Universidade de Brasília
Departamento de Ciência da Computação

# Laboratório de Circuitos Lógicos - 5º Experimento

## IMPLEMENTAÇÃO DE CIRCUITOS COMBINACIONAIS COM MULTIPLEXADORES

**OBJETIVO:** Os conceitos de multiplexação de demultiplexação são apresentados bem como sua utilização para implementar funções lógicas e uma introdução à Linguagem de Descrição de Hardware Verilog. É realizado o projeto de um somador completo com o uso de multiplexadores e diretamente em Verilog.

### 1. INTRODUÇÃO TEÓRICA

#### 1.1. MULTIPLEXAÇÃO E DEMULTIPLEXAÇÃO

**Multiplexar** significa selecionar dados dentre diversas fontes. A **Figura 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.

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

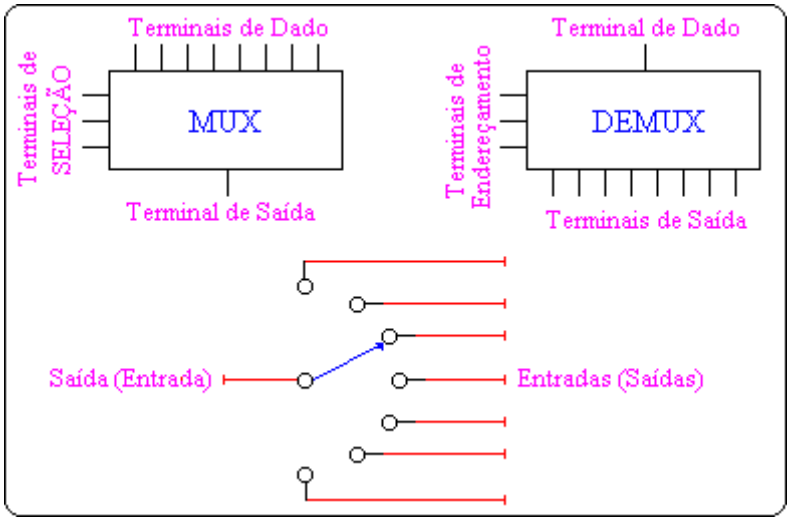



**Figura 1 – Dispositivos eletrônicos de manuseio de dados e seu equivalente mecânico.**

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

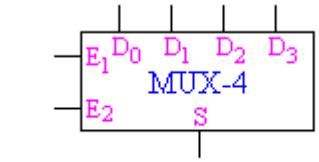
#### 1.2. EXEMPLOS ILUSTRATIVOS

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.

A **Tabela I** mostra a tabela da verdade em forma compacta de um multiplexador de 4 terminais de dado (MUX-4); a completa teria 64 linhas e, portanto, não seria uma maneira eficiente de exprimir seu funcionamento. Se por exemplo, $E_1 = 1$ e $E_2 = 0$, tem-se $S = D_1$. Este circuito pode ser visto como um selecionador de dados.

| $E_2$ | $E_1$ | S |
|---|---|---|
| 0 | 0 | $D_0$ |
| 0 | 1 | $D_1$ |
| 1 | 0 | $D_2$ |
| 1 | 1 | $D_3$ |

**Tabela I – Tabela da verdade do MUX-4**



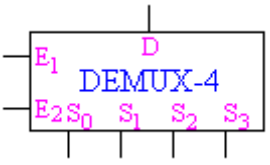

$$S = \overline{E}_1\overline{E}_2D_0 + E_1\overline{E}_2D_1 + \overline{E}_1E_2D_2 + E_1E_2D_3$$

**Figura 2 – MUX-4**

O demultiplexador com 4 terminais de saída da **Figura 3** pode ser visto como um decodificador onde D é um terminal de ativação.

| 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 |

**Tabela II – Tabela da verdade do DEMUX-4**




$$S_0 = \overline{E}_1\overline{E}_2D$$
$$S_1 = E_1\overline{E}_2D$$
$$S_2 = \overline{E}_1E_2D$$
$$S_3 = E_1E_2D$$

**Figura 3 – DEMUX-4**

## 1.3. Implementação de Multiplexadores e Demultiplexadores

Os multiplexadores e demultiplexadores podem ser implementados com as portas lógicas conhecidas.

