Experimento 4:
IMPLEMENTAÇÃO DE CIRCUITOS COMBINACIONAIS COM MULTIPLEXADORES

1 MULTIPLEXAÇÃO E DEMULTIPLEXAÇÃO

Multiplexar significa selecionar dados dentre diversas fontes. A Fig. 1 mostra o esquema funcional generalizado de um multiplexador lógico. Nesse dispositivo, os terminais de seleção determinam o terminal de entrada de dados que terá seu conteúdo transferido para a saída.

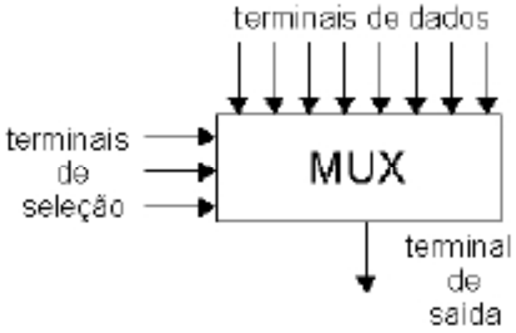

Figura 1 - Dispositivos eletrônicos de manuseio de dados: multiplexador.

A operação inversa é denominada demultiplexação. Como será mostrado adiante, o demultiplexador lógico (Fig. 2) é quase equivalente a um decodificador.

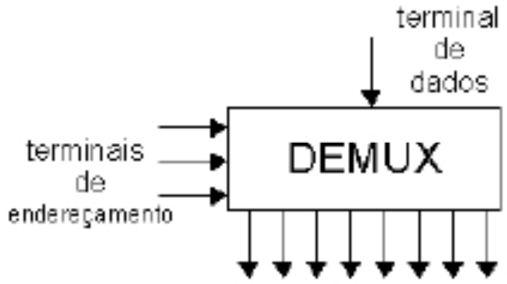

Figura 2 - Dispositivos eletrônicos de manuseio de dados: Demultiplexador.

As operações de multiplexação e demultiplexação são realizadas quando diversas fontes de dados compartilham de uma mesma unidade de processamento ou canal de transmissão (Fig. 3).

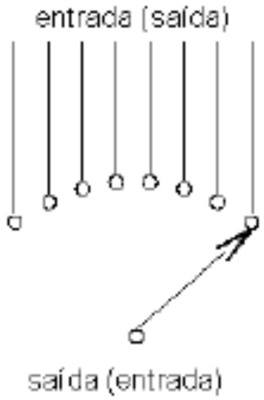

Figura 3 - Equivalente mecânico dos dispositivos eletrônicos de manuseio de dados.

1.1 Exemplos ilustrativos

Os multiplexadores e demultiplexadores também podem ser encarados simplesmente como circuitos combinacionais com diversos terminais de entrada e um de saída, ou vice-versa. O conceito de seleção de dados é mais uma questão de aplicação e ponto de vista do que de funcionamento.

Exemplo 1: Multiplexador com 4 terminais de dados (MUX-4)

Esse circuito é mostrado na Fig. 4. A tabela 1 mostra a tabela-verdade desse multiplexador em forma compactada. A tabela completa teria 64 linhas e, portanto, não seria uma maneira eficiente de exprimir seu funcionamento. Se, por exemplo, $E_1 = 1$ $E_2 = 0$, tem-se $S = D_1$. Esse circuito pode ser visto como um selecionador de dados.

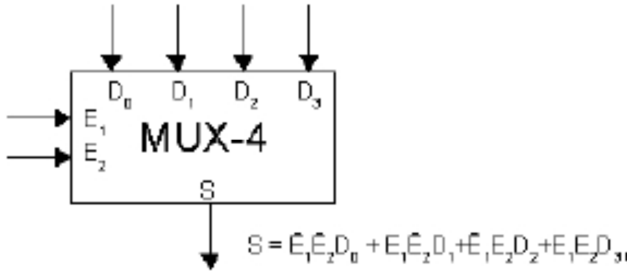

Figura 4 - MUX-4 e sua expressão booleana.

Tabela 1 - Tabela verdade compactada do MUX-4.

