ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO
Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos

PMR3303 Eletrônica Digital para Mecatrônica 1º semestre de 2019

Nome: _____ Nº USP: _____

Experiência 1 PORTAS LÓGICAS (REV. A)

Objetivo: conhecer as chamadas portas lógicas – circuitos integrados que implementam funções lógicas elementares, e implementar funções lógicas mais complicadas a partir delas portas.

Para esta experiência, você deve estudar as seções 4.1 e 4.2 do livro-texto (Wakerly, "Digital Design"). Como esta é a nossa primeira experiência e ninguém é de ferro, a PARTE A contém um breve resumo da teoria desta aula, mas leia o livro-texto também.

ATENÇÃO:

- Estude a apostila **com antecedência**. Sua compreensão será avaliada na aula por **ARGUIÇÃO ORAL**.
- Faça os **EXERCÍCIOS** contidos na apostila e tire dúvidas com os professores **com antecedência**.
- Traga para a aula a apostila **IMPRESSA**. Os pontos importantes devem estar **destacados ou grifados**.

PARTE A 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. Por exemplo, o valor lógico *falso* (ou *zero*, ou *desligado*, etc.) pode ser associado à tensão 0 V, enquanto que *verdadeiro* (ou *um*, ou *ligado*, etc) pode ser associado a 5 V.

Na prática, os níveis lógicos não são representados por tensões precisas, mas sim por *faixas de tensão*. Em eletrônica digital, costuma-se indicar essas faixas por *L* (*Low*) e *H* (*High*). Além disso, em prol de uma notação mais limpa, representaremos o nível *L* pelo símbolo lógico '0' (zero), e ao nível de *H* o símbolo '1' (um).

Os limites das faixas de tensão *L* e *H* dependem da tecnologia empregada na fabricação das portas lógicas. No laboratório, usaremos componentes conhecidos como *CMOS* (*Complementary Metal Oxide Silicon*), em que os limites variam com a tensão com que são alimentados – veremos isso melhor na próxima experiência.

Por hora, podemos adiantar que a tensão de alimentação costuma ser representada por V_{DD} e que as tensões dos níveis lógicos são limitadas pela regra geral dada abaixo (conhecida como "um terço, dois terços"):

- *L* : aproximadamente, de 0 a $V_{DD}/3$
- *H* : aproximadamente, de $2\,V_{DD}/3$ a V_{DD}

Há apenas TRÊS funções lógicas elementares: NOT, AND e OR. Com elas, é possível implementar qualquer circuito digital.

1.1.1 Porta NOT (inversora, complemento ou negação lógica)

A porta NOT possui uma entrada (*A*) e uma saída (*Z*). A Figura 1.1 mostra o símbolo da porta NOT, também conhecida como *inversora*. A saída fornece o nível lógico complementar ao da entrada, ou seja

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

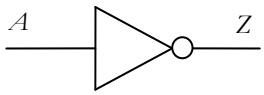


Figura 1.1 Símbolo da porta lógica NOT (inversora)

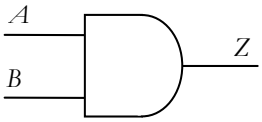
Textualmente, a função NOT costuma ser indicada de diversas formas diferentes. Algumas mais comuns:

$$\mathrm{NOT}(A) = A' = \overline{A} = \_A = /A = \sim A = !A\,.$$

A primeira e a segunda são as mais empregadas em textos como esta apostila. As notações "_*Y*" e "/*Y*" aparecem mais em desenhos, enquanto que as duas últimas são usadas em linguagens de programação.

1.1.2 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.

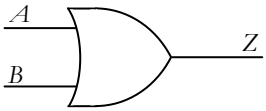


A	*B*	*Z = A.B*
0	0	0
0	1	0
1	0	0
1	1	1

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

1.1.3 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

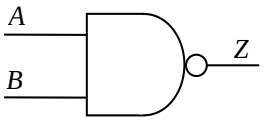
Figura 1.3 Símbolo da Porta OR e sua Tabela da Verdade

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 - combinações bastante simples, mas de extrema importância.

1.2.1 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.

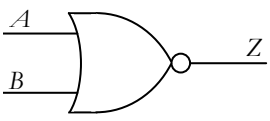


A	*B*	*Z = (A.B)'*
0	0	1
0	1	0
1	0	0
1	1	0

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

1.2.2 NOR

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

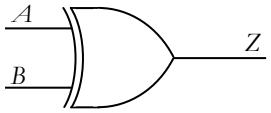


A	*B*	*Z = (A+B)'*
0	0	0
0	1	1
1	0	1
1	1	1

Figura 1.5 Símbolo e tabela da verdade da porta NOR.

