Universidade Federal de Viçosa

Trabalho Prático 02 – Máquinas de Estado Finito
CCF 251 – Introdução aos Sistemas Lógicos
Ciência da Computação – Campus UFV-Florestal
Prof. José Augusto Miranda Nacif

Integrantes:
Tiemy Shibuya Watanabe - 5214, Vinícius de Oliveira Mendes - 3881, Igor Melo - 3889.

1. Introdução

Neste semestre os alunos da Ciência da Computação começaram a discutir sobre o que fazer depois de se formar no curso. Então eles decidiram começar uma arrecadação em dinheiro desde já, para que quando se formarem, façam uma grande festa, mas como conseguir esse dinheiro?

Depois de muitas discussões eles decidiram desenvolver uma máquina para vender doces e colocar na universidade. Para isso eles se dividiram em grupos para montar esta máquina. Ela deve vender doces no valor de 30 centavos e aceitar moedas de 5, 10 ou 25 centavos. A máquina também deve indicar quando o saldo for suficiente para comprar o doce e quando houver troco. Você e seu grupo devem usar Verilog ou SystemVerilog para descrever o hardware da máquina de doces.

2. Desenvolvimento

Primeiramente, para resolver o problema, foi feita a elaboração da máquina de estados (Figura 1) e a tabela de estados (Figura 2) apresentadas a seguir. Para a representação do “Estado atual” e o “Estado próximo” foi utilizada uma notação de 4bits. Cada bit representa um único valor (5 centavos, 10 centavos, 15 centavos, 25 centavos) e a combinação desses bits é o valor do estado atual/próximo. Na máquina de estados foi possível identificar 11 estados no total e no final, elas voltam para o início (0 centavos).

