ID: 112269

Máquina de Estados de *Mealy* em *Verilog*

São José dos Campos - Brasil

Outubro de 2018

ID: 112269

Máquina de Estados de *Mealy* em *Verilog*

Relatório apresentado à Universidade Federal de São Paulo como parte dos requisitos para aprovação na disciplina de Laboratório de Sistemas Computacionais: Circuitos Digitais.

Aluno: Willian Dihanster Gomes de Oliveira

Docente: Prof. Dr. Lauro Paulo da Silva Neto

Universidade Federal de São Paulo - UNIFESP

Instituto de Ciência e Tecnologia - Campus São José dos Campos

São José dos Campos - Brasil

Outubro de 2018

Resumo

Este projeto teve como principal objetivo o desenvolvimento de uma Máquina de Estados (utilizando uma Máquina de *Mealy*) para um contador númerico temporizado na linguagem de descrição de *harware*, *Verilog*. A máquina desenvolvida possui 4 funções, que podem ser controladas pelas entradas *UP* e *DOWN*, sendo elas: contagem crescente, contagem decrescente, manter a contagem e a função *blank* (onde todos os LEDs são apagados). Além disso, o contador segue a sequência númerica 6-9-0-2-4-6-5-3-8, fazendo uma transição de estados a cada $1s$ e sua saída é mapeada para o display de 7 segmentos do Kit FPGA. Para isso, será construído as 3 partes da máquina de estados (registradores, próximo estado e saída), um temporizador e um decodificador para o display de 7 segmentos, em *Verilog*.

Palavras-chaves: Circuitos Digitais, Contador Automático, Máquina de *Mealy*, Kit FPGA, *Verilog*

Lista de ilustrações

Lista de tabelas

Sumário

1 Introdução

Circuitos eletrônicos são usualmente separados entre duas categorias: digitais e analógicos. Enquanto que na digital, trabalha-se com grandezas com valores discretos, na analógica há o uso de grandezas contínuas (1).

Sendo assim, circuitos e sistemas digitais trabalham com dois estados possíveis, que são representados por dois níveis de tensão diferentes: um ALTO e um BAIXO, que também podem ser representados por '0' e '1'. Dessa forma, circuitos digitais servem como base para diversas aplicações, como por exemplo na construção de computadores, na automação, robótica e etc.

Um exemplo mais simples, são as máquinas de estados finitos, que representam comportamentos de um sistema por um número finito de estados e de transições (2). Assim, pode-se representar contadores, como o desenvolvido neste projeto, que fazem a contagem com uma sequência númerica, fazendo uma transição de estados a cada $1s$, de acordo com a entrada definida pelo usuário.

O trabalho está organizado como segue: Capítulo 2 detalha os objetivos do trabalho, Capítulo 3, a fundamentação teórica, Capítulo 4, o desenvolvimento, Capítulo 5, os resultados obtidos e discussões e por fim, no Capítulo 6 as considerações finais.

2 Objetivos

2.1 Geral

Este projeto tem como principal objetivo o desenvolvimento de um contador númerico atráves de uma Máquina de Estados de *Mealy*. Esta maquina será implementada em linguagem *Verilog* e o contador segue a sequência 6-9-0-2-4-6-5-3-8, com transição de estados a cada $1s$ e com 4 formas de contagem.

2.2 Específico

Para o pleno desenvolvimento do objetivo desejado, há uma sequência de passos, definidos a seguir:

- Confecção do diagrama de estados para a sequência definida.
- Implementação em *Verilog*, de uma Máquina de Estados de *Mealy*, sendo essa máquina composta por 3 partes principais (registradores, próximo estado e saída).
- Implementação do módulo do divisor de frequências, que simula um temporizador de $1s$, em *Verilog*.
- Decodificação da saída do estado da máquina para seu valor correspondente (em binário) da sequência numérica, para o display de 7 segmentos.
- Criação de um módulo principal para organização e junção das partes do projeto.
- Avaliação com testes de *waveforms* no *software Quartus Prime* para cada bloco do projeto.
- Avaliação do programa principal (todas as partes integradas) no Kit FPGA.

3 Fundamentação Teórica

Neste capítulo serão detalhados alguns dos principais conceitos utilizados para a realização deste projeto.

3.1 Princípios de Circuitos Digitais

Circuitos eletrônicos são geralmente separados entre duas categorias: digitais e analógicos. Na digital, há o uso de grandezas com valores discretos, já na analógica, de grandezas contínuas (1).

Sendo assim, circuitos e sistemas digitais trabalham com dois estados possíveis, que são representados por dois níveis de tensão diferentes: um ALTO e um BAIXO, que também podem ser representados por '0' e '1'. Em circuitos digitais há duas classes principais de circuitos: circuitos combinacionais e circuitos sequencias.

Em circuitos combinacionais há o uso de portas lógicas interconectadas para produzir uma saída especificada, usando a combinação de variáveis de entrada, sem o envolvimento de armazenamento de dados. Já os circuitos sequenciais há a existência de uma seção com lógica combinacional e de uma seção de memória (*flip-flops*).

Sendo assim, fazendo o uso de porta lógicas, tabela-verdade e álgebra booleana é possível desenvolver sistemas de circuitos digitais para as mais diversas aplicações.

3.2 *Flip-flops*

Um *flip-flop* ou multivibrador biestável, pode ser definido como um elemento de memória e, geralmente, é feito de uma configuração de portas lógicas. Comumente, possuem duas saídas: Q e $\bar{Q}$, a saída normal e saída invertida, respectivamente. No estado *ALTO*, temos $Q = 1$ e $\bar{Q} = 0$, e este estado pode ser chamado de *SET*. Já no estado *BAIXO*, temos $Q = 0$ e $\bar{Q} = 1$, e pode ser chamado de *CLEAR* ou *RESET*. (3) Há diversos tipos de *flip-flops*, como o tipo *D*, *J-K*, *T* e podem ser utilizados por exemplo, para a criação de contadores, registradores ou temporizadores.