1.3 Porta XOR (OU-EXCLUSIVO)

A função OU-EXCLUSIVO (conhecida como XOR) é uma função lógica que tem várias aplicações. 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 expressão abaixo é valida (ela será demonstrada na seção 1.6).

$$A \oplus B = \overline{A}.B + A.\overline{B}. \qquad (1.1)$$

1.4 Circuito Integrado (CI)

A maioria dos circuitos digitais, desde portas lógicas até os microprocessadores de última geração, são fabricados em lâminas silício – um material semicondutor. 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 e possuem pinos metálicos ligados aos pontos de entrada e saída do circuito.

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

É o caso por exemplo do 74HC08, uma pastilha de 14 pinos que contém em seu interior quatro portas AND, cada uma com duas entradas. Veja o seu *datasheet* para ver como as entradas e saídas das portas estão conectadas aos pinos do CI. Por exemplo, repare que a primeira porta tem suas entradas (denominadas A1 e B1) ligadas aos pinos 1 e 2, enquanto que a saída (Y1) 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 *datasheet* do CI 74HC04. Os CIs que veremos neste texto são:

- 74HC04 – *hex inverter* (ou seja, seis portas inversoras)
- 74HC08 – *quad 2-input* AND (ou seja, quatro portas AND de duas entradas)
- 74HC32 – *quad 2-input* OR
- 74HC86 – *quad 2-input* XOR

1.4.1 Alimentação

Portas lógicas são circuitos eletrônicos e precisam ser alimentados eletricamente para funcionar. No 74HC08 por exemplo, o pino 14 deve ser ligado ao terminal positivo da fonte de tensão e o pino 7 ao terminal negativo, como ilustra a Figura 1.7. O lado do pino 1 costuma ser indicado por um chanfro ou por um ponto.

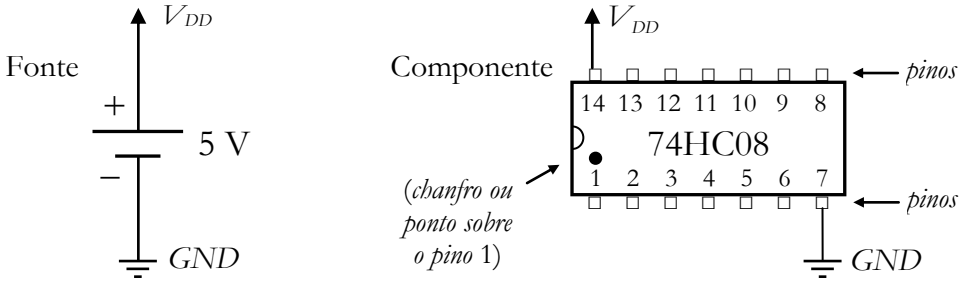


Figura 1.7 Alimentação elétrica do integrado 74HC08.

Nos diagramas elétricos, costumam-se representar os pontos ligados à tensão de alimentação (V_{DD}) por uma **seta para cima**. O tradicional símbolo "⏚" (*terra*) representa os pontos ligados ao negativo da fonte, a qual

atribuímos a tensão de 0 V. Na Figura 1.7, não estão desenhadas as ligações entre os pontos V_{DD} da fonte e do CI para deixar o diagrama mais limpo, e o mesmo acontece com os pontos de *GND*

As letras "HC" presentes no código dos componentes indicam que estes fazem parte de uma subfamília tecnológica do padrão CMOS. Esses componentes devem ser alimentados com uma tensão entre 2 a 6 V (confira nos anexos III ou IV). Na figura, temos V_{DD} = 5 V, que é a tensão nominal que usaremos no laboratório.

Nota: nos *datasheets* dos CIs em anexo, os pinos de alimentação são indicados por GND e VCC. Neste texto chamamos a tensão de alimentação de V_{DD}, que é mais apropriado para componentes CMOS. O nome VCC é uma herança da tecnologia anterior, denominada *TTL* (*Transistor-Transistor Logic*). Veremos isso na próxima experiência.

E por coincidência, os pinos de alimentação são os mesmos em três dos componentes que usaremos: inferior direito e superior esquerdo, mas isso não é regra geral – consulte sempre o *datasheet* do componente.

1.4.2 Níveis lógicos e entradas em aberto

Como dissemos, um componente 74HC entende tensões próximas a 0 V em suas entradas como nível *LOW*. Assim, para impor nível lógico *L* em uma entrada, basta ligá-la a 0 V (ou seja, à malha de terra do circuito). Já uma tensão próxima à tensão de alimentação V_{DD} corresponde ao nível *HIGH*, e para impor nível lógico *H* numa entrada podemos ligá-la a V_{DD}. É o que ilustra os dois primeiros diagramas da Figura 1.8.

