UNIVERSIDADE DE BRASÍLIA

Adrianne Alves da Silva - 160047595
Ana Paula Gomes de Matos - 160023629

TURMA "F"
PRÁTICA DE ELETRÔNICA DIGITAL - 119466
Experimento 04: Circuitos codificadores

Brasília - DF
19 de maio de 2017

UNIVERSIDADE DE BRASÍLIA

PRÁTICA DE ELETRÔNICA DIGITAL - 119466

Relatório referente ao quarto experimento da disciplina Prática de Eletrônica Digital, ministrada pela professora Lourdes Mattos Brasil, realizado no dia 12 de maio de 2017.

Alunas:

Adrianne Alves da Silva

Ana Paula Gomes de Matos

Brasília - DF

SUMÁRIO

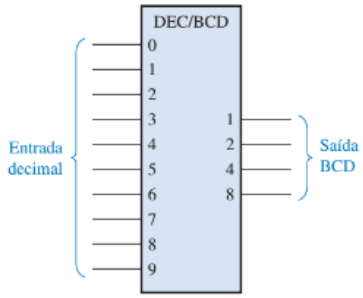
1 INTRODUÇÃO

1.1 Circuitos Codificadores

Para otimizar o tempo de execução de determinada atividade têm-se investido em diversos tipos de tecnologia, pois o ritmo de vida se tornou mais acelerado nas últimas décadas. Nesse sentido, os circuitos codificadores vêm para facilitar essas operações, o que acontece, por exemplo, em teclados que codificam os dígitos inseridos para BCD para que eles possam ser utilizados em outros circuitos. Dessa forma, os codificadores possuem uma ampla gama de atuação.

O codificador é de maneira simplificada um circuito lógico que transforma os valores de sua entrada em determinada base para outra, como por exemplo, decimal para binário. Sendo assim, a sua funcionalidade depende das bases das quais se tratam o conversor, dessa maneira, há conversor binário-BCD, BCD – binário, binário-código gray, entre outros. Esses comportamentos podem ser expressos nas figuras a seguir.[3]

Figura 01: Codificador Decimal BCD



Podemos perceber que há 10 entradas pois correspondem aos 10 símbolos do sistema decimal e uma saída em 4 bits, que é o número necessário para representação em BCD. Isso pode ser observado através da tabela verdade que representa essa transformação (tab. 01).

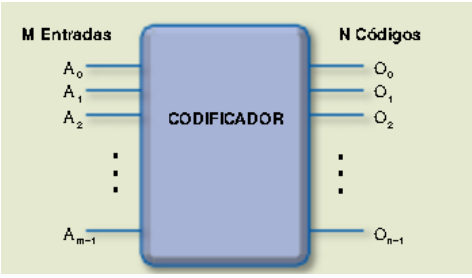
Tabela 01: Tabela verdade Decimal – BCD

DÍGITO DECIMAL	CÓDIGO BCD A_3	A_2	A_1	A_0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

O mesmo ocorre com outros processos de conversão, dedica-se às entradas o objeto de transformação e através de uma relação bem definida consegue-se estabelecer uma saída conveniente. É importante salientar, entretanto, que no codificador apenas uma das entradas

recebe nível lógico alto por vez.

Figura 02: Codificador genérico



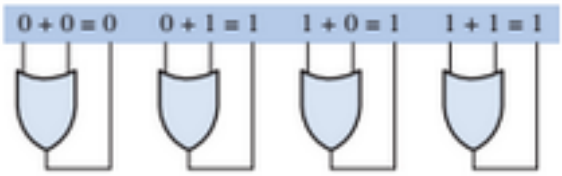
1.1.1 Simplificação de funções booleanas

Em posse das relações de conversão é possível gerar uma tabela verdade e dela extrair uma expressão lógica, entretanto, nem sempre essa expressão encontra-se suficientemente simplificada, e essa situação pode ocasionar gastos financeiros ou energéticos. Essa preocupação surge da constante evolução tecnológica de pequenos dispositivos como smartphones e tablets que aumentaram a sua capacidade, mas enfrentam problemas para comportar uma bateria eficiente. Por esse motivo procura-se simplificar esses circuitos o máximo possível e a álgebra booleana ou mapas de karnaugh são as ferramentas utilizadas com esse propósito. [5]

1.1.1.1 Álgebra booleana

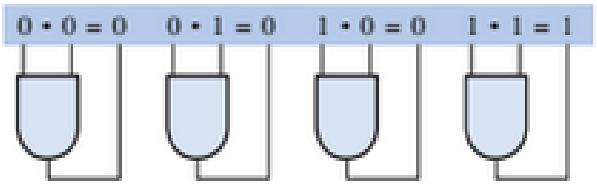
A álgebra booleana representa nada a mais nada a menos que a matemática dos sistemas digitais, obedecendo algumas regras básicas de equivalência que quando aplicadas podem simplificar uma função booleana. Cada operação tem relação com as portas lógicas básicas NOT, AND e OR, a soma, por exemplo, possui equivalência à porta OR e a multiplicação à AND. Dessa maneira, é a partir da lógica por trás das portas lógicas que a álgebra booleana se fundamenta (figura 03). [3]

Figura 03: Soma de binários em termos de portas OR



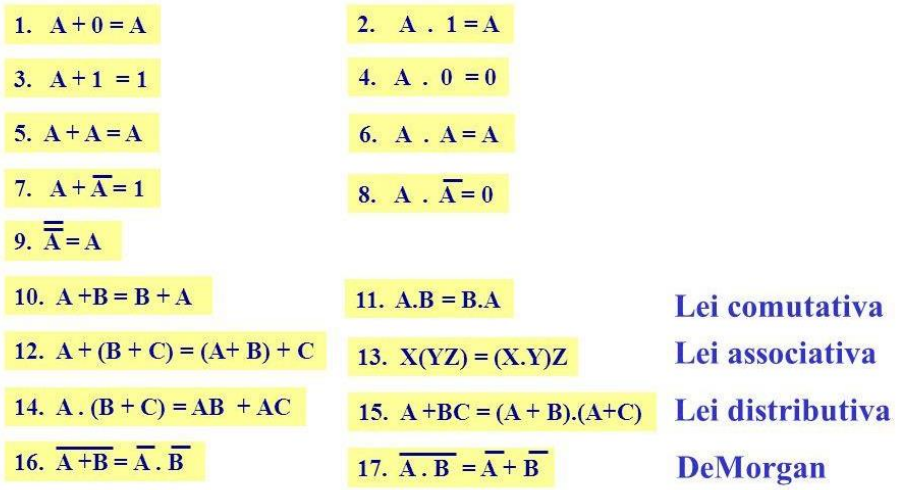
Como dito anteriormente, a multiplicação booleana se dá através da porta AND, esse conceito está expresso na figura a seguir.

