

ID:

Projeto de um contador numérico específico utilizando Máquina de Estados do tipo Moore

São José dos Campos - Brasil

6 de Dezembro de 2019

ID:

Projeto de um contador numérico específico utilizando Máquina de Estados do tipo Moore

Relatório técnico apresentado como documentação do projeto de Máquina de Estados em *Verilog* proposto pela disciplina Laboratório de Sistemas Computacionais: Circuitos Digitais oferecida pela Universidade Federal de São Paulo.

Discente: Claudio Jorge Lopes Filho

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

6 de Dezembro de 2019

Resumo

Este relatório explica em detalhes o projeto de uma **Máquina de Estados Finitos** do tipo Moore para a implementação de um contador numérico específico. A sequência não-usual de contagem 1-4-7-8-9-0-1-2-5 foi definida por sorteio e devia apresentar-se de quatro maneiras: crescente, decrescente, mantendo o número atual, exibindo nada e também podendo voltar para o número inicial 1 dependendo das entradas UP, DOWN e RESET com frequência de 1 Hz. Além disso, a contagem foi exibida em um dos displays de sete segmentos do kit *Altera DE2-115* e apresentada ao docente. Para isso, além dos códigos da Máquina de Estados foram desenvolvidos um divisor de frequência para transformar os 50 MHz nativos do kit para 1 Hz; e um decodificador para transformar *Binary Coded Decimal* (BCD) para sete segmentos. O projeto foi desenvolvido no *software Intel Quartus Prime* utilizando a linguagem de descrição de *Hardware Verilog*, simulado com sucesso e mostrado ao professor.

Palavras-chaves: Contador. *Altera*. *Quartus Prime*. Máquina de Estados Finitos. BCD. *Verilog*, HDL.

Lista de ilustrações

Figura 1 – Circuito Integrado CD74HC73. Implementa flip-flops JK.	9
Figura 2 – Kit <i>Altera DE2-115</i>	13
Figura 3 – Detalhes de um número na base 2	14
Figura 4 – Portas lógicas e seu funcionamento	14
Figura 5 – Exemplo de implementação do Flip-Flop D	16
Figura 6 – Moore x Mealy	18
Figura 7 – Display Ânodo Comum	19
Figura 8 – Display de 7 Segmentos	19
Figura 9 – Temporizador	22
Figura 10 – Decodificador BCD para Decimal pt.1	24
Figura 11 – Decodificador BCD para Decimal pt.2	24
Figura 12 – Decodificador Decimal para Display 7 Segmentos pt.1	25
Figura 13 – Decodificador Decimal para Display 7 Segmentos pt.2	25
Figura 14 – Diagrama de Estados	27
Figura 15 – Recorte da Tabela Verdade Clássica	28
Figura 16 – Registrador	29
Figura 17 – Circuito de Próximo Estado - Bit E3	30
Figura 18 – Circuito de Próximo Estado - Bit E2	30
Figura 19 – Circuito de Próximo Estado - Bit E1	31
Figura 20 – Circuito de Próximo Estado - Bit E0	31
Figura 21 – Circuito de Saída - Bit S3	32
Figura 22 – Circuito de Saída - Bit S2	33
Figura 23 – Circuito de Saída - Bit S1	33
Figura 24 – Circuito de Saída - Bit S0	34
Figura 25 – Projeto Final	34
Figura 26 – Forma de onda do Decodificador	35
Figura 27 – Forma de onda da Máquina de Estados (UP = 1/DOWN = 0)	36
Figura 28 – Forma de onda da Máquina de Estados (UP = 0/DOWN = 1)	36
Figura 29 – Forma de onda da Máquina de Estados (UP = 1/DOWN = 1)	37
Figura 30 – Forma de onda da Máquina de Estados (UP = 0/DOWN = 0)	37
Figura 31 – Forma de onda da Máquina de Estados (RESET = 1)	38

Lista de tabelas

Tabela 1 – Sequência de contagem	11
Tabela 2 – Operadores básicos e operações em Verilog	16
Tabela 3 – Tabela verdade do Flip-Flop D	17
Tabela 4 – Tabela verdade do Flip-Flop T	17
Tabela 5 – Decodificação BCD/Decimal	22
Tabela 6 – Decodificação Decimal/Display 7 Segmentos	23
Tabela 7 – Expressões do Decodificador Decimal/Display 7 Segmentos	23
Tabela 8 – Entradas UP e DOWN	26
Tabela 9 – Estados	26
Tabela 10 – Tabela Verdade	28
Tabela 11 – Expressões para os bits de Próximo Estado	29
Tabela 12 – Expressões para os bits de Saída	32

Sumário

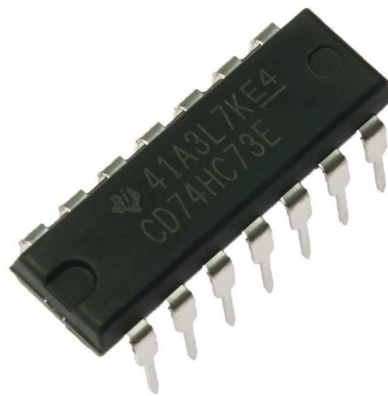
1	INTRODUÇÃO	9
2	OBJETIVOS	11
2.1	Gerais	11
2.2	Específicos	11
3	FUNDAMENTAÇÃO TEÓRICA	13
3.1	FPGA	13
3.2	Base Binária	14
3.3	Portas Lógicas	14
3.4	Verilog	15
3.4.1	Sintaxe Básica	15
3.4.2	Bloqueante e Não-Bloqueante	15
3.5	Flip-Flops e Memória	15
3.6	Máquina de Estados Finitos	16
3.7	Divisor de Frequência	17
3.8	Display de 7 Segmentos	18
4	DESENVOLVIMENTO	21
4.1	Temporizador	21
4.2	Decodificador BCD para 7 Segmentos	22
4.3	Máquina de Estados	26
4.3.1	Requisitos	26
4.3.2	Tabelas Verdade, Estados e Diagrama de Estados	26
4.3.3	Registrador	28
4.3.4	Próximo Estado	29
4.3.5	Função de Saída	32
4.4	Projeto Final	34
5	RESULTADOS OBTIDOS E DISCUSSÕES	35
5.1	Temporizador	35
5.2	Decodificador	35
5.3	Máquina de Estados	36
6	CONSIDERAÇÕES FINAIS	39

REFERÊNCIAS	41
--------------------	-----------

1 Introdução

Na atualidade, somos permeados por sistemas digitais. Desde celulares, relógios, exploração espacial até geladeiras inteligentes podemos observar como a evolução da abstração digital foi essencial para o desenvolvimento da humanidade como um todo.

Figura 1 – Circuito Integrado CD74HC73. Implementa flip-flops JK.



Fonte: <<https://www.autocorerobotica.com.br/cd74hc73-ci-flip-flop-jk>>

O aumento da complexidade das aplicações fez necessário a criação de elementos eletrônicos e circuitos específicos como os de memória que servem especificamente para guardar informação para uso posterior como visto na [Figura 1](#). Com isso, surgem as Máquinas de Estado Finito. Estas, são circuitos que combinam os dados guardados pelos elementos de memória para mudar a informação armazenada e formar uma saída de acordo com o que foi projetado.

Muitos dos sistemas digitais avançados como semáforos, elevadores e alguns processadores envolvem máquinas de estados finitos. Com o fim de aplicar a fundamentação teórica obtida na disciplina de Circuitos Digitais e obter conhecimento prático sobre o projeto de uma máquina de estados, implementaremos uma do tipo Mealy com os objetivos explicados no capítulo a seguir.

2 Objetivos

2.1 Gerais

Implementar um contador para a sequência descrita pela [Tabela 1](#) com 1 Hz de frequência utilizando uma Máquina de Estados Finitos do tipo Moore. A contagem deve responder a três entradas (UP, DOWN e RESET), podendo manter o número atual, apagar o display, seguir a sequência ou inverter a sequência, além de voltar para o número inicial a qualquer instante.

Tabela 1 – Sequência de contagem

1	4	7	8	9	0	1	2	5
---	---	---	---	---	---	---	---	---

Fonte: Autor

2.2 Específicos

Utilizar o *software* **Quartus Prime** para realizar os seguintes itens:

- Desenvolver em *Verilog* uma Máquina de Estados do tipo Moore.
- Desenvolver em *Verilog* um decodificador de BCD para display de sete segmentos.
- Desenvolver em *Verilog* um divisor de frequência de 50 MHz para 1 Hz.
- Utilizar instâncias dos componentes criados.
- Analisar os resultados simulando por *waveforms*.
- Mapear, descarregar o projeto no FPGA do kit *Altera DE2-115* e apresentar ao professor.

3 Fundamentação Teórica

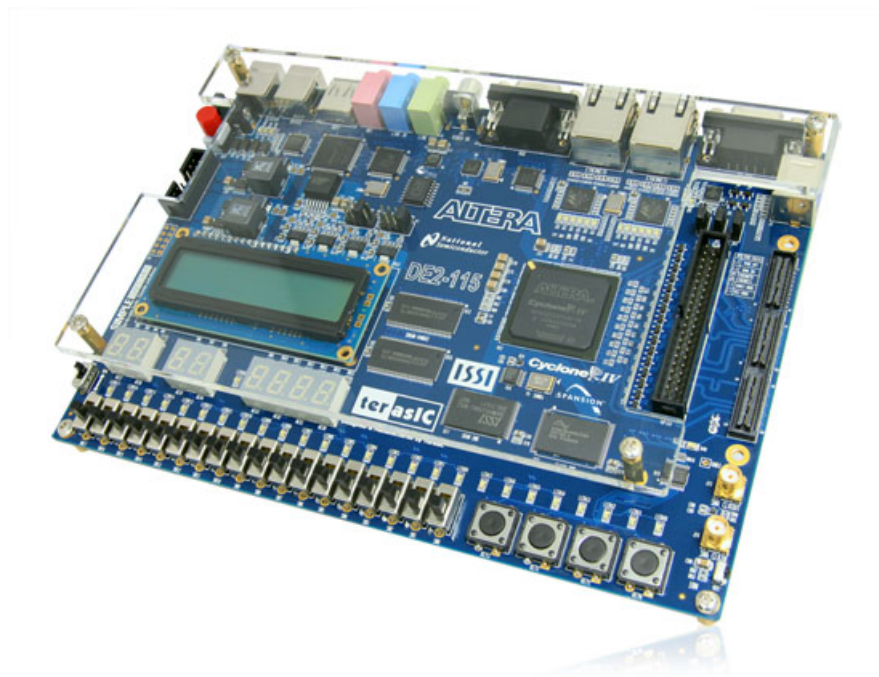
Nesse capítulo, introduziremos os conceitos teóricos necessários para implementarmos o projeto com os objetivos desejados.

3.1 FPGA

Segundo Harris e Harris (2013, p. 274) "*A field programmable gate array (FPGA) is an array of reconfigurable gates.*"¹ (1). São muito utilizados em sistemas complexos, embarcados e conseguem implementar circuitos tanto em esquemático quanto em *Hardware Description Languages* (HDLs) possibilitando a facilidade de uso em projetos.

