

120.432

***LABORATÓRIO DE SISTEMAS COMPUTACIONAIS
CIRCUITOS DIGITAIS***

**CONTADOR NUMÉRICO A PARTIR DE UMA
MÁQUINA DE ESTADOS DO TIPO MOORE**

Relatório apresentado a Universidade Federal de São Paulo como parte da disciplina de Laboratório de Sistemas Computacionais: Circuitos Digitais

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 2019**

1 Resumo

Atualmente a tecnologia tem ganhado espaço em todos os aspectos do dia a dia: Televisores, satélites, celulares, computadores, controles elétricos, VANTs, etc. Dentro de todos esses equipamentos encontram-se os componentes de circuitos digitais. O estudo desses componentes então se tornaram algo de vital importância. Esses circuitos utilizam apenas sinais de entrada binários: alto ou baixo. A ideia é gerar uma lógica baseada no funcionamento de portas e sinais binários de forma a modelar o circuito final requisitado. Esse projeto visa abordar os principais temas recorrentes aos componentes e funcionamento de circuitos digitais. Será desenvolvido um circuito complexo capaz de fazer uma contagem sequencial em 4 modos de operação utilizando a linguagem verilog. Esse circuito consistirá de 3 principais subcircuitos: Divisor de frequência, máquina de estados e decodificador+display. Em conjunto será demonstrado um método para desenvolvimento de circuitos mais complexos: Máquina de estados de Moore, suas especificações e principais cuidados a serem tomados durante o desenvolvimento. Ao final será ratificado o funcionamento do circuito completo.

Sumário

1	Resumo	1
2	Introdução	5
3	Objetivos	6
3.1	Geral	6
3.1.1	Casos Especiais	6
3.2	Específico	6
4	Fundamentação Teórica	8
4.1	Kit DE2-115	8
4.2	Portas Lógicas e Tabela Verdade	8
4.3	Verilog	9
4.4	Decodificador BCD para display 7 segmentos	9
4.5	Divisor de frequência	10
4.6	Máquina de Estados de Moore	11
5	Desenvolvimento	13
5.1	Máquina de Estados de Moore	13
5.2	Divisor de Frequência	17
5.3	Decodificador BCD para 7 Segmentos	18
6	Resultados e Discussões	19
7	Considerações Finais	23

Lista de Figuras

1	Kit FPGA DE2-115	8
2	Esquemático das principais portas lógicas e suas tabelas verdades	9
3	Display de 7 segmentos	10
4	Etapas de um circuito de Moore	11
5	Diagrama da máquina de estado	14
6	Definição dos argumentos e parâmetros	16
7	Início da Memória	16
8	Memória	17
9	Circuito combinacional de saída	18
10	Circuito divisor de frequência	18
11	Implementação do decodificador	19
12	Forma de onda do display 7 segmentos	19
13	Forma de onda do divisor de frequência	20
14	Forma de onda da máquina parcial crescente	20
15	Forma de onda da máquina parcial decrescente	21
16	Forma de onda da máquina parcial com operação constante	21
17	Forma de onda da máquina parcial com operação BLANK	21
18	Forma de onda da máquina de Moore	22

Lista de Tabelas

1	Conversão BCD para 7 segmentos	10
2	Representação do estado e seu correspondente código em binário	13
3	Tabela Verdade relacionando o estado atual, a entrada e o próximo estado	15
4	Tabela Verdade relacionando o estado atual e a saída	15

2 Introdução

O surgimento do transistor bipolar de junção em 1947[1] foi um dos marcos que possibilitou um avanço no ramo tecnológico. Entre as vantagens de se utilizar um transistor, pode-se citar: aumento da velocidade, diminuição de componentes e do próprio circuito, flexibilidade, possibilidade de se programar através de linguagens de descrição de hardware, etc. Todas essas características facilitaram o desenvolvimento tecnológico, e permitiram a criação de sistemas eficientes e muito mais complexos.

Os circuitos digitais tornaram-se uma ferramenta indispensável na realização de projetos de hardware nos dias de hoje, seus componentes evoluíram de transistores individuais para circuitos integrados e serviram de alavanca para circuitos lógicos programáveis, como o caso da FPGA (*Field Programmable Gate Array*)[2].

Para compreender o funcionamento de componentes complexos e entender a lógica de como são feitos os sistemas tecnológicos, é importante o estudo de circuitos digitais, seus conceitos e aplicações. Dessa forma, esse trabalho faz uma abordagem geral aos temas mais recorrentes na área de sistemas digitais: Portas lógicas (AND, OR, NOT, NOR, NAND, XOR, XNOR), circuito contador, decodificador, divisor de frequência e máquina de estados.

Para a realização do circuito foi utilizado o software de simulação de circuitos digitais: Quartus Prime 1.7.1 e uma placa do kit de desenvolvimento FPGA DE2-115. Além disso durante o processo foi utilizado o wiRed Panda como software de apoio de simulações digitais e o software JFlap para o desenvolvimento do diagrama de estados.

3 Objetivos

3.1 Geral

O objetivo geral do projeto é a implementação de um circuito baseando-se em uma máquina de estados do tipo Moore. Esse circuito deve conter um display que será atualizado a cada segundo (frequência de atualização do display = 1Hz). Além disso, foi definido uma sequência numérica, o qual será apresentado no display e quatro modos de funcionamento distintos.

sequência: 7-3-8-3-5-9-1-8-3

- **Crescente** - Apresenta a sequência de número em sua ordem natural a partir do índice em que estiver.
- **Decrescente** - Apresenta a sequência de números em ordem decrescente ao que foi definido a partir do índice em que estiver.
- **Constante** - Mantém o valor do último número apresentado no display.
- **Vazio** - Nenhum número é apresentado no display.

O circuito deve ser capaz de realizar todas as operações e alternar entre elas em tempo de processamento de acordo com a vontade do usuário.

3.1.1 Casos Especiais

- **Primeiro e último elemento** - Caso esteja selecionada a operação crescente ou decrescente, a sequência de números deverá apresentar o resultado continuamente, portanto ao chegar no último elemento deverá recomençar a contagem como em um loop.
- **Transição de estado vazio** - Ao fazer a transição de um estado vazio para uma operação de contagem, o próximo estado deverá ser o primeiro elemento, tanto no caso crescente, como no caso decrescente.
- **Estado vazio** - O estado vazio não deve ser representado pelo número zero. Caso esse estado seja atingido, todos os segmentos do display deverão se desligar. Para que isso ocorra, foi definido o valor decimal 10, por ser o primeiro valor a ultrapassar o limite do display.