Figura 04: Multiplicação em termos da porta AND



A partir desses conceitos iniciais é possível estabelecer leis ou propriedades booleanas que permitam a manipulação de expressões booleanas. Essas relações contemplam a soma, a multiplicação e o uso de identidades, expressando relações semelhantes ao que é trabalhado com o sistema decimal, equivalentes em colocar em evidência, por exemplo, com as leis de Morgan. Esse conjunto de possibilidades pode ser observado na figura a seguir (figura 05).

Figura 05: Leis e Regras da álgebra booleana



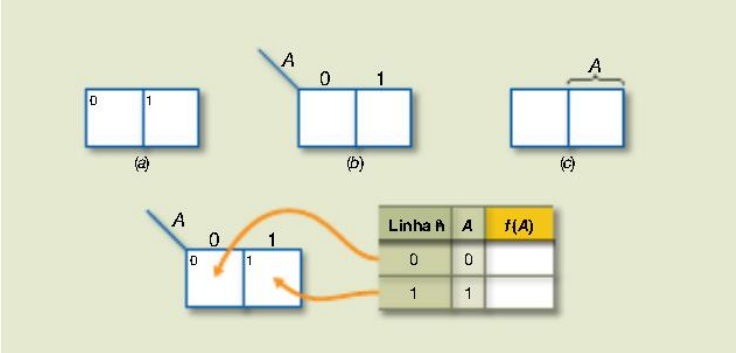
Fonte: http://slideplayer.com.br/slide/338032/

Em posse dessas propriedades é possível reduzir qualquer circuito, entretanto, a convergência para o circuito mais simplificado é muitas vezes lenta e depende muito do raciocínio do projetista, não sendo possível garantir que o circuito final obtido seja o mais simplificado possível. Por esse motivo, foram desenvolvidos outros métodos mais eficientes, como por exemplo, os mapas de karnaugh, que serão apresentados a seguir.

1.1.1.2 Mapas de Karnaugh

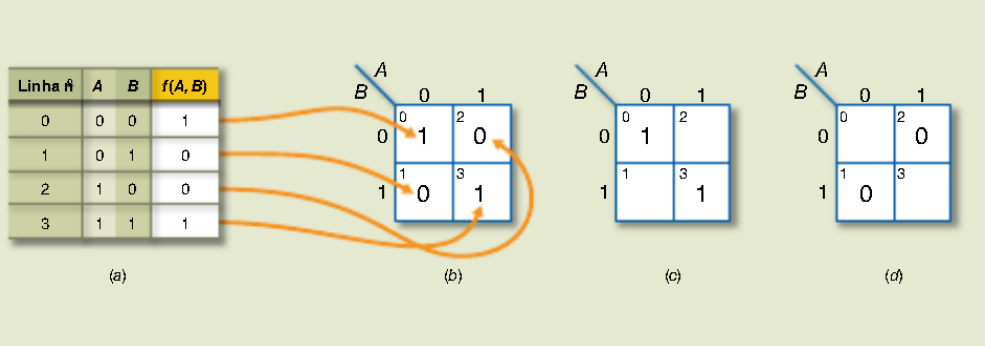
O mapa de Karnaugh, apesar de não possuir um nome tão simples, possui uma lógica certeira e ordenada para simplificação de uma expressão lógica. Sua principal vantagem é garantir que o circuito obtido corresponda ao menor ou mais simplificado circuito lógico. Além disso, é construído diretamente da tabela verdade, não sendo necessárias operações anteriores.[2]
A sua aplicação abrange desde circuitos mais simples, de duas variáveis, a circuitos com 6 variáveis, apesar de compensar, quando acima de 4, o uso de um software. A representação dessa técnica se dá por meio de uma tabela que é dividida em células. Essas células são de proporção 2n, em que n é o número de variáveis de entrada do circuito em questão (figura 06).

Figura 06: Mapa de Karnaugh de 1 variável



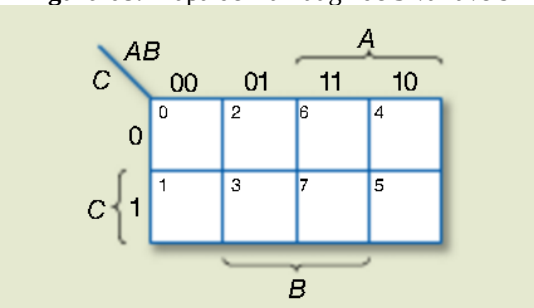
A disposição dos números corresponde aos valores das linhas e colunas (variáveis envolvidas), que representam um valor de saída na tabela verdade. Esse valor de saída é que preenche a tabela do mapa de Karnaugh.