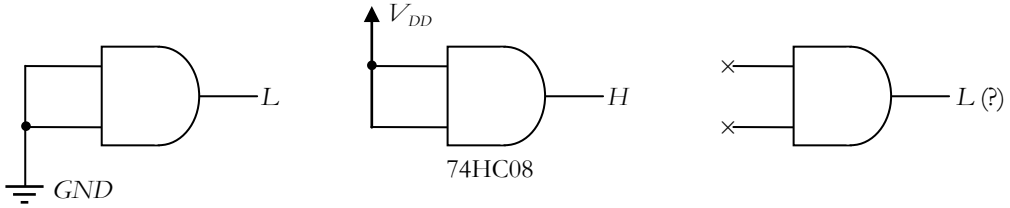


Figura 1.8 Porta AND com entradas fixas em *L* e saída em *L* (à esquerda) e entradas fixas em *H* e saída em *H* (centro). Com entradas em aberto, o nível de saída *esperado* é *L* (direita).

Resta então uma questão instigante: que nível lógico estaria associado a entradas deixadas em aberto? Ou seja, ligadas a NADA?

Em termos técnicos, diz-se que uma entrada em aberto se encontra no estado de *alta impedância*. E a resposta para a pergunta é: por motivos construtivos, no padrão HC uma entrada em alta impedância é interpretada como uma entrada em nível *LOW*. Assim, na porta AND à direita na Figura 1.8, o "×" indica que as entradas estão em aberto e a saída esperada será *L*.*L* = *L*.

No entanto, deixar uma entrada em aberto não é o mesmo que ligá-la 0 V. A condição de alta impedância deixa a entrada susceptível a ruído e interferências. Como não se garante o nível de tensão na entrada, o nível lógico pode mudar inesperadamente.

Em um bom projeto, as entradas não usadas de um componente NÃO DEVEM ser deixadas em aberto, mas ligadas a 0 V ou V_{DD}, conforme o caso.

1.5 Diagrama Lógico

A Figura 1.9 mostra o esquema de um circuito que implementa a função XOR de duas entradas a partir da expressão 1.1. Um esquema como este é chamado de *diagrama lógico* – ou, carinhosamente, de *DL*. O circuito utiliza três CIs. O CI 74HC04, por exemplo, contém seis portas NOT em seu interior, sendo que apenas duas delas (indicadas por U1a e U1b) são usadas. Veja o *datasheet* desse componente em anexo.

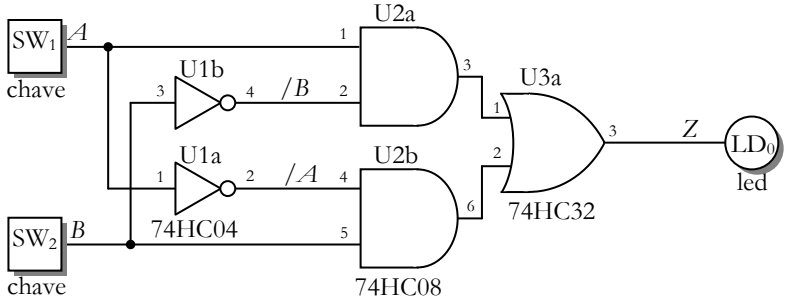


Figura 1.9 Diagrama lógico de um circuito XOR construído com portas elementares

O DL da Figura 1.9 exemplifica algumas boas normas de documentação de circuitos lógicos:

- Os componentes possuem um identificador (U1, U2, U3) e o respectivo código comercial (74HCxx).
- Múltiplas portas de um mesmo CI são diferenciadas por letras (U1a, U2b, etc).
- Os pinos dos CIs ligados às portas estão numerados
- Os sinais importantes são identificados por nomes (*A*, */A*, etc)

Nota: outra boa prática de projeto – a chamada *representação hierárquica*, será explicada mais a frente.

Incluímos também símbolos para representar as chaves (quadrados) que fornecem os sinais de entrada e o led (círculo) que é acionado pela saída do circuito. Adote essa prática ao fazer os diagramas pedidos no pré-relatório e no relatório. Isso vai facilitar sua vida na hora de montar e testar o circuito.

Preste atenção nos detalhes para desenhar as portas corretamente!

- Porta AND: entrada reta e saída **redonda.**
- Porta OR, entrada curva e saída **pontiaguda!**

1.6 Tabela da Verdade

A Tabela 1.1 é a chamada *tabela da verdade* completa do circuito. Ela contém o valor das saídas de cada porta lógica para cada combinação possível de entradas. Como você pode ver, a saída do circuito (pino 3 do componente U3) atende a definição da função XOR, dada na tabela da Figura 1.6.

Tabela 1.1 Tabela da verdade completa do circuito da Figura 1.9.