Utilizaremos o *software Quartus Prime* escrever os códigos, sintetizar os circuitos e integrar ao FPGA *Altera Cyclone®IV 4CE115* do kit *Altera DE2-115* visto na figura [Figura 2](#).

Figura 2 – Kit *Altera DE2-115*



Fonte: <https://www.intel.com/content/www/us/en/programmable/solutions/partners/partner-profile/terasic-inc-/board/altera-de2-115-development-and-education-board.html>

¹ Uma matriz de portas programáveis em campo, field programmable gate array(FPGA), é um arranjo de portas reconfiguráveis. Em tradução livre do inglês.

3.2 Base Binária

Vivemos em um mundo no qual a base numérica 10 é predominante. Isso se deve ao fato de possuímos 10 dedos nas mãos e facilitar a contagem. Por outro lado, em sistemas digitais, a base binária é a mais recomendada, visto que a representação sim/não, aberto/fechado, alta tensão/baixa tensão é adequada quando trabalhamos com circuitos elétricos.

Figura 3 – Detalhes de um número na base 2

$$10110_2 = 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 22_{10}$$

one sixteen
no eight
one four
one two
no one

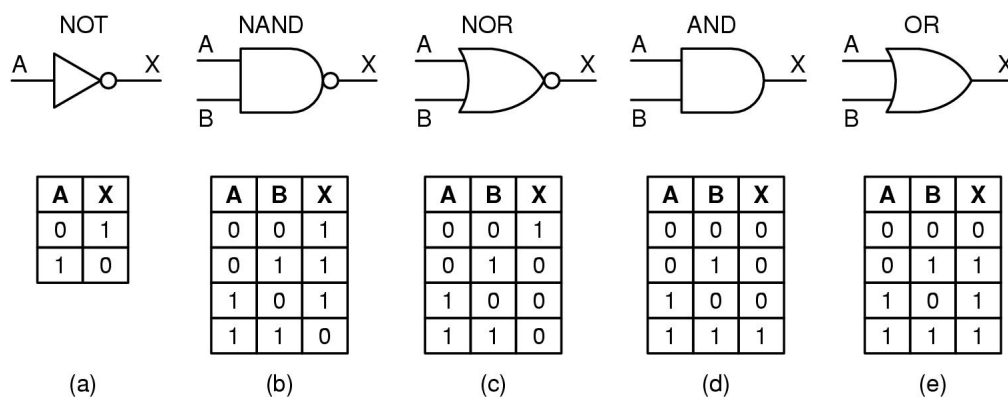
Fonte: Digital Design and Computer Architecture (1)

Na base 2, tratamos cada dígito como a existência ou não de uma potência de 2, começando de 2^{N-1} até 2^0 , onde N é o número de dígitos. Tome como exemplo a Figura 3 na qual dissecamos o número 10110_2 . Os dígitos de valor 1 indicam que $10110_2 = 2^4 + 2^2 + 2^1$ (1).

3.3 Portas Lógicas

Se aproveitando da base binária, George Boole, em 1854, publicou *An Investigation of the Laws of Thought* no qual descreveu operações com variáveis binárias que assumem o valor VERDADEIRO ou FALSO, dando origem ao que conhecemos como variáveis e lógica booleana (1).

Figura 4 – Portas lógicas e seu funcionamento



A e B representam as entradas e X a saída. Fonte: <http://www.dpi.inpe.br/~carlos/Academicos/Cursos/ArqComp/aula_5bn1.html>

A partir disso a eletrônica digital, com base na lógica criada por Boole, deu origem as portas lógicas ([Figura 4](#)), elementos básicos de qualquer sistema digital.

3.4 Verilog

Verilog é uma linguagem de descrição de *Hardware* que visa elevar o nível de abstração na hora de projetar circuitos digitais. Com o auxílio de *Computer-aided design* (CAD) para otimizar a escolha de funções lógicas, a criação do *Verilog* acelerou e melhorou consideravelmente a produção de sistemas digitais. Essa HDL é muito utilizada no ensino e nas indústrias e se tornou IEEE² standard em 1995 ([1](#)).

Os projetos em *Verilog* são compostos por **modules**, estruturas de código que possuem informações de entrada, saída e intermediários que possivelmente necessitem ser gerados. Abaixo segue um exemplo de código simples feito em *Verilog*.

```
1 module introducao(A, B, C, Saida);  
2  
3 input A, B, C;  
4 output Saida;  
5  
6 assign S = A & B | ~C;  
7  
8 endmodule
```

3.4.1 Sintaxe Básica

Como mostrado acima, todo código em *Verilog* tem um *module* e um *endmodule*, além de um nome próprio que deve ser igual ao nome do arquivo gerado. Não é possível ter mais de um *module* em um mesmo arquivo. Entre parênteses deve ser especificado os elementos de entrada e de saída que serão utilizados no módulo criado.

A palavra-chave *input* separa as entradas entre os elementos especificados e a palavra-chave *output* indica as saídas. Intermediários gerados usualmente tem o tipo *wire* ou *reg* dependendo de como vão ser usados.

Para utilizarmos as portas lógicas vistas na seção [seção 3.3](#) temos operadores nativos na linguagem como visto na tabela [Tabela 2](#)

3.4.2 Bloqueante e Não-Bloqueante

3.5 Flip-Flops e Memória

Os flip-flops, também conhecidos como multivibradores biestáveis, são extremamente importante para os circuitos digitais, pois são capazes de armazenar dados devido a sequencia-

² The Institute of Electrical and Electronic Engineers.

Tabela 3 – Tabela verdade do Flip-Flop D

D	Clock	Preset	Clear	Q
x	x	0	1	1
x	x	1	0	0
0	Subida	1	1	0
1	Subida	1	1	1

Fonte: Autor

Tabela 4 – Tabela verdade do Flip-Flop T

T	Clock	Preset	Clear	Q
x	x	0	1	1
x	x	1	0	0
0	Subida	1	1	Q
1	Subida	1	1	Q'

Fonte: Autor

por três blocos: Circuito combinacional de próximo estado, circuito combinacional de saída e um registrador de estados. A cada ciclo de *clock*, a Máquina de Estados é atualizada com o resultado computado pelo circuito de próximo estado e a saída é formulada com o novo valor adquirido pelo registrador.

Existem dois tipos de Máquinas de Estado: Moore e Mealy. Os dois nomes são dados em homenagem aos seus criadores *Edward F. Moore* autor do artigo *Gedanken-experiments on Sequential Machines* em 1956 e *George H. Mealy* autor de *A Method of Synthesizing Sequential Circuits* em 1955 (1). Entender as diferenças entre os dois tipos é essencial. A máquina de Moore possui a saída dependente apenas do estado atual guardado pelo registrador e quando é utilizada como contador não precisa de um circuito combinacional de saída, pois cada estado pode representar o número a ser contado. Já a máquina de Mealy tem a saída dependente do estado atual e das entradas. Isso pode causar mudanças assíncronas na saída já que as entradas não precisam de um ciclo de *clock* para serem alteradas (Figura 6).

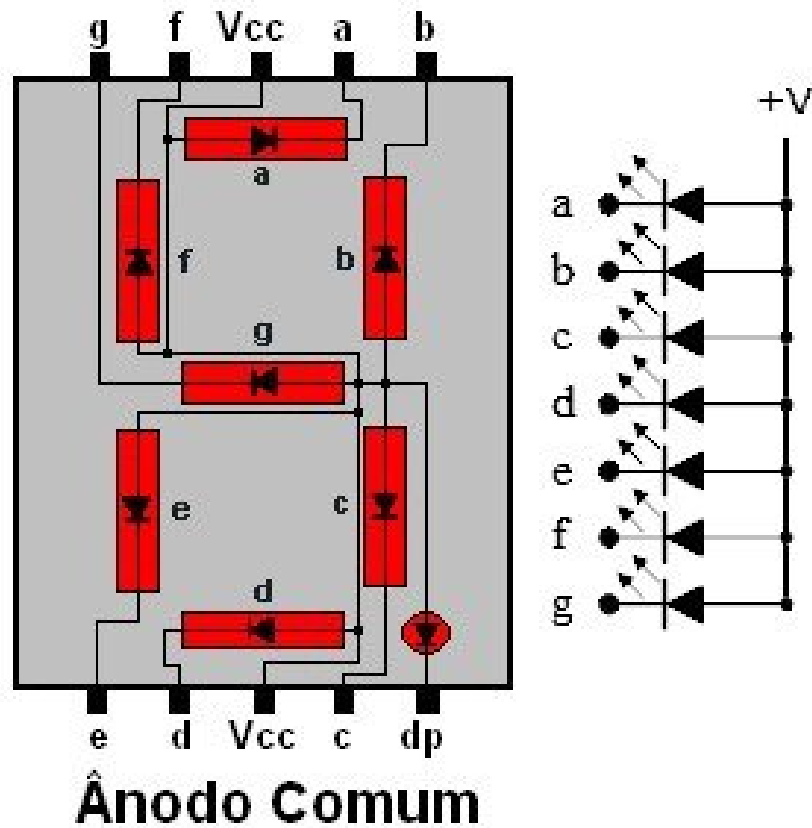
3.7 Divisor de Frequência

Existem casos, como neste projeto, em que o *clock* fornecido vai além do necessário. Para adaptar o *clock* que temos disponível precisamos de um divisor de frequência.

Esse tipo de circuito digital nada mais é do que um contador que replica o sinal de *clock* um determinado número de vezes. Segundo Tocci, Widmer e Moss (2007, p.364) "*In any counter, the signal at the output of the last FF (i.e., the MSB) will have a frequency equal to the input clock frequency divided by the MOD number of the counter.*"³ (2).

³ Em qualquer contador, o sinal na saída do último FF (flip-flop) terá a frequência igual a frequência de

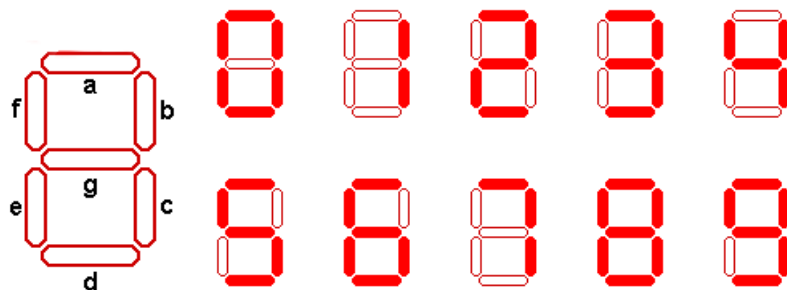
Figura 7 – Display Ânodo Comum



Fonte: <<https://www.electronica-pt.com/eletronica-digital/display-7-segmentos>>

para um sinal nomeado com o decimal correspondente. A partir disso, criamos outro circuito para acender os segmentos baseado no decimal desejado.

Figura 8 – Display de 7 Segmentos



Fonte: <<http://tot.eng.br/multiplexacao-display-7-segmentos-arduino/>>

4 Desenvolvimento

Neste capítulo será explicado em detalhes a implementação da Máquina de Estados utilizando a fundamentação teórica desenvolvida no capítulo anterior.