Figura 07: Mapa de Karnaugh de 2 variáveis



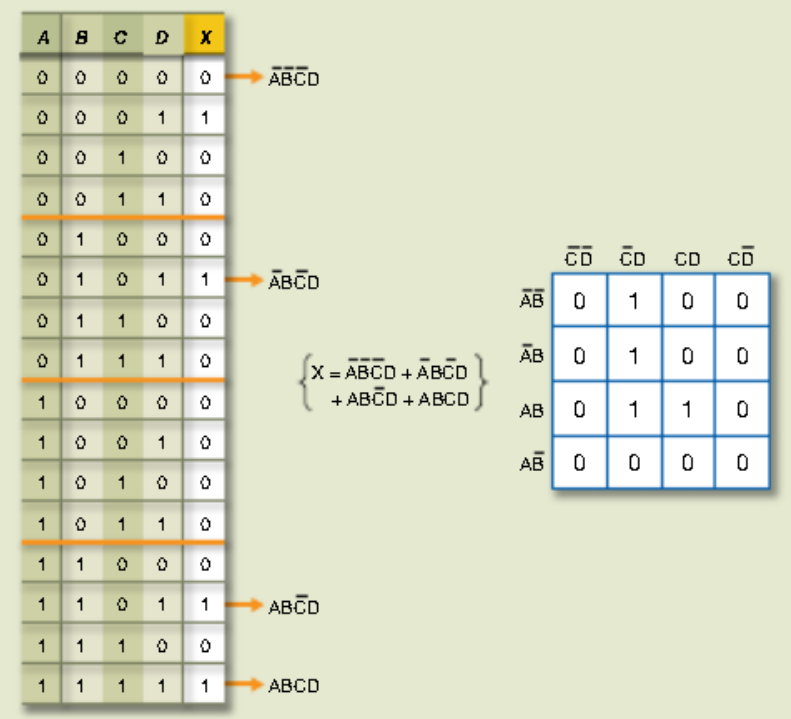
Para duas variáveis fica mais visível o que acontece durante o preenchimento, conforme a figura 07. Quando aumentam o número de variáveis e consequentemente o número de linhas da tabela, a lógica permanece a mesma (figura 8).

Figura 08: Mapa de Karnaugh de 3 variáveis



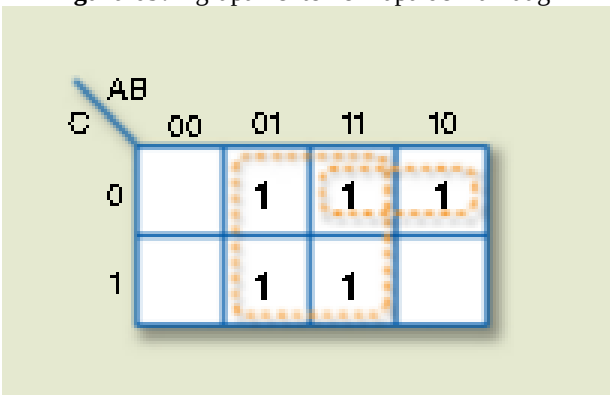
Apesar do modo de preencher o mapa ser o mesmo, o que pode ser visto na figura 08, e o modo de resolver ser similar, cada ordem de grandeza do mapa possui as suas particularidades, como será dissertado a seguir.

Figura 08: Mapa de Karnaugh de 4 variáveis



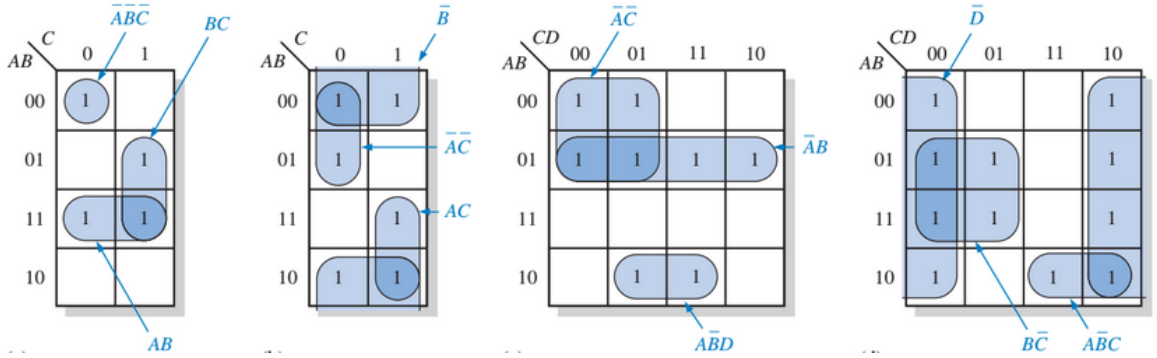
Para obter o circuito mais simplificado é necessário ter em mente que a tarefa principal é unir os números ‘1’ em grupos com múltiplos de 2, ou seja, 1,2,4,8,16, ..., números ‘1’. Nesse passo, é preciso garantir que esteja formando o maior grupo possível, mesmo que seja necessário selecionar uma mesma posição duas vezes. Entrará na expressão final, em relação a cada grupo, a entrada que não mudou de nível lógico, sendo que esse agrupamento se dará através da porta AND, e a junção com os demais grupos será através de portas OR.

Figura 09: Agrupamento no mapa de Karnaugh



Observa-se através da figura 09 que foram realizados agrupamentos de acordo à regra, utilizando-se um número de membros múltiplos de dois. Para obtenção dessa expressão lógica, observa-se cada conjunto formado, tal que S = B + A . not(C). Para um maior número de variáveis, como citado anteriormente, ocorre o mesmo processo, o que diferencia são as possibilidade de combinação, isso por que com 2 variáveis o mapa já se comporta como um cilindro, com 3 é possível unir também os cantos e assim sucessivamente.

