► 2.9 DECODIFICADORES E MULTIPLEXADORES

Dois componentes adicionais, o decodificador e o multiplexador, também são comumente usados como blocos construtivos de circuitos digitais, mesmo que eles próprios possam ser construídos usando portas lógicas.

Decodificadores

Um decodificador é um bloco construtivo de nível mais elevado comumente usado em circuitos digitais. Um ***decodificador***, como o nome diz, decodifica um número binário de n bits de entrada colocando exatamente uma das 2^n saídas do decodificador em 1. Por exemplo, um decodificador de duas entradas, ilustrado na Fig. 2.50, tem $2^2 = 4$ saídas, `d3`, `d2`, `d1` e `d0`. Se as duas entradas `i1i0` forem `00`, então `d0` será 1 e as demais saídas serão 0. Se `i1i0=01`, `d1` será 1. Se `i1i0=10`, `d2` será 1. Se `i1i0=11`, `d3` será 1.

O projeto interno de um decodificador é imediato. Considere um decodificador 2x4. Cada saída `d0`, `d1`, `d2` e `d3` é uma função distinta. A saída `d0` deve ser 1 apenas quando `i1=0` e `i0=0` de modo que `d0 = i1'i0'`. De modo semelhante, `d1=i1'i0`, `d2=i1i0'` e `d3=i1i0`. Assim, construímos o decodificador com uma porta AND para cada saída, ligando os valores de `i1` e `i0` ou seus complementos a cada porta, como mostrado na Fig. 2.50.

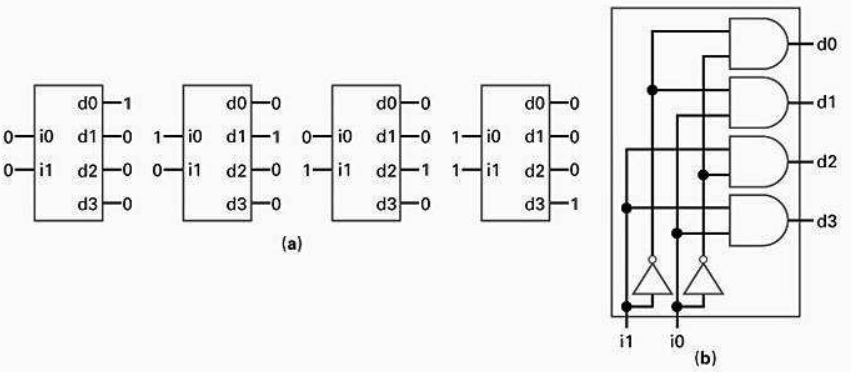


Figura 2.50 Um decodificador 2x4: (a) saídas para as combinações possíveis de entrada, (b) projeto interno.

O projeto interno de um decodificador 3x8 é semelhante: `d0=i2'i1'i0'`, `d1=i2'i1'i0`, etc.

Um decodificador freqüentemente tem uma entrada extra chamada *enable* (habilitar). Quando *enable* é 1, o decodificador atua normalmente, mas, quando *enable* é 0, o decodificador coloca todas as suas saídas em 0 – nenhuma saída é 1. O sinal de habilitação (*enable*) é útil quando algumas vezes você não quer que nenhuma das saídas seja ativada. Sem uma entrada de habilitação, uma das saídas do decodificador *deve* ser 1, porque o decodificador tem uma saída para cada um dos valores possíveis da entrada de n bits do decodificador. Na Fig. 2.44, tínhamos criado e usado um decodificador com habilitação. A Fig. 2.51 mostra o diagrama de blocos de um decodificador com habilitação.

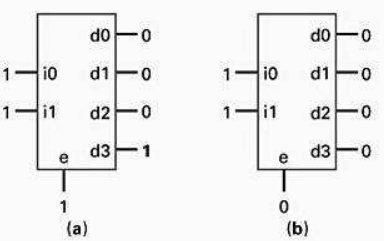


Figura 2.51 Decodificador com habilitação: (a) `e=1`: decodificação normal e (b) `e=0`: todas as saídas em 0.

Quando projetamos um sistema em particular, verificamos se parte (ou o todo) da funcionalidade do sistema poderia ser realizada com um decodificador. O uso de um decodificador diminui a quantidade de projeto lógico combinacional que precisamos executar, como você verá no Exemplo 2.30.