3.3 Contadores

Contadores são circuitos que podem ser implementados a partir de N *flip-flops* conectados, sendo síncronos ou assíncronos. O valor de N e a forma na qual são conectados

determina o número de estados (ou módulo) e a sequência de estados do contador. Por exemplo, com N *flip-flops* é possível implementar um contador de 0 até $2^N - 1$. (1)

3.4 Temporizadores

Em certas aplicações, é utilizado a frequência de *clock* do dispositivo. Mas essa frequência pode ser muita alta para determinados casos. Sendo assim, um circuito temporizador pode ser usado. Uma implementação possível é a divisão de frequências, em que são usados *flip-flops* ou contadores para passar a informação, gerando um certo *delay*. (1)

3.5 Máquina de Estados Finitos

As máquinas de estados finitos representam comportamentos de um sistema por um número finito de estados e de transições (2). Há dois principais modelos amplamente utilizadas, sendo elas: Máquina de *Moore* e a Máquina de *Mealy*. Estas máquinas de estados possuem 3 funções principais, sendo elas: registradores, próximo estado e saída.

Nos registradores, o objetivo é guardar o estado atual da máquina ou também resetá-lo, caso seja requirido. Na função de próximo estado, deve-se verificar as entradas e o estado atual e então, definir qual será o próximo estado da máquina, no próximo ciclo. Já na função de saída, o valor que cada estado representa é decodificado e essa saída pode por exemplo, ser enviada ao display de 7 segmentos.

As principais diferenças entre as duas é sua função de saída, onde na Máquina de *Moore*, depende apenas do estado atual, já na Máquina de *Mealy*, a saída depende do estado atual e das entradas.

3.6 FPGA - *Field-Programmable Gate Array*

O FPGA - *field-programmable gate array* (matriz de portas programáveis) é um dispositivo lógico programável em que se é possível trabalhar e simular circuitos digitais. O Kit FPGA disponível no Laboratório de Circuitos Digitais é o Kit Altera DE2-115 Development and Education Board, com o FPGA Altera Cyclone IV E: EP4CE115F29C7. Esse kit contém, também, um display de 7 segmentos, que será detalhado a seguir. (4)

3.7 Display de 7 Segmentos

Um display de 7 segmentos é uma placa composta por segmentos com LEDs que podem ser ligados ou desligados individualmente, e podem, por exemplo, serem utilizados para a visualização de números decimais de 0 a 9.

3.8 *Intel Quartus Prime Software*

o *Intel Quartus Prime Software* é um software da *Intel* para o desenvolvimento de sistemas programáveis. Com ele é possível, por exemplo, criar circuitos atráves de esquemáticos ou com alguma linguagem de descrição de *hardware* como a *Verilog*. (5)

3.9 *Verilog*

O *Verilog* é uma *HDL - Hardware Description Language*, ou seja, uma linguagem de descrição de *hardware*. Isto é, não é uma linguagem de programação comum, mas serve para descrever circuitos, por exemplo. Neste caso, pode-se em vez de fazer tabelas verdades e etc, tentar reproduzir a lógica do circuito.

4 Desenvolvimento

Neste capítulo, serão detalhados os passos para a implementação da máquina de estados de *Mealy* que represente a sequência ciclíca 6-9-0-2-4-6-5-3-8, com transição de estados a cada 1s e com 4 formas de contagem diferentes, como configurado na Tabela 1.

Tabela 1 – Exemplo de entradas e forma de contagem

Contagem	UP	$DOWN$
Mantém	0	0
Descrecente	0	1
Crescente	1	0
Blank	1	1

Fonte: O Autor

Portanto, considerando que estamos no estado inicial A, temos que:

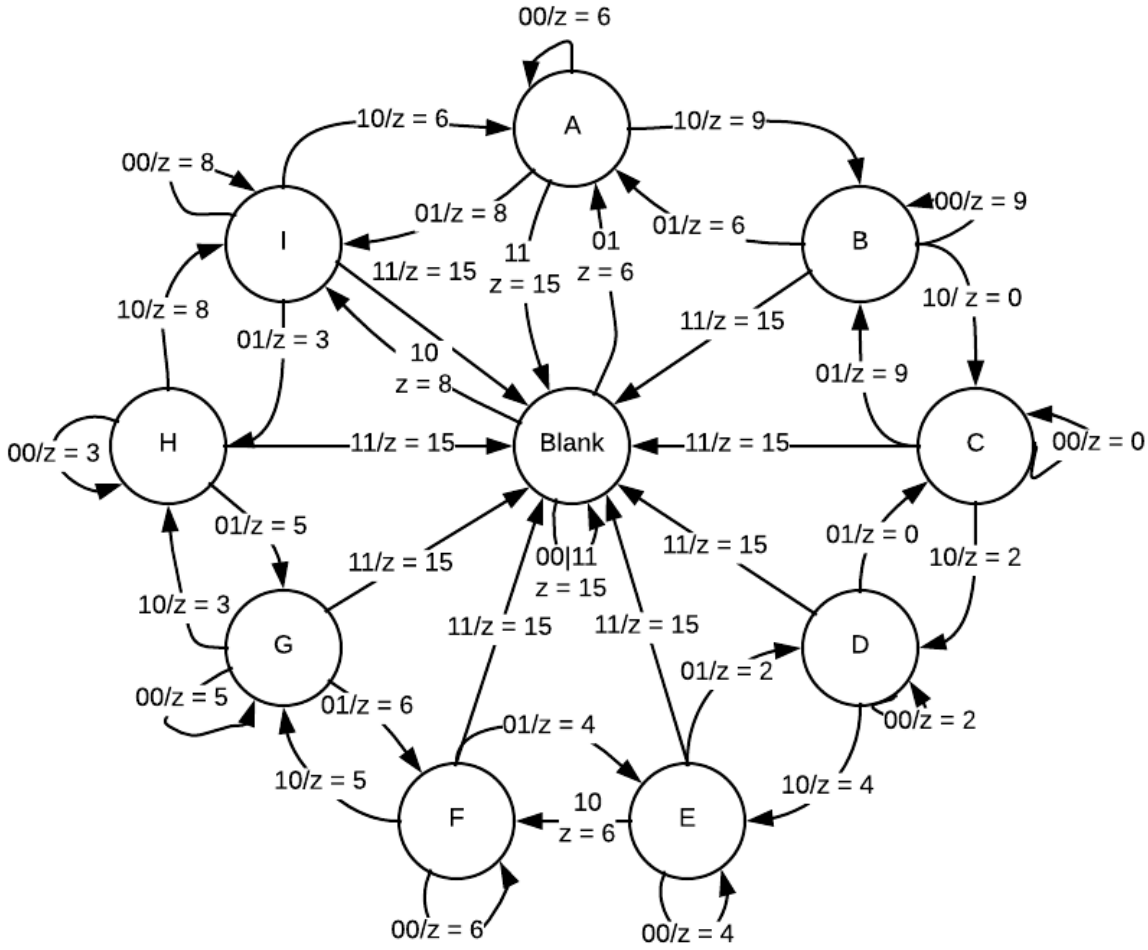
- se $UP = 0$ e $DOWN = 0$, o estado A será mantido.
- se $UP = 0$ e $DOWN = 1$, a máquina volta para o estado anterior, ou seja, a sequência apresentada será 8-3-5-6-4-2-0-9-6.
- se $UP = 1$ e $DOWN = 0$, temos a sequência crescente, ou seja, a sequência ciclíca será 6-9-0-2-4-6-5-3-8.
- se $UP = 1$ e $DOWN = 1$, a máquina mudará para o estado *Blank*, onde todos os LEDs são apagados.