Figura 10: Representação das possibilidades de agrupamento

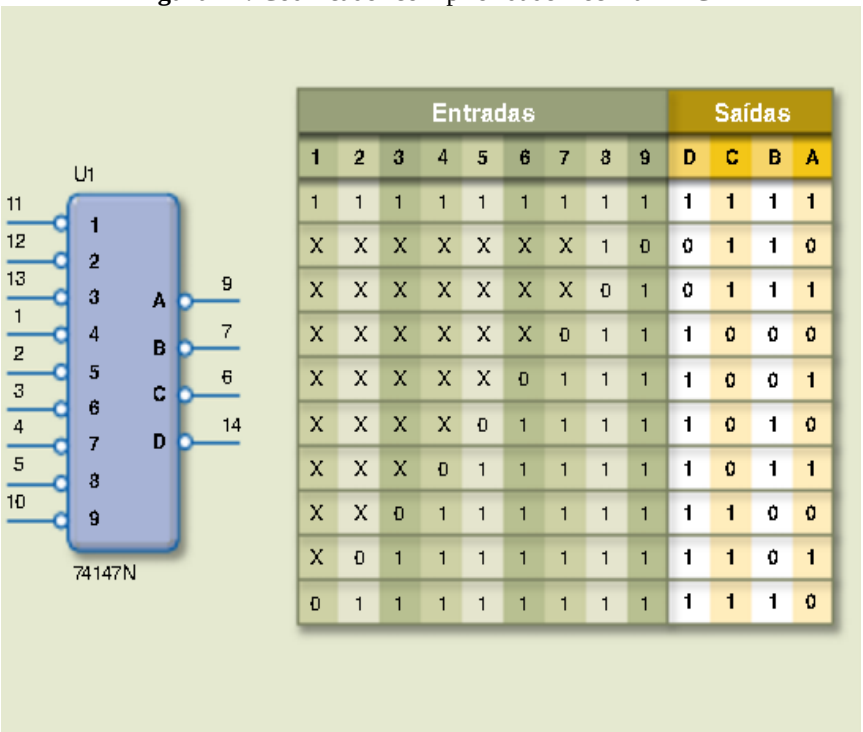


O domínio de todas essas técnicas facilita a implementação dos circuitos codificadores, uma vez que em posse de uma tabela da verdade que caracterize a conversão é possível montar uma expressão lógica e simplifica-la.

1.1.2 Circuitos codificadores com prioridades

O conceito de circuitos codificadores surge do conflito previsto caso duas entradas fossem ativadas ao mesmo tempo, culminando no aparecimento de uma saída diferente da esperada. Para resolver esse impasse seria necessário estabelecer uma prioridade, esta garantiria que independentemente do número de portas ativas, apenas uma entrada seja codificada. Esta entrada escolhida seria a de maior precedência. Um exemplo clássico diz respeito à um codificador com prioridade decimal – BCD.

Figura 11: Codificador com prioridade Decimal - BCD

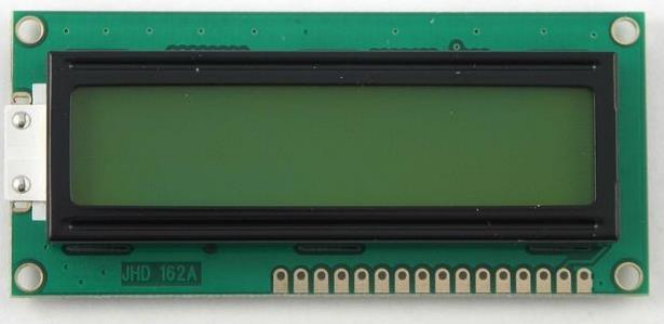


Entradas									Saídas			
1	2	3	4	5	6	7	8	9	D	C	B	A
1	1	1	1	1	1	1	1	1	1	1	1	1
X	X	X	X	X	X	X	1	0	0	1	1	0
X	X	X	X	X	X	X	0	1	0	1	1	1
X	X	X	X	X	X	0	1	1	1	0	0	0
X	X	X	X	X	0	1	1	1	1	0	0	1
X	X	X	X	0	1	1	1	1	1	0	1	0
X	X	X	0	1	1	1	1	1	1	0	1	1
X	X	0	1	1	1	1	1	1	1	1	0	0
X	0	1	1	1	1	1	1	1	1	1	0	1
0	1	1	1	1	1	1	1	1	1	1	1	0

1.1.3 Displays

A palavra *Display* significa apresentação no inglês[6] , e se configura, não obstante, como um dispositivo eletrônico utilizado para apresentar informações ao usuário. Esse dispositivo possui diversas aplicações e está presente nos eletrônicos de uso diário. É importante considerar que por trás da informação passada por esse dispositivo, estão presentes os circuitos, sejam para controlar os pixels ou acender leds, por exemplo.

Figura 12: Display LCD



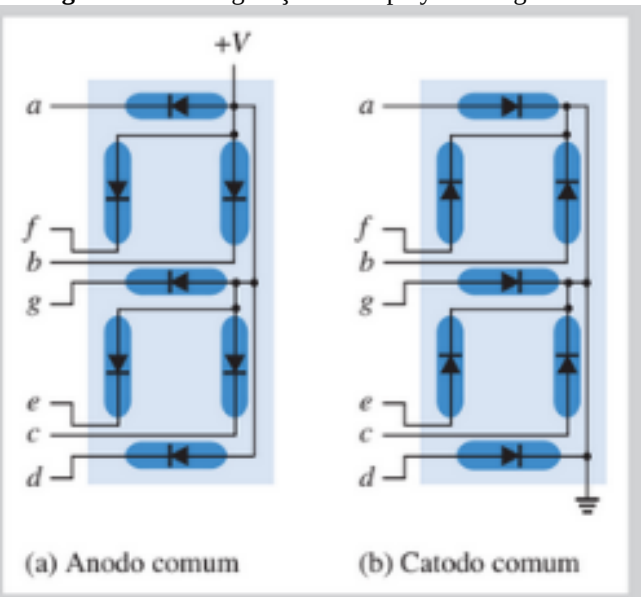
É mais comum no dia-a-dia a interação com os displays de LED, seja em computadores, laptops, smartphones, tablets ou afins. Esse tipo de display, em conjunto com o demonstrado na figura 13, configura – se como display de sete segmentos (figura 13), objeto de estudo deste experimento.

Figura 13: Display comum de 7 segmentos

1.1.3.1 Display LED de 7 segmentos

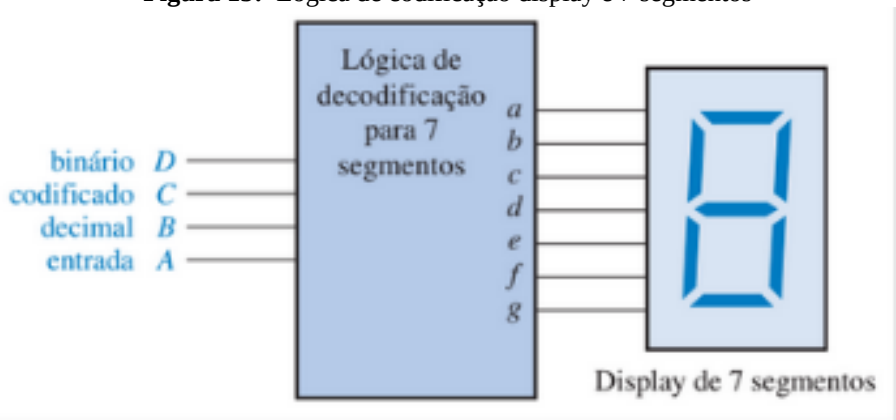
O display de 7 segmentos consiste em um display com 7 LEDS posicionados de modo a formar o número 8. Ele é utilizado para representação de dados numéricos, estando presente em relógios e calculadoras, por exemplo. A formação dos números depende do modo como forem polarizados os LEDS.