▶ EXEMPLO 2.29 Perguntas básicas sobre decodificadores

1. Quais seriam os valores de saída de um decodificador 2x4 quando as entradas são 00? *Resposta*: `d0=1`, `d1=0`, `d2=0` e `d3=0`.
2. Quais seriam os valores de saída de um decodificador 2x4 quando as entradas são 00? *Resposta*: `d0=1`, `d1=0`, `d2=0` e `d3=1`.
3. Quais são os valores de entrada de um decodificador 2x4 que fazem com que mais de uma das saídas estejam em 1 ao mesmo tempo? *Resposta*: Tais valores de entrada não existem. Em um dado momento, apenas uma das saídas do decodificador pode estar em 1.
4. Quais serão os valores de entrada de um decodificador se os valores de saída forem `d0=0`, `d1=1`, `d2=0` e `d3=0`. *Resposta*: Os valores de entrada devem ser `i1=0` e `i0=1`.
5. Quais serão os valores de entrada de um decodificador se os valores de saída forem `d0=1`, `d1=1`, `d2=0` e `d3=0`. *Resposta*: Esta pergunta não é válida. Em qualquer momento, um decodificador tem apenas uma saída em 1.
6. Quantas saídas teria um decodificador de cinco entradas? *Resposta*: 2^5 ou 32. ◀

▶ EXEMPLO 2.30 Display de contagem regressiva para véspera de ano novo

Um *display* de contagem regressiva para véspera de ano novo pode usar um decodificador. O *display* pode ter 60 lâmpadas dispostas em um poste. Queremos que uma lâmpada seja acesa a cada segundo (e a anterior apagada), começando com a lâmpada 59 na base do poste e terminando com a lâmpada 0 no topo. Poderemos usar um microprocessador para fazer a contagem regressiva de 59 a 0, mas provavelmente o microprocessador não terá os 60 pinos de saída que precisaríamos para controlar cada lâmpada. Ao invés disso, o nosso microprocessador poderia fornecer os números 59, 58, ..., 2, 1 e 0 em binário, por meio de uma porta de saída de seis bits (produzindo assim 111011, 111010, ..., 000010, 000001 e 000000). Poderíamos ligar esses seis bits a um decodificador de seis entradas e 64 (2^6)-saídas. A saída `d59` do decodificador acenderia a lâmpada 59, a `d58`, a lâmpada 58, etc.

Provavelmente, gostaríamos de ter um sinal de habilitação para o decodificador deste nosso exemplo, pois queremos que todas as lâmpadas estejam apagadas até começarmos a contagem regressiva. Inicialmente, o microprocessador colocará o sinal de habilitação (*enable*) em 0 de modo que todas as lâmpadas fiquem apagadas. Quando começar a contagem regressiva de 60 segundos, o microprocessador ativará o sinal *enable* com 1 e, então, colocará 59 na saída, em seguida 58 (um segundo após), 57, etc. O sistema final ficaria como o mostrado na Fig. 2.52.

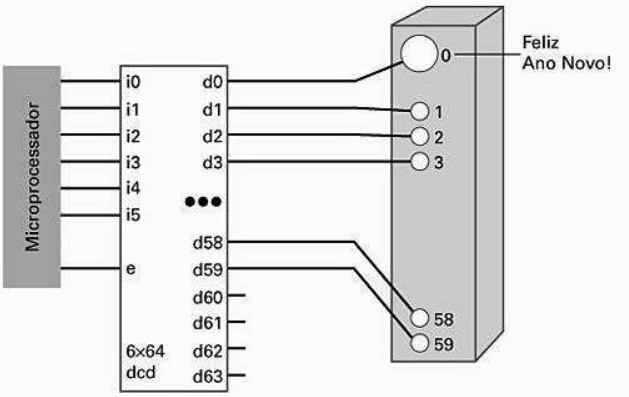


Figura 2.52 Uso de um decodificador 6x64 no interfaceamento entre um microprocessador e um poste de lâmpadas com um *display* de véspera de ano novo. Quando a contagem do minuto final começa, o microprocessador faz `e=1` (habilitação) e então coloca os valores da contagem regressiva em binário nos pinos `i5 .. i0`. Observe que o microprocessador nunca coloca os valores 60, 61, 62 ou 63 em `i5 .. i0` e, desse modo, essas saídas do decodificador ficam sem uso.

Observe que implementamos este sistema sem necessidade de projetar qualquer lógica combinacional em nível de portas básicas – simplesmente usamos um decodificador e o conectamos às entradas e saídas apropriadas. ◀

Sempre que você tiver saídas em que apenas uma delas estará em 1, tomando como base os valores das entradas que estão representando um número binário, pense em usar um decodificador.

Multiplexador (mux)

Um multiplexador ("mux", em forma abreviada) é um outro bloco construtivo de nível mais elevado usado em circuitos digitais. Um ***multiplexador*** *M*x1 tem *M* entradas de dados e 1 saída, permitindo que apenas uma das entradas seja passada para a saída. Algumas vezes, os multiplexadores são chamados de ***seletores*** porque selecionam uma das entradas para ser passada à saída.