A	***B***	***/A*** (U1 pino 2)	***/B*** (U1 pino 4)	***A./B*** (U2 pino 3)	***/A.B*** (U2 pino 6)	***A./B + /A.B*** (U3 pino 3)
0	0	1	1	0	0	0
0	1	1	0	0	1	1
1	0	0	1	1	0	1
1	1	0	0	0	0	0

Tabelas da verdade completas como essa são muito úteis para localizar defeitos em circuitos lógicos. Por exemplo, se fazemos $A = 0$ e $B = 0$ no circuito da Figura 1.9 e a saída resulta em 1, uma ou mais portas estão com defeito ou pode haver algum mal contato. Nestas condições, podemos verificar o valor da saída de cada porta lógica, em busca das que não reproduzem o valor previsto na Tabela 1.1.

1.7 Introdução à Álgebra de Boole

As funções lógicas AND e OR constituem uma álgebra, denominada *Álgebra de Boole*. É por esse motivo que as representamos matematicamente pelos habituais operadores algébricos '.' e '+' respectivamente.

A definição formal da álgebra determina que o seu conjunto domínio inclua pelo menos dois elementos: 0 (elemento neutro da operação '+') e 1 (elemento neutro da operação '.'). No caso particular em que o conjunto de domínio é composto por apenas 0 e 1, temos a *Álgebra de Chaveamento*, que constitui a base teórica dos sistemas digitais.

Como toda boa álgebra, a álgebra de Boole pode ser formalmente definida por um conjunto de postulados (ou axiomas), a partir do qual derivam-se propriedades e teoremas que constituem uma "caixa de ferramentas" para manipular expressões algébricas. A seguir listamos algumas propriedades básicas, que você pode confirmar analisando as tabelas da verdade das funções elementares (seção 1.1). Outras propriedades serão estudadas nas aulas teóricas e nas próximas experiências.

$$\textit{Elemento neutro} \begin{cases} 1.a = a.1 = a \\ 0 + a = a + 0 = a \end{cases} \tag{1.2}$$

$$\textit{Complemento} \begin{cases} a.\bar{a} = \bar{a}.a = 0 \\ a + \bar{a} = \bar{a} + a = 1 \end{cases} \tag{1.3}$$

$$\textit{Comutatividade} \begin{cases} a.b = b.a \\ a + b = b + a \end{cases} \tag{1.4}$$

$$\textit{Idempotência} \begin{cases} a.a = a \\ a + a = a \end{cases} \tag{1.5}$$

$$\textit{Absorção} \begin{cases} 0.a = a.0 = 0 \\ 1 + a = a + 1 = 1 \end{cases} \tag{1.6}$$

$$\textit{Involução} \quad \left(\overline{\overline{a}}\right) = a \tag{1.7}$$

$$\textit{Distributividade} \begin{cases} (a+b).c = a.c + b.c \\ (a.b) + c = (a+c).(b+c) \end{cases} \tag{1.8}$$

A segunda igualdade da propriedade 1.8 pode causar estranheza, pois mostra que a operação '+' é distributiva sobre a operação '.' na álgebra de Boole.

1.8 Verificador de Paridade

Vamos ver um aplicação da função XOR. É muito comum sistemas digitais necessitarem trocar informações com outros, e nesses casos a mensagem enviada por um lado está sempre sujeita a ser recebida com erros pelo outro lado devido a algum mal contato, ruído ou interferência eletromagnética.

O mecanismo de paridade é o seguinte: o emissor envia ao receptor uma mensagem de *n* bits e mais um bit adicional chamado *bit de paridade,* que é função dos *n* bits da mensagem propriamente dita.

Existem dois tipos de paridade: par e 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 **0001** (*n* = 4 bits). Adotando-se paridade **par**, a mensagem seria transmitida como **00011** – com o bit de paridade igual a 1 para que o número total de 'uns' transmitidos seja par (dois, neste caso). Analogamente, no caso de paridade ímpar, transmitir-se-ia **00010**.

Se a mensagem for recebida sem erros, o bit de paridade estará coerente com os *n* bits recebidos. Caso contrário, isso não acontece. Note que erros em até um bit são detectados, mas não há como corrigi-los. Ao detectar um erro, o receptor deve solicitar a retransmissão da mensagem ou ignorá-la.

