Laboratório de Sistemas Computacionais: Circuitos Digitais

Projeto 07 - Máquina de Estados Finitos em Verilog

Aluna: Thauany Moedano

Universidade Federal de São Paulo - UNIFESP
Instituto de Ciência e Tecnologia, São José dos Campos

1. Introdução

Projetos de circuitos digitais são extremamente importantes para a implementação de processadores, hardwares, circuitos integrados e tecnologias com sistemas digitais em geral. Desde um simples relógio digital a supercomputadores, o primeiro passo é pensar no projeto lógico do circuito. Este trabalho apresenta a construção e implementação de um circuito digital que mostra em um display a sequência **4-2-5-2-0-1-4-3-5** em ordem crescente, ordem decrescente, capaz de manter um único número aparecendo no display e também mostrar o display apagado de acordo com as entradas.

A sessão 2 disserta sobre a fundamentação teórica utilizada para desenvolver o trabalho. A sessão 3 apresenta o desenvolvimento e implementação do código em Verilog. A próxima sessão discute os resultados obtidos com a implementação do circuito.

2. Fundamentação Teórica

Esta sessão aborda os principais conceitos utilizados para projetar a descrição de hardware em verilog. O projeto pode ser concluído em uma única fase onde escreve-se um código utilizando uma linguagem de descrição (Verilog) para simular uma máquina de estados. Por fim, o funcionamento do circuito é testado em uma placa FPGA.

2.1. Verilog

Verilog não é uma linguagem de programação. Verilog é uma *HDL - Hardware Description Languange* ou seja, é uma linguagem utilizada para descrever um circuito. Desta maneira, a implementação de circuitos fica muito simples uma vez que não é necessário pensar em tabelas verdade e fazer mapas de karnaught. Basta entender qual é o conceito chave do circuito e aplicá-lo utilizando todos os recursos que Verilog oferece. Os ”black box”de um esquemático são equivalentes aos módulos em Verilog.

2.2. Máquinas de Estados Finitos

Uma Máquina de Estados Finitos é um circuito lógico que pode ser divididos em estados, como o próprio nome sugere. A saída de cada estado é definida por um estado anterior (no caso de uma Máquina de Moore) e também pelas entradas do circuito (No caso da máquina de Mealy). Portanto, a máquina de Mealy tem os estados atualizados imediatamente quando as entradas mudam ao contrário da Máquina de Moore que só faz transições a cada ciclo de clock. [Tocci et al. 2003]

Uma Máquina de Estados em Verilog tem três blocos principais: Um registrador, uma lógica para o próximo estado e uma para a saída. O bloco do registrador tem como função armazenar qual é o estado atual. Se a Máquina de Estados implementa um reset

(função que faz a máquina voltar imediatamente para o estado inicial), é neste bloco que o reset deve ser verificado.

O bloco de próximo estado como o nome sugere verifica quem é o próximo estado que deve ser atribuído de acordo com o estado atual e as entradas.

O último bloco trata da lógica de saída que faz a conversão do estado atual para a saída desejada.

2.3. Temporizador e Display de Sete Segmentos

O clock da Máquina de Estados Finitos pode ser controlado a partir de um temporizador. Um temporizador em verilog pode ser construído utilizando uma contagem para dividir a frequência do clock e retornar um novo clock com a frequência correta.

O Display de Sete Segmentos é uma pequena placa com setes LEDs capazes de formar números de 0 a 9.

2.4. Testes em FPGA

FPGA (Field Programmable Gate Array) é um dispositivo cujo sua arquitetura pode ser programada: Ou seja, suas funcionalidades não vêm pré definidas como ocorre em geladeiras, tevelisões, celulares entre outros aparelhos eletrônicos. Isso permite que o FPGA possa ser reprogramado diversas vezes e tenha funcionalidades alternadas.

3. Desenvolvimento

3.1. Diagrama de Estados

O primeiro passo para desenvoler o projeto é desenhar o diagrama de estados de acordo com as entradas. As entradas são **Up** e **Down** e ditam o que devem acontecer com cada estado seguindo a tabela:

Up	Down	Contagem
1	0	Crescente
0	1	Descrecente
0	0	Mantém
1	1	Blank

Note que o estado blank representa um estado inválido das entradas, ou seja: o display deve aparecer aparecer apagado quando se manter nesse estado. Isto foi representado com a saída 1111 no display cujo saída será todos os leds apagados.