Um multiplexador é como um aparelho de mudança de via em um parque ferroviário de manobras, que põe em conexão diversas vias de entrada com uma única via de saída, como mostrado na Fig. 2.53. A alavanca de controle do aparelho de mudança de via estabelece a conexão entre a via de entrada adequada e a via de saída. O surgimento de um trem na saída dependerá de se há um trem presente na via de entrada selecionada no momento. Em um multiplexador, o controle não é uma alavanca, mas sim entradas de seleção, as quais representam a conexão desejada em binário. Ao invés de um trem aparecer ou não na saída, um multiplexador produz um 1 ou um 0 na saída, dependendo de se a entrada selecionada tem um 1 ou um 0.

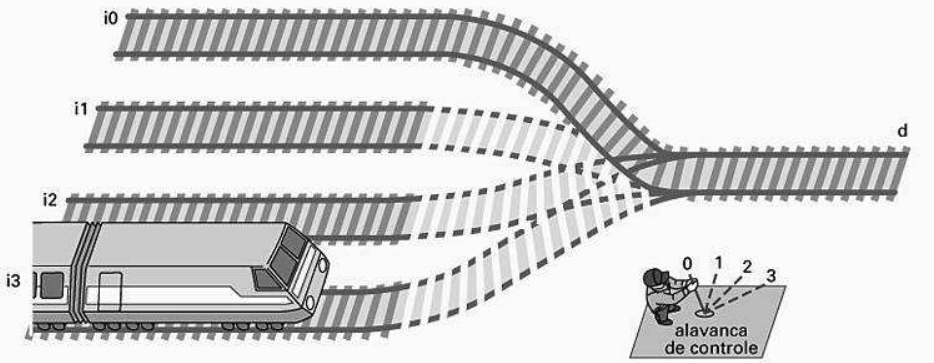


Figura 2.53 Um multiplexador é como um aparelho de mudança de via em um parque ferroviário de manobras. Ele determina qual via de entrada será conectada à única via de saída, de acordo com a alavanca de controle do aparelho de mudança de via.

Um multiplexador de duas entradas, conhecido como multiplexador 2x1, tem duas entradas de dados `i1` e `i0`, uma entrada de seleção `s0` e uma saída de dados `d`, como mostrado na Fig. 2.54. Se `s0=0`, o valor de `i0` passará para a saída. Se `s0=1`, o valor que passará será o de `i1`.

A estrutura interna de um multiplexador 2x1 está mostrada na Fig. 2.54. Quando `s0=0`, a porta AND superior gera uma saída dada por `1*i0=i0` e a porta AND inferior produz uma saída dada por `0*i1=0`. Desse modo, a porta OR fornece a saída dada por `i0+0=i0` e, assim, `i0` passa à saída, como desejado. Do mesmo modo, quando `s0=1`, a porta inferior deixa passar `i1`, ao passo que a porta superior coloca a saída em 0, resultando que a porta OR deixa passar `i1`.

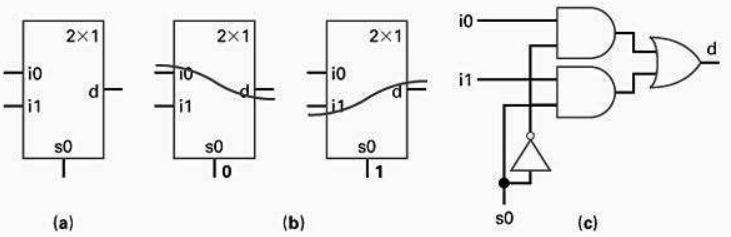


Figura 2.54 Multiplexador 2x1: (a) símbolo na forma de bloco, (b) conexões para s0=0 e s0=1, (c) estrutura interna.

Um multiplexador de quatro entradas, conhecido como multiplexador 4x1, tem quatro entradas de dados i3, i2, i1 e i0, duas entradas de seleção s1 e s0, e uma saída de dados d (um multiplexador *sempre* tem apenas uma saída de dados, não importando quantas entradas). Um diagrama de blocos de um multiplexador está mostrado na Fig. 2.55.

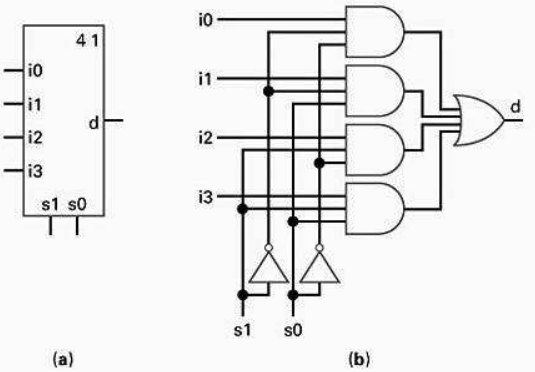


Figura 2.55 Multiplexador 4x1: (a) símbolo na forma de bloco e (b) estrutura interna.