Além disso, se a entrada *reset* for requisitada, a contagem reiniciará do primeiro estado, A.

4.1 Diagrama de Estados

Inicialmente, para desenvolvimento deste projeto, foi realizada a confecção do diagrama de estados com as configurações possíveis para as entradas (UP e $DOWN$) e os estados da máquina, que pode ser conferido na Figura 1, a seguir. Este diagrama foi feito com a ajuda do *software on-line, Lucid Chart* (6).

Figura 1 – Diagrama de Estados



Fonte: Autor

Sendo assim, foi usado uma esquema de codificação de 4 *bits* para a representação dos estados possíveis da máquina. Cada estado tem sua representação binária, conforme a Tabela 2.

Tabela 2 – Tabela de Codificação de Estados.

Número	Estado	$Q3$	$Q2$	$Q1$	$Q0$
6	A	0	0	0	0
9	B	0	0	0	1
0	C	0	0	1	0
2	D	0	0	1	1
4	E	0	1	0	0
6	F	0	1	0	1
5	G	0	1	1	0
3	H	0	1	1	1
8	I	1	0	0	0
Blank	*Blank*	1	0	0	1

Fonte: O Autor

Em *Verilog*, definiu-se os estados e seus respectivos valores por *parameter* da linguagem, como pode ser visto na Figura 2.

Figura 2 – Definição da Codificação dos Estados em *Verilog*

```
parameter
    A = 4'b0000,
    B = 4'b0001,
    C = 4'b0010,
    D = 4'b0011,
    E = 4'b0100,
    F = 4'b0101,
    G = 4'b0110,
    H = 4'b0111,
    I = 4'b1000,
    Blank = 4'b1001;
```

Fonte: Autor

4.2 Função do Próximo Estado

Um dos principais componentes de uma máquina de estados finitos é a função de próximo estado. Esta função tem como objetivo ditar qual o próximo estado em que a máquina deve estar. Temos, então, na Tabela 3, as possibilidades de entradas, estado atual e próximo estado.

Tabela 3 – Tabela de próximo estado, dado uma entrada.

Estado Atual	Entradas		Estado Atual				Próximo Estado				Próximo Estado
	UP	$DOWN$	$Q3$	$Q2$	$Q1$	$Q0$	$D3$	$D2$	$D1$	$D0$	
A	0	0	0	0	0	0	0	0	0	0	A
B	0	0	0	0	0	1	0	0	0	1	B
C	0	0	0	0	1	0	0	0	1	0	C
D	0	0	0	0	1	1	0	0	1	1	D
E	0	0	0	1	0	0	0	1	0	0	E
F	0	0	0	1	0	1	0	1	0	1	F
G	0	0	0	1	1	0	0	1	1	0	G
H	0	0	0	1	1	1	0	1	1	1	H
I	0	0	1	0	0	0	1	0	0	0	I
Blank	0	0	1	0	0	1	1	0	0	1	*Blank*
A	0	1	0	0	0	0	0	0	0	1	I
B	0	1	0	0	0	1	0	0	1	0	A
C	0	1	0	0	1	0	0	0	1	1	B
D	0	1	0	0	1	1	0	1	0	0	C
E	0	1	0	1	0	0	0	1	0	1	D
F	0	1	0	1	0	1	0	1	1	0	E
G	0	1	0	1	1	0	0	1	1	1	F
H	0	1	0	1	1	1	1	0	0	0	G
I	0	1	1	0	0	0	0	0	0	0	H
Blank	0	1	1	0	0	1	0	0	0	0	A
A	1	0	0	0	0	0	1	0	0	0	B
B	1	0	0	0	0	1	0	0	0	0	C
C	1	0	0	0	1	0	0	0	0	1	D
D	1	0	0	0	1	1	0	0	1	0	E
E	1	0	0	1	0	0	0	0	1	1	F
F	1	0	0	1	0	1	0	1	0	0	G
G	1	0	0	1	1	0	0	1	0	1	H
H	1	0	0	1	1	1	0	1	1	0	I
I	1	0	1	0	0	0	0	1	1	1	A
Blank	1	0	1	0	0	1	0	0	0	0	A
A	1	1	0	0	0	0	1	0	0	1	*Blank*
B	1	1	0	0	0	1	1	0	0	1	*Blank*
C	1	1	0	0	1	0	1	0	0	1	*Blank*
D	1	1	0	0	1	1	1	0	0	1	*Blank*
E	1	1	0	1	0	0	1	0	0	1	*Blank*
F	1	1	0	1	0	1	1	0	0	1	*Blank*
G	1	1	0	1	1	0	1	0	0	1	*Blank*
H	1	1	0	1	1	1	1	0	0	1	*Blank*
I	1	1	1	0	0	0	1	0	0	1	*Blank*
Blank	1	1	1	0	0	1	1	0	0	1	*Blank*

Fonte: O Autor

Para a implementação desta função, podemos definir um bloco *always* sensível ao estado e as entradas *UP* e *DOWN*. Em seguida, um *case* para todos os estados possíveis e dentro deles, verificar quais são as entradas no momento e definir o próximo estado.