4.1 Temporizador

Para a execução do projeto, foi solicitado que a transição de contagem acontecesse em uma frequência de 1 Hz. Por outro lado, a frequência nativa do kit *Altera DE2-115* é de 50 MHz e precisamos construir um divisor de frequência. Como fundamentado na seção 3.7, podemos utilizar a equação proposta para identificarmos o MOD do contador que deveremos implementar.

$$N = \frac{clock_{disponível}}{clock_{desejado}}$$

Com $clock_{disponível} = 50 \text{ MHz}$ e $clock_{desejado} = 1 \text{ Hz}$ temos que:

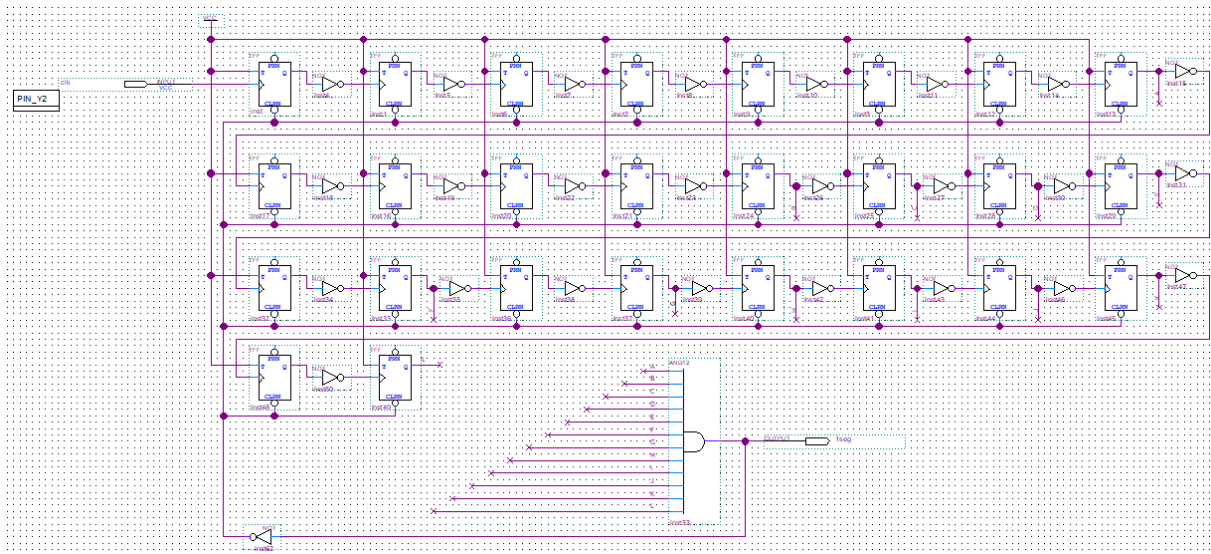
$$N = \frac{50 \text{ MHz}}{1 \text{ Hz}} = 50.000.000$$

Desse modo, foi construído um contador assíncrono MOD-50.000.000 com entrada de 50 MHz.

Primeiramente devemos notar que $50.000.000_{10} = 10111110101111000010000000_2$ e são necessários 26 bits para representá-lo. Desse modo, precisamos de 26 flip-flops do tipo T de modo a alcançar a representação.

Para que o contador funcione, ligamos o pino VCC à entrada T de todos os elementos e conectamos os flip-flops em sequência de modo que a saída negada de um conecte-se à entrada de clock do outro. Teremos um contador MOD- 2^{26} , ou seja, MOD-67.108.863 que não é o que desejamos. Precisamos resetar a contagem quando o conjunto de flip-flops alcançar o valor 50.000.000 e para isso basta juntar a saída dos flip-flops 8, 13, 14, 15, 16, 18, 20, 21, 22, 23, 24 e 26 (contando a partir do *clock* original) em uma porta AND, negar e conectar à entrada CLEAR de todos os flip-flops do contador como pode ser visto no Temporizador final da [Figura 9](#).

Figura 9 – Temporizador



Fonte: Autor

4.2 Decodificador BCD para 7 Segmentos

Para possibilitar a exibição utilizando o display de 7 segmentos disponível no kit *Altera DE2-115* foi necessário criar um decodificador BCD para display de 7 segmentos. A execução foi feita em duas partes: Primeiramente, decodificar o BCD para decimal e depois decodificar o decimal para display de 7 segmentos.

Para decodificar BCD para decimal é simples, pois cada número equivale a apenas uma expressão booleana como podemos ver pela [Tabela 5](#).

Tabela 5 – Decodificação BCD/Decimal

Entradas				Saídas									
W	X	Y	Z	S0	S1	S2	S3	S4	S5	S6	S7	S8	S9
0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	0	0	1	0	1	0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0	0	0	0	0	0	0
0	0	1	1	0	0	0	1	0	0	0	0	0	0
0	1	0	0	0	0	0	0	1	0	0	0	0	0
0	1	0	1	0	0	0	0	0	1	0	0	0	0
0	1	1	0	0	0	0	0	0	0	1	0	0	0
0	1	1	1	0	0	0	0	0	0	0	1	0	0
1	0	0	0	0	0	0	0	0	0	0	0	1	0
1	0	0	1	0	0	0	0	0	0	0	0	0	1

Fonte: Autor

A decodificação do decimal também é simples, porém agora a expressão de cada segmento é maior (Tabela 6). Selecionamos todos os bits de valor 0 e passamos por uma porta NOR, garantindo a saída ativa em baixa do display ânodo comum visto na seção 3.8 além de desativar todos os segmentos para qualquer número entre 10 e 15. A implementação das expressões definidas na Tabela 7 se encontra nas figuras 10, 11, 12, 13.

Tabela 6 – Decodificação Decimal/Display 7 Segmentos

	a	b	c	d	e	f	g
S0	0	0	0	0	0	0	1
S1	1	0	0	1	1	1	1
S2	0	0	1	0	0	1	0
S3	0	0	0	0	1	1	0
S4	1	0	0	1	1	0	0
S5	0	1	0	0	1	0	0
S6	0	1	0	0	0	0	0
S7	0	0	0	1	1	1	1
S8	0	0	0	0	0	0	0
S9	0	0	0	0	1	0	0
S10	1	1	1	1	1	1	1
S11	1	1	1	1	1	1	1
S12	1	1	1	1	1	1	1
S13	1	1	1	1	1	1	1
S14	1	1	1	1	1	1	1
S15	1	1	1	1	1	1	1

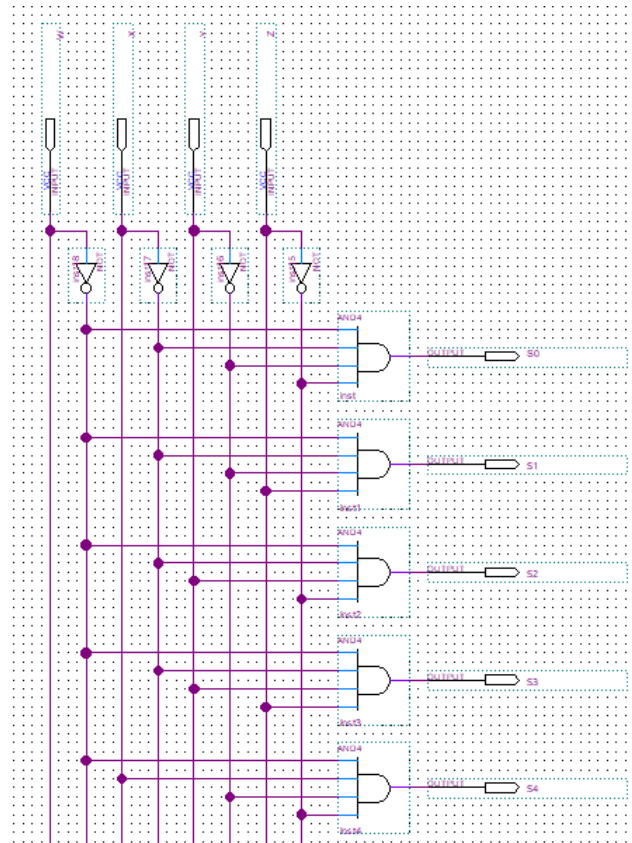
Fonte: Autor

Tabela 7 – Expressões do Decodificador Decimal/Display 7 Segmentos

	Expressões
a	$(S0 + S2 + S3 + S5 + S6 + S7 + S8 + S9)'$
b	$(S0 + S1 + S2 + S3 + S4 + S7 + S8 + S9)'$
c	$(S0 + S1 + S3 + S4 + S5 + S6 + S7 + S8 + S9)'$
d	$(S0 + S2 + S3 + S5 + S6 + S8 + S9)'$
e	$(S0 + S2 + S6 + S8)'$
f	$(S0 + S4 + S5 + S6 + S8 + S9)'$
g	$(S2 + S3 + S4 + S5 + S6 + S8 + S9)'$