Dada sua grande aplicação na prática, os multiplexadores e demultiplexadores também são fabricados em forma integrada, com 2, 4, 8 ou mais terminais de entrada de dados. Os demultiplexadores são decodificadores com um terminal de ativação e, portanto também são encontrados em forma de circuitos integrados em média escala (MSI).

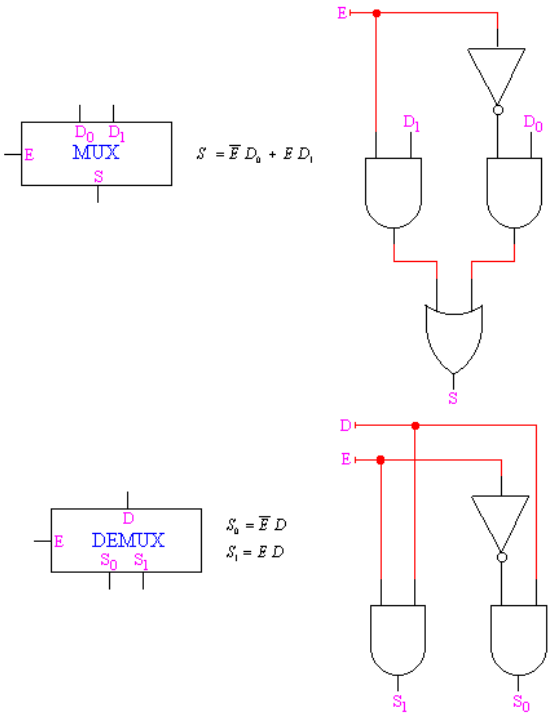



**Figura 4.1 – Implementação de multiplexadores e demultiplexadores com componentes SSI.**

A linguagem de descrição de hardware (HDL) Verilog, possibilita a descrição de sistemas digitais complexos de forma simples. A **Figura 4.2** apresenta as descrições comportamentais em Verilog dos circuitos Multiplexador e Demultiplexador mostrados na **Figura 4.1**.

mux2.v

```
module mux2(
        input [1:0] Dado,
        input Escolha,
        output Saida
);

 assign Saida = Dado[Escolha];

endmodule
```

demux2.v

```
module demux2(
        input Dado,
        input Escolha,
        output [1:0] Saida
        );

 assign Saida[0]= Escolha==1'b0 ? Dado : 1'b0;
 assign Saida[1]= Escolha==1'b1 ? Dado : 1'b0;

endmodule
```

**Figura 4.2 – Implementação de multiplexador 2x1 e demultiplexador 1x2 em Verilog.**

Você deve incluir os arquivos Verilog (mux2.v e demux2.v) no seu projeto, criar os respectivos blocos (Create Symbol) e assim usá-los diretamente no seu diagrama esquemático.

A criação de multiplexadores e demultiplexadores maiores é feita de maneira análoga à apresentada, conforme os exemplos mostrados na **Figura 4.3**

```
module mux8(
        input [7:0] Dado,
        input [2:0] Escolha,
        output Saida
);

 assign Saida = Dado[Escolha];

endmodule
```

```
module demux16(
        input Dado,
        input [3:0] Escolha,
        output [15:0] Saida
        );

 assign Saida[0]= Escolha==4'd0 ? Dado : 1'b0;
 assign Saida[1]= Escolha==4'd1 ? Dado : 1'b0;
...
assign Saida[14]= Escolha==4'd14 ? Dado : 1'b0;
assign Saida[15]= Escolha==4'd15 ? Dado : 1'b0;
endmodule
```

**Figura 4.3 – Implementação de multiplexador 8×1 e demultiplexador 1×16 em Verilog.**

Lembre-se:
Um *decodificador* pode ser implementado como um demultiplexador com Dado de entrada fixo em 1!