Figura 14: Configuração do display de 7 segmentos



A ativação dos LEDS estará sujeita ao modo de funcionamento do circuito, caso seja ânodo comum, os LEDS serão acesos quando a entrada for nível lógico baixo, caso seja cátodo comum ligará quando a entrada for nível lógico alto. Além disso, considerando que os displays utilizam controladores, eles dependem de circuitos codificadores para converter os dígitos hexadecimais para os 7 segmentos.

Figura 15: Lógica de codificação display e 7 segmentos



Essa lógica relaciona-se ao fato de que nenhum segmento é de fato usado em todos os dez dígitos decimais, por esse motivo, é necessário que cada segmento possua um circuito próprio de decodificação. E a expressão lógica de cada circuito pode ser encontrada através da tabela verdade que relaciona os códigos BCD ao número Hexadecimal.

1.2 Objetivo

Mostrar as vantagens do uso de técnicas de simplificação de funções boleanas; apresentar o funcionamento e familiarizar o aluno com projetos codificadores.

1.3 Justificativa

A implementação de um circuito em um *software* pode ser realizada de diferentes formas, contudo, vale lembrar que nem todas as formas garantem a maior eficiência energética e menor gasto financeiro, para isso faz-se importante o uso das simplificações boleanas que podem ser obtidas por meio de manipulações algébricas ou por meio do mapa de Karnaugh, essas simplificações garantem sistemas mais simples, eficientes e baratos

No atual momento do curso, é trivial saber que o computador não trabalha no sistema decimal com o qual estamos acostumados e sim no binário, porém, para que aconteça a interação entre usuário e máquina é crucial o uso de algum "tradutor de código" mais conhecido como conversor, um bastante conhecido é o **codificador** o qual é capaz de transformar informações obtidas em decimal, por exemplo, para binário.

2 PARTE EXPERIMENTAL

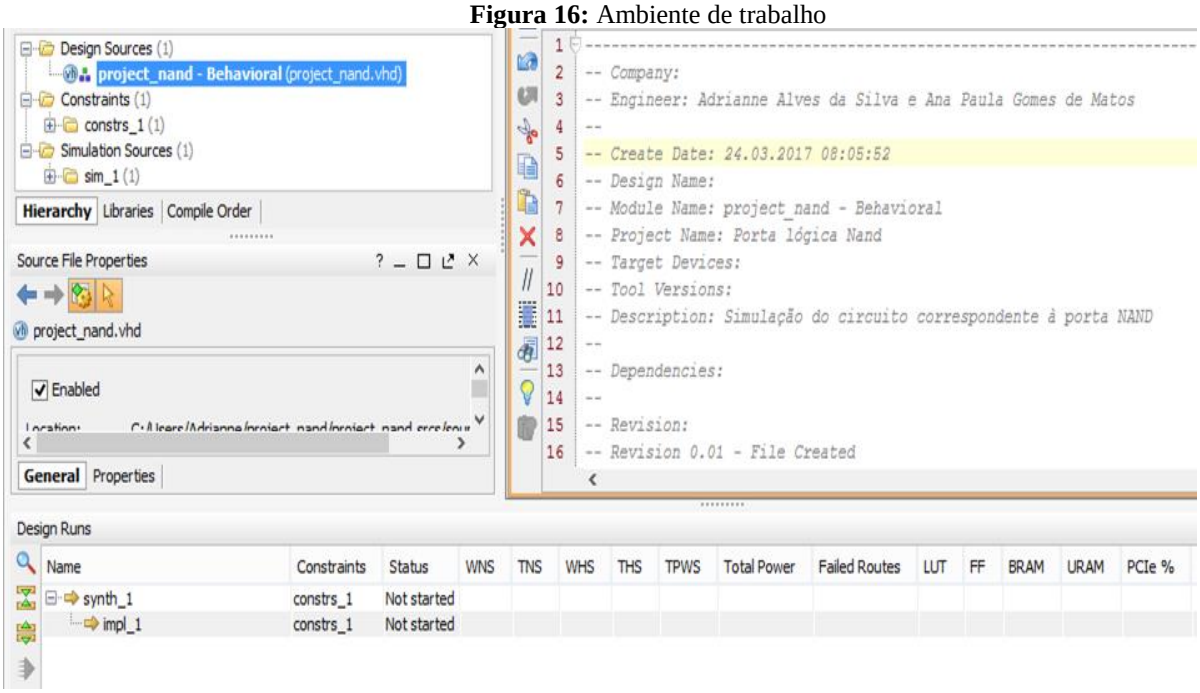
O software utilizado para codificar e simular os experimentos foi o VIVADO , versão 2013.4, instalado nos computadores da universidade. O experimento foi realizado em sala.

2.1 Preparação do ambiente de simulação

O passo inicial para a realização do experimento foi criar um novo projeto no *software* Vivado. A configuração do mesmo foi realizada adicionando a placa Basys 3 ao projeto, por meio da aba "*add or create constraints*". Após isso foi necessário apenas criar o arquivo relativo ao *design* , adicionando o arquivo com o tipo vhdl, para que o código pudesse ser implementado nesta linguagem. O módulo foi então definido de acordo com o número de portas de entrada e saída do circuito referente à cada atividade solicitada, nomeadas as entradas, em sua maioria como A, B, C,..., e Saída como SA, SB, SC e assim sucessivamente.