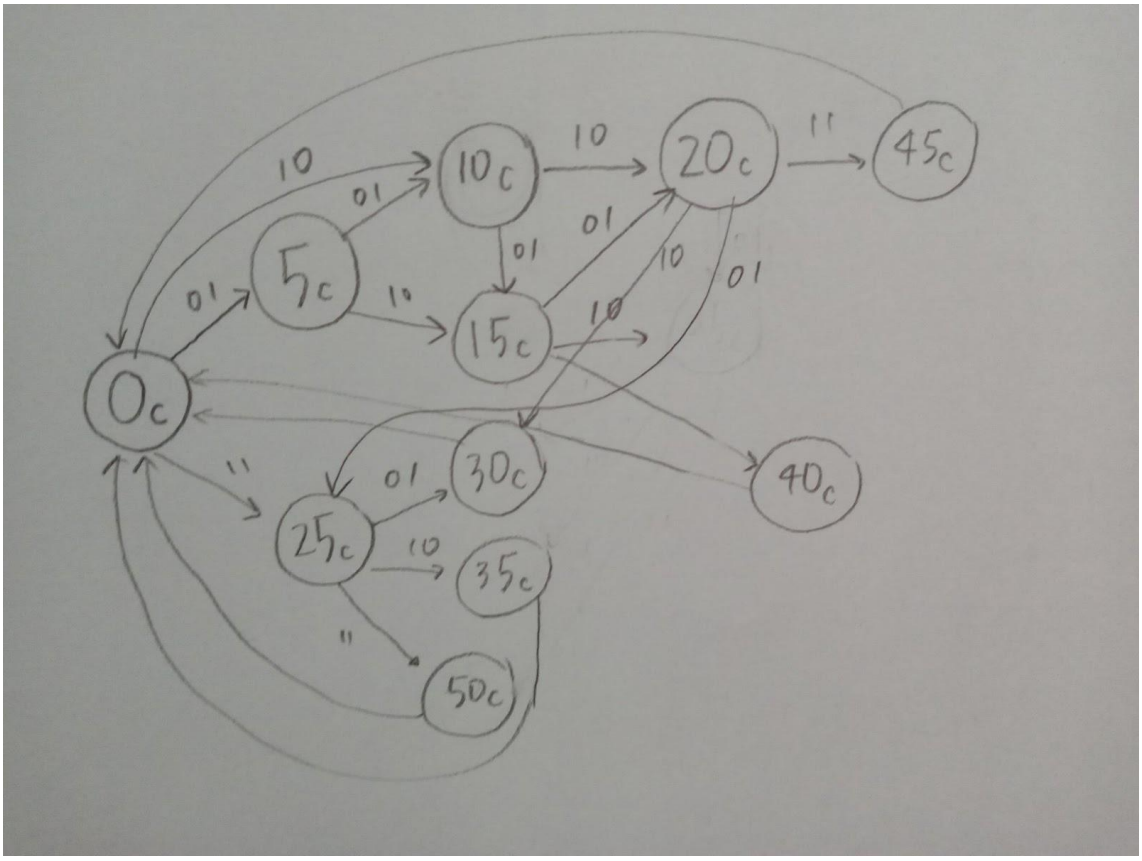

Figura 1 - Máquina de Estados
Fonte: Elaborado pelos autores

Estado Atual				Entrada		Próximo Estado			
5c	10c	15c	25c			5c	10c	15c	25c
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	1	0	0	0
0	0	0	0	1	0	0	1	0	0
0	0	0	0	1	1	0	0	0	1
0	0	0	1	0	0	0	0	0	1
0	0	0	1	0	1	1	0	0	1
0	0	0	1	1	0	0	1	0	1
0	0	0	1	1	1	0	1	1	1
0	0	1	0	0	0	0	0	1	0
0	0	1	0	0	1	1	0	1	0
0	0	1	0	1	0	0	1	1	0
0	0	1	0	1	1	0	0	1	1
0	0	1	1	0	0	0	0	1	1
0	0	1	1	0	1	0	0	0	0

0	0	1	1	1	0	0	0	0	0
0	0	1	1	1	1	0	0	0	0
0	1	0	0	0	0	0	1	0	0
0	1	0	0	0	1	1	1	0	0
0	1	0	0	1	0	0	1	1	0
0	1	0	0	1	1	0	0	1	1
0	1	0	1	0	0	0	1	0	1
0	1	0	1	0	1	0	0	0	0
0	1	0	1	1	0	0	0	0	0
0	1	0	1	1	1	0	0	0	0
0	1	1	0	0	0	0	1	1	0
0	1	1	0	0	1	1	1	1	0
0	1	1	0	1	0	0	1	0	1
0	1	1	0	1	1	0	1	1	1
0	1	1	1	0	0	0	1	1	1
0	1	1	1	0	1	0	0	0	0
0	1	1	1	1	0	0	0	0	0
0	1	1	1	1	1	0	0	0	0

Estado Atual				**Entrada**		**Próximo Estado**			
5c	10c	15c	25c			5c	10c	15c	25c
1	0	0	0	0	0	1	0	0	0
1	0	0	0	0	1	0	1	0	0
1	0	0	0	1	0	0	0	1	0
1	0	0	0	1	1	1	0	0	1
1	0	0	1	0	0	1	0	0	1
1	0	0	1	0	1	0	0	0	0
1	0	0	1	1	0	0	0	0	0
1	0	0	1	1	1	0	0	0	0
1	0	1	0	0	0	1	0	1	0
1	0	1	0	0	1	0	0	0	1
1	0	1	0	1	0	1	0	0	1
1	0	1	0	1	1	1	0	1	1

1	0	1	1	0	0	1	0	1	1
1	0	1	1	0	1	0	0	0	0
1	0	1	1	1	0	0	0	0	0
1	0	1	1	1	1	0	0	0	0
1	1	0	0	0	0	1	1	0	0
1	1	0	0	0	1	1	0	1	0
1	1	0	0	1	0	0	0	0	1
1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	0	1	1	0	1
1	1	0	1	0	1	0	0	0	0
1	1	0	1	1	0	0	0	0	0
1	1	0	1	1	1	0	0	0	0
1	1	1	0	0	0	1	1	1	0
1	1	1	0	0	1	0	0	0	0
1	1	1	0	1	0	0	0	0	0
1	1	1	0	1	1	0	0	0	0
1	1	1	1	0	0	0	0	0	0
1	1	1	1	0	1	0	0	0	0
1	1	1	1	1	0	0	0	0	0
1	1	1	1	1	1	0	0	0	0

Figura 2 - Tabela de estados
Fonte: Elaborado pelos autores

Como próximo passo, foi feita a implementação do código utilizando a linguagem *verilog*, as partes principais do código estão representadas nas Figuras 3,4,5,6,7 e 8. Na Figura 3 temos o nosso módulo “maquina” com os *inputs*, *output* e os registradores declarados que foram utilizados e na última linha estão todos os estados com os seus respectivos valores (retirados da tabela de estados).

```
module maquina(clk, in, out);

input clk, in;
output out;

reg out;
reg [2:0] state;


//parameter saldoInsuf = 2'b00, trocoRec = 2'b01, doceComp = 2'b10, doceEtroco = 2'b11;
parameter 0c = 4'b0000, 5c = 4'b1000, 10c = 4'b0100, 15c = 4'b0010, 20c = 4'b1010 25c = 4'b0001, 30c = 4'b1001, 35c =
```