## 1.4. Aplicação de Multiplexadores na Geração de Funções

Além de sua aplicação natural como selecionadores de dados, os multiplexadores podem ser usados 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 convencionais resulta em um projeto mais fácil, mais compacto e mais flexível.

### 1.4.1. Exemplo

Projetar um circuito que realize a tabela da verdade abaixo:

| 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 | $f = \overline{D}$ |
| 0 | 1 | 0 | 1 | 0 | |
| 0 | 1 | 1 | 0 | 0 | $f = D$ |
| 0 | 1 | 1 | 1 | 1 | |
| 1 | 0 | 0 | 0 | 1 | $f = \overline{D}$ |
| 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 | |

**Tabela III – Tabela da verdade da função**

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

| AB \ CD | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | | 1 | | 1 |
| 01 | 1 | | 1 | |
| 11 | | 1 | | 1 |
| 10 | | | | 1 |

**Tabela IV – Mapa de Karnaugh da função *f***

Trata-se de uma função pouco simplificável usando-se portas AND, OR e NOT, como mostra seu mapa de Karnaugh. Por isso as técnicas convencionais (minimização por mapa de Karnaugh) são descartadas e a função é realizada com apenas um MUX-8 e uma porta NOT adicional. Três variáveis (A, B e C, no exemplo) são arbitrariamente escolhidas para acionar os terminais de seleção. A tabela da verdade é dividida nos $2^3 = 8$ blocos onde essas variáveis são mantidas constantes. A saída *f*, dentro de cada um destes blocos, é função de D apenas e, portanto, existem somente quatro possibilidades: $f = 0$, $f = 1$, $f = D$ e $f = \overline{D}$. Cada terminal de dados do MUX é acionado com a função de bloco correspondente. Veja a **Figura 5**.

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

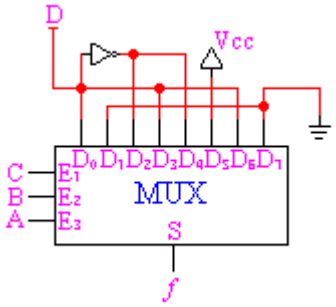



**Figura 5 – Implementação da função *f* com um MUX-8**

1.4.2. Técnica Geral

A técnica geral de implementação introduzida acima pode ser generalizada para uma função de *n* variáveis. Considere, por exemplo, um multiplexador de 8 entradas de dado como o da **Figura 5**. A expressão booleana da saída S é:

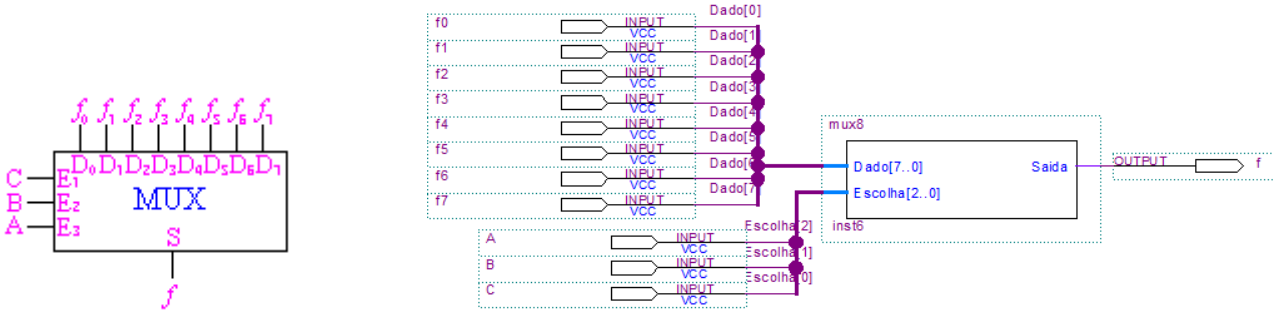
$$S = \overline{E_1}.\overline{E_2}.\overline{E_3}.D_0 + E_1.\overline{E_2}.\overline{E_3}.D_1 + \cdots + E_1.E_2.E_3.D_7$$

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

$$f(A,B,C,D,E,\cdots) = \overline{A}.\overline{B}.\overline{C}.f_0(D,E,\cdots) + A.\overline{B}.\overline{C}.f_1(D,E,\cdots) + \cdots + A.B.C.f_7(D,E,\cdots)$$

onde A, B e C são 3 variáveis selecionadas arbitrariamente dentre as *n*, e $f_0$, $f_1$, ..., $f_7$ 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 **Figura 6**. 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, $D$ e $\overline{D}$), como já foi visto. Também se *n* = 3, *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.