Na Figura 3, temos uma demonstração do bloco responsável pela função de próximo estado da máquina.

Figura 3 – Bloco para função de próximo estado

```
always@(estado or UP or DOWN)
   begin
      case(estado)
      A: if(UP  ==  1 && DOWN == 0)
         prox_estado = B;
         else if(UP == 0 && DOWN == 0)
         prox_estado = A;
         else if(UP == 0 && DOWN == 1)
         prox_estado = I;
      B: if(UP == 1 && DOWN == 0)
         prox_estado = C;
         else if(UP == 0 && DOWN == 0)
         prox_estado = B;
         else if(UP == 0 && DOWN == 1)
         prox_estado = A;
      C: if(UP == 1 && DOWN == 0)
         prox_estado = D;
         else if(UP == 0 && DOWN == 0)
         prox_estado = C;
         else if(UP == 0 && DOWN == 1)
         prox_estado = B;
      D: if(UP == 1 && DOWN == 0)
         prox_estado = E;
         else if(UP == 0 && DOWN == 0)
         prox_estado = D;
         else if(UP == 0 && DOWN == 1)
         prox_estado = C;
```

Fonte: Autor

Por exemplo, caso estejamos no estado A, e a entrada no momento é $UP = 1$ e $DOWN = 0$, o próximo estado deve ser B. O código completo está disponível no Apêndice A.

4.3 Registradores

Uma outra parte importante de uma máquina de estados, são os registradores, que têm como principal função armazenar o estado atual. Além disso, essa função pode conter uma entrada de *reset*, reinicializando a contagem da máquina.

Em *Verilog*, esta função pode ser implementada criando um bloco *always* sensível ao *clock*. Neste bloco deve-se verificar se a entrada de *reset* está ativa, se estiver, então o estado (que é um registrador) é setado para o estado inicial, A. Senão, o estado atual é agora o próximo estado.

Uma visão geral do bloco desenvolvido para esta função, pode ser conferido na Figura 4.

Figura 4 – Circuito para os Registradores.

```
always@(posedge clock)
   begin
      if(reset)
         estado <= A;
      else
         estado <= prox_estado;
   end
```

Fonte: Autor

4.4 Função Saída

Em uma Máquina de *Mealy*, a saída depende do estado atual e das entradas. Sendo assim, temos na Figura 5, o bloco responsável pela função de saída.

Ou seja, depois que o próximo estado é definido (em um bloco sensível as entradas ou estados), há um bloco *always* para a saída, com o estado e as entradas na lista de sensibilidade. Nesse módulo há um *case*, onde é verificado o estado da máquina e entao o registrador de saida z é setado com o valor que o estado representa.

Figura 5 – Circuito para função de saída

```
always@(estado or UP or DOWN)
   begin
      z = 4'b0000;
      case(estado)
         A: z = 4'd6;
         B: z = 4'd9;
         C: z = 4'd0;
         D: z = 4'd2;
         E: z = 4'd4;
         F: z = 4'd6;
         G: z = 4'd5;
         H: z = 4'd3;
         I: z = 4'd8;
         Blank: z = 4'd15;
         default z = 4'bxxxx;
      endcase
   end
```

Fonte: Autor

4.5 Junção dos Componentes da Máquina de Estados

As 3 partes principais da máquina (registradores, próximo estado e saída) foram separados em blocos, mas agrupados em um único módulo, representando a Máquina de *Mealy*.

Este módulo tem como parâmetro o *clock*, *reset*, as entradas *UP* e *DOWN* e a variável de saída. Uma visão geral do módulo pode ser conferido na Figura 6 e o código completo está disponível no Apêndice A.

Neste módulo, depois das definições das entradas e saída, há uma definição dos parâmetros ou constantes, que representarão os estados. Em seguida, há o bloco da função de próximo estado, mais embaixo o bloco de saída da máquina e por fim, o bloco de registradores, em que verificado se o reset está ativo ou se devemos atualizar o estado atual.

Figura 6 – Máquina de Estados

```
module MaquinaDeMealy(clock, reset, UP, DOWN, z);
   input clock, reset, UP, DOWN;
   output reg [3:0] z;
   reg [3:0] estado, prox_estado;

   parameter
      A = 4'b0000,
      B = 4'b0001,
      C = 4'b0010,
      D = 4'b0011,
      E = 4'b0100,
      F = 4'b0101,
      G = 4'b0110,
      H = 4'b0111,
      I = 4'b1000,
      Blank = 4'b1001;

   always@(estado or UP or DOWN)
      begin
         case(estado)
            A: if(UP  ==  1 && DOWN == 0)
               prox_estado = B;
               else if(UP == 0 && DOWN == 0)
               prox_estado = A;
               else if(UP == 0 && DOWN == 1)
               prox_estado = I;
               else if(UP == 1 && DOWN == 1)
               prox_estado = Blank;
            B: if(UP == 1 && DOWN == 0)
               prox_estado = C;
               else if(UP == 0 && DOWN == 0)
               prox_estado = B;
               else if(UP == 0 && DOWN == 1)
               prox_estado = A;
               else if(UP == 1 && DOWN == 1)
               prox_estado = Blank;
```

Fonte: Autor

4.6 Temporizador

Uma outra parte importante para o desenvolvimento deste projeto é o temporizador, já que a frequência de *clock* do Kit FPGA é alta para que algo seja visualizado no display. Sendo assim, uma ideia é dividir essa frequência para valores mais baixos.