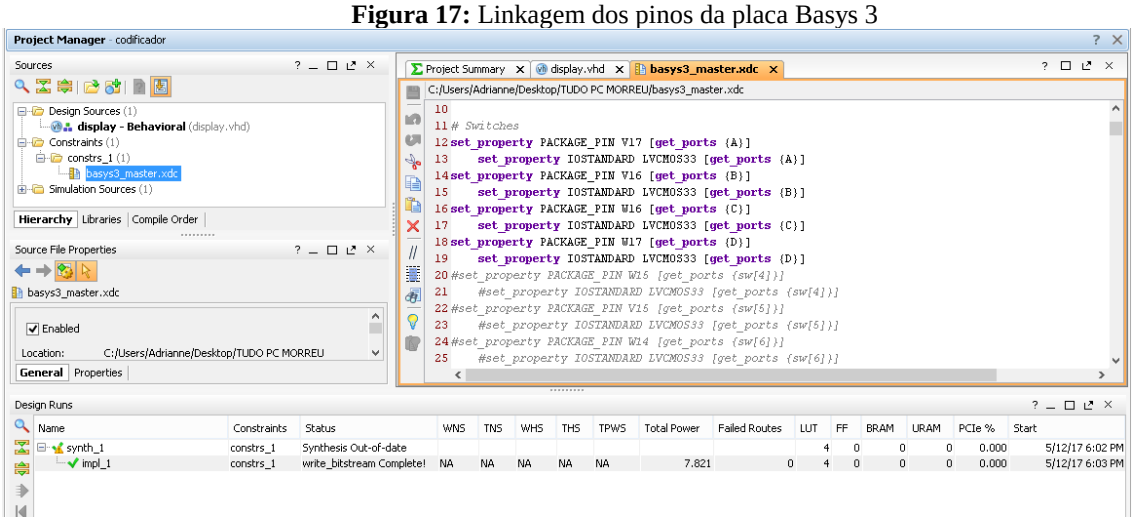
É necessário então ativar na aba de *project settings* , *bitstream* o arquivo bin_file que possibilita escrita binária sem uso do cabeçalho. Assim, obteve-se o ambiente demonstrado na figura a seguir.

Figura 16: Ambiente de trabalho



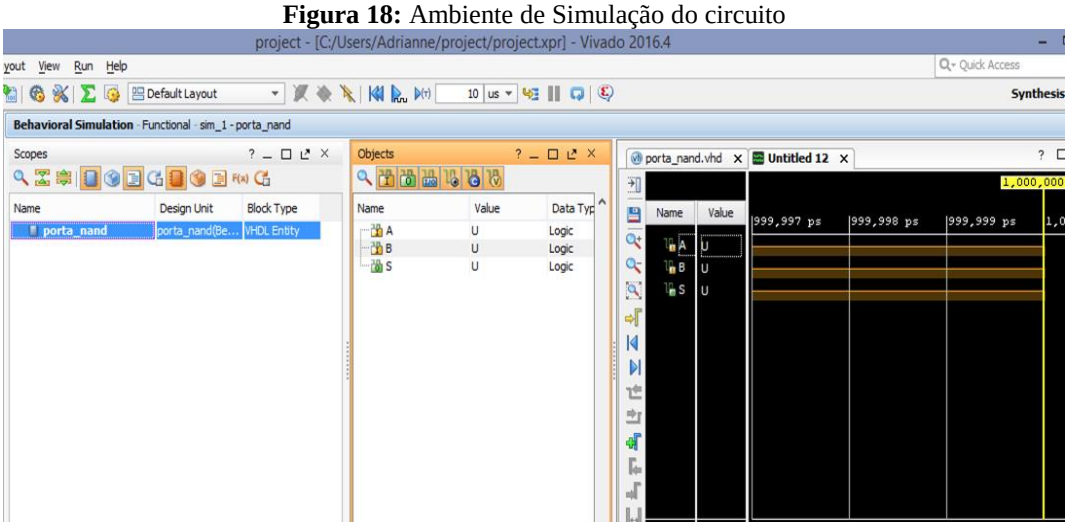
Sendo assim, foi possível abrir o arquivo vhdl definido e implementá-lo. Para cada atividade, forneceu-se ao arquivo o código e as expressões booleanas adequadas. Por sequência do passo-a-passo fornecido, foram linkados os pinos de entrada e saída da placa Basys com os dispositivos de entrada e saída implementadas. No arquivo basys3.xdc (**figura 16**), isso foi feito por meio da retirada do comentário das linhas referentes ao número de entradas e saídas presentes em cada atividade, além da adequada atribuição de cada uma das portas, segundo a nomenclatura adotada, essa atividade seria essencial para a verificação diretamente numa FPGA.

Figura 17: Linkagem dos pinos da placa Basys 3



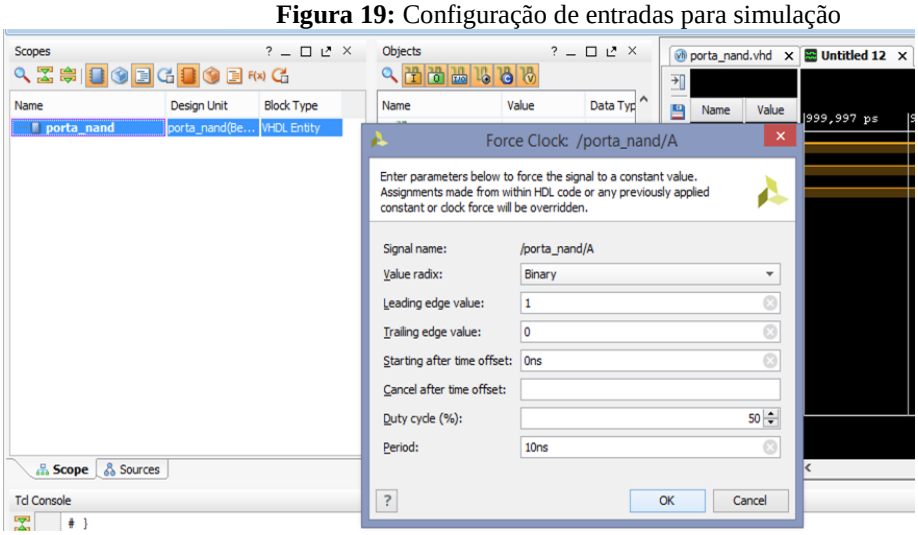
Após todos esses passos foi necessário sintetizar o circuito por meio do *Run Synthesis* , gerando um esquemático facilmente visualizado no *RTL Analysis* . Nesse ponto, é que tornou-se possível realizar a simulação do experimento, bastando clicar em *run simulation* que gera um ambiente de simulação (**figura 18**).