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.

$$PP = b_{n-1} \oplus b_{n-2} \oplus \ldots b_0 \ , \tag{1.9}$$

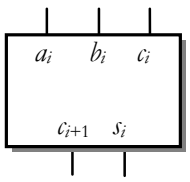
$$PI = \overline{PP} = \overline{b_{n-1} \oplus b_{n-2} \oplus \ldots b_0} \ . \tag{1.10}$$

Note que para implementar essas funções seriam necessárias portas XOR de várias entradas, o que não é comum. Por exemplo, cada porta do CI 74HC86 possui apenas duas entradas. Isso não é problema, sabendo que a função XOR é *associativa*! Ou seja,

$$b_2 \oplus b_1 \oplus b_0 = (b_2 \oplus b_1) \oplus b_0 = b_2 \oplus (b_1 \oplus b_0) \ .$$

1.9 Somador Binário Completo

O somador completo – ou *full adder*, é um circuito digital com três bits de entrada (a_i, b_i e c_i) e dois bits de saída (c_{i+1} e s_i). O esquema desse circuito e a tabela da verdade estão na Figura 1.10.



a_i	b_i	c_i	c_{i+1}	s_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Figura 1.10 Somador completo e sua tabela da verdade

Repare que as saídas c_{i+1} e s_i compõem um número em binário de dois bits de 00 a 11 (0 a 3 em decimal) que corresponde à **soma aritmética** dos valores (0 ou 1) dos três bits de entrada! Por exemplo, para $a_i = 0$, $b_i = 1$ e $c_i = 1$ a soma desses três bits em binário é 0 + 1 + 1 = 10 (2 em decimal), que é representado nas saídas por $c_{i+1} = 1$ e $s_i = 0$.

Nota: os nomes das entradas e as saídas parecem estranhos, mas lembre-se que são apenas nomes – podem ser escolhidos arbitrariamente. Veremos mais a frente que há um motivo importante para chamá-los assim.

Você saberia dizer qual é a função lógica correspondente ao bit de saída s_i? Aí vai uma dica.

Como s_i é o bit menos significativo do resultado, seu valor será igual a 1 somente quando a soma for ímpar. Ou seja, quando houver um número ímpar de entradas em 1. Portanto... s_i será um detector de paridade ímpar – um XOR de todas as entradas, como vimos na seção 1.7.

$$s_i = a_i \oplus b_i \oplus c_i \,. \tag{1.11}$$

A equação do bit c_{i+1} é menos óbvia. Por ser o bit mais significativo da soma, podemos concluir que c_{i+1} será igual a 1 sempre que a soma resultar maior ou igual a dois (10 em binário). Portanto, c_{i+1} deve detectar situações em que pelo menos 2 das entradas sejam iguais a 1. Ou seja, c_{i+1} é 1 quando: a_i e b_i valem 1, ou a_i e c_i valem 1, ou a_i e c_i valem 1. Matematicamente, temos

$$c_{i+1} = a_i.b_i + a_i.c_i + b_i.c_i \,. \tag{1.12}$$

Analisando as equações 1.11 e 1.12, vemos que saídas s_i e c_{i+1} podem ser implementadas por meio de dois circuitos independentes, sem que partes de um sejam aproveitadas para implementar o outro (iremos montá-las dessa maneira no laboratório). Assim podemos representar cada delas por um símbolo próprio, como mostra a Figura 1.11.

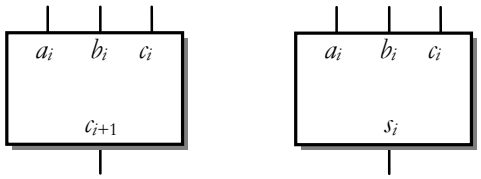


Figura 1.11 Símbolos lógicos das funções c_{i+1} (à esquerda) e s_i.

1.10 Somador de *n* bits com *ripple-carry*

Como construir então um circuito para somar dois números binários de *n* bits cada? Por extensão, podemos definir o somador completo de *n* bits, cujo símbolo lógico é mostrado na Figura 1.12.

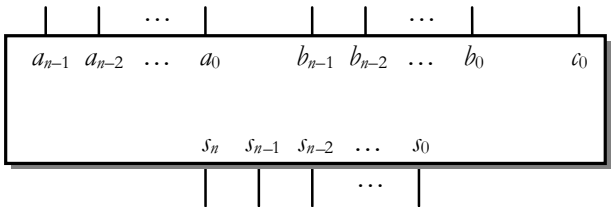


Figura 1.12 Símbolo do somador completo de *n* bits.

Sendo $a[n{-}1{:}0] = a_{n-1}\, a_{n-2}..\, a_0$ e $b[n{-}1{:}0] = b_{n-1}\, b_{n-2}..\, b_0$ dois números em binário de *n* bits, o somador gera um número $s[n{:}0] = s_n\, s_{n-1}..s_0$ de *n*+1 bits como resultado da soma aritmética dada por

$$s[n{:}0] = a[n{-}1{:}0] + b[n{-}1{:}0] + c_0 \,.$$

O bit de entrada c_0 é o chamado ***carry-in*** (vem-um) e permite somar um ao resultado da soma fazendo-se $c_0 = 1$.

Para entender como esse somador funciona, vamos fazer manualmente a soma em binário "11 + 11 = 110" (em decimal, "3 + 3 = 6"), desconsiderando a entrada c_0 para simplificar. Fazendo a soma "coluna a coluna" da forma que faríamos para somar dois números em decimal, temos o procedimento ilustrado na Figura 1.13.