Neste projeto, a ideia é que a transição de estados ocorra a cada $1s$. Sendo assim,

como a frequência do *clock* é de 50MHz, podemos contar de 0 até 50.000.000, gerando um sinal alto. Isto é, basta criarmos um registrador que guarda o valor da soma (de 0 até 50.000.000), e cada chamado da função, fazer a verificação. Caso a soma seja igual a 50.000.000, então lançamos um sinal alto e reiniciamos o registrador para 0. Caso contrário, o registrador é incrementado em 1 unidade e o sinal continua em 0.

O módulo do temporizador utilizado foi a versão disponibilizada pelo professor, que pode ser visualizada na Figura 7 e também no Apêndice C.

Figura 7 – Modulo do Temporizador

```
module DivisorFrequencias(clk, S);
   input clk;
   output reg S;
   reg [25:0] OUT;

   always @ (posedge clk)
      if (OUT == 26'd50000000)
         begin
            OUT <= 26'd0;
            S <= 1;
         end
      else
         begin
            OUT <= OUT + 1;
            S <= 0;
         end

endmodule
```

Fonte: Autor

4.7 Display de 7 Segmentos

Para mapear as saídas da máquina para serem mostradas no Kit FPGA, deve-se decodificar essas saídas (numeros binários) para os 7 segmentos do display, conforme a tabela 4.

Tabela 4 – Tabela de Saída, dado um estado.

Entradas					Saídas						
W	X	Y	Z	Decimal	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	1	0	1	1	0	0	0	0
0	0	1	0	2	1	1	0	1	1	0	1
0	0	1	1	3	1	1	1	1	0	0	1
0	1	0	0	4	0	1	1	0	0	1	1
0	1	0	1	5	1	0	1	1	0	1	1
0	1	1	0	6	1	0	1	1	1	1	1
0	1	1	1	7	1	1	1	0	0	0	0
1	0	0	0	8	1	1	1	1	1	1	1
1	0	0	1	9	1	1	1	1	0	1	1
1	1	1	1	*Blank*	0	0	0	0	0	0	0

Fonte: Autor

Em questões de implementação, pode-se criar um novo módulo, que receba como entrada a saída da máquina e converta essa entrada nos segmentos que devem ser ligados. Para isso, há um bloco *always* em que a entrada é verificada em um *case*, que define quais segmentos devem estar ligados ou não, sendo o inverso dos bits da Tabela 4, já que o display é ativo em '0'. O módulo desenvolvido pode ser conferido na Figura 8. O código também está disponível no Apêndice B.

Figura 8 – Módulo que mapeia para o display de 7 segmentos

```
module Decodificador(entrada, segmentos);
   input [3:0] entrada;
   output reg[0:6] segmentos;

   always@(*)
      begin
         case(entrada)
            4'b0000: segmentos = 7'b0000001;
            4'b0001: segmentos = 7'b1001111;
            4'b0010: segmentos = 7'b0010010;
            4'b0011: segmentos = 7'b0000110;
            4'b0100: segmentos = 7'b1001100;
            4'b0101: segmentos = 7'b0100100;
            4'b0110: segmentos = 7'b0100000;
            4'b0111: segmentos = 7'b0001111;
            4'b1000: segmentos = 7'b0000000;
            4'b1001: segmentos = 7'b0000100;
            default: segmentos = 7'b1111111;
         endcase
      end

endmodule
```

Fonte: Autor

5 Resultados Obtidos e Discussões

Finalizada a implementação dos componentes e a junção da máquina de estados, o temporizador e o decodificador para o display de 7 segmentos, criou-se um modulo principal capaz de trabalhar com essas partes, conforme a Figura 9 e também está disponível no Apêndice D.

Figura 9 – Módulo principal.

```
module Principal(clock, reset, UP, DOWN, segmentos, saida_estado);
   input clock, reset, UP, DOWN;
   output wire [0:6] segmentos;
   output wire [0:3] saida_estado;
   wire novo_clock;

   DivisorFrequencias df(clock, novo_clock);
   MaquinaMealy mq(novo_clock, reset, UP, DOWN, saida_estado);
   Decodificador d(saida_estado, segmentos);

endmodule
```

Fonte: Autor

Neste módulo, as entradas são o *clock*, reset, *UP* e *DOWN*. Já como saída temos os segmentos e a saída do estado. Também há uma variável que funciona como o novo *clock* temporizado.

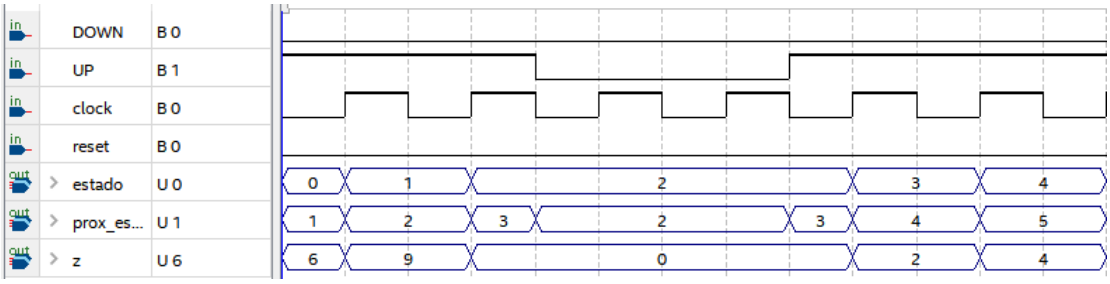
Neste módulo principal, a entrada do *clock* é o *clock* do Kit FPGA de 50MHz. Então, esse sinal é enviado para o divisor de frequências, a máquina de estados utiliza esse novo sinal e em seguida a saída da máquina é enviada para o decodificador.

Para validar o projeto desenvolvido, foram feitas simulações atráves de $waveforms$ no *software Quartus*. Além disso, também foi feita uma simulação e apresentação do pleno funcionamento do projeto no Kit FPGA. As simulações de forma de onda podem ser conferidas a seguir.

5.1 Mantém - $UP = 0$ e $DOWN = 0$

Com $UP = 0$ e $DOWN = 0$, é esperado que a sequência pare e mantenha seu resultado, como mostra a simulação da Figura 10, onde a sequência seguia de forma crescente, até que a entrada mudou para o mantém, onde ficou parada no estado 2, até que a entrada mudasse.