Baseando-se nas informações apresentadas, um **Diagrama de Estados** pode ser elaborado. Cada saída deve ser representada por um estado por uma combinação única de

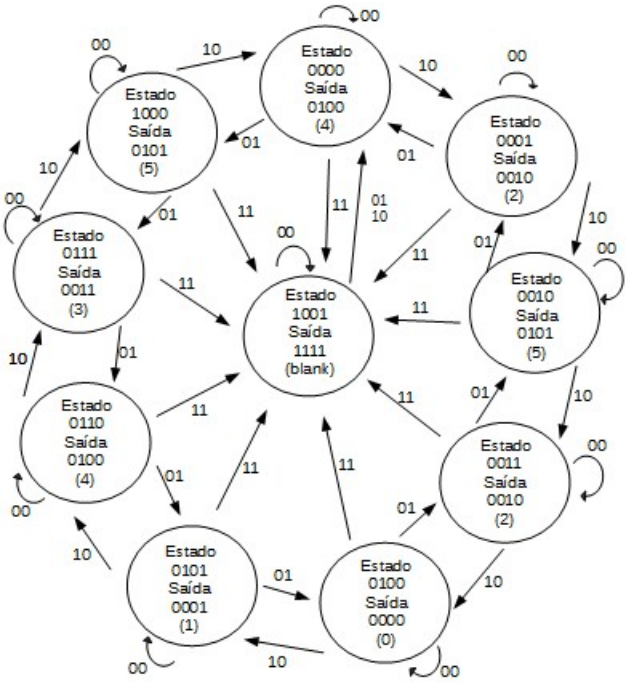

Figura 1. Diagrama de Estados

bits. Portanto, para cada número da contagem bem como o *blank* devem ser representados por uma única combinação de bits.

A função do próximo estado é obtida a partir das entradas UP e DOWN e as entradas do estado atual. Já a função de saída depende apenas das entradas do estado atual uma vez que esta é uma Máquina de Moore. Portanto, as saídas só mudam em transições de clock.

3.2. Parâmetros de uma Máquina de Estados em Verilog

Dado o diagrama de estados é possível construir a Máquina de Estados em Verilog. O primeiro passo é declarar um módulo para a Máquina de Estados. Todo projeto em Verilog exige que um módulo seja criado. Os parâmetros a serem passados para o módulo são parâmetros de entrada e saída. Portanto teremos como parâmetros: **Clk, reset, UpDown e display**.

- Clk: Será a entrada de clock da máquina de estados. O Clock ainda deve ser temporizado na frequência desejada.
- reset: Serve para arbitrar se a contagem deve ser realizada ou não. Quando reset está em 1 a contagem é realizada. Caso contrário, a Máquina de Estados deve retornar imediatamente para o primeiro estado.
- UpDown: São as entradas de dois bits que ditam se a Máquina deve ir para o próximo estados, voltar para o anterior, permanecer no mesmo estado ou ir para o blank (LEDs apagados)

- display: É a saída final da Máquina de Estados já decodificada para o display de sete segmentos.

Além disso há as declarações dentro da Máquina de Estados que serão utilizadas para realizar as lógicas de próximo estado e saída.

Os parâmetros que foram passado no módulo devem ser declarados como entrada ou saída dentro do módulo da Máquina de Estados (**input / output**). Como **UpDown e display** tem mais de um bit, é necessário informar quantos bits podem variar para aquela saída.

4 novas entradas são atribuídas dentro do módulo da Máquina de Estados: **newClk, saida, EstadoAtual, ProxEstado**.

Há 10 parâmetos(E1...E9, BLANK) são atribuídos os estados de acordo com o que foi descrito no diagrama de estados.

- newClk: É o fio que vai receber o clock com a frequência correta
- saida: São os bits convertidos do estado atual que serão decodificados para o display de sete segmentos.
- EstadoAtual: Vai guardar os bits correspondente ao estado atual.
- ProxEstado: vai guardar os bits correspondente ao próximo estado.

```
module MaqEstados(clk, reset, UpDown, display);
    input clk, reset;
     input[1:0] UpDown;
     output[6:0] display;
     wire newClk;
     reg[3:0] saida;
     reg[3:0] EstadoAtual, ProxEstado;

       parameter E1 = 4'b0000, E2 = 4'b0001, E3 = 4'b0010, E4 = 4'b0011,
                      E6 = 4'b0101, E7 = 4'b0110, E8 = 4'b0111, E9 = 4'b1000
```

3.3. Montando o código

O primeiro bloco indica quem será o estado atual. É representado a partir de um comando **always@**. Este comando faz com que sempre o que está dentro dos parâmetros mude, tudo que está dentro do bloco **always@** deve ser executado. Fazemos uma verificação simples: Se o reset está em um realiza-se a contagem e ao EstadoAtual deve ser atribuído ProxEstado. Caso contrário, o EstadoAtual deve ser obrigatoriamente o primeiro estado (E1).

```
always@(posedge newClk)
begin
    if(reset)
        EstadoAtual <= ProxEstado;
    else
        EstadoAtual <= E1;
end
```

O próximo bloco corresponde a lógica de próximo estado. Para cada (E1...E9, BLANK) deve-se verificar qual é a entrada UpDown atual e associar o ProxEstado de acordo com UpDown e EstadoAtual. O estado BLANK tem uma condição especial e interage de forma diferente dos outro estados. O BLANK caso UpDown = (00) ou (11), o estado se mantém em BLANK. Qualquer outra variação faz com que a Máquina Retorne para o primeiro estado (E1).

