. If the number of literals is more than five, but fewer than 50, use a "one-hot" encoding. Otherwise, use a "gray" encoding.
"sequential"	Use a binary encoding in which the first enumeration literal in the Enumeration Type has encoding 0 and the second 1.
"gray"	Use an encoding in which the encodings for adjacent enumeration literals differ by exactly one bit. An N -bit gray code can represent 2^N values.
"johnson"	Use an encoding similar to a gray code. An N -bit Johnson code can represent at most 2^N states, but requires less logic than a gray encoding.
"one-hot"	The default encoding style requiring N bits, in which N is the number of enumeration literals in the Enumeration Type.
"compact"	Use an encoding with the fewest bits.
"user"	Encode each state using its value in the Verilog source. By changing the values of your state constants, you can change the encoding of your state machine.

5. Conclusão

As máquinas de estados são essenciais para o projeto de circuitos digitais, com seu uso em HDL podemos abstrair tarefas de um sistema de modo a tornar mais simples o seu projeto. Ao fim deste relatório podemos concluir que os tipos de implementações para FSM disponibilizados pela Altera são importantes graças aos mecanismos específicos para síntese presentes no modelo "user" e "safe". Vale destacar também os modelos de máquinas de estados moore e mealy, que possuem cada uma suas vantagens e desvantagens.