Laboratório de Eletrônica Digital

Prof.: José Valentim dos Santos Filho

Pré-Relatório/ Relatório #1

Lab 1 - PORTAS LÓGICAS

O objetivo desta aula é conhecer os circuitos integrados digitais que implementam as chamadas portas lógicas – funções lógicas elementares. Veremos como implementar funções lógicas mais complicadas a partir dessas portas, com alguns exemplos práticos de aplicação.

Atenção:
Esta apostila contém alguns exercícios que constituem um pré-relatório, o qual deve ser feito INDIVIDUALMENTE e entregue ao professor no início de cada aula.

PARTE I TEORIA

1.1 Portas Lógicas Elementares

Portas lógicas (*gates*) são circuitos eletrônicos que implementam funções lógicas elementares por meio de sinais elétricos. Os valores lógicos são representados por tensões elétricas padronizadas. Por exemplo, o valor lógico *falso* (ou *zero*, ou *desligado*, ou ... como você preferir) pode ser associado a uma tensão igual a 0 V, enquanto que o valor lógico *verdadeiro* (ou *um*, ou *ligado*, ou... etc) pode ser associado a 5 V.

Em eletrônica digital, costuma-se designar esses níveis de tensão por

- *L* (*LOW*) - nível lógico baixo,
- *H* (*HIGH*) - nível lógico alto.

Dentro do padrão conhecido como *TTL* (*Transistor-Transistor Logic*), por exemplo, um nível *L* válido corresponde a uma tensão elétrica entre 0 V e 0,8 V, enquanto o nível *H* vai de 2,4 V a 5,0 V. Estudaremos esse padrão com mais detalhes na próxima experiência. Para esta aula, basta saber que os níveis lógicos não são representados por tensões exatas, mas sim por duas faixas de tensão que não se sobrepõem.

Em prol de uma notação mais limpa, neste texto vamos representar o nível de tensão *L* pelo símbolo lógico '0' (zero). Ao nível de tensão *H*, associamos o símbolo '1' (um).

A seguir, vamos ver as portas que implementam as três funções lógicas elementares: NOT, AND e OR.

♦ Porta NOT (inversora, complemento ou negação lógica)

A porta NOT possui uma entrada (*Y*) e uma saída (*Z*). A Figura 1.1 mostra o símbolo da porta NOT, também conhecida como porta inversora. O valor presente na saída é complementar ao valor imposto à entrada, ou seja

- se $Y = 0$ então $Z = 1$
- se $Y = 1$ então $Z = 0$

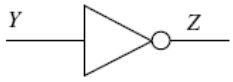


Figura 1.1 Símbolo da porta lógica NOT (inversora)

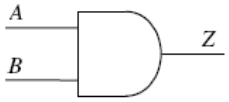
Matematicamente, a função NOT costuma ser indicada de diversas formas diferentes. As mais comuns são

$$\text{NOT}(Y) = Y' = \overline{Y} = /Y = \_Y = \sim Y = !Y \,. \qquad (1.1)$$

A primeira e a segunda são as mais utilizadas em textos, como esta apostila. As notações "*/Y*" e "*_Y*" aparecem mais em desenhos de circuitos digitais, enquanto que as duas últimas são utilizadas por linguagens de programação.

♦ Porta AND (E lógico)

A porta AND possui duas entradas (*A* e *B*) e uma saída (*Z*). A Figura 1.2 mostra o símbolo da porta AND e a sua *Tabela da Verdade* (valores de saída tabelados para cada possível combinação de entradas). A notação matemática para se representar uma operação AND é a de um produto.

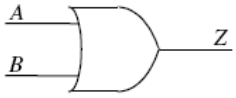


A	*B*	*Z = A.B*
0	0	0
0	1	0
1	0	0
1	1	1

Figura 1.2 Símbolo da porta lógica AND e sua Tabela da Verdade

♦ Porta OR (OU lógico)

A porta OR também possui duas entradas e uma saída. A Figura 1.3 mostra o símbolo da porta OR e a tabela da verdade desta. Matematicamente, denota-se a operação com o símbolo '+'.