A estrutura interna de um multiplexador 4x1 está mostrada na Fig. 2.55. Quando s1s0=00, a porta AND superior gera uma saída dada por i0*1*1=i0, a próxima porta AND gera i1*0*1=0, a próxima, i2*1*0=0, e a porta AND inferior produz uma saída dada por i3*0*0=0. A porta OR fornece i0+0+0+0=i0 e, assim, i0 passa à saída, como desejado. Do mesmo modo, quando s1s0=01, a segunda porta AND deixa passar i1, ao passo que as todas as demais portas AND produzem um 0 em suas saídas. Quando s1s0=10, a terceira porta AND deixa passar i2 e as demais portas AND produzem um 0 em suas saídas. Quando s1s0=11, a porta AND inferior deixa passar i3, e as outras portas AND produzem um 0 em suas saídas. Para qualquer valor de s1s0, apenas uma porta AND terá dois 1s em suas entradas de seleção, deixando passar assim o valor presente em sua entrada de dados: as demais portas AND terão no mínimo um 0 em suas entradas de seleção e, conseqüentemente, produzirão um 0 em suas saídas.

Um multiplexador 8x1 tem oito entradas de dados i7...i0, três entradas de seleção s2, s1 e s0, e uma saída de dados. De forma mais genérica, um multiplexador *M*x1, tem *M* entradas de dados, $\log_2(M)$ entradas de seleção e uma saída de dados. Lembre-se de que um multiplexador sempre tem somente uma saída.

▶ EXEMPLO 2.31 Perguntas básicas sobre multiplexadores

Assuma que, no momento, as quatro entradas de um multiplexador 4x1 têm os seguintes valores: i0=1, i1=1, i2=0 e i3=0. Qual será o valor na saída d de um multiplexador para os seguintes valores de entrada de seleção?

1. `s1s0=01`. *Resposta*: Como `s1s0=01` deixa passar a entrada `i1` para `d`, então `d` terá o valor de `i1` que é 1, no momento.
2. `s1s0=01`. *Resposta*: Essa configuração de valores de entrada da linha de seleção deixa passar `i3`, então `d` terá o valor de `i3` que é 0, no momento.
3. Quantas entradas de seleção devem estar presentes em um multiplexador 16x1? *Resposta*: Quatro entradas de seleção são necessárias para indicar qual das 16 entradas deverá ser passada para a saída porque $\log_2(16)=4$.
4. Quantas entradas de seleção há em um multiplexador 4x2? *Resposta*: A pergunta não é válida – não existe nada como um multiplexador 4x2. Um multiplexador tem exatamente uma saída.
5. Quantas entradas há em um multiplexador que tem cinco entradas de seleção? *Resposta*: Cinco entradas de seleção podem indicar uma única entrada entre $2^5=32$ entradas. Essa entrada deverá ser passada à saída. ◀

► EXEMPLO 2.32 Sistema para exibir o voto de um prefeito usando multiplexador

Considere uma pequena cidade com um prefeito muito impopular. Durante as assembléias municipais*, o moderador da reunião apresenta quatro proposições ao prefeito, que então vota cada uma delas (aprovando ou rejeitando). Seguidamente, logo após o prefeito votar, os cidadãos passam a vaiá-lo e a gritar palavras impróprias – independentemente de como ele votou. Depois de muito abuso desse tipo, o prefeito construiu um sistema digital simples (acontece que o prefeito tinha feito um curso de projeto digital), mostrado na Fig. 2.56. Ele providenciou quatro chaves para o seu próprio uso que podem ser posicionadas em cima ou em baixo, dando uma saída 1 ou 0, respectivamente. Durante as assembléias, quando chega o momento de votar a primeira proposição, ele leva a primeira chave para cima (aprovação) ou para baixo (rejeição) – mas ninguém mais consegue ver a posição da chave. Quando chega o momento de votar a segunda proposição, ele leva a chave para cima ou para baixo. E, assim por diante. Depois de votar todas as proposições, ele sai da assembléia e vai tomar um café. Sem a presença do prefeito, o moderador liga um grande painel luminoso de cor verde ou vermelha. Quando o sinal de entrada do painel é 0, o painel fica vermelho. Quando a entrada é 1, o painel fica verde. O moderador controla duas chaves com as quais pode escolher e conectar qualquer uma das saídas das chaves, usadas pelo prefeito, com o painel luminoso. Desse modo, o moderador usa todas as configurações das chaves, começando com a configuração 00 (e dizendo "Nesta proposição, o voto do prefeito foi ..."), então 01, 10 e finalmente 11. Em cada configuração, o painel fica aceso com a cor verde ou vermelha, de acordo com as posições das chaves do prefeito. O sistema pode ser facilmente implementado usando um multiplexador 4x1, como mostrado na Fig. 2.56. ◀