Figura 18: Ambiente de Simulação do circuito



Nesse passo, é possível configurar os valores referentes à cada entrada, assim como o tempo de mudança de sinal (0 ou 1) para a geração de ondas . Isso é feito clicando sobre cada entrada com o botão direito, na opção de *Force Clock*, conforme a figura a seguir.

Figura 19: Configuração de entradas para simulação



A primeira configuração realizada é a modificação do argumento *value* para o tipo binário informando entradas diferenciadas para os argumento *Leading edge value* e *Trailing edge value*, que seja 0 ou 1 . Define-se um período para as ondas, diferentes para cada entrada. É legal se o valor de uma for o dobro da outra, por exemplo. E então, basta executar a simulação.

2.2 Experimento

O experimento consistiu em apenas uma atividade envolvendo o display de 7 segmentos, que leva em consideração quatro entradas em BCD e retorna a resposta para cada LED individualmente, contabilizando um total de 7 circuitos. Inicialmente, em posse da tabela verdade que representa essa conversão chegou-se aos 7 circuitos por meio de mapas de Karnaugh, e só então foi passado para VHDL e por fim testado na placa *Basys3*.

2.2.1 Tabela Verdade

A tabela verdade desta conversão nada mais é que a relação de conversão BCD – Hexadecimal, tal que obedece a seguinte tabela verdade :

Figura 20: Tabela verdade display de 7 segmentos

Entrada	HEX	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 0 1 0	A	1 1 1 0 1 1 1
1 0 1 1	B	0 0 1 1 1 1 1
1 1 0 0	C	1 0 0 1 1 1 0
1 1 0 1	D	0 1 1 1 1 0 1
1 1 1 0	E	1 0 0 1 1 1 1
1 1 1 1	F	1 0 0 0 1 1 1

Através de 7 mapas de karnaugh com 4 variáveis de entrada foi possível obter as expressões lógicas utilizadas para a implementação do código em VHDL que será representado a seguir. Deve-se relatar que esquecemos que a placa basys3 é anôdo e por tanto, realizamos as expressões considerando cátodo, entretanto, identificamos a tempo de negar as expressões e garantir que estivessem corretas.

2.2.2 Código

Foi necessário apenas um código para a implementação desses circuitos, utilizando-se apenas e simplesmente das funções booleanas extraídas, conforme está apresentado a seguir:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity display is
   Port ( A : in STD_LOGIC;
          B : in STD_LOGIC;
          C : in STD_LOGIC;
          D : in STD_LOGIC;
          SA : out STD_LOGIC;
          SB : out STD_LOGIC;
          SC : out STD_LOGIC;
          SD : out STD_LOGIC;
          SE : out STD_LOGIC;
          SF : out STD_LOGIC;
          SG : out STD_LOGIC
          );
end display;

architecture Behavioral of display is

begin

SA <= NOT((NOT(B) AND NOT(D)) OR (NOT(A) AND C) OR (B AND C) OR (A AND
NOT(D)) OR (NOT(A) AND B AND D) OR (A AND NOT(B)AND NOT(C)));

SB <= NOT((NOT(A) AND NOT(B)) OR (NOT(B) AND NOT(D)) OR (NOT(A) AND NOT(C)
AND NOT(D)) OR (NOT(A) AND C AND D) OR (A AND NOT(C) AND D));

SC <= NOT( (NOT(A) AND NOT(C)) OR (NOT(A) AND D) OR (NOT(C) AND D) OR
(NOT(A) AND B) OR (A AND NOT(B)));

SD <= NOT((A AND NOT(C)) OR (NOT(A) AND NOT(B) AND NOT(D)) OR (NOT(B) AND
C AND D) OR (B AND NOT(C) AND D) OR (B AND C AND NOT(D)));

SE <= NOT( (NOT(B) AND NOT(D)) OR (C AND NOT(D)) OR (A AND C) OR (A AND B));