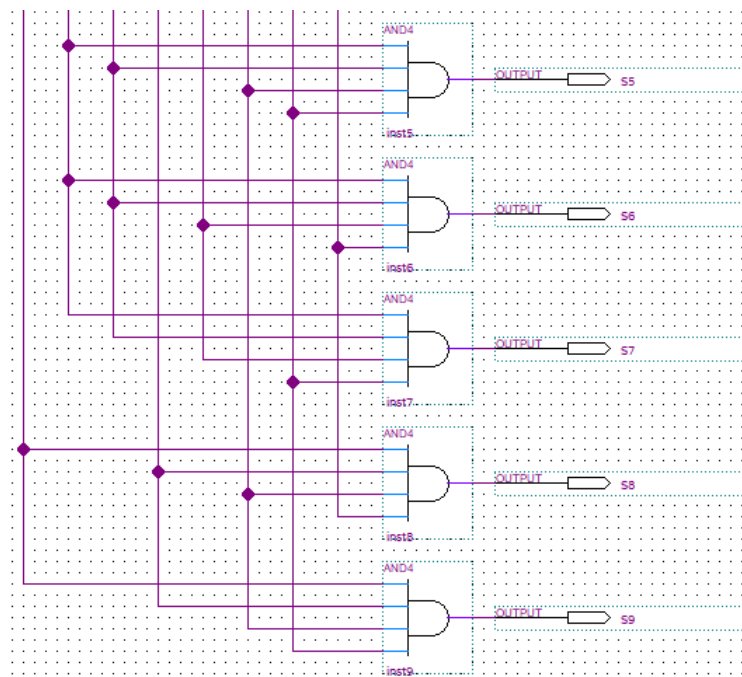
Fonte: Autor

Figura 10 – Decodificador BCD para Decimal pt.1



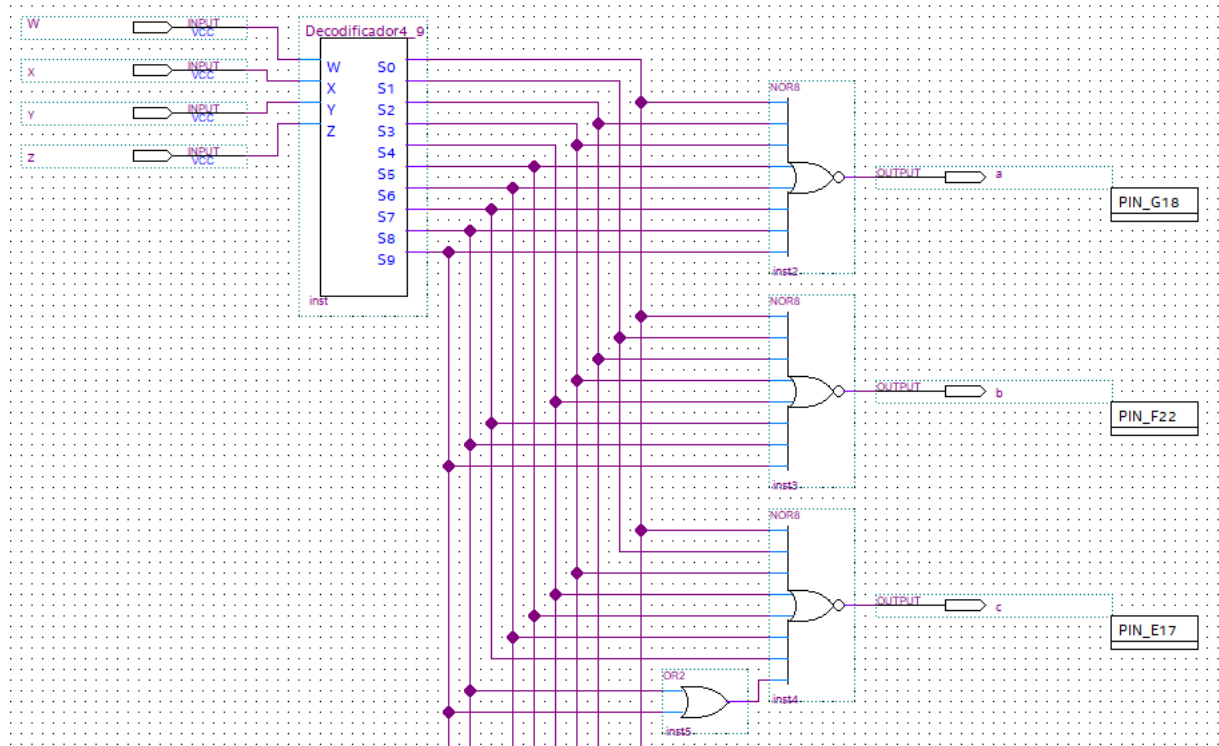
Fonte: Autor

Figura 11 – Decodificador BCD para Decimal pt.2



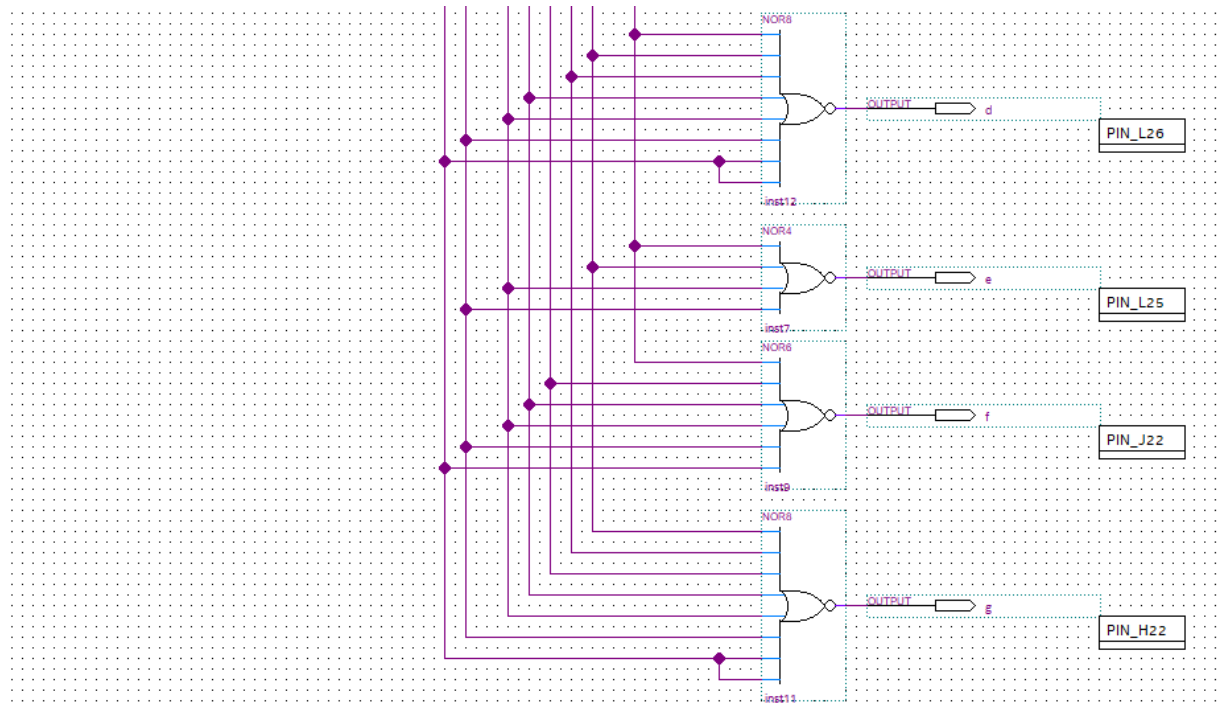
Fonte: Autor

Figura 12 – Decodificador Decimal para Display 7 Segmentos pt.1



Fonte: Autor

Figura 13 – Decodificador Decimal para Display 7 Segmentos pt.2



Fonte: Autor

4.3 Máquina de Estados

Esta seção vai tratar da Máquina de Estados como uma unidade. Cada subseção cuidará de uma parte específica de sua montagem.

4.3.1 Requisitos

A Máquina de Estados deve ter três entradas, UP, DOWN e RESET. A entrada RESET é independente das outras duas e a sequência deve voltar para o início quando esta estiver em nível alto. Já as outras respondem à [Tabela 8](#):

Tabela 8 – Entradas UP e DOWN

UP	DOWN	Ação
0	0	Manter Estado
0	1	Decrescer
1	0	Crescer
1	1	Ir para o <i>Blank</i>

Fonte: Autor

Além disso, ao crescer ou decrescer quando se encontra no estado *Blank* a sequência deve sempre voltar para o primeiro número, no caso 1.

4.3.2 Tabelas Verdade, Estados e Diagrama de Estados

Para desenvolver o contador utilizando Máquina de Mealy, precisamos decidir a quantidade de estados necessários. Isso define também a quantidade de flip-flops que o registrador precisa ter. Temos nove números e um estado de blank, portanto 10 estados no total ([Tabela 9](#)). A quantidade de bits necessários é igual a $\lceil \log_2 10 \rceil$, ou seja, 4 bits.

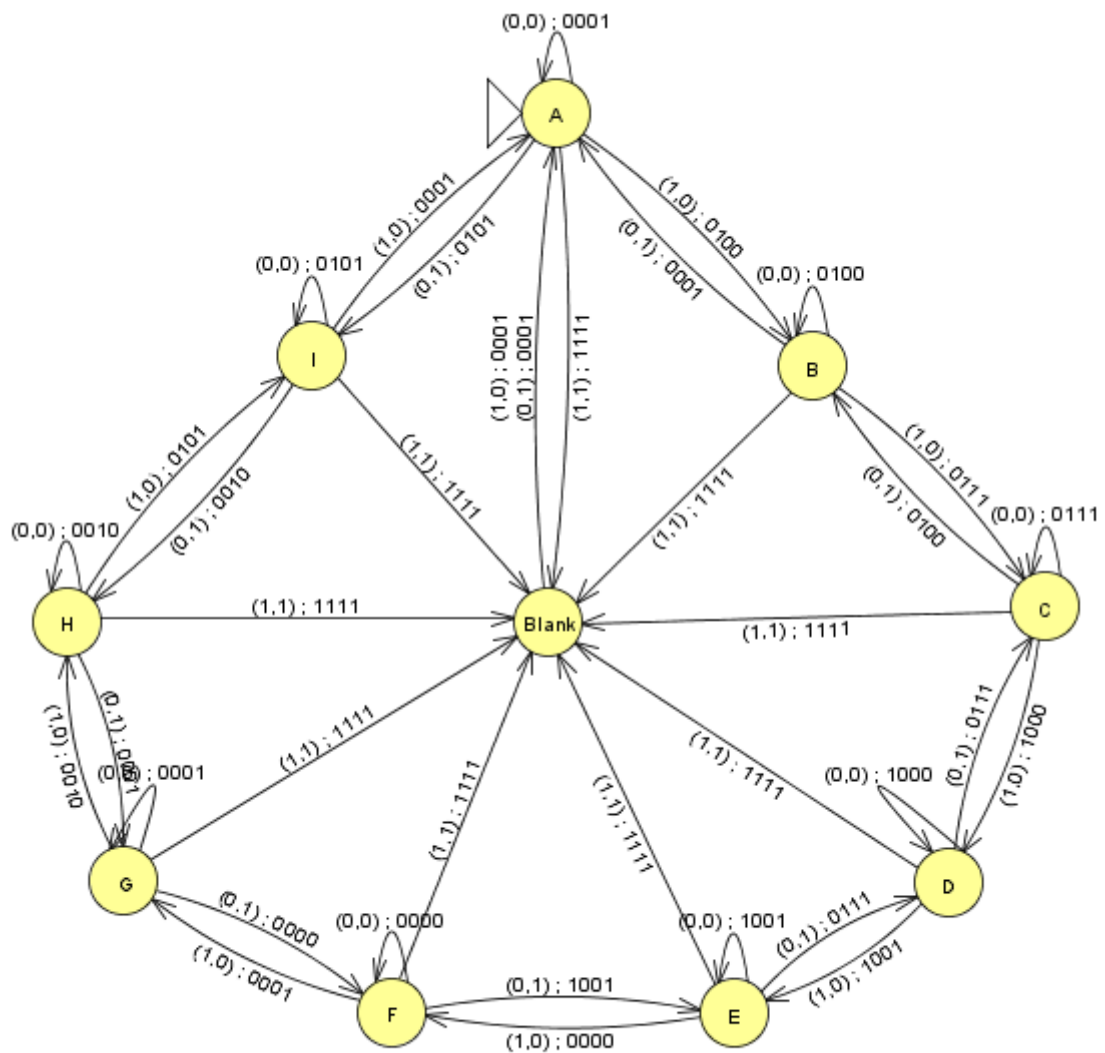
Tabela 9 – Estados

Estado	Codificação
A	0000
B	0001
C	0010
D	0011
E	0100
F	0101
G	0110
H	0111
I	1000
Blank	1001

Fonte: Autor

Além da codificação dos estados. O projeto de uma Máquina de Estados também requer um diagrama de estado, mostrando todas as transições possíveis dependendo das entradas. O diagrama mostrado na [Figura 14](#) segue o modelo de Mealy, no qual cada aresta representa uma combinação de entrada em conjunto com a saída esperada. O triângulo ao lado do estado *A* indica que ele é o estado inicial. O diagrama foi criado utilizando o *software JFLAP*. (3)

Figura 14 – Diagrama de Estados



Fonte: Autor

Para a organização execução do projeto, o diagrama de estado foi convertido em uma tabela verdade afim de encontrarmos e simplificarmos expressões para cada elemento. A [Tabela 10](#) mostra uma versão simplificada desta, enquanto a [Figura 15](#) mostra um recorte da tabela original.