E_2	E_1	S
0	0	D0
0	1	D1
1	0	D2
1	1	D3

Exemplo 2: Demultiplexador com 4 saídas (DEMUX-4)

Esse circuito, mostrado na Fig. 5, é um decodificador onde D é um terminal de ativação. A tabela verdade é apresentada na tabela 2.

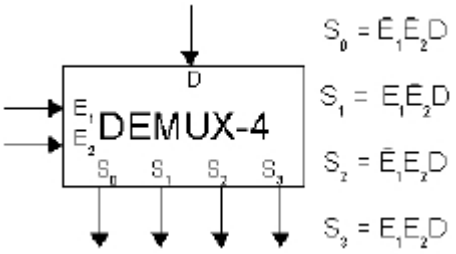

Figura 5 - DEMUX-4 com suas expressões booleanas.

Tabela 2 - Tabela verdade compactada do DEMUX-4.

D	E_2	E_1	S_0	S_1	S_2	S_3
0	X	X	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

2 APLICAÇÃO DE MULTIPLEXADORES

Além de sua aplicação natural como selecionadores de dados, os multiplexadores podem ser utilizados para se implementar uma função booleana genérica. Eles são particularmente convenientes para tal fim quando a função a ser implementada é de natureza irregular, e não permite muita simplificação. Em um caso desses, o uso de multiplexadores em lugar de portas lógicas convencionais resulta em um projeto mais fácil, mais compacto e mais flexível.

2.1 Exemplo

Projetar um circuito que realize a função *f*, descrita pela tabela-verdade apresentada na tabela 3, pelo mapa de Karnaugh apresentado na Fig. 6, e pela equação lógica apresentada na equação 1.

Tabela 3 - Tabela da função *f*.

A	B	C	D	F	
0	0	0	0	0	f = D
0	0	0	1	1	
0	0	1	0	0	f = 0
0	0	1	1	0	
0	1	0	0	1	$f = \bar{D}$
0	1	0	1	0	
0	1	1	0	0	f = D
0	1	1	1	1	
1	0	0	0	1	$f = \bar{D}$
1	0	0	1	0	
1	0	1	0	1	f = 1
1	0	1	1	1	
1	1	0	0	0	f = D
1	1	0	1	1	
1	1	1	0	0	f = 0
1	1	1	1	0	

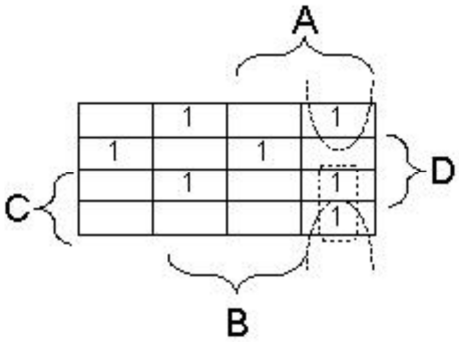

Figura 6 - Mapa de Karnaugh da função *f*.

$$f = \bar{A}.\bar{B}.\bar{C}.D + \bar{A}.B.\bar{C}.\bar{D} + \bar{A}BCD + A.\bar{B}.\bar{C}.\bar{D} + \\ + A\bar{B}C\bar{D} + A\bar{B}CD + AB\bar{C}D$$

Equação 1 - Equação de Excitação da função *f*.

A Fig. 7 apresenta uma implementação da função *f* com um multiplexador de 8 entradas (MUX-8). Três das quatro variáveis independentes são escolhidas para acionar os terminais de seleção. Cada terminal de dado é então acionado por uma as quatro funções lógicas que a variável restante pode formar (D, /D, 1 ou 0).

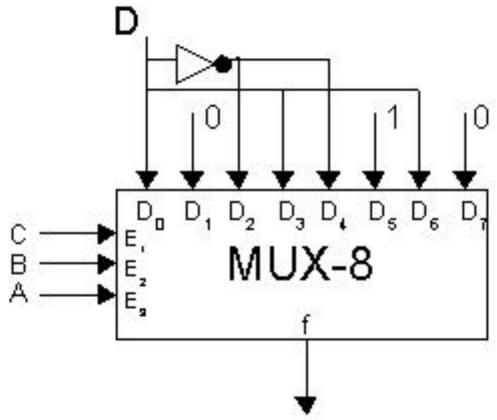

Figura 7 - Implementação da função *f* com um MUX-8.