3.2 Específico

Para facilitar a realização do objetivo geral, o circuito pode ser dividido em três outros principais esquemáticos a serem desenvolvidos:

- **Decodificador+Display** - Circuito responsável pela decodificação de um número binário para uma saída específica, representando o valor em decimal em uma faixa de 0 a 9. Esse valor será utilizado de modo coerente no display, o qual irá demonstrar o número correspondente em decimal. Esse circuito tem o objetivo de representar corretamente valores de 0 a 9 em um display. Considerando também que valores fora dessa faixa resultem no display completamente apagado.

- **Divisor de frequência** - Esse circuito é responsável por manter a frequência do display em 1Hz. Analisando de outro modo, nessa parte ocorre a conversão da frequência do ciclo de clock natural da placa FPGA que será usada nesse trabalho para uma frequência desejada.
- **Máquina de estados** - É a parte principal do trabalho. Aqui ocorrem a seleção do estado atual, bem como o cálculo do próximo estado e também o valor de saída (em binário).

Esses três circuitos são montados paralelamente, bem como o teste individual de cada um. Ao final são usados em conjunto com a finalidade do objetivo geral já mencionado na subseção anterior.

4 Fundamentação Teórica

4.1 Kit DE2-115

Foi utilizado a placa do kit DE2-115 de desenvolvimento FPGA produzido pela Altera. O kit é uma plataforma programável de desenvolvimento, o projeto final foi descarregado para essa placa. A Fig. 1 ilustra esse kit.

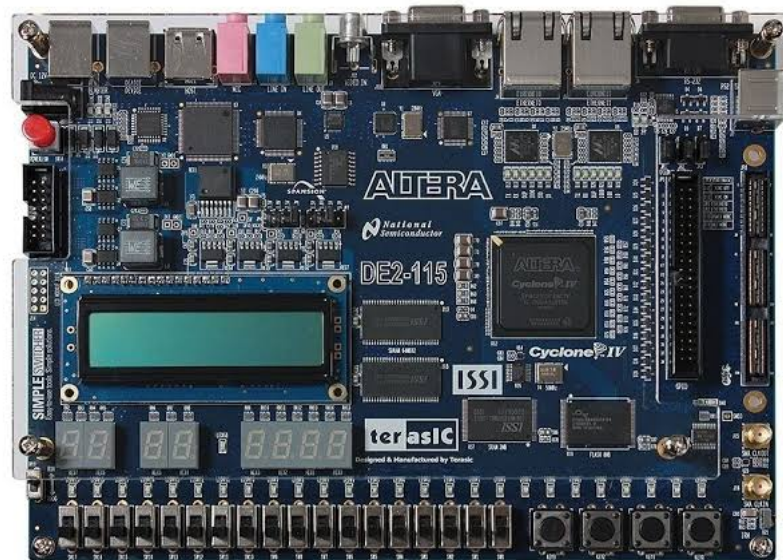


Figura 1: Kit FPGA DE2-115

É possível atribuir aos pinos da placa com as variáveis de entrada e saída do Quartus. Para isso é necessário consultar a pinagem correta, a qual pode ser verificada no manual disponibilizado pela Intel[3]

4.2 Portas Lógicas e Tabela Verdade

As portas lógicas são os elementos mais básicos de um circuito digital. Elas são responsáveis pela forma como as entradas se relacionam gerando uma determinada saída.

Para entender o comportamento de uma porta lógica ou de um circuito é realizado o seguinte experimento: primeiro é anotado o valor do sinal de entrada em uma tabela. O sinal é então enviado para o sistema, onde em seguida será processado e irá gerar uma saída correspondente. O resultado é anotado ao lado do sinal de entrada. Esse procedimento é realizado para todas os valores de entrada. Ao final, é concluída uma tabela conhecida como Tabela Verdade. Ela estará relacionando a entrada e a saída para todas as possíveis combinações de entrada[4], dessa forma é mais fácil analisar a lógica e manter o controle por trás de qualquer sistema.








<p>AND</p>  <p>A AND B = C</p> <table><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	0	0	0	0	1	0	1	0	0	1	1	1	<p>NAND</p>  <p>A NAND B = C</p> <table><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	0	0	1	0	1	1	1	0	1	1	1	0	<p>OR</p>  <p>A OR B = C</p> <table><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	0	0	0	0	1	1	1	0	1	1	1	1	<p>NOR</p>  <p>A NOR B = C</p> <table><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	0	0	1	0	1	0	1	0	0	1	1	0
0	0	0																																																	
0	1	0																																																	
1	0	0																																																	
1	1	1																																																	
0	0	1																																																	
0	1	1																																																	
1	0	1																																																	
1	1	0																																																	
0	0	0																																																	
0	1	1																																																	
1	0	1																																																	
1	1	1																																																	
0	0	1																																																	
0	1	0																																																	
1	0	0																																																	
1	1	0																																																	
<p>XOR</p>  <p>A XOR B = C</p> <table><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	0	0	0	0	1	1	1	0	1	1	1	0	<p>XNOR</p>  <p>A XNOR B = C</p> <table><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	0	0	1	0	1	0	1	0	0	1	1	1	<p>NOT</p>  <p>A NOT = C</p> <table><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	0	1	1	0																					
0	0	0																																																	
0	1	1																																																	
1	0	1																																																	
1	1	0																																																	
0	0	1																																																	
0	1	0																																																	
1	0	0																																																	
1	1	1																																																	
0	1																																																		
1	0																																																		

Figura 2: Esquemático das principais portas lógicas e suas tabelas verdades

Considerando uma análise das portas lógicas para apenas duas entradas:

- **AND** - Saída alta apenas quando as duas entradas estão em alta.
- **NAND** - Saída alta se qualquer uma das entradas for baixa.
- **OR** - Saída alta se qualquer uma das duas entradas estão em alta.
- **NOR** - Saída alta se as duas entradas estiverem em alta.
- **XOR** - Saída alta se houver obrigatoriamente apenas uma entrada alta.
- **XNOR** - Saída alta se as entradas tiverem o mesmo sinal.
- **NOT** - Inverte o valor da saída.

4.3 Verilog

Da subseção anterior, é possível montar diversos sistemas, e estes podem acabar ficando muito complexo e difícil de implementar. A fim de facilitar a modelagem de sistemas elétricos surgiu a linguagem de descrição de hardware Verilog.

4.4 Decodificador BCD para display 7 segmentos

Como apresentado nos objetivos, será mostrado em um display os valores de uma sequência numérica. O número a ser mostrado será calculado em nível de código entretanto a conversão desse número para um display não é compreendida pelo computador de forma automática.