Figura 10 – Simulação contagem mantém.

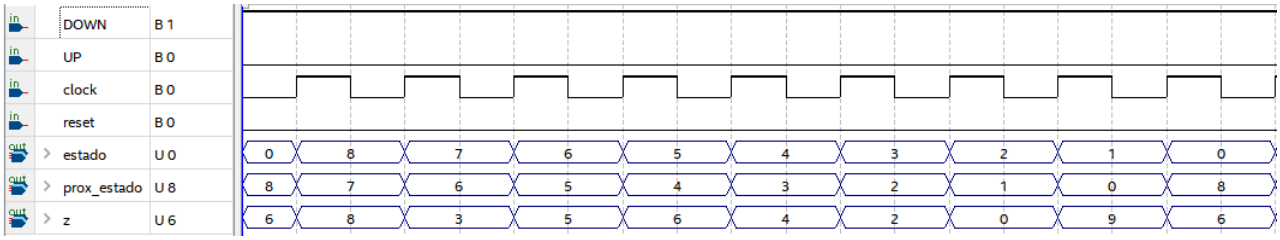


Fonte: Autor

5.2 Decrescente - $UP = 0$ e $DOWN = 1$

Para $UP = 0$ e $DOWN = 1$, a sequência apresentada será da forma decrescente. Sendo assim, a sequência 8-3-5-6-4-2-0-9-6 deve ser apresentada, como pode ser comprovado pela simulação da Figura 11.

Figura 11 – Simulação contagem decrescente.

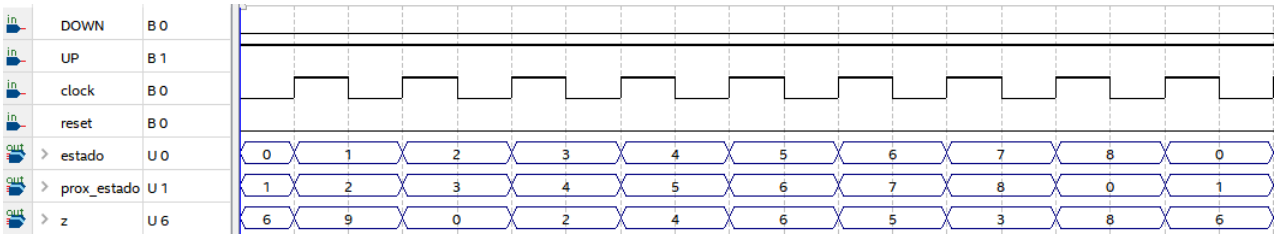


Fonte: Autor

5.3 Crescente - $UP = 1$ e $DOWN = 0$

Para $UP = 1$ e $DOWN = 0$, a contagem será da forma crescente. Sendo assim, a sequência 6-9-0-2-4-6-5-3-8 deve ser apresentada, como pode ser visualizado na simulação da Figura 12.

Figura 12 – Simulação contagem crescente.

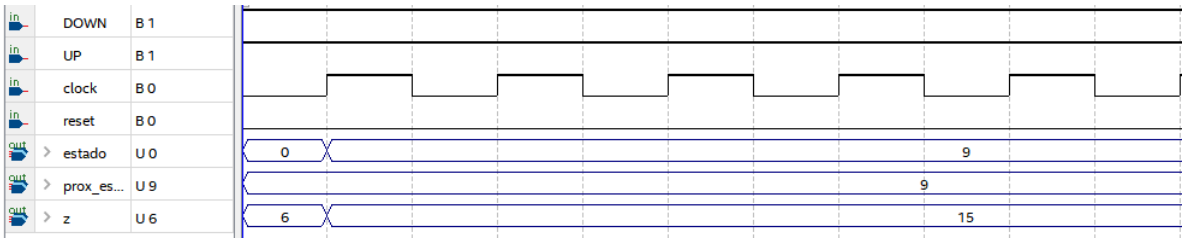


Fonte: Autor

5.4 *Blank* - $UP = 1$ e $DOWN = 1$

Já para $UP = 1$ e $DOWN = 1$, é esperado que a máquina avance para um estado de *blank*, onde todos os LEDs estarão apagados. Isso pode ser implementado atráves de um estado inválido para o display, como o número 15, representado na simulação da Figura 13. Nesse exemplo, a entrada foi alterarada para a entrada *blank* e então, a máquina muda para o estado 9, com valor 15.

Figura 13 – Simulação contagem *blank*.

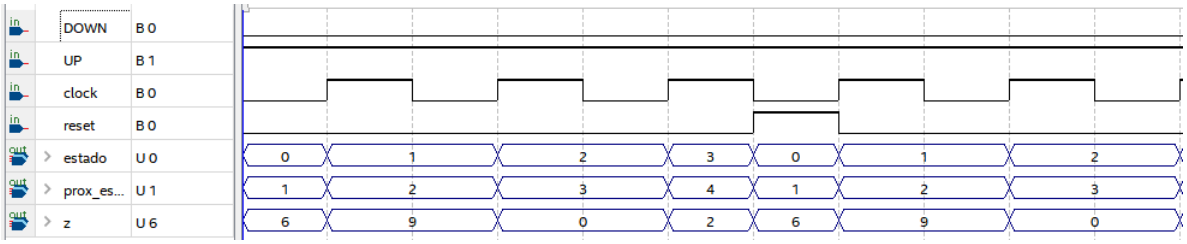


Fonte: Autor

5.5 Uso do *reset*

Ainda, há uma opção de *reset*, onde não importando qual seja o estado atual da máquina, ela deve voltar ao estado inicial (A = 6), e então, após desligado o *reset*, a contagem deve prosseguir. Essa simulação pode ser visualizada na Figura 14, onde a contagem seguia a ordem crescente e um *reset* foi acionado, forçando a máquina a voltar ao estado inicial e prosseguindo a contagem crescente, como o esperado.

Figura 14 – Simulação contagem *reset*.