As três variáveis a serem ligadas nos terminais de seleção de dados (A, B e C, no exemplo) são escolhidas. Então, a tabela-verdade é dividida nos 2^3 = 8 blocos onde essas variáveis são mantidas constantes. A saída *f*, dentro de cada um desses blocos, é função de D apenas e, portanto, existem somente quatro possibilidades: *f* = 0, *f* = 1, *f* = D ou *f* = $\overline{D}$. Cada terminal de dados do MUX é acionado com a função do bloco correspondente.

Exercício: Realize a mesma função, porém, usando as variáveis B, C e D para acionar os terminais de seleção.

2.2 Técnica geral

A técnica de implementação introduzida acima pode ser generalizada para uma função de *n* variáveis. Considere por exemplo um multiplexador de 8 terminais de dados como o da Fig. 8. A expressão booleana da saída S é:

$$S = \overline{E_1}\,\overline{E_2}\,\overline{E_3}D_0 + E_1\overline{E_2}\,\overline{E_3}D_1 + \ldots + E_1E_2E_3D_7$$

Qualquer função de *n* > 3 variáveis pode ser reescrita na forma

$$f(A,B,C,D,E,\ldots) = \overline{A}\,\overline{B}\,\overline{C}F_0(D,E,\ldots) +$$
$$A\overline{B}\,\overline{C}F_1(D,E,\ldots) + \ldots + ABCF_7(D,E,\ldots)$$

onde A, B e C são variáveis selecionadas arbitrariamente dentre as *n* variáveis, F_0, F_1, ..., F_7 são funções das (n−3) variáveis restantes, portanto mais simples que a função original *f*.

A identificação dessas duas expressões conduz à forma geral de implementação mostrada na Fig. 8. No caso particular em que *n* = 4, as funções F_0, F_1, ..., F_7 são funções da única variável restante e existem apenas 4 possibilidades (1, 0, D, /D), como já foi visto. Também, se *n* = 3, então *f* estará na própria forma canônica de mintermos e, portanto, os fatores F_0, F_1, ..., F_7 só podem ser ou identicamente iguais a 0 ou identicamente iguais a 1.

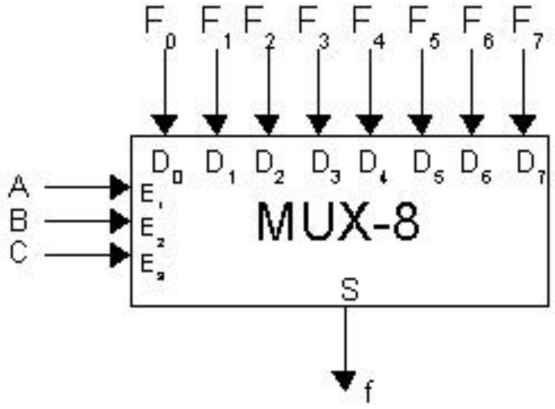

Figura 8 - Ligações de um MUX-8 para a implementação de uma função de *n* > 3 variáveis. No caso *n* = 3, é mais econômico o uso de um MUX-4.

A mesma técnica pode ser naturalmente estendida a multiplexadores com mais entradas. Um multiplexador de 16 entradas de dados, por exemplo, pode implementar qualquer uma das 2^{32} funções diferentes de 5 variáveis, com apenas uma única porta inversora adicional.

3 Aplicação de Decodificadores

O decodificador também é um circuito útil à implementação de funções complicadas, pois cada uma de suas saídas constitui um dos mintermos das variáveis de entrada. Observe, por exemplo, que qualquer função de 4 variáveis pode ser implementada com um DECOD-4 e uma porta OU (Fig. 9).

$$f = \overline{A}\cdot\overline{B}\cdot\overline{C}\cdot\overline{D} + \overline{A}\cdot\overline{B}\cdot C\cdot D + \overline{A}\cdot B\cdot\overline{C}\cdot D + A\cdot\overline{B}\cdot\overline{C}\cdot D$$