A	*B*	*Z = A+B*
0	0	0
0	1	1
1	0	1
1	1	1

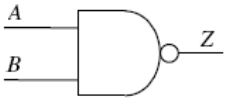
Figura 1.3 Símbolo da Porta OR e sua Tabela da Verdade

1.2 Portas NAND e NOR

Combinando as três funções lógicas elementares (NOT, AND e OR), podemos construir outras funções. É o caso, por exemplo, das funções NAND e NOR. Duas combinações bastante simples, mas de extrema importância.

♦ NAND

Uma porta NAND é uma porta AND seguida por um inversor. Seu símbolo lógico e tabela da verdade estão na Figura 1.4. Note que o símbolo do inversor foi simplificado, restando apenas a bolinha.

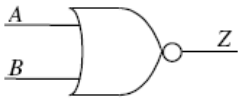


A	*B*	*Z = (A.B)'*
0	0	1
0	1	1
1	0	1
1	1	0

Figura 1.4 Símbolo lógico e tabela da verdade da porta NAND.

♦ NOR

Continuando, uma porta NOR é constituída por uma porta OR seguida por um inversor, conforme mostra a Figura 1.5.



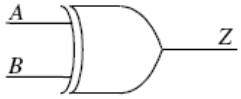
A	*B*	*Z = (A+B)'*
0	0	1
0	1	0
1	0	0
1	1	0

Figura 1.5 Símbolo e tabela da verdade da porta NOR.

Verdade seja dita: na realidade, é a porta AND que é constituída por uma porta NAND seguida de um inversor! Isso porque, por motivos tecnológicos, é mais fácil construir com transistores uma porta NAND do que uma AND. Por isso, estas últimas empregam mais transistores em sua construção, e são mais lentas – apresentam maior *atraso de propagação*. O mesmo pode ser dito a respeito das portas OR e NOR.

1.3 Porta XOR (OU-EXCLUSIVO)

A função OU-EXCLUSIVO (conhecida como XOR) é uma função lógica que tem grandes aplicações práticas. Como seu nome indica, a função XOR difere da função OR por excluir a condição em que ambas as entradas estão em um. A Figura 1.6 mostra o símbolo da função XOR e a sua tabela da verdade. Denotamos esta operação com o símbolo '⊕'.



A	*B*	*Z = A⊕B*
0	0	0
0	1	1
1	0	1
1	1	0

Figura 1.6 Símbolo da Porta XOR e sua Tabela da Verdade

Como qualquer função lógica não elementar, a função XOR pode ser construída a partir das funções elementares. Por hora, acredite que

$$A \oplus B = \overline{A}.B + A.\overline{B}. \qquad (1.2)$$

Na seção 1.5, mostraremos que esta igualdade é verdadeira.

1.4 Circuito Integrado (CI)

A maioria dos circuitos digitais, desde as portas lógicas vistas nas seções anteriores até os poderosos microprocessadores de última geração, são fabricados em lâminas silício – um material semi-condutor. Em um único centímetro quadrado de silício são integrados milhões de transistores e outros dispositivos eletrônicos em escala microscópica, constituindo os chamados circuitos integrados (CI), também conhecidos como *chips*. Por serem muito frágeis, os *chips* são encapsulados em pastilhas de epoxi ou cerâmica, contendo pinos metálicos ligados aos pontos de entrada e saída do circuito.

No laboratório, utilizaremos alguns circuitos integrados básicos. Em anexo, são fornecidas partes selecionadas das folhas de especificações (os famosos *data sheets*) desses componentes.

É o caso, por exemplo, do **74LS00**, uma pastilha de 14 pinos que contém em seu interior quatro portas NAND, cada uma com duas entradas. Dê uma olhada no seu *data sheet*, para ter uma idéia de como as portas estão dispostas na realidade dentro do CI. Por exemplo, repare a primeira inversora tem suas entradas (denominadas A1 e B1) ligadas internamente aos pinos 1 e 2, enquanto que a saída está ligada ao pino 3.