O display possui 7 segmentos que recebem sinais alto ou baixo, o qual irá definir se o segmento irá acender ou apagar. Para o caso desse kit de desenvolvimento em específico,

para um sinal de nível alto o segmento estará apagado e para um sinal de nível baixo o segmento irá acender.

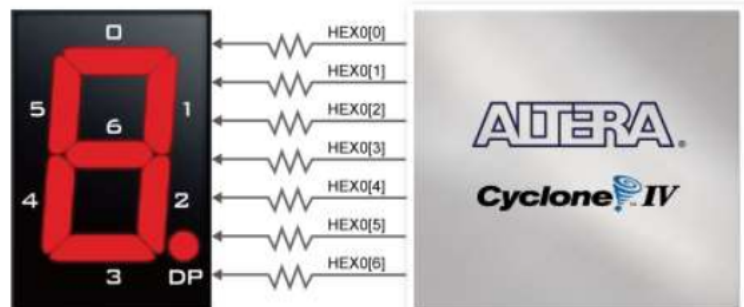


Figura 3: Display de 7 segmentos

Dessa forma, com o auxílio da Fig. 3 e com valores decimais de 0 a 9, é necessário fazer a conversão de sinais como mostra a Tab. 1. Considerando a sequência do bit mais significativo para o menos significativo da saída, esses valores correspondem respectivamente aos segmentos $\{0,1,2,3,4,5,6\}$

Decimal	BCD	Saída
0	0000	0000001
1	0001	1001111
2	0010	0010010
3	0011	0000110
4	0100	1001100
5	0101	0100100
6	0110	0100000
7	0111	0001111
8	1000	0000000
9	1001	0000100

Tabela 1: Conversão BCD para 7 segmentos

Além disso, por padrão qualquer valor fora da margem decimal citada ativará todos os segmentos, fazendo com que o display fique apagado.

4.5 Divisor de frequência

O projeto funciona de acordo com a atualização constante de seu estado atual. Essa atualização é síncrona e portanto é dependente do clock intrínseco do kit FPGA. Tendo em vista que o clock funciona a uma frequência de 50MHz, não é interessante utilizá-lo diretamente. Para resolver esse problema, foi desenvolvido um circuito divisor de frequência.

O circuito divisor de frequência basicamente cria um clock (será chamado de clock artificial por conveniência) baseado no clock da FPGA. De acordo com as especificações desse projeto, o estado atual deve ser atualizado a cada 1 segundo.

Seja n o número de clocks, t o tempo desejado de atualização, e f a frequência do

clock, segue da Eq. 1 a quantidade de ciclos de clocks necessárias para que o clock artificial atualize uma única vez.

$$n = \frac{t}{f} \quad (1)$$

4.6 Máquina de Estados de Moore

Uma máquina de estados de Moore é uma forma de implementar um circuito sequencial finito. Ao se referir a esse tipo de sistema, é comumente utilizado o conceito de estado. Basicamente o estado é a forma como o circuito se encontra. Uma das principais características da máquina de Moore é o fato da saída depender apenas do estado atual. Outra característica que provém do mesmo fato, é que os valores de saída ocorrem dentro dos próprios estados, e não durante a transição dos estados (Como seria em uma máquina de Mealy).

A Fig. 4 mostra um esquemático geral de como é montado um circuito de Mealy.

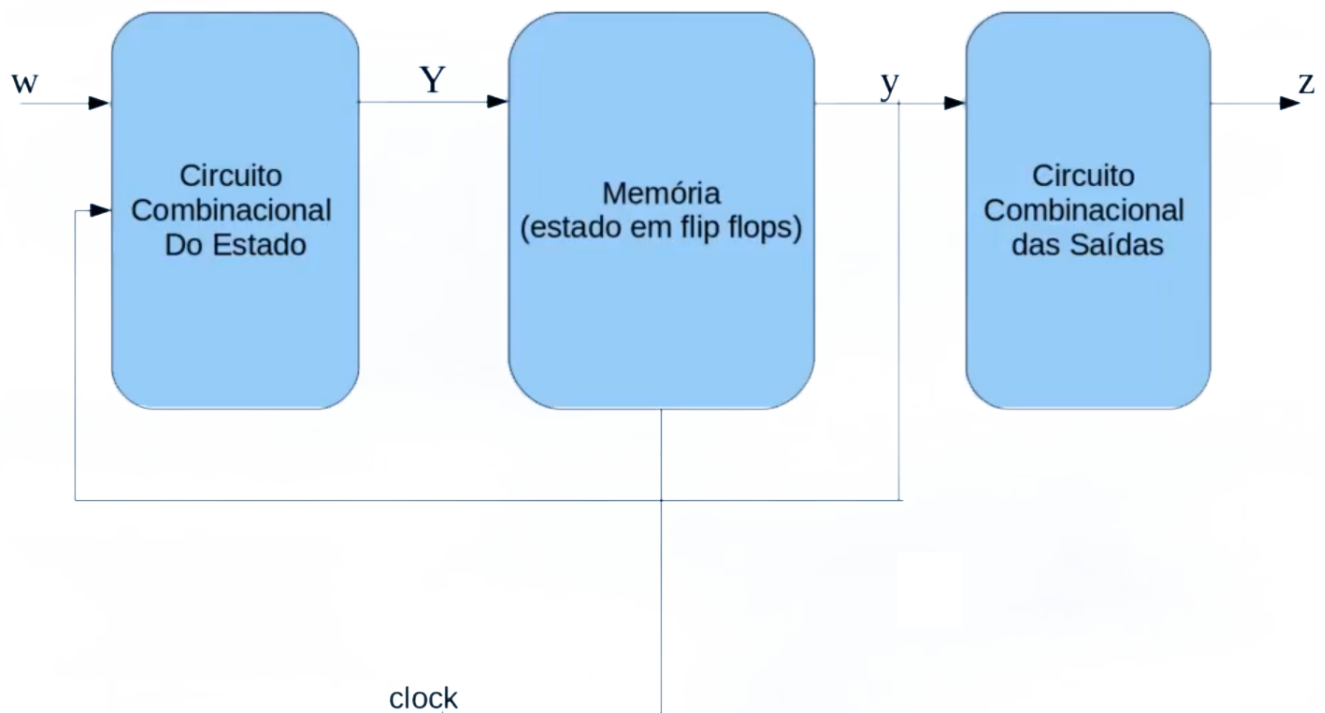


Figura 4: Etapas de um circuito de Moore

Para o funcionamento da máquina de estados é necessário o acesso a três informações: O estado atual, o próximo estado e a saída. Seguindo a Fig. 4, a primeira etapa se passa pelo circuito combinacional do estado. Nessa parte os sinais tanto de entrada do sistema (representado por w na figura), como o do estado atual (y) são usados para se calcular o próximo estado.