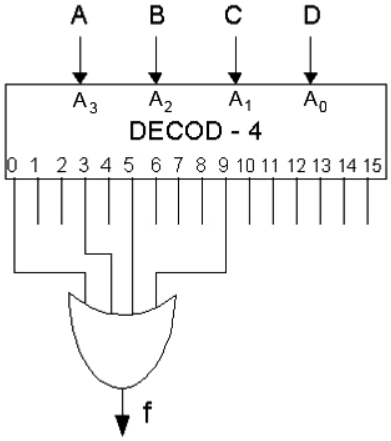

Figura 9 - Uso de um decodificador como gerador de mintermos para implementação de funções. Se o decodificador tiver saídas invertidas, a porta OU é substituída por uma porta NÃO-E.

4 PROJETO MODULAR EM VHDL

Um dos aspectos mais importantes da linguagem é a possibilidade de incluir uma unidade VHDL na descrição de uma outra unidade, levando à modularização dos programas.

Como exemplo, suponha que desejamos construir uma entidade chamada `sistema`, ilustrada na Figura 10(a), que implementa o diagrama de blocos mostrado na Figura 10(b).

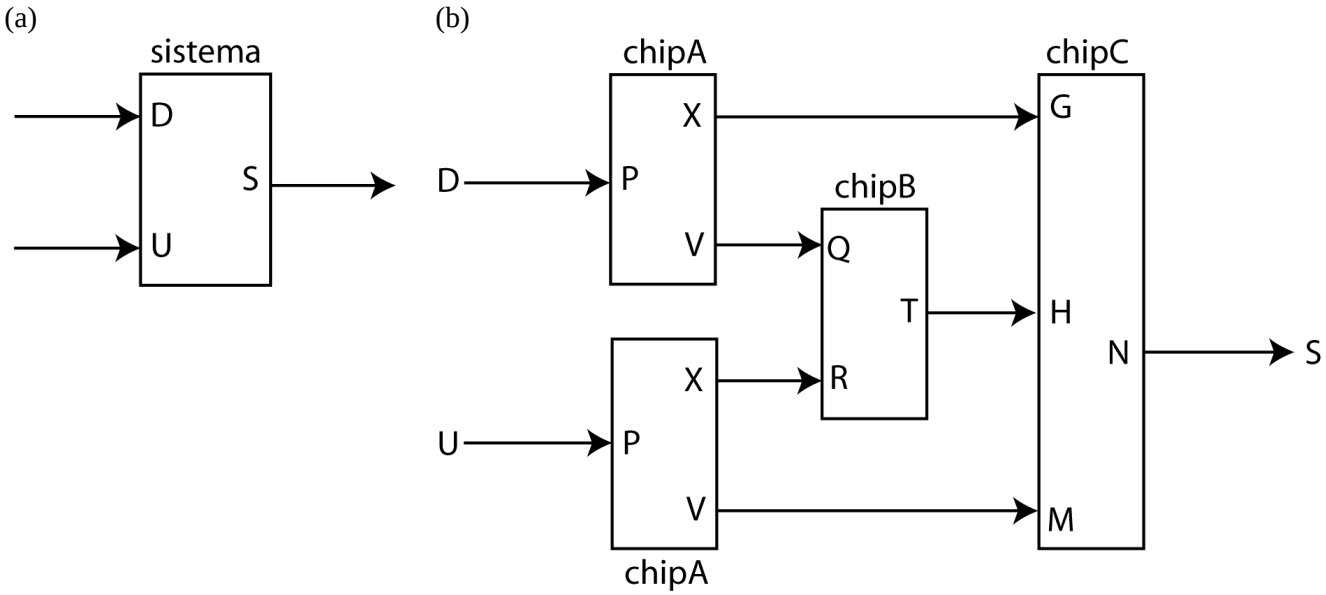

Figura 10 – Entidade `sistema` (a) e sua arquitetura interna (b).