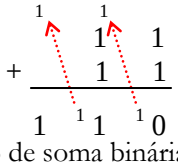


Figura 1.13 Exemplo de soma binária de 2 números de 2 bits

Na primeira coluna da direita, temos a soma dos bits menos significativos "1 + 1" que resulta em "10" (2 em binário), sendo que o "0" compõe o resultado da soma da coluna e o "1" é o ***carry-out*** (vai-um). Esse bit passa a ser o *carry-in* da segunda coluna e deve ser somado aos dois bits dessa coluna, de forma que temos "1 + 1 + 1 = 11" (3 em binário). Isso gera novo *carry-out*, que passa para a terceira coluna, e assim por diante. Elementar, não?

Conclusão: é possível construir um somador de *n* bits por **cascateamento** de *n* somadores completos de um bit , nos quais a saída s_i fornece o *i*-ésimo bit do resultado da soma e a saída c_{i+1} gera o vai-um da coluna *i*+1. Ou seja, basta conectar a saída *carry-out* de um somador à entrada *carry-in* do seguinte, como mostra a Figura 1.14. Essa solução é conhecida como ***ripple-carry***.

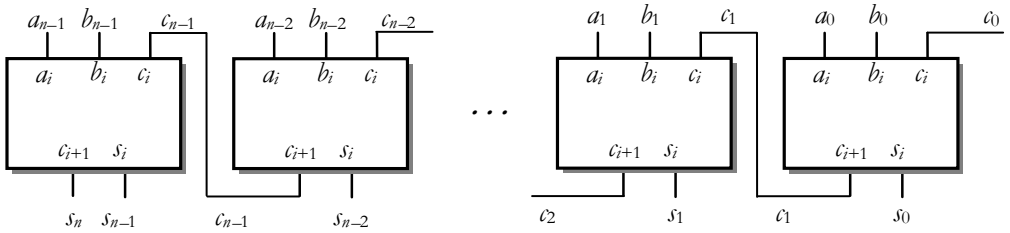


Figura 1.14 Somador *ripple-carry* de n bits.

Cabe uma ressalva importante: o somador *ripple-carry* funciona, mas **não é usado na prática** devido ao problema de *atraso de propagação*. Como as saídas de cada somador completo levam algum tempo para se estabilizarem após uma variação nas entradas, o bit mais significativo da soma somente se estabiliza após *n* vezes esse atraso de propagação. A solução envolve circuitos dedicados para calcular em paralelo cada um dos bits de *carry-in*, os chamados *Geradores de Vai-Um*, que estudaremos mais a frente.

1.11 Representação Hierárquica

Vamos aproveitar o somador completo de *n* bits para ilustrar uma prática importante de documentação de circuitos digitais: a *representação hierárquica*.

No topo da hierarquia, o somador é representado pelo símbolo da Figura 1.12, que contém apenas os sinais de entrada e saída, sem mostrar qualquer detalhe da implementação (a famosa "caixa-preta") . Descendo um nível de hierarquia – isto é, de detalhamento, temos a Figura 1.14 que mostra como implementar o "somador de *n* bits" usando-se *n* exemplares do circuito "somador de um bit".

Note na Figura 1.14 que os somadores de um bit são representados por símbolos lógicos (as "caixas pretas" deste nível), todos iguais. Detalhe de (boa) documentação: do lado de dentro dos símbolos, aparecem os nomes genéricos das entradas e saídas (sempre os mesmos) e do lado de fora indicam-se os nomes dos sinais do circuito. **Siga esse padrão** quando for desenhar os seus diagramas lógicos!

Descendo mais um nível, tem-se que cada somador de um bit por sua vez é composto por dois circuitos: c_{i+1} e s_i, cujos símbolos são mostrados na Figura 1.11. Por fim, o nível mais baixo de detalhamento mostraria o diagrama lógico desses circuitos na forma de portas lógicas – mas isso fica para você fazer no pré-relatório.

1.12 Materiais e Equipamentos

1.12.1 O *protoboard*

Nesta experiência, usaremos a placa para a montagem e teste de circuitos eletrônicos conhecida como *protoboard* mostrada na Figura 1.15.

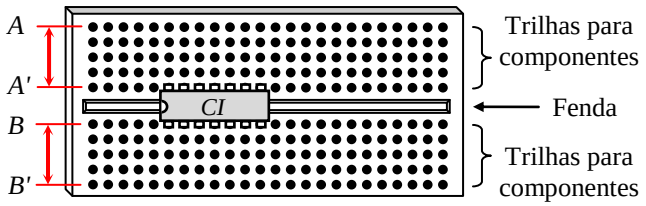


Figura 1.15 Barra de conexões *protoboard*

O *protoboard* é uma barra com vários pontos de conexão espaçados de 0,1 polegadas, onde podem ser encaixados os terminais de circuitos integrados e componentes discretos. Uma fenda central divide a barra em duas metades. Em cada metade estão dispostas várias trilhas verticais de cinco furos.