Para fazer esta associação, novamente usa-se o comando **always@** passando como parâmetro a borda de subida do newClk. Depois usa-se o comando **case** para verificar cada caso de estado e é feito as atribuições comparando qual é a entrada UpDown. O comando **default** indica uma saída padrão do **case** e evita a criação de **latches parasitas**.

```
always@(posedge newClk)
        begin
                case(EstadoAtual)
                    E1:
                        if(UpDown == 2'b00)
                            ProxEstado = E1;
                        else if(UpDown == 2'b01)
                            ProxEstado = E9;
                        else if(UpDown == 2'b10)
                            ProxEstado = E2;
                        else
                            ProxEstado = BLANK;
                    E2:
                        if(UpDown == 2'b00)
                            ProxEstado = E2;
                        else if(UpDown == 2'b01)
                            ProxEstado = E1;
                        else if(UpDown == 2'b10)
                            ProxEstado = E3;
```

```
        else
            ProxEstado = BLANK;
    E3:
        if(UpDown == 2'b00)
            ProxEstado = E3;
        else if(UpDown == 2'b01)
            ProxEstado = E2;
        else if(UpDown == 2'b10)
            ProxEstado = E4;
        else
            ProxEstado = BLANK;
    E4:
        if(UpDown == 2'b00)
            ProxEstado = E4;
        else if(UpDown == 2'b01)
            ProxEstado = E3;
        else if(UpDown == 2'b10)
            ProxEstado = E5;
        else
            ProxEstado = BLANK;
    E5:
        if(UpDown == 2'b00)
            ProxEstado = E5;
        else if(UpDown == 2'b01)
            ProxEstado = E4;
        else if(UpDown == 2'b10)
            ProxEstado = E6;
        else
            ProxEstado = BLANK;
    E6:
        if(UpDown == 2'b00)
            ProxEstado = E6;
        else if(UpDown == 2'b01)
            ProxEstado = E5;
        else if(UpDown == 2'b10)
            ProxEstado = E7;
        else
```

```
            ProxEstado = BLANK;
        E7:
            if(UpDown == 2'b00)
                ProxEstado = E7;
            else if(UpDown == 2'b01)
                ProxEstado = E6;
            else if(UpDown == 2'b10)
                ProxEstado = E8;
            else
                ProxEstado = BLANK;
        E8:
            if(UpDown == 2'b00)
                ProxEstado = E8;
            else if(UpDown == 2'b01)
                ProxEstado = E7;
            else if(UpDown == 2'b10)
                ProxEstado = E9;
            else
                ProxEstado = BLANK;
        E9:
            if(UpDown == 2'b00)
                ProxEstado = E9;
            else if(UpDown == 2'b01)
                ProxEstado = E8;
            else if(UpDown == 2'b10)
                ProxEstado = E1;
            else
                ProxEstado = BLANK;
        BLANK:
            if(UpDown == 2'b00)
                ProxEstado = BLANK;
            else if(UpDown == 2'b01)
                ProxEstado = E1;
            else if(UpDown == 2'b10)
                ProxEstado = E1;
            else
                ProxEstado = BLANK;
```

```
                default : EstadoAtual = 4'bXXXX;

            endcase
        end
```

O último bloco corresponde a conversão do EstadoAtual para a saída em binário. De maneira similar ao bloco anterior basta utilizar a função **case** para converter os estados nas saídas desejadas de acordo com a sequência: **4-2-5-2-0-1-4-3-5**.

```
    always@(EstadoAtual)
    begin
        saida = 4'b0100;
        case(EstadoAtual)
            E1: ;
            E2 : saida = 4'b0010;
            E3 : saida = 4'b0101;
            E4 : saida = 4'b0010;
            E5 : saida = 4'b0000;
            E6 : saida = 4'b0001;
            E7 : saida = 4'b0100;
            E8 : saida = 4'b0011;
            E9 : saida = 4'b0101;
            BLANK : saida = 4'b1111;
            default :
                begin
                    saida = 4'bXXXX;
                end
        endcase
end


endmodule
```

3.4. Configurando o clock automático - Temporizador

A Máquina de Estados só tem sua saída modificada durante as transições de clock. Deseja-se que a máquina funcione com um clock automático. A placa de FPGA oferece um clock que opera em 50MHz. Portanto, é necessário configurar um temporizador para que as

transições ocorram em um segundo. A seguinte conta foi tomada para saber como deveria ser configurado o temporizador:

$$\frac{50Mhz}{2^x} = 1segundo$$

$$50Mhz = 1.2^x$$

$$\log 50MHz = \log 2^x$$

$$x \approx 26,6$$

Em Verilog é possível fazer um temporizador de forma simples. A partir da função **always@**, basta utilizar uma contagem sobre uma entrada de 26 bits (uma vez que o resultamento da conta deu aproximadamente 26,6) e atualizar a frequência toda vez que o clock de 50 Mhz varia. No fim atribui-se a frequência para o clock novo que passa a variar de um e um segundo.