Note, na Figura 10(b), que `sistema` utiliza em sua arquitetura interna três outras entidades: `chipA` (usada duas vezes), `chipB` e `chipC`, apresentadas a seguir.

```
entity chipA is
    port ( P : in STD_LOGIC_VECTOR(3 downto 0);
           X : out STD_LOGIC;
           V : out STD_LOGIC_VECTOR(2 downto 0) );
end chipA;

entity chipB is
    port ( Q : in STD_LOGIC_VECTOR(2 downto 0);
           R : in STD_LOGIC;
           T : out STD_LOGIC_VECTOR(4 downto 0) );
end chipB;

entity chipC is
    port ( G : in STD_LOGIC;
           H : in STD_LOGIC_VECTOR(4 downto 0);
           M : in STD_LOGIC_VECTOR(2 downto 0);
           N : out STD_LOGIC_VECTOR(1 downto 0) );
end chipC;
```

As arquiteturas das entidades `chipA`, `chipB` e `chipC`, serão omitidas aqui, pois são irrevelantes para nosso objetivo didático, uma vez que estas entidades são tratadas neste caso como “caixas pretas”. Dito isso, a entidade `sistema` pode ser implementada como mostrado a seguir.

```
entity sistema is
    port ( D, U : in STD_LOGIC_VECTOR(3 downto 0);
           S : out STD_LOGIC_VECTOR(1 downto 0) );
end sistema;

architecture sistema_arch of sistema is

component chipA is
    port ( P : in STD_LOGIC_VECTOR(3 downto 0);
           X : out STD_LOGIC;
           V : out STD_LOGIC_VECTOR(2 downto 0) );
end component;

component chipB is
    port ( Q : in STD_LOGIC_VECTOR(2 downto 0);
           R : in STD_LOGIC;
           T : out STD_LOGIC_VECTOR(4 downto 0) );
end component;

component chipC is
    port ( G : in STD_LOGIC;
           H : in STD_LOGIC_VECTOR(4 downto 0);
           M : in STD_LOGIC_VECTOR(2 downto 0);
           N : out STD_LOGIC_VECTOR(1 downto 0) );
end component;

signal fio1, fio2 : STD_LOGIC;
signal barramento1, barramento2 : STD_LOGIC_VECTOR(2 downto 0);
signal barramento3 : STD_LOGIC_VECTOR(4 downto 0);

begin
   U0: chipA port map(D,fio1,barramento1);
   U1: chipA port map(U,fio2,barramento2);
   U2: chipB port map(barramento1,fio2,barramento3);
   U3: chipC port map(fio1,barramento3,barramento2,S);
end sistema_arch;
```

Note que, na arquitetura da entidade `sistema`, as entidades `chipA`, `chipB` e `chipC` são incluídas como "components". A seguir, são criados os sinais que representam as conexões internas da entidade `sistema`, que conectarão os componentes `chipA`, `chipB` e `chipC`.

Após o `begin`, utiliza-se o comando `port map` para criar diferentes instâncias dos componentes. Os labels `U0`, `U1`, `U2` e `U3` indicam os nomes dados a cada instância criada (quaisquer outros nomes poderiam ter sido utilizados). Note que são criadas duas instâncias do componente `chipA`: a primeira instância foi rotulada como `U0` e a segunda como `U1` (note, ainda que, embora o componente `chipA` esteja sendo instanciado duas vezes, a declaração de `chipA` como componente, antes do begin, só é feita uma vez). São criadas também uma instância de `chipB`, rotulada como `U2`, e uma instância de `chipC`, rotulada como `U3`.

Para cada instância de cada componente, é utilizada a palavra chave "`port map`" para mapear sinais e entradas e saídas da entidade `sistema` às entradas e saídas (*ports*) do componente instanciado. No caso da instância `U0`, por exemplo, a entrada `D` da entidade `sistema` é associada à entrada `P` desta instância do componente `chipA`, o sinal `fio1` é associado à saída `X` e o sinal `barramento1` é associado à saída `V`. O que garante essas associações é a ordem em que esses sinais aparecem após o comando `port map`. Neste caso, a ordem dos sinais é `D`, `fio1` e `barramento1`; portanto, `D` é associado ao primeiro *port* de `chipA` (a entrada `P`) , `fio1` é associado ao segundo *port* de `chipA` (a saída `X`), e `barramento1` é associado ao terceiro *port* de `chipA` (a saída `V`). Assim, cada uma das conexões mostradas na Figura 10(b) é implementada.