As setas indicam de que forma os pontos estão conectados internamente entre si por lâminas metálicas condutoras. Uma trilha vertical de 5 furos na posição (*A-A'* ou *B-B'*) NÃO ESTÁ em curto com outras trilhas paralelas a ela e também NÃO HÁ conexão entre trilhas verticais *A-A'* e *B-B'* de cada lado da fenda .

Para montar os circuitos no *protoboard*, coloque integrados com encapsulamento DIP (*Dual In Paralell*) longitudinalmente sobre a fenda de uma barra de conexões, como mostra a figura. Desta forma, cada pino ocupará um furo de uma trilha vertical isolada das demais, e você terá os quatro furos restantes da trilha para ligar o pino a outro ponto do circuito.

1.12.2 Placa XLA

No laboratório, usaremos a placa XLA mostrada na Figura 1.16 para montar e testar os circuitos digitais. Ela contém vários componentes, mas para esta experiência precisaremos apenas do *protoboard*, das chaves e leds.

Como fonte de alimentação, usa-se um conversor de entrada 127 V AC e saída **5 V DC** nominais.

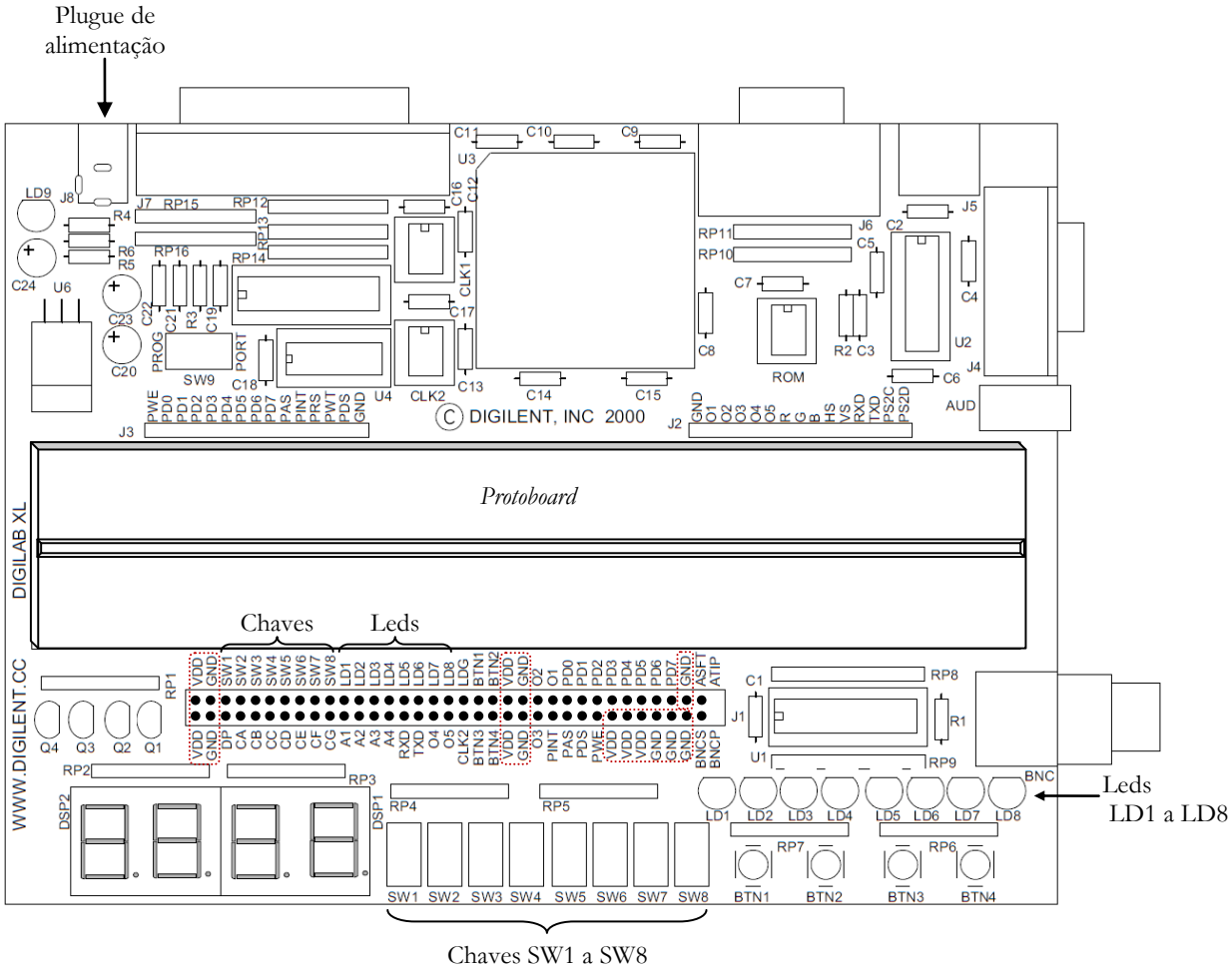


Figura 1.16 Placa didática XLA

A placa contém oito chaves do tipo liga-desliga na parte inferior, numeradas de SW1 a SW8. Na barra de terminais situada logo abaixo do *protoboard*, há um ponto ligado a cada uma delas. Num ponto tem-se 0 V (nível *LOW*) quando a chave correspondente está desligada e 5 V (nível *HIGH*) quando ligada. A chave permanece desligada quando posicionada para frente e é ligada quando acionada para trás. Os oito leds, numerados de LD1 a LD8, se encontram na parte inferior direita da placa. Para cada led também há um ponto correspondente na barra de terminais. Aplicando-se uma tensão de 5 V a um ponto, acende-se o led correspondente.

Atente também para os pontos "VDD" e "GND" na barra de terminais da Figura 1.16 (indicados pelas linhas pontilhadas). Em VDD tem-se a **tensão de alimentação de 5 V** e em GND tem-se 0 V. Use-os para alimentar os CIs montados no *protoboard*.

1.12.3 Multímetro

O multímetro é um instrumento de laboratório voltado à medição de tensões, correntes e resistências. No caso do multímetro digital Minipa ET-2060 que usaremos (mostrado na Figura 1.17), podem ser feitas também medições de frequência, capacitância, teste de diodo, teste de transistor.

Figura 1.17 Multímetro Minipa ET-2060

A chave rotativa permite selecionar as funções e escalas. Tensões e correntes podem ser medidas em valor contínuo (DC) ou alternado senoidal (AC), posicionando-se a pequena chave seletora a esquerda do aparelho.