No segundo elemento da figura está representado a memória, onde guarda o código do estado atual. Para o seu funcionamento é importante a conexão com um clock que define a frequência de atualização (será usado o clock artificial) e um sinal (Y) que irá setar o

próximo estado como estado atual.

O circuito combinacional de saída da figura está relacionado ao sinal de interesse do sistema. Sabe-se por definição que a saída depende apenas do estado atual (y).

5 Desenvolvimento

5.1 Máquina de Estados de Moore

Para o desenvolvimento da máquina de estado de moore, primeiramente foram listados todos os possíveis estados e seus códigos correspondente. Essas informações podem ser consultadas na Tab. 2.

Estado	Código em binário
S1	0000
S2	0001
S3	0010
S4	0011
S5	0100
S6	0101
S7	0110
S8	0111
S9	1000
BLANK	1001

Tabela 2: Representação do estado e seu correspondente código em binário

Segundo as especificações do projeto, precisam haver 4 diferentes tipos de operações, além disso as operações são escolhidas por chaves. Dessa forma como cada combinação binária de chave seletora pode corresponder a uma diferente operação, têm-se da Eq. 2 a quantidade de chaves seletoras necessárias para o projeo.

$$Q_s = \log_2 N \quad (2)$$

Onde Q_s é a quantidade de chaves seletoras e N o número de operações. Aplicando a Eq. 2 temos que são necessárias $\log_2 4 = 2$ chaves seletoras.

Para descobrir a quantidade de bits necessário para representar todos os possíveis estados, basta calcular de modo análogo ao anterior. Sabendo que são 10 diferentes estados (Tab. 1), seriam necessários $\log_2 10 = 3,32$ bits, esse valor é arredondado para o maior inteiro seguinte, resultando em 4 bits.

Os valores de saída possuem uma faixa de valores entre 0-10, logo a quantidade de bits necessárias para representar seriam $\log_2 10$, o que cai exatamente no mesmo caso que anteriormente, resultando em 4 bits.

Tendo definido a nomenclatura usada no circuito, foi montado um diagrama de estados (Fig. 5) para auxiliar no desenvolvimento restante do projeto. Note que são representados apenas 10 estados enquanto poderiam estar sendo representados 16 devido aos 4 bits dos estados. Isso ocorre porque as outras combinações são estados inalcançáveis, e portanto não são de interesse para o projeto. Seria possível definir todos os estados, porém todos seriam estados de blank, sendo redundante a utilização deles.

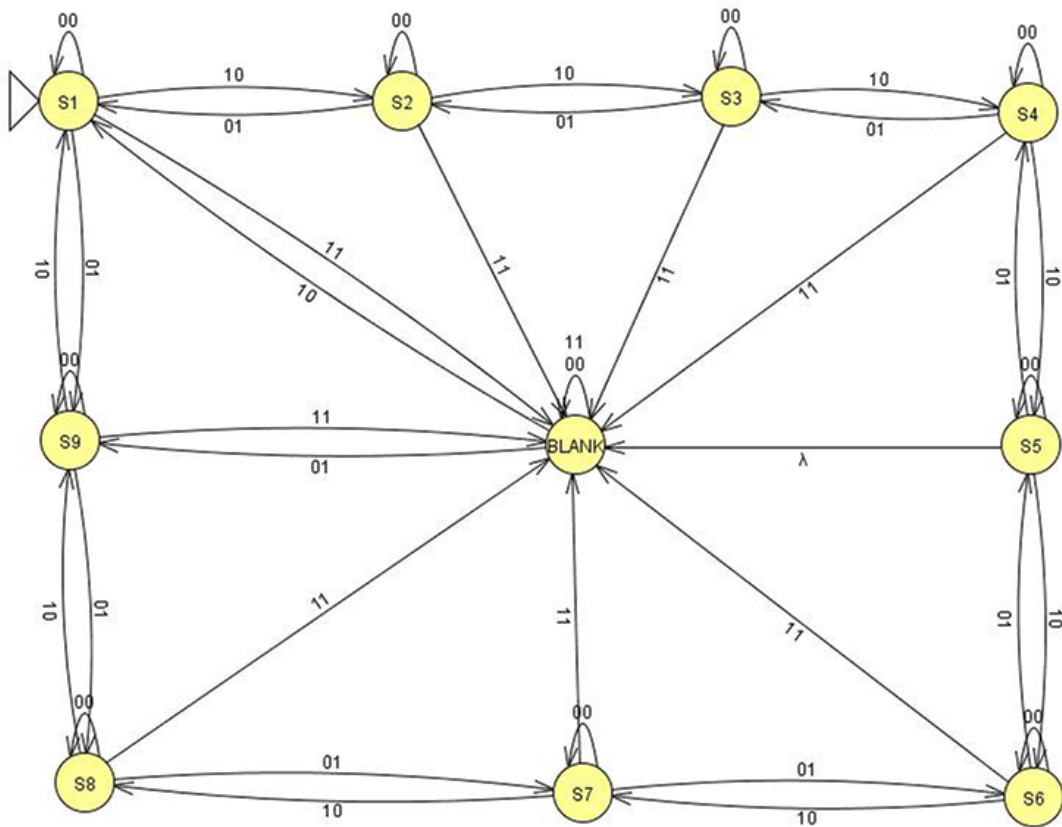


Figura 5: Diagrama da máquina de estado

Nota-se um certo padrão ao observar o diagrama. A análise inicia na posição do estado inicial "S1". A partir dele, para uma ordem crescente, o fluxo das transições seguem sempre para o próximo estado, por isso formam uma sequência circular na imagem. De mesmo modo ocorre quando está selecionado a opção decrescente, o fluxo percorre o sentido anti-horário. Além disso todos possuem um loop para a própria posição devido a operação de manter o valor. Já a operação "limpa" ocorre na transição de qualquer estado para o "BLANK", por isso todos possuem uma conexão em direção a esse estado.

Como cada transição depende de uma das 4 entradas de operação, outra forma visual de garantir que o diagrama esteja correto é verificar se cada estado possui 4 transições saindo dele. As Tab. 3 e Tab. 4 mostram a Tabela Verdade dos valores de estado atual e saída.