Tabela 10 – Tabela Verdade

Estado Atual	Próximo Estado/Saída			
	UD(0,0)	UD(0,1)	UD(1,0)	UD(1,1)
A(0000)	A(0000)/0001	I(1000)/0101	B(0001)/0100	Blank(1001)/1111
B(0001)	B(0001)/0100	A(0000)/0001	C(0010)/0111	Blank(1001)/1111
C(0010)	C(0010)/0111	B(0001)/0100	D(0011)/1000	Blank(1001)/1111
D(0011)	D(0011)/1000	C(0010)/0111	E(0100)/1001	Blank(1001)/1111
E(0100)	E(0100)/1001	D(0011)/1000	F(0101)/0000	Blank(1001)/1111
F(0101)	F(0101)/0000	E(0100)/1001	G(0110)/0001	Blank(1001)/1111
G(0110)	G(0110)/0001	F(0101)/0000	H(0111)/0010	Blank(1001)/1111
H(0111)	H(0111)/0010	G(0110)/0001	I(1000)/0101	Blank(1001)/1111
I(1000)	I(1000)/0101	H(0111)/0010	A(0000)/0001	Blank(1001)/1111
Blank(1001)	Blank(1001)/1111	A(0000)/0001	A(0000)/0001	Blank(1001)/1111

Fonte: Autor

Figura 15 – Recorte da Tabela Verdade Clássica

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	E3	E2	E1	E0	U	D		E3'	E2'	E1'	E0'		S3	S2	S1	S0
2	0	0	0	0	0	0		0	0	0	0		0	0	0	1
3	0	0	0	0	0	1		1	0	0	0		0	1	0	1
4	0	0	0	0	1	0		0	0	0	1		0	1	0	0
5	0	0	0	0	1	1		1	0	0	1		1	1	1	1
6	0	0	0	1	0	0		0	0	0	1		0	1	0	0
7	0	0	0	1	0	1		0	0	0	0		0	0	0	1
8	0	0	0	1	1	0		0	0	1	0		0	1	1	1
9	0	0	0	1	1	1		1	0	0	1		1	1	1	1
10	0	0	1	0	0	0		0	0	1	0		0	1	1	1
11	0	0	1	0	0	1		0	0	0	1		0	1	0	0
12	0	0	1	0	1	0		0	0	1	1		1	0	0	0
13	0	0	1	0	1	1		1	0	0	1		1	1	1	1

Fonte: Autor

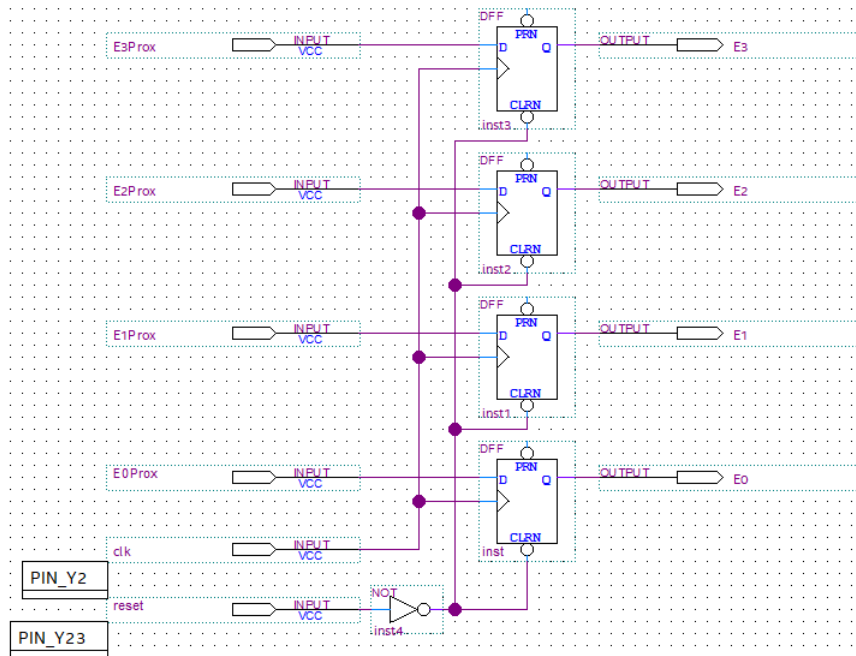
Na [Figura 15](#), E3, E2, E1, E0 representam os bits do estado sendo E3 o mais significativo, U representa a entrada UP e D a entrada DOWN. E3', E2', E1' e E0' indicam o próximo estado a ser visitado dado o estado atual e as entradas. S3, S2, S1 e S0 são os bits do número de saída.

4.3.3 Registrador

Para armazenar os valores do estado atual, foi criado um banco de flip-flops do tipo D que reagem ao *clock* de 1 Hz, guardam o resultado do circuito de Próximo de Estado a cada ciclo de *clock* e transmitem o valor atual para o circuito combinacional de saída. Agregado a isso vem a entrada RESET, que passa por uma porta NOT e vai para as entradas CLEAR

dos flip-flops, setando o valor de estado atual para 0000 como podemos ver implementado na Figura 16.

Figura 16 – Registrador



Fonte: Autor

4.3.4 Próximo Estado

A cada ciclo de *clock* o circuito de próximo estado define qual estado deve ser acessado com base nas entradas fornecidas. Simplificando a tabela verdade mostrada na Tabela 10 através das técnicas conhecidas, como por exemplo o mapa de Karnaugh, conseguimos as expressões (Tabela 11) de cada bit de estado. Para facilitar o trabalho foi utilizada a ferramenta online que utiliza essa técnica. (4)

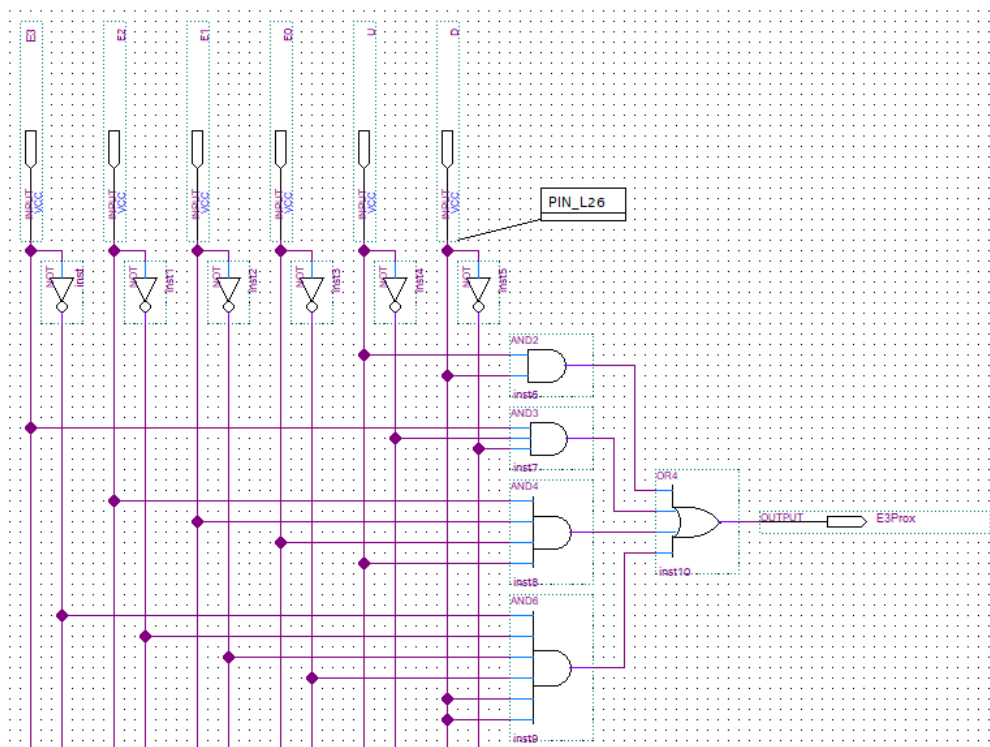
Tabela 11 – Expressões para os bits de Próximo Estado

Bit	Expressão
E3	$U.D + E3.U'D' + E2.E1.E0.U + E3'E2'E1'E0'D$
E2	$E2.E1'D' + E2.E0'D' + E2.E0.U' + E2.E1.U' + E3.E0'U'D + E2'E1.E0.U.D'$
E1	$E1.E0'D' + E1.E0.U' + E3.E0'U'D + E3'E1'E0.U.D' + E2.E1'E0'U'D$
E0	$U.D + E3'E0'U + E0.U'D' + E1.E0'D + E2.E0'D + E3.E0'D$

Fonte: Autor

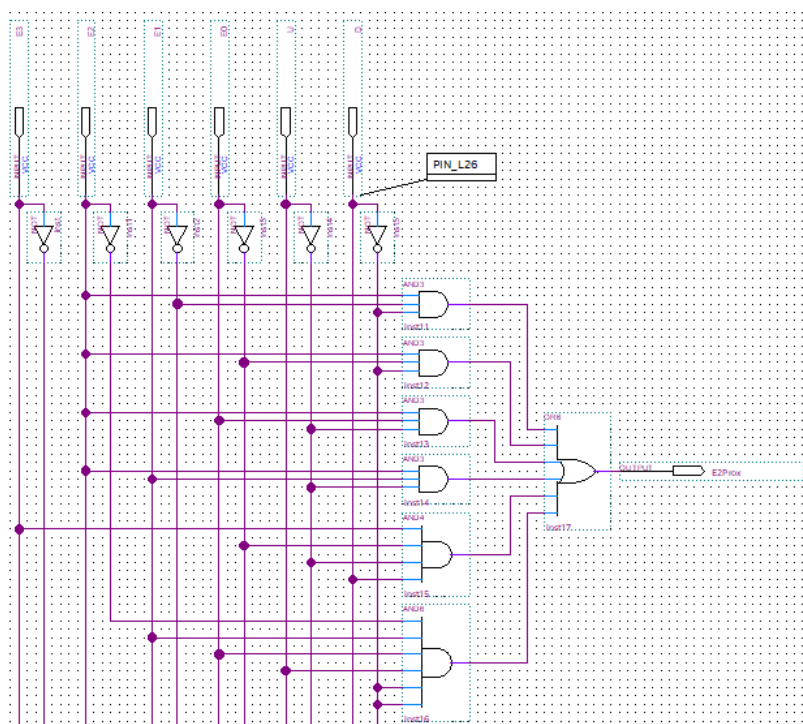
As figuras 17, 18, 19, 20 mostram a implementação final do circuito de próximo estado de maneira simplificada para facilitar a visualização.