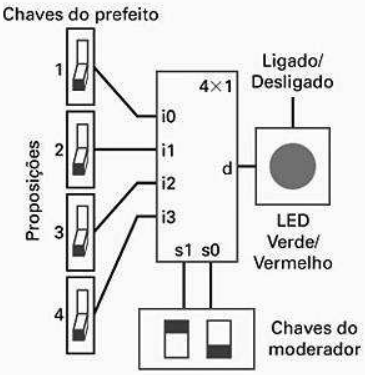


Figura 2.56 Implementação com multiplexador 4x1 de um sistema para exibir o voto de um prefeito.

Multiplexador Mx1 de *N* bits

Os multiplexadores freqüentemente são usados para seletivamente deixar passar não só bits isolados, mas também itens com *N* bits de dados. Por exemplo, um conjunto de entradas `A` pode consistir em quatro bits `a3`, `a2`, `a1` e `a0`, e um outro conjunto de entradas `B` também pode consistir em quatro bits `b3`, `b2`, `b1` e `b0`. Queremos multiplexar essas entradas conduzindo-as

* N. de T: Nos Estados Unidos, há governos municipais em que as decisões importantes são tomadas por todos os cidadãos reunidos em uma assembléia (*town meeting*).

a uma saída de quatro bits C consistindo em c3, c2, c1 e c0. A Fig. 2.57(a) mostra como obter essa multiplexação usando quatro multiplexadores 2x1.

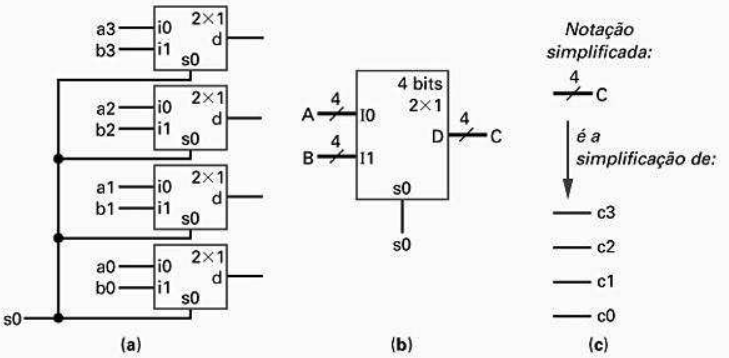


Figura 2.57 Multiplexador 2x1 de quatro bits: (a) estrutura interna usando quatro multiplexadores 2x1 para selecionar entre os itens A ou B de quatro bits de dados e (b) símbolo para diagrama de blocos de um multiplexador 2x1 de quatro bits. (c) O diagrama de blocos usa uma notação simplificada comum, consistindo em uma linha cheia com um traço inclinado e o número 4, para representar quatro fios simples.

Como a multiplexação de dados é muito usada, um outro bloco construtivo comum é um multiplexador Mx1 com N bits de largura de dados. Assim, em nosso exemplo, usaríamos um multiplexador 2x1 de 4 bits. Não fique confuso, contudo – um multiplexador Mx1 de N bits é na realidade simplesmente o mesmo que N multiplexadores Mx1 separados, mas que compartilham as mesmas entradas de seleção. A Fig. 2. 57(b) mostra o símbolo de um multiplexador 2x1 de 4 bits.

▶ EXEMPLO 2.33 Display multiplexado de automóvel colocado acima do espelho retrovisor

Alguns carros vêm com um *display* colocado acima do espelho retrovisor, como mostrado na Fig. 2.58. O motorista pode pressionar um botão chamado `modo` para mostrar a temperatura exterior, o consumo médio do carro em quilômetros por litro, o consumo instantâneo em quilômetros por litro ou a distância restante aproximada que ainda pode ser percorrida antes que acabe o combustível. Assuma que o computador central do carro envia os dados ao *display* na forma de quatro números binários de oito bits, `T` (a temperatura), `M` (consumo médio em km/litro), `I` (consumo instantâneo em km/litro) e `Q` (quilômetros restantes). `T` consiste em oito bits: `t7`, `t6`, `t5`, `t4`, `t3`, `t2`, `t1` e `t0`. O mesmo ocorre com `M`, `I` e `Q`. Assuma que o sistema de *display* tem duas entradas adicionais `x` e `y`, que sempre mudam de valor seguindo a seqüência seguinte – `00`, `01`, `10` e `11` – a cada vez que o botão de `modo` é apertado (veremos em um capítulo posterior como criar tal seqüência). Quando `xy = 00`, queremos exibir `T`, quando `xy=01`, queremos exibir `M`, quando `xy=10`, queremos exibir `I` e quando `xy=11`, queremos exibir `Q`. Assuma que as saídas `D` vão a um *display* que sabe como converter esse número binário `D` de oito bits em um número que pode ser exibido e lido pelas pessoas, como na Fig. 2.58*.