Fonte: Autor

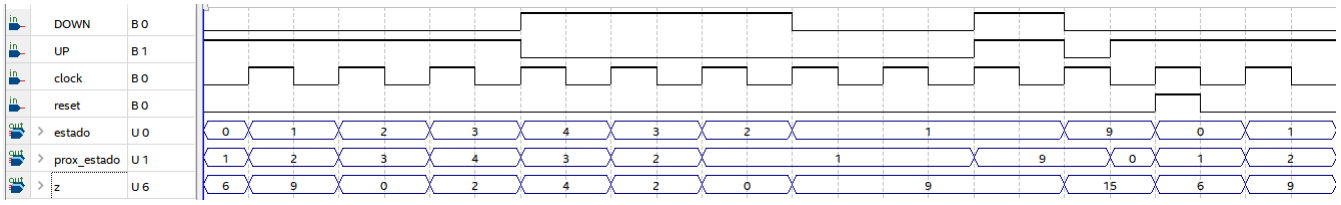
5.6 Contagem Genérica

Foi simulado, também, um caso mais genérico, com uma mistura de combinações das entradas em uma única simulação, como mostrado na Figura 15.

Pode-se notar que a contagem se mantém crescente, com $UP = 1$ e $DOWN = 0$, até que há uma troca, para $UP = 0$ e $DOWN = 1$, e a contagem segue de modo decrescente.

Após isso, a entrada $DOWN$ é trocada para $DOWN = 0$, e a contagem se mantém no estado atual. Então, há uma nova troca, e agora $UP = 1$ e $DOWN = 1$, e então, a máquina vai para o estado *blank*. Por último, temos $UP = 1$ e $DOWN = 0$, onde a contagem segue crescente, até que o *reset* é acionado, e então, a contagem é reiniciada do primeiro estado, conforme esperado.

Figura 15 – Simulação contagem genérica

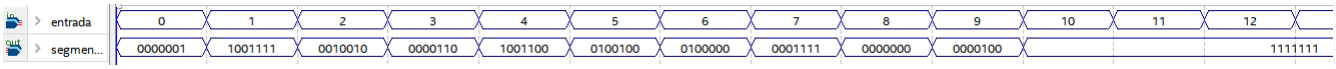


Fonte: Autor

5.7 Display de 7 Segmentos

Para este bloco, foram testadas os números possíveis (0-9), como pode ser visto pela *waveforms* da Figura 16. Como o esperado, o circuito opera corretamente. Por exemplo, o número 1 tem todos os segmentos ligados (em 0) exceto pelo último segmento. Já o número 8 tem todos segmentos ligados.

Figura 16 – Simulação display de 7 segmentos



Fonte: Autor

5.8 Simulação no Kit FPGA

A sintetização deste projeto no Kit FPGA foi realizada e apresentada ao professor, no dia 12/12.

6 Considerações Finais

Com o desenvolvimento deste projeto, foi possível unir todo o conhecimento adquirido na disciplina de Circuitos Digitais e os do Laboratório de Sistemas Computacionais: Circuitos Digitais.

Após a modelagem da máquina, as simulações e os resultados obtidos, pode-se concluir o sucesso em sua implementação, pois a máquina possui todas as suas funcionalidades sendo executadas de acordo com o esperado.

Sendo assim, pode-se concluir a eficiência de uma máquina de estados para o problema apresentado e também suas diversas aplicações. Assim como, a de de um Kit FPGA para essas simulações e ver sua aplicabilidade em outros sistemas.

Em comparação com o projeto anterior (desenvolvido atráves de circuitos esquemáticos), utilizar uma linguagem de descrição de *hardware* como a *Verilog* para seu desenvolvimento, é certamente mais fácil e rápido.

Além disso, com o desenvolvimento deste projeto, foi possível ver mais na prática uma aplicação de circuitos digitais, ter novas noções de *hardware*, integração *software* e *hardware* e o conhecimento de uma linguagem de descrição de *hardware*.

Referências

1 FLOYD, T. *Sistemas digitais: fundamentos e aplicações*. [S.l.]: Bookman Editora, 2009. Citado 3 vezes nas páginas 7, 11 e 12.

2 GILL, A. et al. Introduction to the theory of finite-state machines. McGraw-Hill, 1962. Citado 2 vezes nas páginas 7 e 12.

3 TOCCI, R. J.; WIDMER, N. S.; MOSS, G. L. *Sistemas digitais: princípios e aplicações*. [S.l.]: Prentice Hall, 2003. v. 8. Citado na página 11.

4 FPGA.Altera DE2-115 Development and Education Board. <https://www.ee.ryerson.ca/~courses/coe608/labs/DE2_115_User_Manual.pdf>. Acessado em 22/10/2018. Citado na página 12.

5 QUARTUS PRIME. *Intel Quartus Prime Software Suite*. <https://www.intel.com/content/www/us/en/software/programmable/quartus-prime/overview.html>. Acessado em 23/10/2018. Citado na página 13.

6 LUCID CHART. *Software para a criação de de fluxogramas e diagramas on-line*. <https://www.lucidchart.com>. Acessado em 20/10/2018. Citado na página 15.