Estado Atual ($Q_3Q_2Q_1Q_0$)	Mantém(00) ($Q'_3Q'_2Q'_1Q'_0$)	Crescente(10) ($Q'_3Q'_2Q'_1Q'_0$)	Decrescente(01) ($Q'_3Q'_2Q'_1Q'_0$)	Limpa(11) ($Q'_3Q'_2Q'_1Q'_0$)
S1 (0001)	S1 (0001)	S2 (0010)	S9 (1001)	BLANK (1010)
S2 (0010)	S2 (0010)	S3 (0011)	S1 (0001)	BLANK (1010)
S3 (0011)	S3 (0011)	S4 (0100)	S2 (0010)	BLANK (1010)
S4 (0100)	S4 (0100)	S5 (0101)	S3 (0011)	BLANK (1010)
S5 (0101)	S5 (0101)	S6 (0110)	S4 (0100)	BLANK (1010)
S6 (0110)	S6 (0110)	S7 (0111)	S5 (0101)	BLANK (1010)
S7 (0111)	S7 (0111)	S8 (1000)	S6 (0110)	BLANK (1010)
S8 (1000)	S8 (1000)	S9 (1001)	S7 (0111)	BLANK (1010)
S9 (1001)	S9 (1001)	S1 (0001)	S8 (1000)	BLANK (1010)
BLANK (1010)	BLANK (1010)	S1 (0001)	S9 (1001)	BLANK (1010)

Tabela 3: Tabela Verdade relacionando o estado atual, a entrada e o próximo estado

Estado Atual ($Q_3Q_2Q_1Q_0$)	Saída ($O_3O_2O_1O_0$)
0001 (S1)	0111
0010 (S2)	0011
0011 (S3)	1000
0100 (S4)	0011
0101 (S5)	0101
0110 (S6)	1001
0111 (S7)	0001
1000 (S8)	1000
1001 (S9)	0011
1010 (BLANK)	1010

Tabela 4: Tabela Verdade relacionando o estado atual e a saída

Focando um pouco na parte da implementação, primeiramente foram definidos os argumentos de entrada e saída do escopo principal. Seriam esses: o clock natural (Que será convertido em um clock artificial no divisor de frequência), o reset, as entradas seletoras (para definir a operação) e os valores de saídas do display. A Fig. 6 demonstra essa implementação. Lembrando que os valores atribuídos aos estados são apenas códigos identificadores para reconhecimento.

Em seguida, é iniciado o bloco always, tendo como parâmetros o clock já ajustado para 1 segundo e a chave de reset. Dessa forma, o reset se encontra assíncrono com o clock. Uma vez que o bloco é atualizado sempre que um dos parâmetros atualizar.

Após isso, é feito todas as possíveis atribuições do estado atual e do próximo estado, de acordo com a entrada seletora. Para cada estado são considerados 4 operações possíveis, por isso existem 4 declarações condicionais para cada um.

Por fim, cada estado é implementado o circuito combinacional de saída, onde baseado no estado atual, é decodificado o valor da saída correspondente. Essa tarefa se torna muito mais simplificada a nível de código. Basta fazer uma verificação para saber o estado atual, e atribuir os valores diretamente para um registrador de saída.

Segue as Fig. 6 a Fig. 9 a implementação principal da máquina de estado.


```

28 module maquinaMoore(clockNatural, reset, seletor, displayout);
29
30 input clockNatural, reset;
31 wire clockArtificial;
32 input [1:0] seletor;
33 reg [3:0] saida;
34 output [6:0] displayout;
35 reg [3:0] estadoAtual;
36
37
38 parameter S1 = 4'd0, S2 = 4'd1, S3 = 4'd2, S4 = 4'd3,
39 S5 = 4'd4, S6 = 4'd5, S7 = 4'd6, S8 = 4'd7, S9 = 4'd8,
40 blank = 4'd9;
41

```

Figura 6: Definição dos argumentos e parâmetros

```

44 always @(posedge clockArtificial, posedge reset)
45 begin
46     if (reset)
47         estadoAtual <= S1;
48

```

Figura 7: Inicio da Memória

<pre> 50 else 51 case (estadoAtual) 52 53 S1: 54 if (seletor == 2'b00) 55 estadoAtual <= S1; 56 else if (seletor == 2'b01) 57 estadoAtual <= S9; 58 else if (seletor == 2'b10) 59 estadoAtual <= S2; 60 else 61 estadoAtual <= blank; 62 63 S2: 64 if (seletor == 2'b00) 65 estadoAtual <= S2; 66 else if (seletor == 2'b01) 67 estadoAtual <= S1; 68 else if (seletor == 2'b10) 69 estadoAtual <= S3; 70 else 71 estadoAtual <= blank; 72 73 S3: 74 if (seletor == 2'b00) 75 estadoAtual <= S3; 76 else if (seletor == 2'b01) 77 estadoAtual <= S2; 78 else if (seletor == 2'b10) 79 estadoAtual <= S4; 80 else 81 estadoAtual <= blank; 82 83 S4: 84 if (seletor == 2'b00) 85 estadoAtual <= S4; 86 else if (seletor == 2'b01) 87 estadoAtual <= S3; 88 else if (seletor == 2'b10) 89 estadoAtual <= S5; 90 else 91 estadoAtual <= blank; 92 93 S5: 94 if (seletor == 2'b00) 95 estadoAtual <= S5; 96 else if (seletor == 2'b01) 97 estadoAtual <= S4; 98 else if (seletor == 2'b10) 99 estadoAtual <= S6; 100 else 101 estadoAtual <= blank; </pre>	<pre> 103 104 S6: 105 if (seletor == 2'b00) 106 estadoAtual <= S6; 107 else if (seletor == 2'b01) 108 estadoAtual <= S5; 109 else if (seletor == 2'b10) 110 estadoAtual <= S7; 111 else 112 estadoAtual <= blank; 113 114 S7: 115 if (seletor == 2'b00) 116 estadoAtual <= S7; 117 else if (seletor == 2'b01) 118 estadoAtual <= S6; 119 else if (seletor == 2'b10) 120 estadoAtual <= S8; 121 else 122 estadoAtual <= blank; 123 124 S8: 125 if (seletor == 2'b00) 126 estadoAtual <= S8; 127 else if (seletor == 2'b01) 128 estadoAtual <= S7; 129 else if (seletor == 2'b10) 130 estadoAtual <= S9; 131 else 132 estadoAtual <= blank; 133 134 S9: 135 if (seletor == 2'b00) 136 estadoAtual <= S9; 137 else if (seletor == 2'b01) 138 estadoAtual <= S8; 139 else if (seletor == 2'b10) 140 estadoAtual <= S1; 141 else 142 estadoAtual <= blank; 143 144 blank: 145 if (seletor == 2'b00) 146 estadoAtual <= blank; 147 else if (seletor == 2'b01) 148 estadoAtual <= S9; 149 else if (seletor == 2'b10) 150 estadoAtual <= S1; 151 else 152 estadoAtual <= blank; 153 154 default: estadoAtual <= 4'bxxxx; 155 endcase </pre>
--	--