Figura 17 – Circuito de Próximo Estado - Bit E3



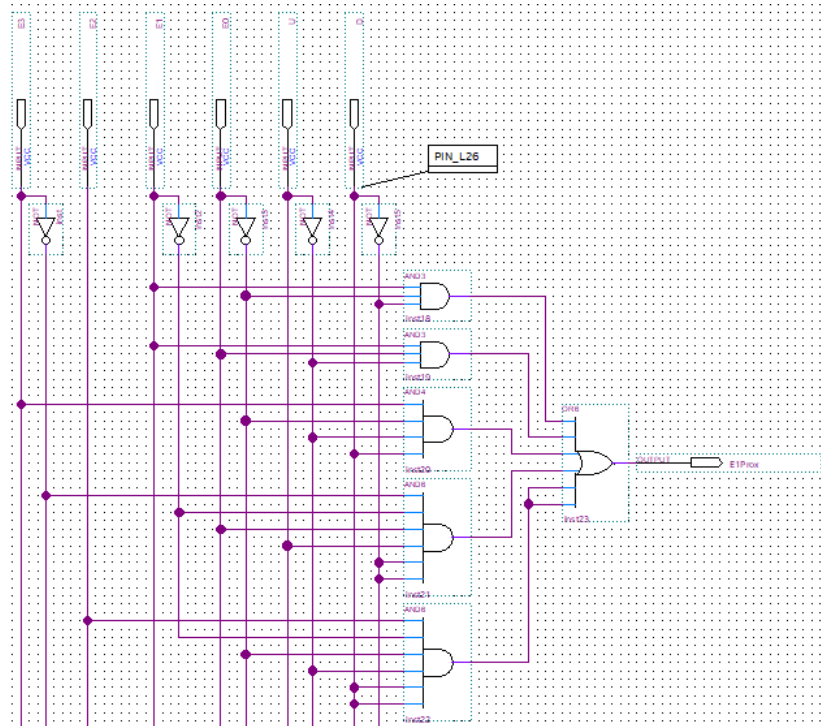
Fonte: Autor

Figura 18 – Circuito de Próximo Estado - Bit E2



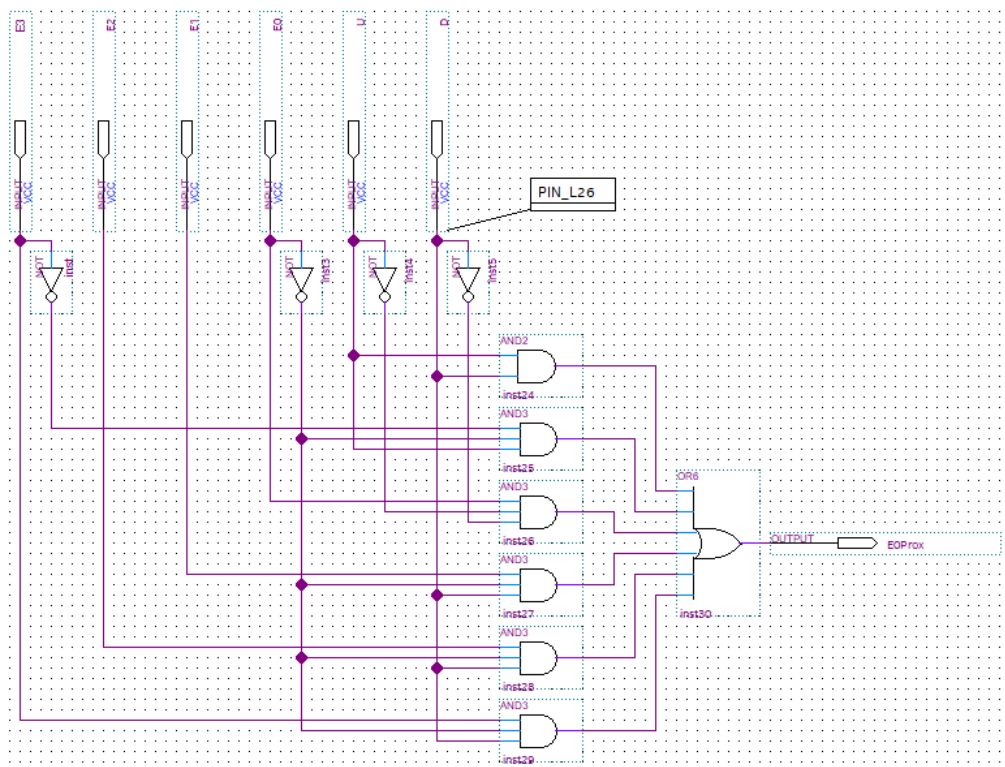
Fonte: Autor

Figura 19 – Circuito de Próximo Estado - Bit E1



Fonte: Autor

Figura 20 – Circuito de Próximo Estado - Bit E0



Fonte: Autor

4.3.5 Função de Saída

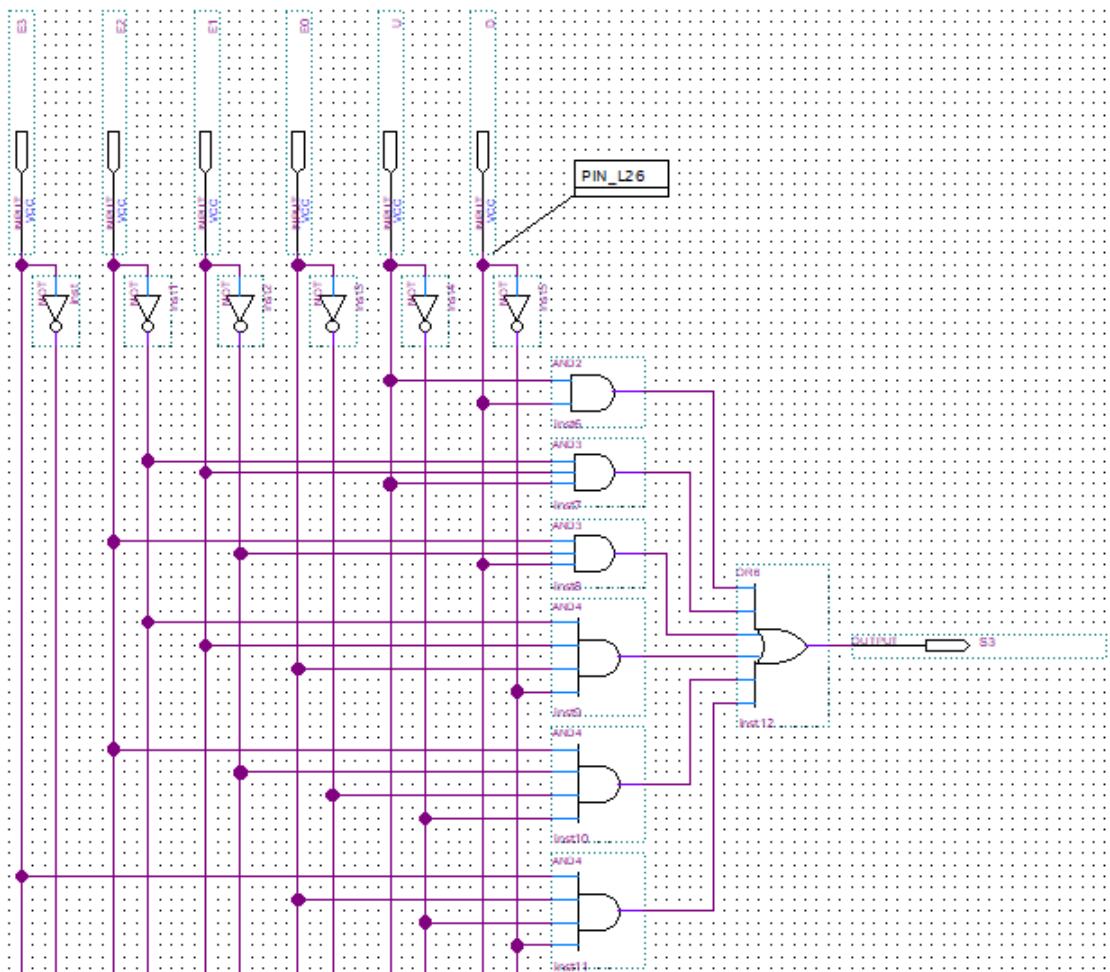
Como fundamentado na seção 3.6, a saída da máquina de Mealy depende de suas entradas além do estado atual. Esse detalhe acabou aumentando significativamente o tamanho das expressões de cada bit, como podemos ver na Tabela 12 e nas figuras 21, 22, 23, 24.

Tabela 12 – Expressões para os bits de Saída

Bit	Expressão
S3	$U.D + E2'.E1.U + E2.E1'.D + E2'.E1.E0.D' + E2.E1'.E0'.U' + E3.E0.U'.D'$
S2	$U.D + E2'.E1.D + E3.U'.D' + E3'.E2'.E0'.D + E3'.E2'.E1'.U + E2'.E1.E0'.U + E2.E1.E0.U + E3'.E2'.E1'.E0.D'$
S1	$U.D + E3.E0'.D + E2'.E1.E0.D + E2.E1.E0'.U + E3.E0.U'.D' + E3'.E2'.E1'.E0.U + E2'.E1.E0'.U'.D' + E2.E1.E0.U'.D'$
S0	$U.D + E0.D + E0.U + E3.D' + E0'.U'.D' + E3'.E2'.E1'.D$