Usaremos componentes encapsulados em pastilha padrão *DIP* (*Dual In-line Package*), cuja ilustração e dimensões físicas estão na segunda página do *data sheet* (numerada como 5). Repare no recorte e na marca à esquerda da primeira vista superior: elas indicam a posição do pino 1.

Note ainda que o CI é um circuito eletrônico ativo, que precisa ser alimentado por uma fonte de tensão externa para poder funcionar. Afinal, de que outra forma uma porta lógica poderia fornecer em sua saída uma tensão não nula mesmo tendo suas entradas mantidas em 0 V? Portanto, o pino 14 (indicado por V_{CC}) desse CI deve ser ligado a uma fonte de +5 V, enquanto que o pino 7 (GND) deve ser conectado ao terra da fonte.

Lista Completa de CIs que serão usados nesta experiência :

74LS00 – quatro NANDs de duas entradas

74LS04 – seis portas inversoras

74LS08 – quatro ANDs de duas entradas

74LS32 – quatro ORs de duas entradas

74LS86 – quatro XORs de duas entradas

OBS: Procurem nos datasheets detalhes sobre os CI's (pinagem, alimentação, etc)

1.6 Exemplos práticos

Vamos dar uma olhada em dois exemplos práticos de utilização da função XOR.

♦ Inversor Controlado

Uma porta XOR também funciona como um inversor controlado. Para entender isso, observe novamente a Figura 1.6. Repare que quando fixamos uma das entradas em zero, a saída simplesmente passa a reproduzir o valor da outra entrada. Por outro lado, se fixamos uma das entradas em um, a saída é igual ao inverso da outra entrada.

Assim, podemos representar uma porta XOR como mostra a Figura 1.8. As entradas da porta XOR foram chamadas de *S* (sinal) e *M* (modo). Analisando a tabela da verdade do XOR (Figura 1.6), temos que

- Se $M = 0$ então $Z = S$
- Se $M = 1$ então $Z = S'$

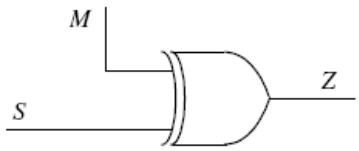


Figura 1.8 Inversor Controlado

♦ Verificador de Paridade

A necessidade de comunicação entre circuitos digitais é bastante comum. A mensagem enviada por um lado está sempre sujeita a erros. Freqüentemente ocorrem ruídos nas linhas de transmissão, interferências ou quebra de fios.

O mecanismo de paridade é o seguinte: o emissor e o receptor trocam mensagens com N bits. A cada conjunto de N bits (uma mensagem), o emissor envia um bit adicional chamado *bit de paridade*. O valor desse bit depende da mensagem trocada.

Existem dois tipos de paridade: paridade par e paridade impar. Na paridade par, o bit de paridade é ajustado para que os N + 1 bits (N de mensagem + 1 bit de paridade) tenham sempre um número par de bits iguais a um. Na paridade ímpar, o bit de paridade é ajustado para que os N + 1 bits tenham um número ímpar de 'uns'. O bit de paridade ímpar, obviamente, é o complemento do bit de paridade par.

Considere, por exemplo, a mensagem **1110** de 4 bits (N = 4). Adotando-se paridade **par**, a mensagem seria transmitida como **11101** – acrescenta-se um bit de paridade igual a 1 para que o número total de 'uns' seja par (quatro bits iguais a 1 em um total de cinco). Analogamente, no caso de paridade ímpar, transmitir-se-ia **11100**.

Se a mensagem enviada foi corretamente recebida, o bit de paridade estará coerente com os N bits recebidos. Se houve erro na recepção de um dos N+1 bits (incluindo o bit de paridade), isso não acontece. Note que os erros de 1 bit são detectados, mas não há como corrigi-los.