(a) (b)

**Figura 6 – Ligações de um MUX-8 para a implementação de uma função de *n* > 3 variáveis. (a) Esquemático (b) Desenho no Quartus-II.**

A mesma técnica pode naturalmente ser estendida a outros multiplexadores. Um multiplexador de 16 entradas de dado, por exemplo, pode implementar qualquer uma das $2^{32}$ funções diferentes de 5 variáveis, com apenas uma única porta NOT adicional.

## 2. PARTE EXPERIMENTAL

2.1. Use o **mux8** escrito em Verilog para implementar um somador completo **(Pré-Projeto 1)**. O somador completo é um circuito com 2 terminais de saída e 3 terminais de entrada que é usado em operações aritméticas. Os terminais de saída exprimem a soma binária das 3 entradas. Projete, desenhe e simule funcional e temporal, sintetize e filme o funcionamento do circuito, usando os Cout=LEDR[1] e S=LEDR[0] e chaves A=SW[2] B=SW[1] e Cin=SW[0] do kit FPGA **(Pós-Experimento 1)**.

| Entradas | | | Saídas | |
|---|---|---|---|---|
| **A** | **B** | **Cin** | **Cout** | **S** |
| 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 |

**Tabela V – Tabela da verdade do somador completo**

2.2. Implemente novamente o somador completo usando a sua descrição comportamental em Verilog **(Pré-Projeto 2)**. Projete, simule funcional e temporal, sintetize, usando os Cout=LEDR[1] e S=LEDR[0] e chaves A=SW[2] B=SW[1] e Cin=SW[0] do kit FPGA **(Pós-Experimento 2)**.
Dica: `assign {Cout,S}=A+B+Cin;`

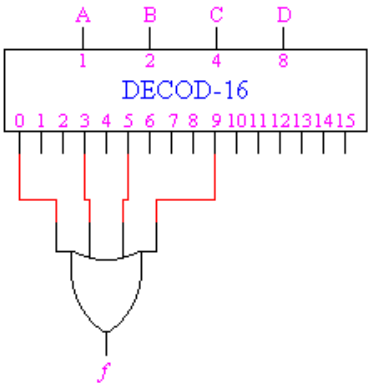
2.3. Implemente a função de 7 variáveis dada abaixo (**Pré-Projeto 3)** usando o multiplexador com 8 entradas de dados (**mux8)** e um decodificador com 16 terminais de saída (**DECOD-16**) com algumas portas adicionais. Projete, simule funcional e temporal, sintetize e filme o funcionamento do circuito, usando os LEDR[0]=$f$ e chaves {*A,B,C,D,E,F,G*}=SW[6:0] do kit FPGA **(Pós-Experimento 3).** Preencha uma tabela da verdade compacta (apenas com os mintermos).

$$f(A,B,C,D,E,F,G) = F.G + A.B.C.D.\overline{E}.\overline{F}.G + \overline{A}.\overline{B}.\overline{C}.\overline{D}.\overline{E}.\overline{F}.G + A.\overline{B}.C.E.F.\overline{G} + \overline{A}.B.C.D.\overline{E}.F.\overline{G} + A.B.C.D.E.\overline{F}.\overline{G} + A.\overline{B}.\overline{C}.D.E.\overline{F}.\overline{G}$$