Figura 8: Memória

```

157 always @(estadoAtual)
158 begin
159     case(estadoAtual)
160         S1:
161             saida = 4'b0111; //7
162
163         S2:
164             saida = 4'b0011; //3
165
166         S3:
167             saida = 4'b1000; //8
168
169         S4:
170             saida = 4'b0011; //3
171
172         S5:
173             saida = 4'b0101; //5
174
175         S6:
176             saida = 4'b1001; //9
177
178         S7:
179             saida = 4'b0001; //1
180
181         S8:
182             saida = 4'b1000; //8
183
184         S9:
185             saida = 4'b0011; //3
186
187         blank:
188             saida = 4'b1010; //10
189     endcase
190 end

```

Figura 9: Circuito combinacional de saída

5.2 Divisor de Frequência

A ideia do divisor de frequência é fazer uma contagem de 0 até 50M, e quando chegar nesse valor a saída será 1 e o contador irá começar novamente do 0. Dessa forma, será garantido que dado um clock de 50MHZ, para cada vez que se passar pelo divisor de frequência, passará 1 segundo para produzir uma saída em nível alto. Para a construção do contador, foi utilizado um vetor de registradores. Logo, para cada iteração do clock natural, o vetor de registradores é aumentado em 1d. Segue a Fig. 10

```
199 module divisorFreq(in, out);
200     input in;
201     output wire out;
202
203     parameter n = 25;
204     reg [n:0] cont;
205     reg value;
206
207     always@(posedge in)
208     begin
209         cont <= cont + 1;
210         if(cont == 26'b10111110101111000010000000) /*26'b0000000000000000000000010*/
211         begin
212             value <= 1;
213             cont <= 26'b0;
214         end
215         else
216             value <= 0;
217     end
218
219     assign out = value;
220 endmodule
```

Figura 10: Circuito divisor de frequência

Como o circuito deve ser capaz de fazer uma contagem até 50M, a quantidade de registradores necessário será a mesma que a quantidade de bits necessários para representar 50M em binário. Logo foi usado um vetor de registradores de tamanho $\lceil \log_2 50M \rceil = 26$.

Além disso, a condição de parada verifica se o contador já atingiu o numero 50M em binário. Caso se tenha atingido, o valor de saída é alterado para nível alto, enquanto que o contador recebe 0 para iniciar a contagem novamente. Caso não se tenha atingido, o valor de saída recebe nível baixo.

Como esse é o circuito para o ajuste do clock artificial, o valor de saída corresponde ao valor do clock artificial.

5.3 Decodificador BCD para 7 Segmentos

Como a saída desse circuito é um display de 7 segmentos, foi definido um vetor saída de tamanho 7, onde cada índice corresponde a um segmento. Para se fazer a conversão de BCD para o display, basta apenas atribuir os sinais ao vetor de acordo com a Tab. 1 mostrada na fundamentação teórica.

Sempre que o valor mudar, será verificado qual a sua magnitude em decimal, e a saída do display receberá os valores de acordo. Caso não seja um valor dentro dos limites definidos (0 a 9), será redirecionado para o *default* onde todos os segmentos terão níveis alto e o display ficará apagado.

Segue a Fig. 11 a implementação do decodificador

```

1  module display(in, out);
2
3      input wire[3:0]in;
4      reg [6:0]disp;
5      output wire [6:0]out;
6
7
8      always@(in)
9      begin
10         case(in)
11             0: disp = 7'b0000001;
12             1: disp = 7'b1001111;
13             2: disp = 7'b0010010;
14             3: disp = 7'b0000110;
15             4: disp = 7'b1001100;
16             5: disp = 7'b0100100;
17             6: disp = 7'b0100000;
18             7: disp = 7'b0001111;
19             8: disp = 7'b0000000;
20             9: disp = 7'b0000100;
21             default: disp = 7'b1111111;
22         endcase
23     end
24     assign out = disp;
25 endmodule

```

Figura 11: Implementação do decodificador

6 Resultados e Discussões

Para analisar os resultados, em um primeiro momento serão separados os três circuitos e gerado a forma de onda respectiva a cada um. Após demonstrar os resultados paralelos, será gerado a forma de onda do circuito completo.

Primeiro temos a forma de onda referente aos displays. Assumindo a Tab. 1 como correta, é esperado que após o número 9, todas as próximas saídas terão os segmentos em nível alto, significando que o display estará apagado. Para a simulação foi usado os valores padrões (clock com período de 10ns).

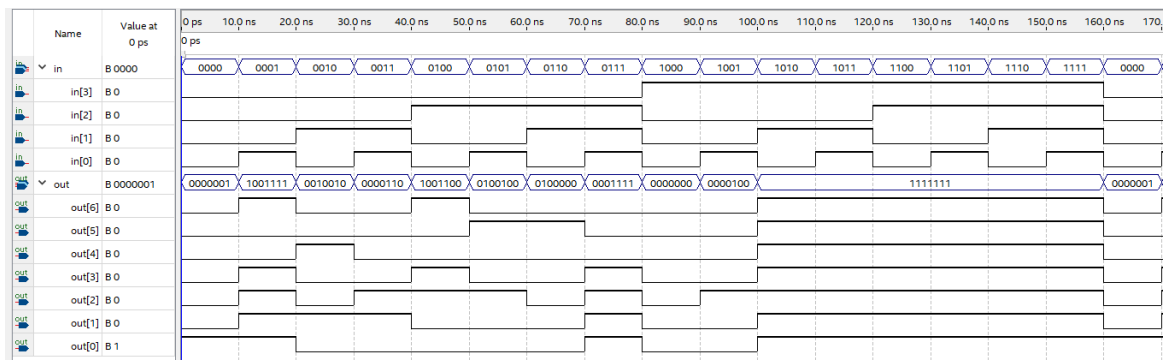


Figura 12: Forma de onda do display 7 segmentos

De fato o display segue os valores apresentados na Tab. 1, e também respeita os limites do display para valores fora de sua faixa de operação. Além disso quando a contagem recomeça o display também volta a funcionar normalmente.