A paridade par (que denotaremos por *PP*) é gerada com uma operação XOR entre todos os bits da mensagem, e a paridade ímpar (*PI*) é simplesmente o complemento desta operação.

$$PP = b_{n-1} \oplus b_{n-2} \oplus \ldots b_0 \,, \tag{1.3}$$

$$PI = \overline{PP} = \overline{b_{n-1} \oplus b_{n-2} \oplus \ldots b_0} \,. \tag{1.4}$$

PARTE II PRÉ-RELATÓRIO

Faça os exercícios listados a seguir, individualmente, e entregue suas respostas ao professor no começo da aula de laboratório. Por favor, utilize folhas de papel quadriculado, tamanho A4 (não é necessário ser milimetrado).

1- Faça a tabela verdade de cada uma das portas lógicas, construa o diagrama lógico para cada porta, monte o circuito no protoboard e verifique o seu funcionamento (construindo a tabela verdade do circuito montado).

Para Relatório: Alimente a entrada do inversor com um trem de pulso de 1KHz e verifique a forma de onda da saída no osciloscópio.

2- Construa uma porta XOR a partir de portas elementares NOT, AND e OR.
Construa o diagrama lógico identificando as ligações e a pinagem dos CI's.
Verifique o seu funcionamento (construindo a tabela verdade do circuito montado).

Substituição da porta NOT por NAND

b- Suponha que não temos à nossa disposição o CI 74LS04 (NOT), mas no seu lugar poderemos usar as portas NAND do 74LS00.
Monte o circuito, obtenha a tabela verdade e compare com a tabela verdade do circuito original.

3- Gerador de Paridade (OBS: Neste exemplo, o autor faz referência a uma figura 1.7 – Esqueçam isso!)

Vamos exercitar o conceito de paridade par e ímpar, usando o circuito esquematizado na Figura 1.11. O circuito tem como entradas uma palavra B de três bits ($B_2B_1B_0$) e duas saídas: ZI e ZP.

- A saída ZI vale 0 sempre que a palavra B tiver um número ímpar de bits iguais a 1, e $ZI = 1$ caso contrário. Com isto, o conjunto de quatro bits (três de entrada, mais ZI) sempre terá paridade ímpar. Em outras palavras, ZI é um *gerador de paridade ímpar*. É também conhecido como *detector de paridade par*, uma vez que $ZI = 1$ sinaliza quando a paridade de B é par. (Confuso? Bem, respire fundo e leia de novo, não é tão complicado assim...)
- ZP é o complementar de ZI, ou seja, ZP gera o bit de paridade par do conjunto de três bits, ou, equivalentemente, detecta a condição de paridade ímpar nos três bits de B.

a) Projete e desenhe o diagrama lógico (DL) de um circuito que implemente o gerador de paridades, usando apenas um CI 74LS86 (se você precisar, pode usar também um 74LS00, mas realmente não é necessário...). Novamente, indique o número dos pinos, a identificação dos componentes (U1a, U2, etc) e o código deles (74LSxx). Siga o exemplo da Figura 1.7.
Note que cada porta do 74LS86 possui apenas duas entradas, e precisaremos fazer um XOR entre os três bits da palavra B. Isso não é problema, sabendo que a função XOR é *associativa*! Ou seja,

$$(B_2 \oplus B_1) \oplus B_0 = B_2 \oplus (B_1 \oplus B_0) = B_2 \oplus B_1 \oplus B_0 . \qquad (1.9)$$

b) Faça a tabela da verdade completa do circuito que você projetou, determinando o valor da saída de cada porta de seu circuito para cada combinação possível das entradas B_2, B_1 e B_0. Siga o exemplo da Tabela 1.1

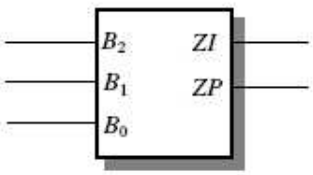


Figura 1.11 Gerador de paridade

4- Projete o circuito pedido na letra b do exercício 3-32 do Tocci.
Monte o circuito, obtenha a tabela verdade e compare com o circuito original.