Figura 2.58 *Display* acima do retrovisor.

* N. de T: No caso, está sendo mostrada a temperatura exterior (*outside*) de 87 graus Fahrenheit (ou 31 graus centígrados).

Podemos projetar o sistema de *display* usando oito multiplexadores 4x1, como mostrado na Fig. 2.59.

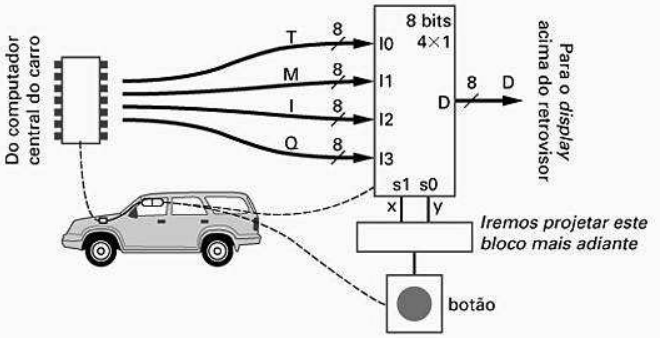


Figura 2.59 *Display* colocado acima do retrovisor, usando um multiplexador 4x1 de oito bits.

Observe quantos fios devem ser estendidos desde o computador central do carro, que pode estar localizado abaixo do capô, até o *display* colocado acima do retrovisor – 8 * 4 = 32 fios. Isso é muito fio. Em um capítulo posterior, veremos como reduzir o número de fios. ◄

Observe no exemplo anterior como um projeto pode ser simples, quando utilizamos blocos construtivos de nível mais elevado. Se tivéssemos de usar multiplexadores comuns de 4x1, teríamos oito deles e muitos fios desenhados. Se tivéssemos de usar portas, teríamos 40 delas. Naturalmente, por baixo do nosso projeto simples da Fig. 2.59 há, de fato, oito multiplexadores 4x1, e por baixo desses há 40 portas e, ainda, por baixo dessas há grandes quantidades de transistores. Vemos que o uso de blocos construtivos de nível mais elevado torna a nossa tarefa de projeto muito mais tratável.

► 2.10 CONSIDERAÇÕES ADICIONAIS

Captura de esquemáticos e simulação

Ao projetar um circuito, como saber se estamos projetando corretamente? Talvez tenhamos criado a tabela-verdade com algum erro, colocando um 0 em alguma casa da coluna de saída quando deveríamos ter posto um 1; ou talvez tenhamos escrito um mintermo errado, escrevendo xyz quando deveríamos ter escrito `xyz'`. Por exemplo, considere o contador do número de 1s do Exemplo 2.25. Criamos uma tabela-verdade, a seguir criamos equações e finalmente, um circuito. O circuito está correto?

Um método de verificarmos o nosso trabalho é fazendo a chamada engenharia reversa da função a partir do circuito – começando com o circuito, poderíamos convertê-lo em equações e, em seguida, essas em uma tabela-verdade. Se obtivermos a mesma tabela-verdade original, então o circuito deverá estar correto. Entretanto, algumas vezes, começamos com uma equação, ao invés de uma tabela-verdade, como no Exemplo 2.24. Poderemos fazer a engenharia reversa de um circuito obtendo uma equação, mas essa equação pode ser diferente da original, especialmente se tivermos manipulado algebricamente a equação original durante o projeto do circuito. Além disso, para verificar se duas equações são equivalentes, pode ser necessário que as equações sejam convertidas para a forma canônica (soma de mintermos). Se a nossa função tiver um grande número de entradas, isso poderá produzir equações enormes.

De fato, mesmo se não fizermos nenhum erro quando estivermos convertendo o que está em nossa mente, a respeito da função desejada, para a forma de tabela-verdade ou equação, como saberemos se o nosso entendimento estava correto?

Um método comumente usado para verificar se um circuito funciona como esperávamos é chamado de simulação. A ***simulação*** de um circuito é o processo de se fornecer valores de entradas ao circuito e executar um programa de computador que calcula a saída do circuito com essas entradas. Poderemos então conferir se a saída está de acordo com o que esperamos. O programa de computador que executa a simulação é chamado de ***simulador***.