O módulo do temporizador foi posteriormente instanciado no módulo da Máquina de Estados.

```
        module Temporizador(clkIn, clkOut);
    input clkIn;
    output wire clkOut;

    parameter nDiv = 25;
    reg[nDiv:0] freq;

    always@(posedge clkIn)
        begin
            freq <= freq + 1;
        end

    assign clkOut = freq[nDiv];
endmodule
        //Instancia no Modulo da Maquina de Estados//
         Temporizador T1 (.clkIn(clk), .clkOut(newClk));
```

3.5. Display de Sete Segmentos

A saída deve ser decodificada em um display de sete segmentos, lembrando que quando se está no estado *blank*, o display deve aparecer apagado. Um módulo para o display pode ser criado. Os parâmetros passado são os bits de **saida** da Máquina de Estados. Um **always@** é utilizado para que sempre quando um dos bits de saída for modificado,

o display seja atualizado. Uma função **case** faz a conversão dos bits de saída para bits correspondentes no display de sete segmentos. Posteriormente o módulo foi instanciado no módulo da Máquina de Estados.

```
    module DisplayVerilog(W,X,Y,Z,out);

input W,X,Y,Z;
wire[3:0] entrada;
output reg[6:0] out;
assign entrada = {W,X,Y,Z};
always@(W or X or Y or Z)
    begin
        case(entrada) //Conversao da saida de 4 bits para 7 bits co
            4'B0000 : out = 7'B0000001;
            4'B0001 : out = 7'B1001111;
            4'B0010 : out = 7'B0010010;
            4'B0011 : out = 7'B0000110;
            4'B0100 : out = 7'B1001100;
            4'B0101 : out = 7'B0100100;
            4'B0110 : out = 7'B0100000;
            4'B0111 : out = 7'B0001111;
            4'B1000 : out = 7'B0000000;
            4'B1001 : out = 7'B0000100;
            default : out = 7'B1111111;
            endcase
        end
endmodule

//Instancia no modulo da Maquina de Estados//
DisplayVerilog D1 (.W(saida[3]), .X(saida[2]), .Y(saida[1]), .Z(sai
```

4. Resultados

Ao testar a forma de onda da Máquina de Estados pode-se observar a corretude do funcionamento. A contagem é crescente quando Up = 1 e Down = 0 e descrecente quando Up = 0 e Down = 1. O Reset também funciona corretamente realizando a contagem quando está ligado em alto.

Os testes foram feitos na placa de FPGA para observar o funcionamento dos LEDs. Os LEDs acenderam corretamente mostrando os números na sequência desejada.

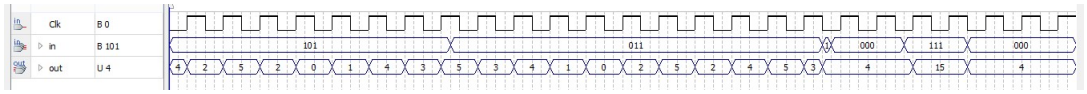

Figura 2. Saída final da Máquina de Estados Finitos

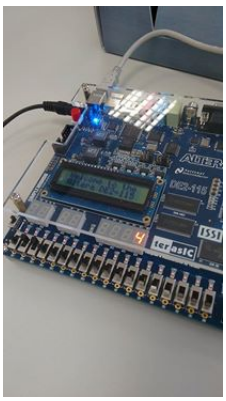

Figura 3. Funcionamento dos LEDs em FPGA

5. Conclusão

Máquina de Estados são úteis para as mais diversas aplicações. Utilizando esta ideia é possível fazer circuitos mais complexos. Para tanto é preciso estar familiarizado com projeto de circuitos e testes em FPGA para confirmar a corretude do circuito. A linguagem Verilog facilita a construção de projetos uma vez que é inviável projetar em esquemático circuitos muito grandes. Uma máquina de estados torna-se bem mais simples em Verilog uma vez que não há a necessidade de fazer tabelas verdades e mapas de karnaught e construir esquemáticos para cada bit dos estados e das saídas.

Portanto é importante ter conhecimentos em projeto de circuitos, máquina de estados, Verilog e configuração do ambiente FPGA.

Referências

Tocci, R. J., Widmer, N. S., and Moss, G. L. (2003). *Sistemas digitais: princípios e aplicações*, volume 8. Prentice Hall.