A ponta de prova preta deve ser ligada ao terminal "COM" (preto) em todos os casos. Para medidas de tensão e resistência, a ponta de prova vermelha deve ser ligada ao terminal "V/Ω" (vermelho) – os outros terminais amarelos da direita se destinam a medidas de corrente e não serão usados neste curso.

LEMBRE-SE: uma medida de tensão sempre se refere a DIFERENÇA DE POTENCIAL (*ddp*) entre DOIS PONTOS. Mas quando nos referimos à *tensão em um ponto* subentendemos que se trata da diferença de tensão entre o tal ponto e um segundo ponto ligado à *tensão de referência* do circuito (o chamado *terra*), ao qual arbitrariamente atribuímos o valor 0 V, e que nesta atividade será qualquer ponto ligado ao terminal negativo da fonte de tensão. Portanto, o medir uma tensão, conecte a ponta preta do multímetro (COMUM) a um ponto de terra.

Um recurso útil: **teste de continuidade**. Essa função é indicada por um símbolo de *bip* na parte inferior da chave rotativa (veja a indicação na Figura 1.17). Nessa condição, o multímetro emite um *bip* sempre que as pontas de provas são postas em curto-circuito. Por exemplo, use o teste de continuidade para verificar se algum cabinho está rompido.

Atenção: o teste de continuidade funciona de forma semelhante à medida de resistência, ou seja, o multímetro aplica uma tensão entre as pontas de prova e mede a corrente que circula entre elas. Portanto, NÃO se deve testar a continuidade ou medir a resistência em partes de um circuito ENERGIZADO. Caso contrário, você pode queimar o multímetro ou o circuito, ou ambos. Cuidado também para não aplicar o teste de continuidade ou a medição de resistência em partes eletricamente muito sensíveis, pois a tensão aplicada pelo multímetro pode ser suficiente para queimá-las.

Quando não estiver usando, mantenha o multímetro desligado (chave seletora na posição "0").

1.13 Pré-Relatório e Relatório

O formulário que se encontra na PARTE B da apostila constitui tanto o pré-relatório como o relatório desta experiência. Existem dois tipos de itens que você deverá responder:

- **Exercícios**: constituem o *pré-relatório* e devem ser feitos *antes* da aula.
- **Anotações**: devem ser feitas individualmente *durante* a aula e constituem o *relatório*.

ATENÇÃO: leia as atividades da PARTE B e não apenas os enunciados dos exercícios do pré-relatório

Muitos detalhes necessários para fazer os exercícios estão descritos nas atividades em que se inserem. Além disso, você já terá uma noção do que deverá fazer e perderá menos tempo com a leitura durante a aula.