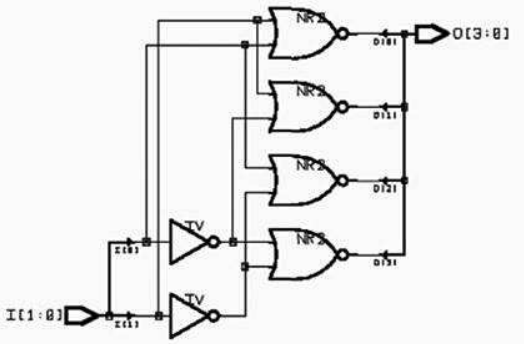


Figura 2.60 Imagem de tela de uma ferramenta comercial de captura de esquemático.

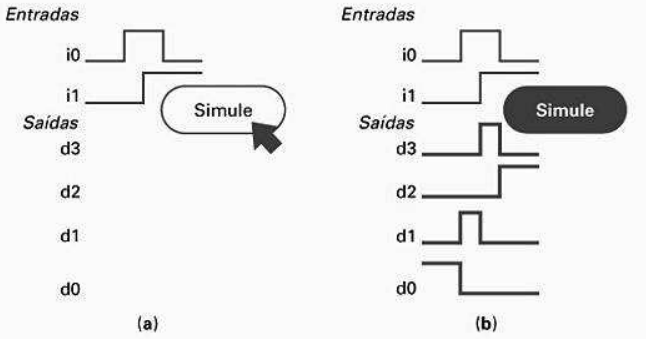


Figura 2.61 Simulação: (a) começa com nós para definir os sinais das entradas ao longo do tempo, (b) gera automaticamente as formas de onda de saída quando solicitamos que o simulador faça a simulação do circuito.

Para usar a simulação na verificação de um circuito, deveremos descrevê-lo usando um método que permita a leitura do circuito pelos programas de computador. Um método para se descrever um circuito é desenhando-o por meio de uma ferramenta de captura de esquemático. Uma ***ferramenta de captura de esquemático*** permite que um usuário coloque portas lógicas em uma tela de computador e desenhe fios para conectar essas portas. A ferramenta permite aos usuários guardar seus desenhos de circuitos na forma de arquivos de computador. Todos os desenhos dos circuitos deste capítulo são exemplos de esquemáticos – por exemplo, o desenho do circuito da Fig. 2.50(b), representando um decodificador 2x4, é um exemplo de esquemático. A Fig. 2.60 mostra um esquemático do mesmo projeto, que foi desenhado usando uma ferramenta comercial bem conhecida de captura de esquemático. A captura de esquemáticos é usada não só para capturar os circuitos que serão usados pelas ferramentas de simulação, como também pelas ferramentas que mapeiam nossos circuitos para implementações físicas, como será discutido no Capítulo 7.

Depois de criar um circuito por meio da captura de esquemático, devemos fornecer ao simulador um conjunto de entradas, para as quais desejamos verificar se as saídas são as corretas. Uma das maneiras de se fornecer as entradas é desenhando as formas de onda das entradas do circuito. Uma ***forma de onda*** de entrada é uma linha que vai da esquerda para a direita, representando o valor da entrada à medida que o tempo avança para a direita. Em momentos diferentes, a linha está alta para representar 1 ou baixa para representar 0, como mostrado na Fig. 2.61(a). Depois de estarmos satisfeitos com as nossas formas de onda de entrada, instruímos o simulador a simular o nosso circuito usando as formas de onda de entrada dadas. O simulador determina quais são as saídas do circuito para cada combinação exclusiva de entradas e gera as formas de onda das saídas, como mostrado na Fig. 2.61(b). Então, podemos conferir se as formas de onda de saída estão em concordância com os valores de saída que estávamos esperando para cada entrada. Essa verificação pode ser feita visualmente ou, então, fornecendo certas expressões de verificação (freqüentemente chamadas de asserções) ao simulador.

A simulação ainda não garante que o nosso circuito está correto, mas aumenta a nossa *confiança*.

Comportamento não ideal das portas – atraso de tempo

Idealmente, as saídas das portas lógicas deveriam mudar de valor instantaneamente em respostas a alterações em suas entradas. Nos diagramas de tempo anteriores deste capítulo, assumiu-se que as portas ideais tinham esse atraso nulo, como mostrado novamente na Fig. 2.62(a) para o caso de uma porta OR. Infelizmente, as saídas das portas reais não mudam de valor imediatamente mas, pelo contrário, após um breve intervalo de tempo. Afinal de contas, mesmo o mais veloz dos automóveis não pode acelerar de 0 a 100 km/h em 0 segundo. O atraso nas portas é devido em parte ao fato de que os transistores não são chaveados instantaneamente do estado de não condução para o de condução (ou vice-versa) – por exemplo, leva algum tempo para que os elétrons se acumulem no canal de um transistor nMOS. Além disso, a corrente elétrica viaja a uma grande velocidade, a qual, embora muito elevada, não é ainda infinitamente rápida. Além disso, os fios não são perfeitos e podem retardar a corrente elétrica devido a características "parasíticas" como capacitância e indutância. O diagrama de tempo da Fig. 2.62(b) ilustra como a saída de uma porta real é ligeiramente diferente da ideal depois de ocorrerem algumas alterações nas entradas. Os atrasos nas portas CMOS modernas podem levar menos de um nanossegundo para responder às alterações – extremamente rápidos, mas ainda não nulos.