SF <= NOT((NOT(C) AND NOT(D)) OR (B AND NOT(D)) OR (A AND NOT(B)) OR (A
```

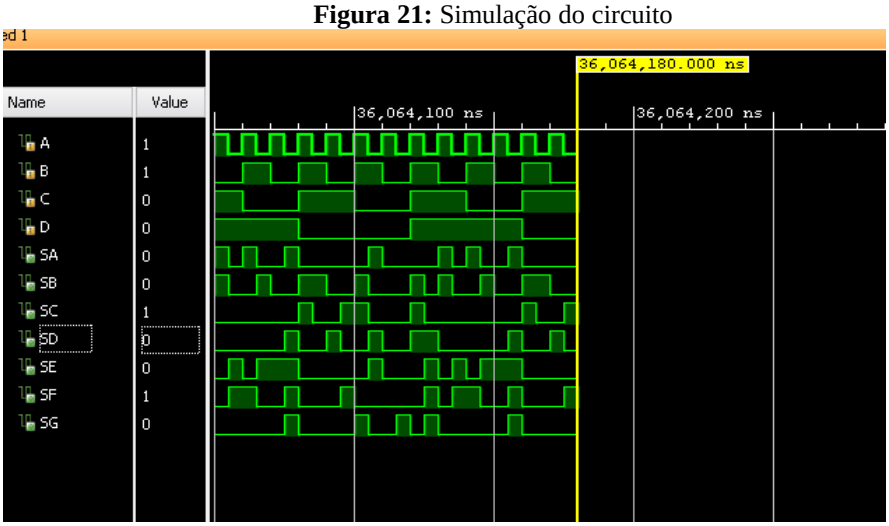
AND C) OR (NOT(A) AND B AND NOT(C)));

SG <= NOT((NOT(B) AND C) OR (C AND NOT(D)) OR (A AND NOT(B)) OR (A AND D) OR (NOT(A) AND B AND NOT(C)));

end Behavioral;

Optamos por não utilizar vetores apenas para tornar o código mais limpo e evitar que confundíssemos os LEDS nomeados, entretanto, bastava declarar uma variável de entrada como vetor de 4 posições e uma variável de saída como vetor de tamanho 7. Da maneira como fizemos pudemos observar, tanto pela simulação (**figura 21**) quando pelo teste na placa que todas as conversões foram feitas corretamente. O esquemático que representa esse circuito encontra-se ao fim deste documento.

Figura 21: Simulação do circuito



3 DISCUSSÃO

A realização do experimento foi feita por meio da implementação da expressão lógica obtida da tabela verdade com a utilização do mapa de Karnouth, vale destacar que a as expressões lógicas resultantes foram jogadas direto no programa, porém, como a placa basys 3 é anôdo os valores na placa foram apresentados com os LED's apagados, para consertar isso, foi necessário negar todas as expressões para que assim os valores fossem expostos de forma correta com os LED's acesos.

4 CONCLUSÕES

Como pode ser observado, o experimento foi concluído de forma eficiente, sem imprevistos e com resultado satisfatório, por se tratar de um experimento simples o mesmo foi finalizado ainda em horário de aula.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] FONSECA FILHO, Cléuzio. **História da computação: O Caminho do Pensamento e da Tecnologia**. EDIPUCRS, 2007. p. 56-58.

[2] AMARAL, Valder Moreira. **Eletrônica Digital**. São Paulo: Fundação Padre Anchieta, 2011. p. 31-33. 4 v.

[3] FLOYD, Thomas. Portas lógicas. In _____. **Sistemas digitais: fundamentos e aplicações**. Bookman Editora, 2009. Cap. 3, p. 134-135.

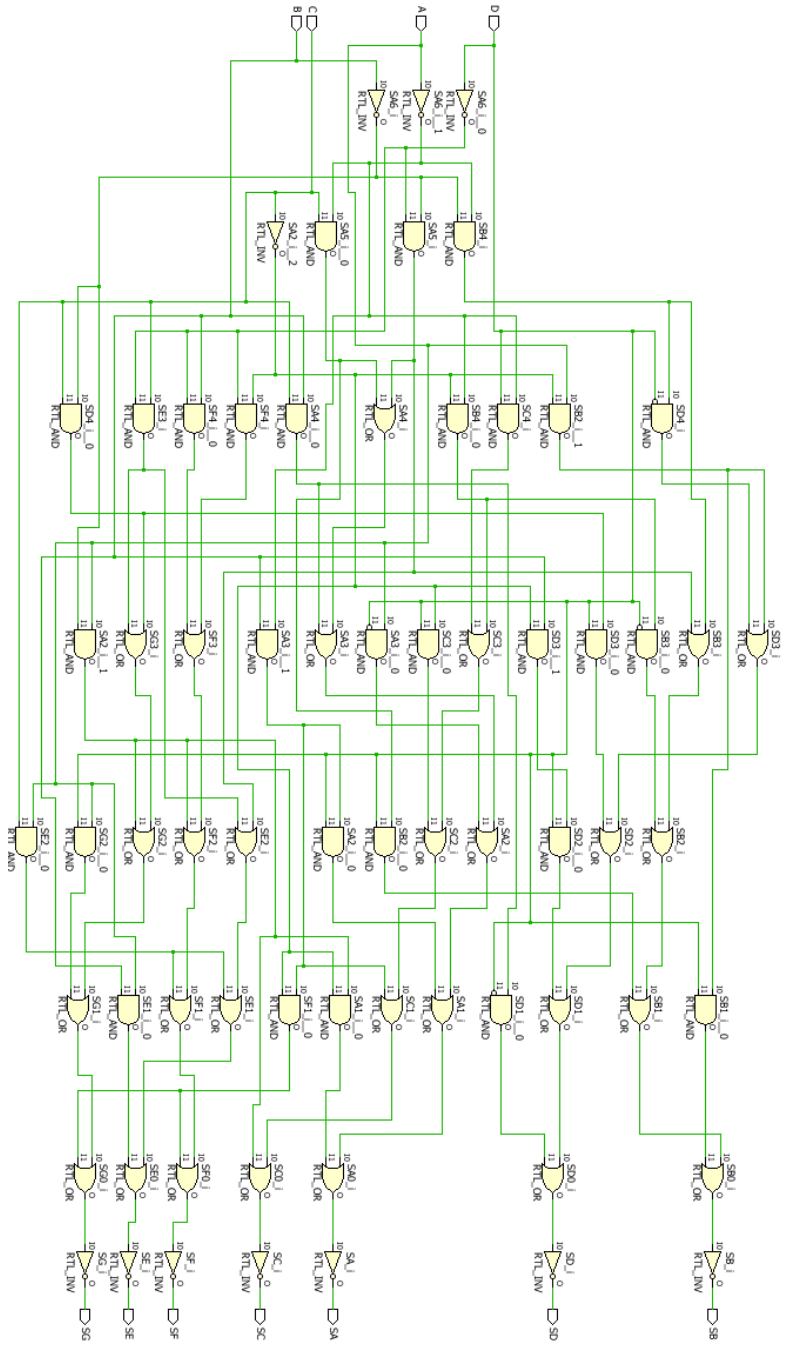
[4] _____. Álgebra booleana e simplificação lógica. In _____. **Sistemas digitais: fundamentos e aplicações**. Bookman Editora, 2009. Cap. 4, p. 244-245.

[5] URRIZA, José M. et al. Economia de energia em dispositivos móveis. In: **VI Workshop de Comunicação sem Fio e Computação Móvel**. 2004.

[6] DISPLAY. **Dicionário online inglês - português**, 18 maio. 2017. Disponível em <http://dictionary.cambridge.org/pt/dicionario/ingles-portugues/display>. Acesso em 18 maio. 2017.

DIAGRAMA ESQUEMÁTICO

Figura 21: Conversor BCD – Hexadecimal e conexão com o display de 7 segmentos



UnB - FGA	**Prancha 1**	**Versão**	**Data**	**Elaborador por:**	Adrianne Alves da Silva
LAB PED 1	Codificadores	1.0	12/04/2017	**Revisado por:**	Ana Paula Gomes de Matos