Agora será analisado o divisor de frequência. Como o seu funcionamento foi implementado para uma quantidade de 50M clock, é impossível verificar qualquer alteração observando a forma de onda, uma vez que o tempo de simulação não seria suficiente para rodar os 50M de ciclos. Devido a esse problema, será considerado uma divisão de 3 ciclos

de clock apenas para as simulações. A única diferença de implementação, será mudar o valor dentro da declaração *if* da Fig. 10 para a quantidade de ciclos desejada. Para facilitar o entendimento, foi comentado no canto direito do código o valor a ser usado em simulação.

O período do ciclo de clock simulado, assim como anteriormente, foi de 10ns. A saída esperada é um sinal em nível alto sempre em clocks de multiplicidade 3 ao de entrada.

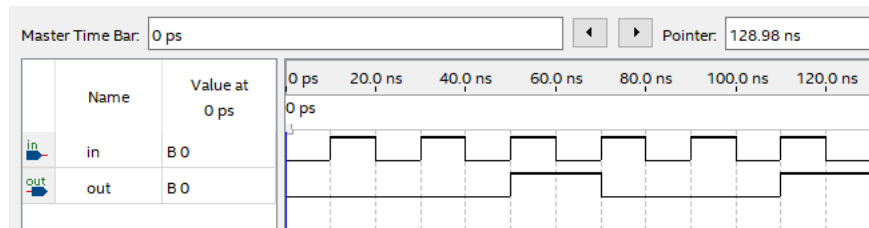


Figura 13: Forma de onda do divisor de frequência

O resultado bate de acordo com o previsto. Dessa forma pode-se esperar que ao trocar a contagem máxima para 50M, o tempo demorado de processamento com um clock de 50MHz seja de 1 segundo.

Por fim, o último circuito refere-se ao funcionamento da parte principal da máquina. Para a simulação houve a necessidade de adicionar uma saída de 4 bits para a geração da forma de onda, e também foi usado o clock natural para atualização dos estados e valores. Como a saída não é a saída do display, como no caso do circuito completo, fica mais fácil a análise do resultado. É esperado, para a operação em modo crescente (10), o valor em binário da sequência de número mostrada nos objetivos (7-3-8-3-5-9-1-8-3)

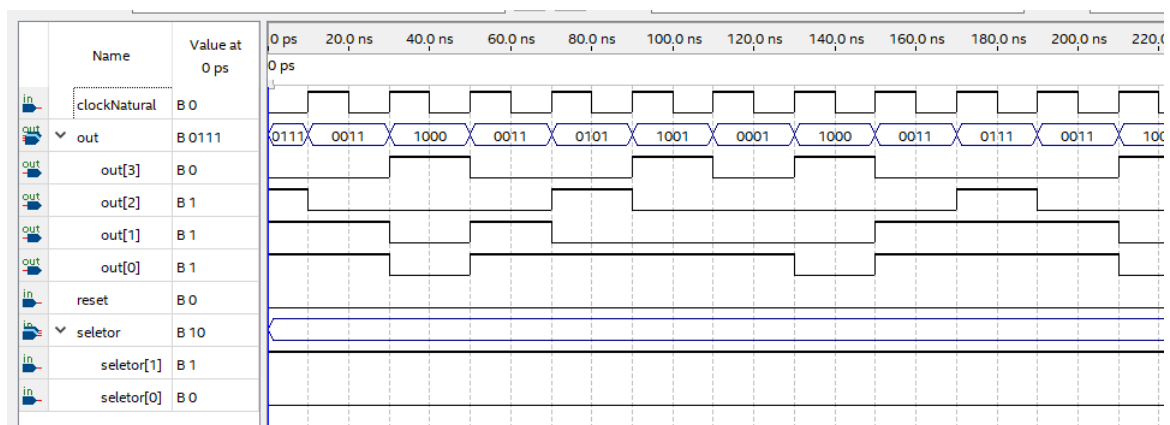


Figura 14: Forma de onda da máquina parcial crescente

De fato os números estão seguindo a sequência esperada, e quando chegam ao último valor a contagem recomeça (no caso, em 170ns).

A fim de demonstrar as outras operações, as Fig. 15, Fig. 16, Fig. 17 representam respectivamente: o caso decrescente, a aplicação do estado constante (00) no meio de uma operação crescente, e a aplicação do estado BLANK (11) tanto no modo crescente quanto no decrescente.