Apêndices

APÊNDICE A – MaquinaMealy.v

Código do arquivo MaquinaMealy.v

```
module MaquinaMealy(clock, reset, UP, DOWN, z, estado, prox_estado);
        input clock, reset, UP, DOWN;
        output reg [3:0] z;
        output reg [3:0] estado, prox_estado;

        parameter
                A = 4'b0000,
      B = 4'b0001,
                C = 4'b0010,
                D = 4'b0011,
                E = 4'b0100,
                F = 4'b0101,
                G = 4'b0110,
                H = 4'b0111,
                I = 4'b1000,
                Blank = 4'b1001;

        always@(estado or UP or DOWN)
                begin
                        case(estado)
                                A: if(UP  ==  1 && DOWN == 0)
                                        prox_estado = B;
                                        else if(UP == 0 && DOWN == 0)
                                        prox_estado = A;
                                        else if(UP == 0 && DOWN == 1)
                                        prox_estado = I;
                                        else if(UP == 1 && DOWN == 1)
                                        prox_estado = Blank;
                                B: if(UP == 1 && DOWN == 0)
                                        prox_estado = C;
                                        else if(UP == 0 && DOWN == 0)
                                        prox_estado = B;
                                        else if(UP == 0 && DOWN == 1)
                                        prox_estado = A;
                                        else if(UP == 1 && DOWN == 1)
                                        prox_estado = Blank;
                                C: if(UP == 1 && DOWN == 0)
                                        prox_estado = D;
                                        else if(UP == 0 && DOWN == 0)
                                        prox_estado = C;
                                        else if(UP == 0 && DOWN == 1)
                                        prox_estado = B;
                                        else if(UP == 1 && DOWN == 1)
                                        prox_estado = Blank;
                                D: if(UP == 1 && DOWN == 0)
                                        prox_estado = E;
                                        else if(UP == 0 && DOWN == 0)
                                        prox_estado = D;
                                        else if(UP == 0 && DOWN == 1)
                                        prox_estado = C;
                                        else if(UP == 1 && DOWN == 1)
                                        prox_estado = Blank;
```

```
53                              E: if(UP == 1 && DOWN == 0)
54                                      prox_estado = F;
55                                      else if(UP == 0 && DOWN == 0)
56                                      prox_estado = E;
57                                      else if(UP == 0 && DOWN == 1)
58                                      prox_estado = D;
59                                      else if(UP == 1 && DOWN == 1)
60                                      prox_estado = Blank;
61                              F: if(UP == 1 && DOWN == 0)
62                                      prox_estado = G;
63                                      else if(UP == 0 && DOWN == 0)
64                                      prox_estado = F;
65                                      else if(UP == 0 && DOWN == 1)
66                                      prox_estado = E;
67                                      else if(UP == 1 && DOWN == 1)
68                                      prox_estado = Blank;
69                              G: if(UP == 1 && DOWN == 0)
70                                      prox_estado = H;
71                                      else if(UP == 0 && DOWN == 0)
72                                      prox_estado = G;
73                                      else if(UP == 0 && DOWN == 1)
74                                      prox_estado = F;
75                                      else if(UP == 1 && DOWN == 1)
76                                      prox_estado = Blank;
77                              H: if(UP == 1 && DOWN == 0)
78                                      prox_estado = I;
79                                      else if(UP == 0 && DOWN == 0)
80                                      prox_estado = H;
81                                      else if(UP == 0 && DOWN == 1)
82                                      prox_estado = G;
83                                      else if(UP == 1 && DOWN == 1)
84                                      prox_estado = Blank;
85                              I: if(UP == 1 && DOWN == 0)
86                                      prox_estado= A;
87                                      else if(UP == 0 && DOWN == 0)
88                                      prox_estado = I;
89                                      else if(UP == 0 && DOWN == 1)
90                                      prox_estado = H;
91                                      else if(UP == 1 && DOWN == 1)
92                                      prox_estado = Blank;
93                              Blank: if(UP == 1 && DOWN == 0)
94                                      prox_estado = A;
95                                      else if(UP == 0 && DOWN == 0)
96                                      prox_estado = Blank;
97                                      else if(UP == 0 && DOWN == 1)
98                                      prox_estado = I;
99                                      else if(UP == 1 && DOWN == 1)
100                                     prox_estado = Blank;
101                             default prox_estado = 4'bxxxx;
102                     endcase
103             end
104
105     always@(estado or UP or DOWN)
106             begin
107                     z = 4'b0000;
108                     case(estado)
109              A: z = 4'd6;
110                             B: z = 4'd9;
111                             C: z = 4'd0;
112                             D: z = 4'd2;
```

```
113                                 E: z = 4'd4;
114                                 F: z = 4'd6;
115                                 G: z = 4'd5;
116                                 H: z = 4'd3;
117                                 I: z = 4'd8;
118                                 Blank: z = 4'd15;
119                                 default z = 4'bxxxx;
120                         endcase
121                 end
122
123         always@(posedge clock or posedge reset)
124                 begin
125                         if(reset)
126                                 estado <= A;
127                         else
128                                 estado <= prox_estado;
129                 end
130
131 endmodule
```

APÊNDICE B – Decodificador.v

Código do arquivo Decodificador.v

```
1  module Decodificador(entrada, segmentos);
2          input [3:0] entrada;
3          output reg[0:6] segmentos;
4
5          always@(*)
6                  begin
7                          case(entrada)
8                                  4'b0000: segmentos = 7'b0000001;
9                                  4'b0001: segmentos = 7'b1001111;
10                                 4'b0010: segmentos = 7'b0010010;
11                                 4'b0011: segmentos = 7'b0000110;
12                                 4'b0100: segmentos = 7'b1001100;
13                                 4'b0101: segmentos = 7'b0100100;
14                                 4'b0110: segmentos = 7'b0100000;
15                                 4'b0111: segmentos = 7'b0001111;
16                                 4'b1000: segmentos = 7'b0000000;
17                                 4'b1001: segmentos = 7'b0000100;
18                                 default: segmentos = 7'b1111111;
19                         endcase
20                 end
21
22 endmodule
```

APÊNDICE C – DivisorFrequencias.v

Código do arquivo DivisorFrequencias.v

```
1  module DivisorFrequencias(clk, S);
2          input clk;
3          output reg S;
4          reg [25:0] OUT;
5
6          always @ (posedge clk)
7                  if (OUT == 26'd50000000)
8                          begin
9                                  OUT <= 26'd0;
10                                 S <= 1;
11                         end
12                 else
13                         begin
14                                 OUT <= OUT + 1;
15                                 S <= 0;
16                         end
17
18 endmodule
```

APÊNDICE D – Principal.v

Código do arquivo Principal.v

```
1  module Principal(clock, reset, UP, DOWN, segmentos, saida_estado);
2          input clock, reset, UP, DOWN;
3          output wire [0:6] segmentos;
4          output wire [0:3] saida_estado;
5          wire novo_clock;
6
7          DivisorFrequencias df(clock, novo_clock);
8          MaquinaMealy mq(novo_clock, reset, UP, DOWN, saida_estado);
9          Decodificador d(saida_estado, segmentos);
10
11 endmodule
```