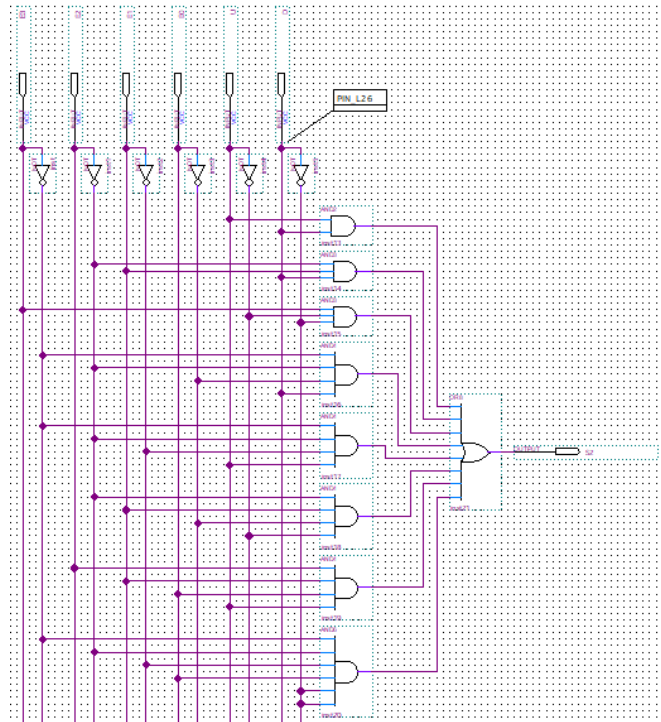
Fonte: Autor

Figura 21 – Circuito de Saída - Bit S3



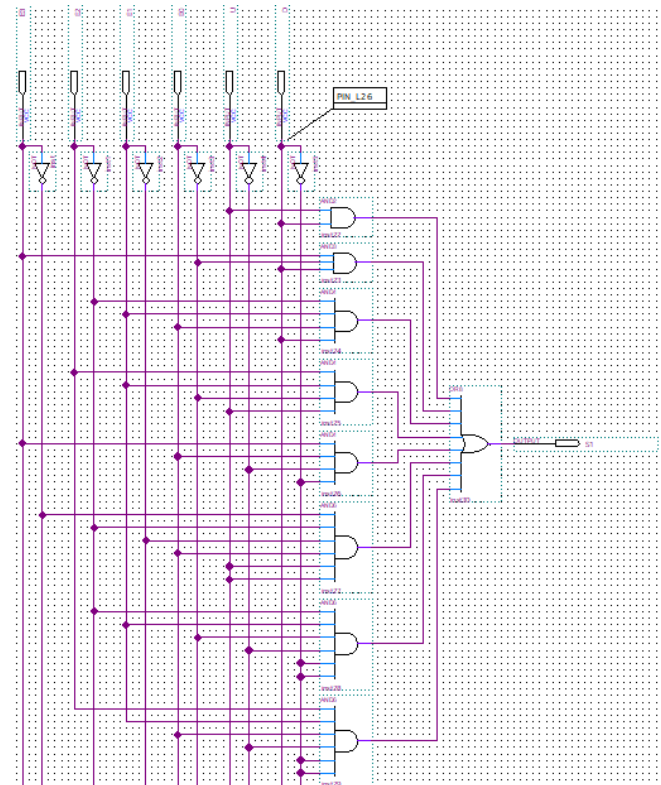
Fonte: Autor

Figura 22 – Circuito de Saída - Bit S2



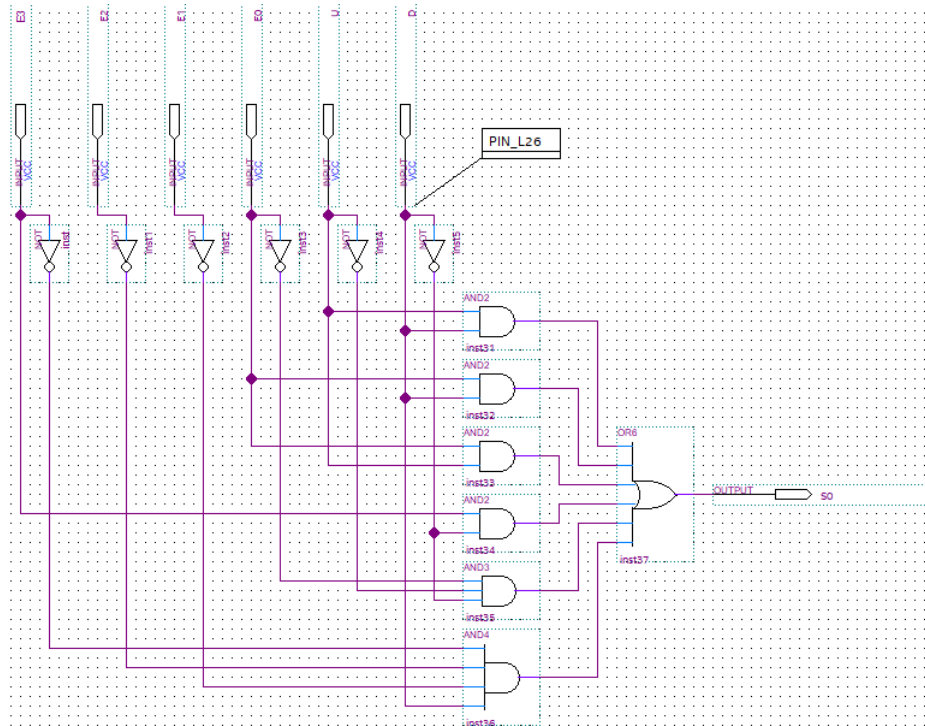
Fonte: Autor

Figura 23 – Circuito de Saída - Bit S1



Fonte: Autor

Figura 24 – Circuito de Saída - Bit S0



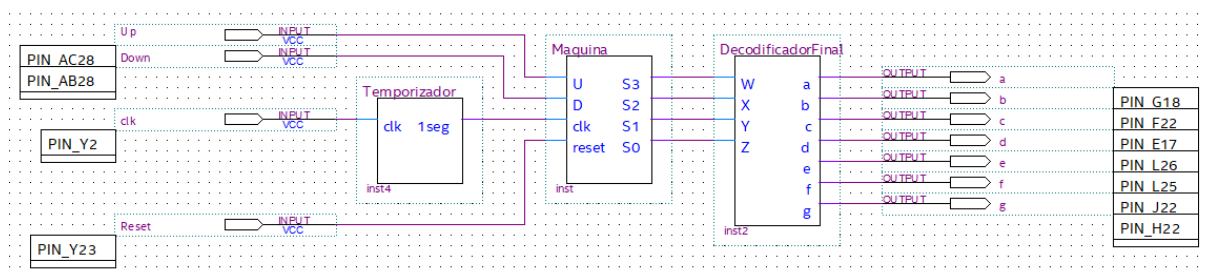
Fonte: Autor

4.4 Projeto Final

Unindo o circuito final de cada seção anterior através de *black-boxes* obtemos a configuração da Figura 25.

A entrada clock passa pela caixa Temporizador, sofre a divisão de frequência e vai para a máquina de estados. A máquina executa as operações a cada ciclo de *clock* e as saídas S3, S2, S1 e S0 vão para o Decodificador que cuida da exibição do valor recebido no display de sete segmentos.

Figura 25 – Projeto Final



Fonte: Autor

5 Resultados Obtidos e Discussões

O projeto foi mapeado e descarregado no kit *Altera DE2-115* e apresentado ao professor com sucesso. Algo observado foi a assincronalidade das saídas já que as mesmas dependem assincronamente das entradas. A seguir utilizaremos do recurso de *waveforms* disponível no *software Quartus Prime* para demonstrar os resultados da implementação da Máquina de Estados.

5.1 Temporizador

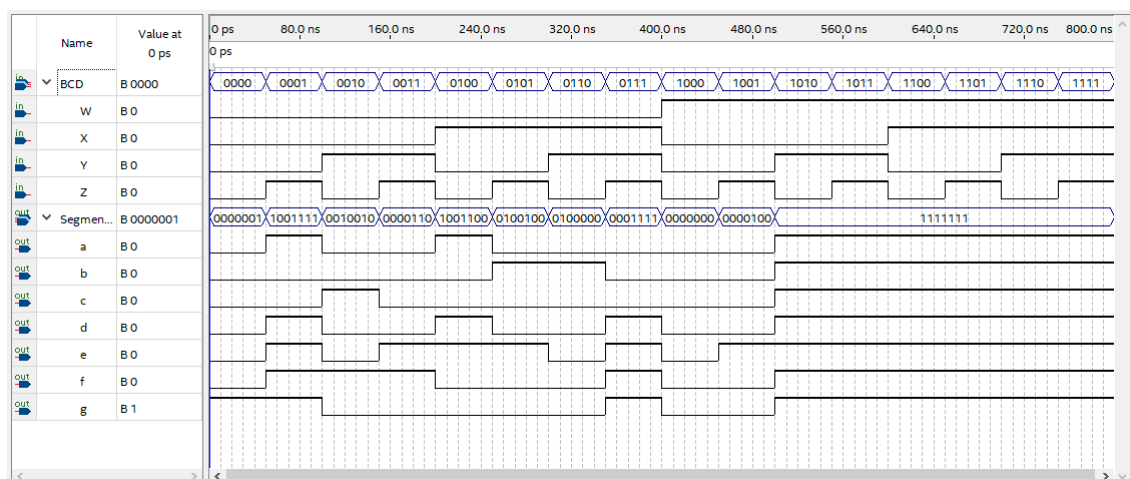
As *waveforms* disponíveis no *Quartus Prime* não suportam a diferença de frequência exigida pelo projeto. Por esse motivo, não será possível mostrar visualmente os resultados do que foi implementado nas seção 4.1.

Para a simulação das outras partes do projeto, serão utilizadas frequências de *clock* convenientes para a exibição dos dados.

5.2 Decodificador

A [Figura 26](#) mostra a simulação do decodificador em execução. Podemos conferir o resultado com a [Tabela 5](#) e a [Tabela 6](#) e verificar que a saída é a esperada. Assim como desejávamos, cada número codificado em BCD acende seus respectivos segmentos e quando os códigos de 10 a 15 são passados o display apaga.

Figura 26 – Forma de onda do Decodificador

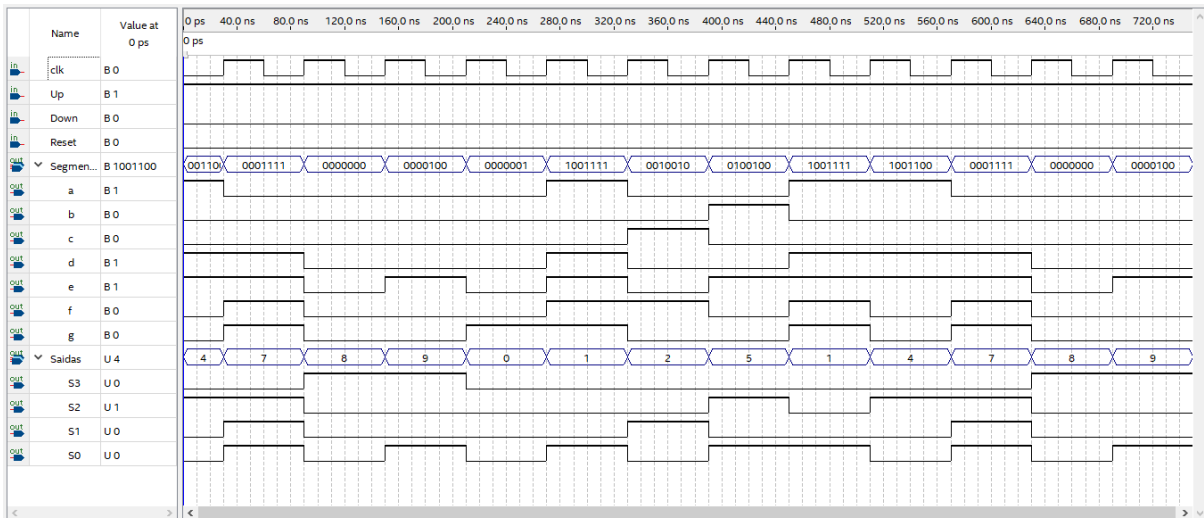


Fonte: Autor

5.3 Máquina de Estados

A [Figura 27](#) mostra o funcionamento do contador com a entrada UP = 1, DOWN = 0 e RESET = 0.