Figura 3 - Código (Parte1)
Fonte: Elaborado pelos autores

Na Figura 4 temos a representação da nossa “*always*”, em que realiza as instruções sempre que houver um clock sinalizando que houve uma entrada de dados. Dentro dela temos o bloco condicional “*case*” para todos os estados (11 estados), ilustrados tanto na Figura 4 como na Figura 5 e 6. No bloco condicional, é verificado para cada caso, quais são as entradas e conforme o valor da entrada é atribuído um valor específico para o próximo estado. Na Figura 5, nos estados 30c, 35c, 45c e 50c como a saída já é pré-definida não tem o bloco condicional “*if*”. É colocado o comando “*out*” e faz com que o próximo estado seja o 0c, voltando assim, para o início.

```
always @(posedge clk or in )begin
    state <= 0c;
    case (state)
        0c:begin
            if (in == 2'b00 )
                out = 2'b01 ;
            else if (in == 2'b01 )
                state = 5c;
            else if (in == 2'b10)
                state = 10c;
            else if(in == 2'b11)
                state = 25c;
            end
        5c:begin
            if (in == 2'b00 )
                out = 2'b01 ;
            else if (in == 2'b01 )
                state = 10c;
            else if (in == 2'b10)
                state = 15c;
            else if(in == 2'b11)
                state = 30c;
            end
```

Figura 4 - Código (Parte2)
Fonte: Elaborado pelos autores

```
        10c:begin
            if (in == 2'b00 )
                out = 2'b01 ;
            else if (in == 2'b01 )
                state = 15c;
            else if (in == 2'b10)
                state = 20c;
            else if(in == 2'b11)
                state = 35c;
            end
        15c:begin
            if (in == 2'b00 )
                out = 2'b01 ;
            else if (in == 2'b01 )
                state = 20c;
            else if (in == 2'b10)
                state = 25c;
            else if(in == 2'b11)
                state = 35c;
            end
```

Figura 5 - Código (Parte3)
Fonte: Elaborado pelos autores

```
        25c:begin
            if (in == 2'b00 )
                out = 2'b01 ;
            else if (in == 2'b01 )
                state = 30c;
            else if (in == 2'b10)
                state = 35c;
            else if(in == 2'b11)
                state = 50c;
            end
        30c:
            state = 0c;
            out = 2'b10;
        35c:
            state = 0c;
            out = 2'b11;
        45c:
            state = 0c;
            out = 2'b11;
        50c:
            state = 0c;
            out = 2'b11;
    endcase
end
```

Figura 6 - Código (Parte4)
Fonte: Elaborado pelos autores

```
always @(state)
    begin
        case (state)
            0c:
                out = 2'b00; //$display("Saldo Induficiente");
            5c:
                out = 2'b00;//$display("Saldo Induficiente");
            10c:
                out = 2'b00;//$display("Saldo Induficiente");
            15c:
                out = 2'b00;//$display("Saldo Induficiente");
            20c:
                out = 2'b00;//$display("Saldo Induficiente");
            25c:
                out = 2'b00;//$display("Saldo Induficiente");
            30c:
                out = 2'b10;//$display("Doce comprado");
            35c:
                out = 2'b11;//$display("Doce comprado e troco recuperado");
            45c:
                out = 2'b11;//$display("Doce comprado e troco recuperado");
            50c:
                out = 2'b11;//$display("Doce comprado e troco recuperado");
            default:
                out = 2'b00;
        endcase
```

Figura 7 - Código (Parte5)
Fonte: Elaborado pelos autores

```
initial
    begin
      $monitor($time," c %b res %b d %b out  %b  out1  %b",c,d,out);
      #1 d=01;
      #1 d=01;
      #1 d=10;
      #1 d=10;
      $finish ;
    end
endmodule
```

Figura 8 - Código (Parte6)
Fonte: Elaborado pelos autores

A Figura 7 representa as saídas de cada estado a partir do bloco condicional "*case*". Dessa forma é possível identificar qual é o estado atual e o que ele representa (saldo insuficiente, troco recuperado, doce comprado e doce comprado e troco recuperado). Por fim, a Figura 8 mostra o *clock*, as entradas e as saídas na tela ao executar o código.

3. Conclusão

Infelizmente, não foi possível obter um resultado satisfatório para o trabalho. Mas estamos enviando até onde nós conseguimos, pois acreditamos que a lógica que nós seguimos não está tão errada assim.