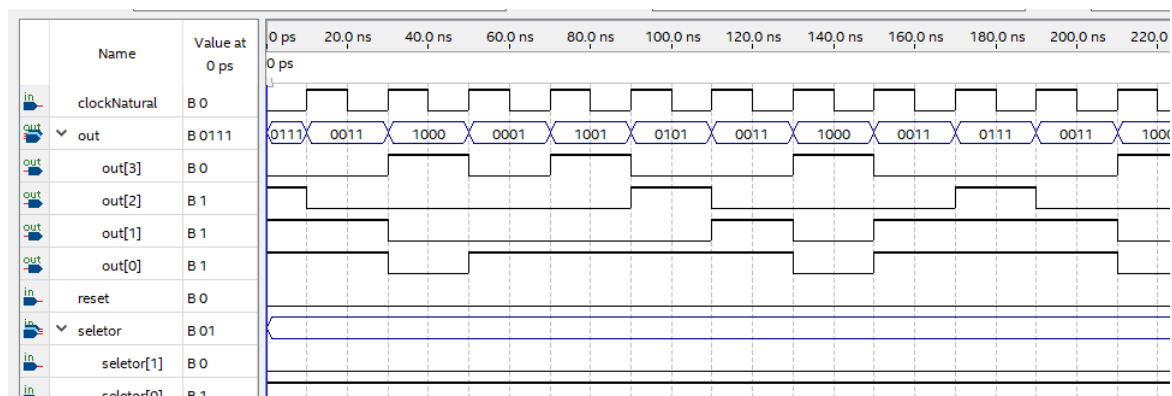


Figura 15: Forma de onda da máquina parcial decrescente

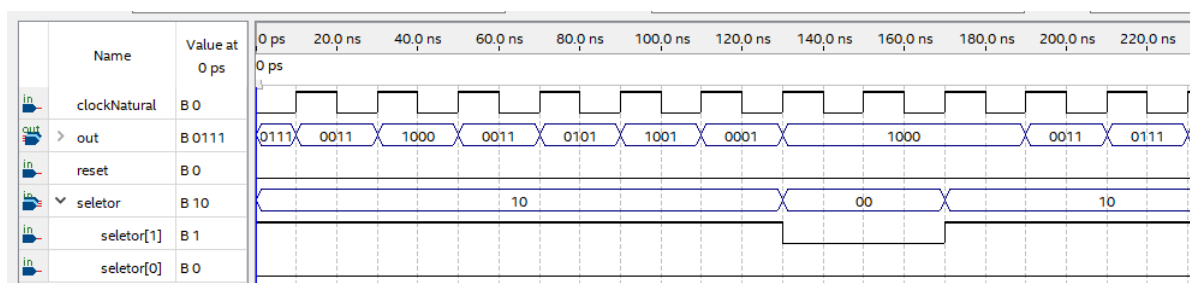


Figura 16: Forma de onda da máquina parcial com operação constante

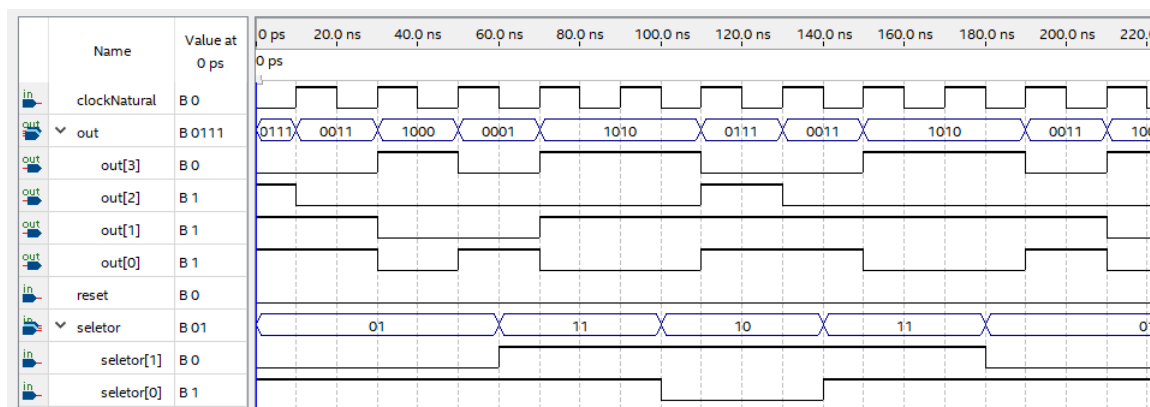


Figura 17: Forma de onda da máquina parcial com operação BLANK

Por fim, a Fig. 18 mostra a forma de onda do circuito completo (Usando um clock artificial relativo a 3 clocks). Para auxiliar na análise também foi plotado o valor em decimal que deveria aparecer no display.

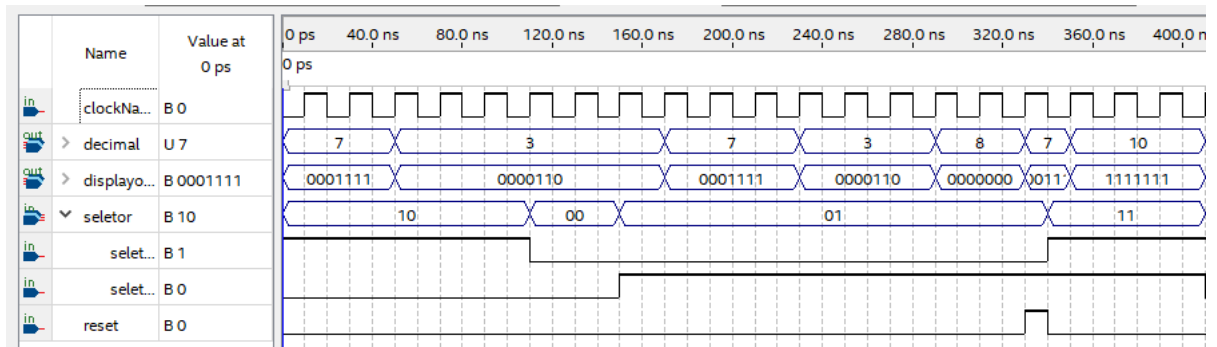


Figura 18: Forma de onda da máquina de Moore

Da forma como se encontra, o estado é sempre verificado a cada 3 ciclos de clock natural, onde irá verificar o estado atual e qual o tipo de operação deverá ser realizada.

Foi mostrado algumas situações como: a passagem de crescente para a operação constante; a posterior retomada partindo desse estado para o funcionamento decrescente; o reset assíncrono retornando para o estado inicial e a passagem para o estado blank.

7 Considerações Finais

Esse projeto desenvolveu um circuito contador sequencial baseado em Máquina de Moore, mostrando os principais pontos, os cuidados a serem tomados, como sintetizar todos os elementos, etc. De início deve-se ter claramente qual o objetivo final a ser atingido (seção 3). Como requisito deve-se saber o funcionamento de elementos e circuitos básicos a serem utilizados (seção 4). Por fim chega na etapa de desenvolvimento do sistema, montando as Tabelas Verdade, diagrama de estados e a implementação em si (seção 5), para então obter e verificar os resultados gerados (seção 6).

Um problema encontrado ao decorrer do projeto foi na definição da transição de um estado. Mais especificamente, um dos requerimentos era que ao sair do estado BLANK para uma sequência decrescente, o próximo estado deveria ser o estado inicial (Requisição apontada na seção 3.1.1 de casos especiais como transição de estado vazio), entretanto ao longo do desenvolvimento do projeto foi considerado que o próximo estado era o último estado (S9). Esse erro pode ser notado tanto na Tab. 3, quanto no diagrama de estado da Fig. 5 e não foi corrigido para manter fiel à apresentação do trabalho (foi apresentado com o erro). Apesar disso, é facilmente corrigível ao se alterar o estado S9 para S1 da linha 146 do código, Fig. 8.

Uma das vantagens na utilização do verilog é na simplicidade e compressão em que os códigos podem ser feitos. Enquanto para um esquemático digital esse circuito seria muito complexo com uma grande quantidade de fios, propiciando aumento de erros em conexões erradas, o mesmo circuito a nível de código fica muito enxuto e fácil de localizar e resolver os erros, como o citado anteriormente.

A utilização do método de Moore é funcional e pode ser de grande ajuda para desenvolvimento de circuitos. A padronização de um método unificado a uma implementação a nível de código pode ajudar no desenvolvimento de circuitos com uma lógica complexa a primeira vista.

Referências

- [1] Ligia Souza Palma. *ENG26 Sistemas Lógicos*. 2007.
- [2] Luiza Maria Romeiro Codá. *DISPOSITIVOS LÓGICOS PROGRAMÁVEIS*. 2014.
- [3] https://www.intel.com/content/dam/altera-www/global/en_US/portal/dsn/42/doc-us-dsnbk-42-1404062209-de2-115-user-manual.pdf.
- [4] Prof. Jorge H. B. Casagrande. *Eletrônica Digital 1*. 2005.