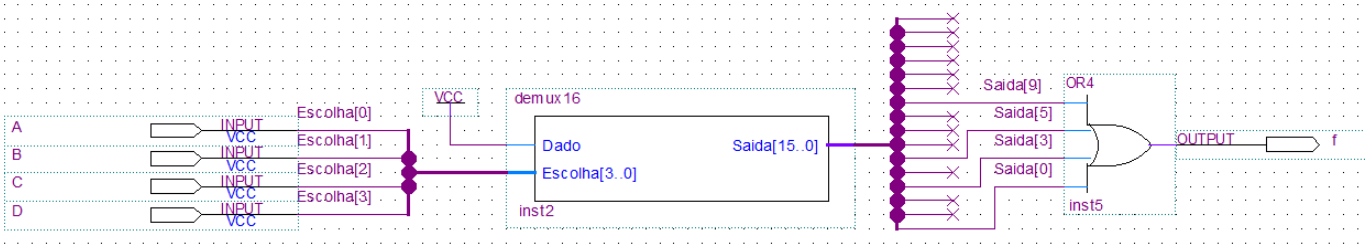
Dica: Use o decodificador como um gerador de mintermos para o multiplexador. O decodificador também é um circuito útil na implementação de funções complicadas, pois cada uma de suas saídas constitui um dos mintermos das variáveis de entrada. Observe, por exemplo, que qualquer função de 4 variáveis pode ser implementada com um DECOD-16 e mais algumas portas OR's.

Por exemplo, a função $f$ abaixo pode ser implementada com um DECOD-16 como mostra a **Figura 8**.

$$f = \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + A\overline{B}\,\overline{C}\,D + A\overline{B}C\overline{D} + AB\overline{C}\,\overline{D}$$
$$\Rightarrow f = m_0 + m_9 + m_5 + m_3$$




(a)




(b)

**Figura 8 – Uso de decodificadores como geradores de mintermos para implementação de funções. (a) Esquemático (b) Desenho no Quartus-II**

## 3. SUMÁRIO

É apresentada a linguagem de descrição de hardware Verilog. A multiplexação e a demultiplexação são realizadas quando diversas fontes de dados compartilham de uma mesma unidade de processamento ou canal de transmissão.

Na presente experiência, os multiplexadores são apresentados como circuitos combinacionais, e é mostrada a relação existente entre os demultiplexadores e decodificadores.

O enfoque principal é a aplicação de multiplexadores na geração de funções booleanas. Essa técnica implementação é bem diferente e mais simples do que as convencionais. Ela é apresentada inicialmente com um exemplo e a seguida sua forma geral.

## 4. EQUIPAMENTOS E MATERIAL

- Software Quartus-II versão 13.0 SP1
- Kit de desenvolvimento em FPGA DE2 ou D2-70 Intel
- PenDrive

**5. TESTE DE AUTOAVALIAÇÃO**

Responda V ou F:

1. ( ) Os multiplexadores e demultiplexadores são circuitos combinacionais.

2. ( ) A tabela da verdade completa de um multiplexador com 2 terminais de seleção de dados tem 64 linhas.

3. ( ) A tabela da verdade completa de um multiplexador com 8 terminais de entrada de dados tem $2^8$ linhas.

4. ( ) A tabela da verdade completa de um demultiplexador de 3 terminais de endereçamento tem 16 linhas e 8 colunas de saída.

5. ( ) A tabela da verdade completa de um multiplexador tem exatamente o mesmo número de 0's e 1's.

6. ( ) O demultiplexador é um decodificador com um terminal de ativação. Assim sendo, os terminais de endereçamento do demultiplexador correspondem aos terminais de entrada do decodificador.

7. ( ) Continuando a questão anterior, o terminal de ativação do decodificador corresponde ao terminal de dados do demultiplexador.

8. ( ) O multiplexador também é equivalente a um codificador.

9. ( ) Um multiplexador com 1 terminal de seleção pode implementar qualquer uma das 16 funções de duas variáveis com apenas uma porta NOT adicional.

10. ( ) Um multiplexador com 4 terminais de dados pode implementar qualquer função de 4 variáveis com apenas uma porta NOT adicional.

11. ( ) Um multiplexador de 16 terminais de dados pode implementar qualquer função de 4 variáveis sem nenhuma porta adicional.