Figura 27 – Forma de onda da Máquina de Estados (UP = 1/DOWN = 0)

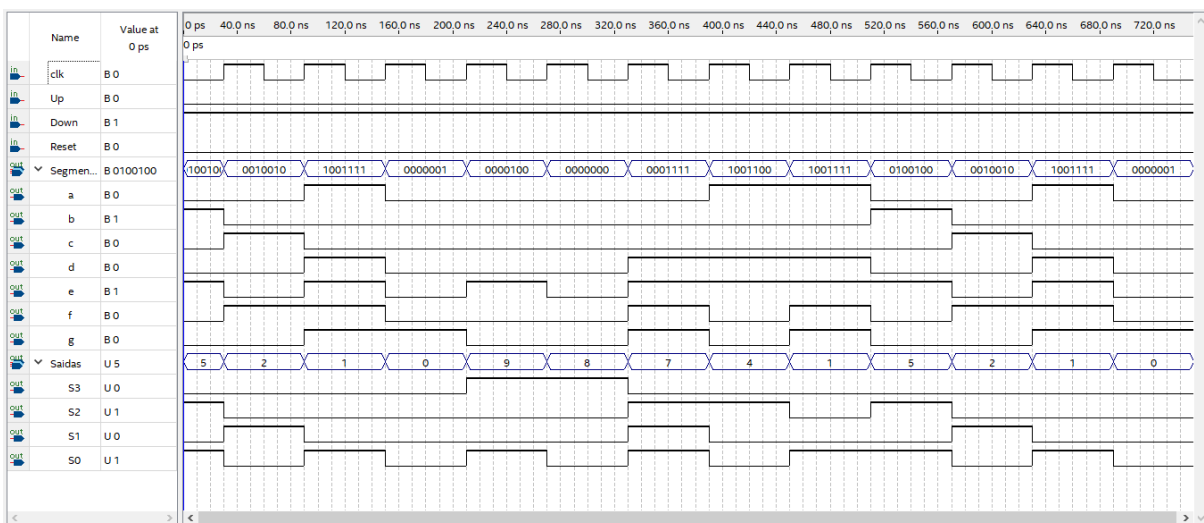


Fonte: Autor

A simulação ocorreu como esperado. Podemos ver que as saídas respeitam a sequência definida na [Tabela 1](#).

Já na [Figura 28](#) a sequência está ao contrário como deveria ser, pois o valor das entradas é DOWN = 1 e UP = 0.

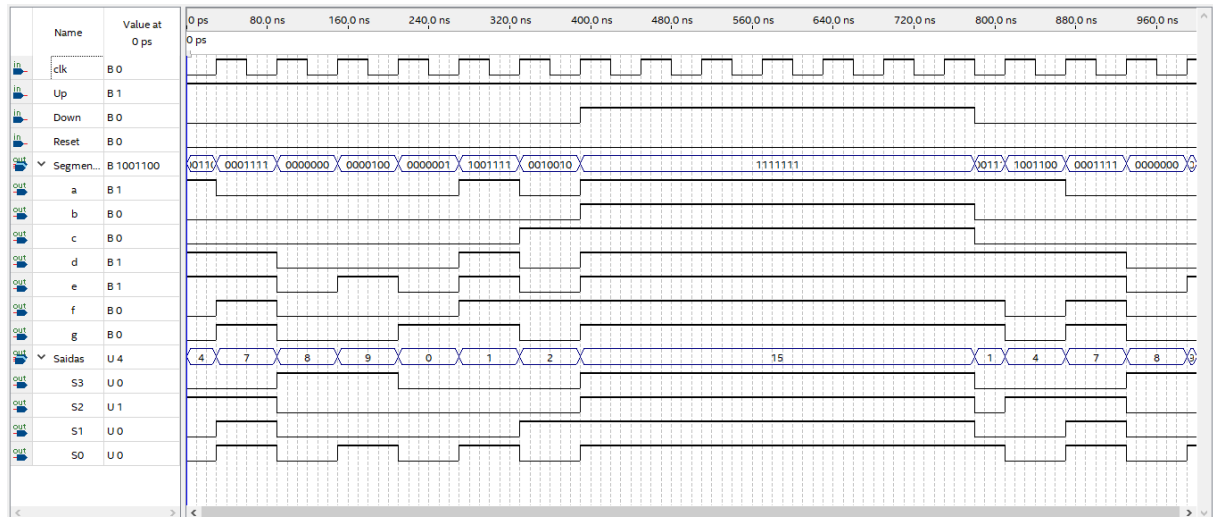
Figura 28 – Forma de onda da Máquina de Estados (UP = 0/DOWN = 1)



Fonte: Autor

A Figura 29 tem como objetivo mostrar o estado *Blank* quando $Up = 1$, $DOWN = 1$ e $RESET = 0$. Nesse caso, a saída 15 é requisitada pela máquina, apagando o display como esperado. A imagem também nos mostra que a contagem, de fato, volta ao número inicial quando saímos do estado *blank* como requisitado.

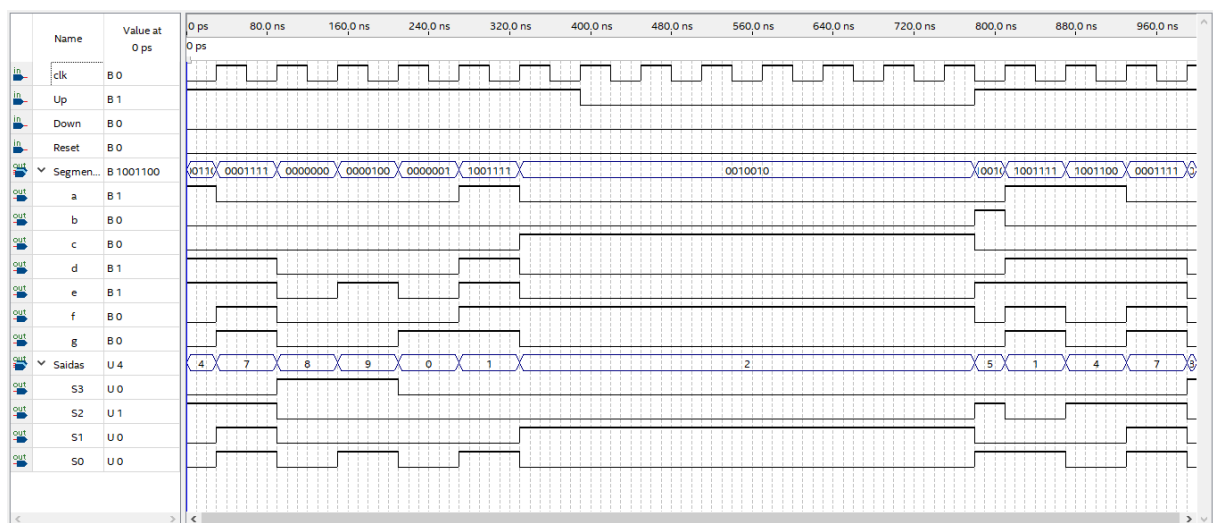
Figura 29 – Forma de onda da Máquina de Estados ($UP = 1/DOWN = 1$)



Fonte: Autor

A Figura 30 demonstra a ação de manter estado descrita na Tabela 8 quando $UP = 0$, $DOWN = 0$ e $RESET = 0$.

Figura 30 – Forma de onda da Máquina de Estados ($UP = 0/DOWN = 0$)

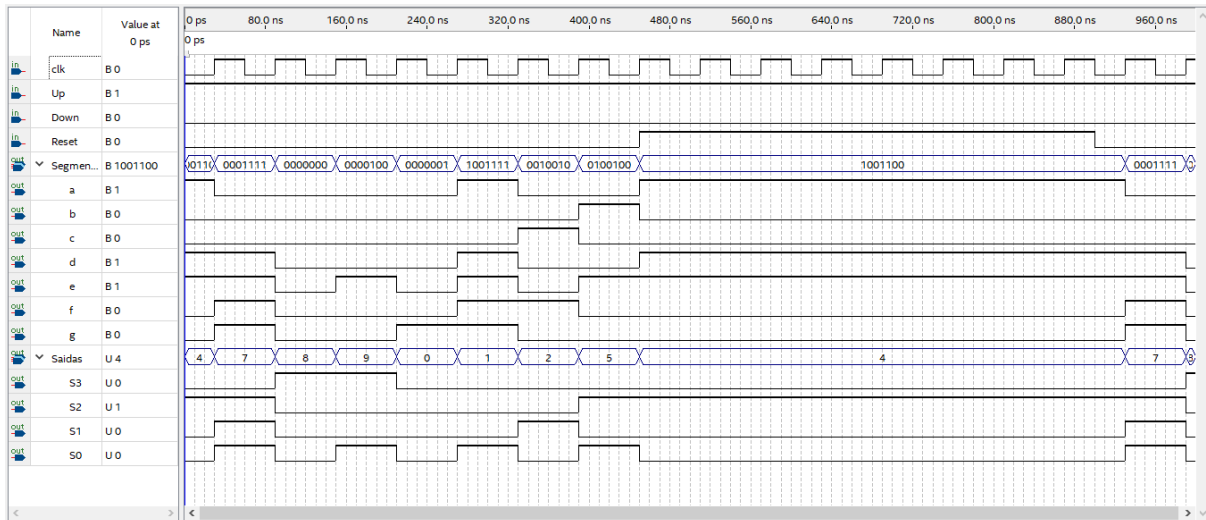


Fonte: Autor

Na Figura 31 podemos ver o funcionamento da entrada RESET. A saída tem valor

igual a 4 pois é o comportamento da máquina do tipo Mealy. Ela se encontra no estado 0000, mas como $UP = 1$ a saída é 4, pois depende diretamente da entrada.

Figura 31 – Forma de onda da Máquina de Estados (RESET = 1)



Fonte: Autor

6 Considerações Finais

Com todos os passos mostrados anteriormente, implementamos, com sucesso, um contador numérico utilizando uma Máquina de Estados do tipo Mealy. O projeto foi trabalhoso, exigindo pesquisa em diversas bibliografias e testes individuais no FPGA durante e fora de aula.

Apesar de ter funcionado com sucesso, utilizar blocos esquemáticos para desenvolver uma Máquina de Estados Finitos não é a melhor opção. A implementação por esquemático é passiva de erros bobos e necessita de mais tempo e atenção dobrada para que obtenhamos sucesso de primeira. Quando algum erro ocorre é extremamente difícil encontrá-lo, como, por exemplo, ao montar os circuitos de saída vistos na [Figura 22](#).

Tendo isso em vista, sob a imposição do modelo esquemático, a melhor estratégia foi desenvolver e testar o projeto por partes antes de uní-lo, mostrando a eficiência das *black-boxes* fornecidas pelo *Quartus Prime*.

Em termos de implementação, a melhor maneira seria através de HDLs. Estas, simplificam a criação de circuitos muito complexos além diminuir consideravelmente o tempo de montagem e facilitar a busca caso ocorra algum erro.

Apesar das dificuldades, o projeto foi de extrema importância, compilando tudo que vimos até o momento e nos testando a um alto nível na projetagem de circuitos digitais.

Referências

- 1 HARRIS, D. M.; HARRIS, S. L. *Digital Design and Computer Architecture*. 2. ed. Waltham/MA, EUA: Morgan Kaufmann, 2013. 690 p. Bibliografia: p. 8, 11, 56, 123, 274. ISBN 978-0-12-394424-5. Citado 4 vezes nas páginas 13, 14, 15 e 17.
- 2 TOCCI, R. J.; WIDMER, N. S.; MOSS, G. L. *Digital Systems: Principles and Applications*. 10. ed. New Jersey, EUA: Pearson, 2007. 921 p. Bibliografia: p. 364, 425-426. ISBN 0-13-172579-3. Citado 3 vezes nas páginas 16, 17 e 18.
- 3 THE NATIONAL SCIENCE FOUNDATION. *JFLAP*. Disponível em: <<http://www.jflap.org/>>. Acesso em: 21 out. 2019. Citado na página 27.
- 4 ONLINE KARNAUGH MAP GENERATOR. *Online Karnaugh Map Generator*. Disponível em: <<http://www.32x8.com/index.html>>. Acesso em: 21 out. 2019. Citado na página 29.