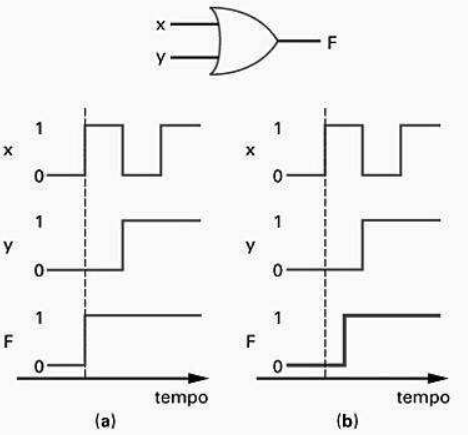


Figura 2.62 Diagrama de tempo de uma porta OR: (a) sem atraso de porta, (b) com atraso.

Demultiplexadores e codificadores

Dois componentes adicionais, demultiplexadores e codificadores, também podem ser considerados blocos construtivos combinacionais. No entanto, esses componentes são bem menos usados do que os seus opostos, os multiplexadores e os decodificadores. No entanto, por uma questão de completude, iremos introduzir brevemente aqui esses componentes adicionais. Você poderá notar que ao longo deste livro os demultiplexadores e codificadores não aparecem em muitos exemplos, se é que aparecem.

Demultiplexador

Um demultiplexador tem aproximadamente o comportamento oposto ao de um multiplexador. De forma específica, um ***demultiplexador*** 1xM tem uma entrada de dados e, com base nos valores de $\log_2(M)$ linhas de seleção, passa essa entrada para uma das M saídas. As outras saídas permanecem em 0.

Codificador

Um codificador tem o comportamento oposto ao de um decodificador. De forma específica, um decodificador $\log_2(n)$ tem n entradas e $\log_2(m)$ saídas. Das n entradas, assume-se que apenas uma delas, em qualquer momento dado, é 1 (esse seria o caso se a entrada consistisse em uma chave deslizante ou rotativa com n posições possíveis, por exemplo). O codificador coloca um valor binário nas $\log_2(n)$ saídas indicando qual das n entradas é 1. Por exemplo, um codificador 4x2 tem quatro entradas `d3`, `d2`, `d1` e `d0` e duas saídas `e1` e `e0`. Quando a entrada

é 0001, a saída é 00. Para 0010, a saída é 01. Para 0100, a saída é 10 e para 1000, a saída é 11. Em outras palavras, d0=1 resulta na saída de 0 em binário, d1=1 resulta na saída de 1 em binário, d2=1 resulta na saída de 2 em binário e d3=1 resulta na saída de 3 em binário.

Um ***codificador de prioridade*** tem comportamento similar, mas lida com situações em que mais de uma entrada são 1 ao mesmo tempo. Um *codificador de prioridade* dá prioridade à entrada mais elevada que tem um 1 e fornece o valor binário dessa entrada. Por exemplo, se um *codificador de prioridade* 4x2 tiver as entradas d3 e d1 iguais a 1 (de modo que as entradas são 1010), o *codificador de prioridade* dará prioridade a d3 e, portanto, produzirá a saída 11.

► 2.11 OTIMIZAÇÕES E TRADEOFFS EM LÓGICA COMBINACIONAL (VEJA A SEÇÃO 6.2)

As seções anteriores deste capítulo descreveram como criar os circuitos combinacionais básicos. Neste livro, o conteúdo desta seção, Seção 2.11, está de fato na Seção 6.2, onde se descreve como melhorar esses circuitos (menores, mais rápidos, etc.) – ou seja, como fazer otimizações e *tradeoffs*. Uma maneira de se usar este livro é tratando das otimizações e *tradeoffs* logo após a introdução do projeto lógico combinacional, ou seja, agora (como Seção 2.11). Um uso alternativo é tratar das otimizações e *tradeoffs* mais tarde (como Seção 6.2), após ter introduzido também o projeto seqüencial básico, os componentes de bloco operacional e o projeto em nível de transferência entre registradores – ou seja, após os Capítulos 3, 4 e 5.

► 2.12 DESCRIÇÃO DE LÓGICA COMBINACIONAL USANDO LINGUAGENS DE DESCRIÇÃO DE HARDWARE (VEJA A SEÇÃO 9.2)

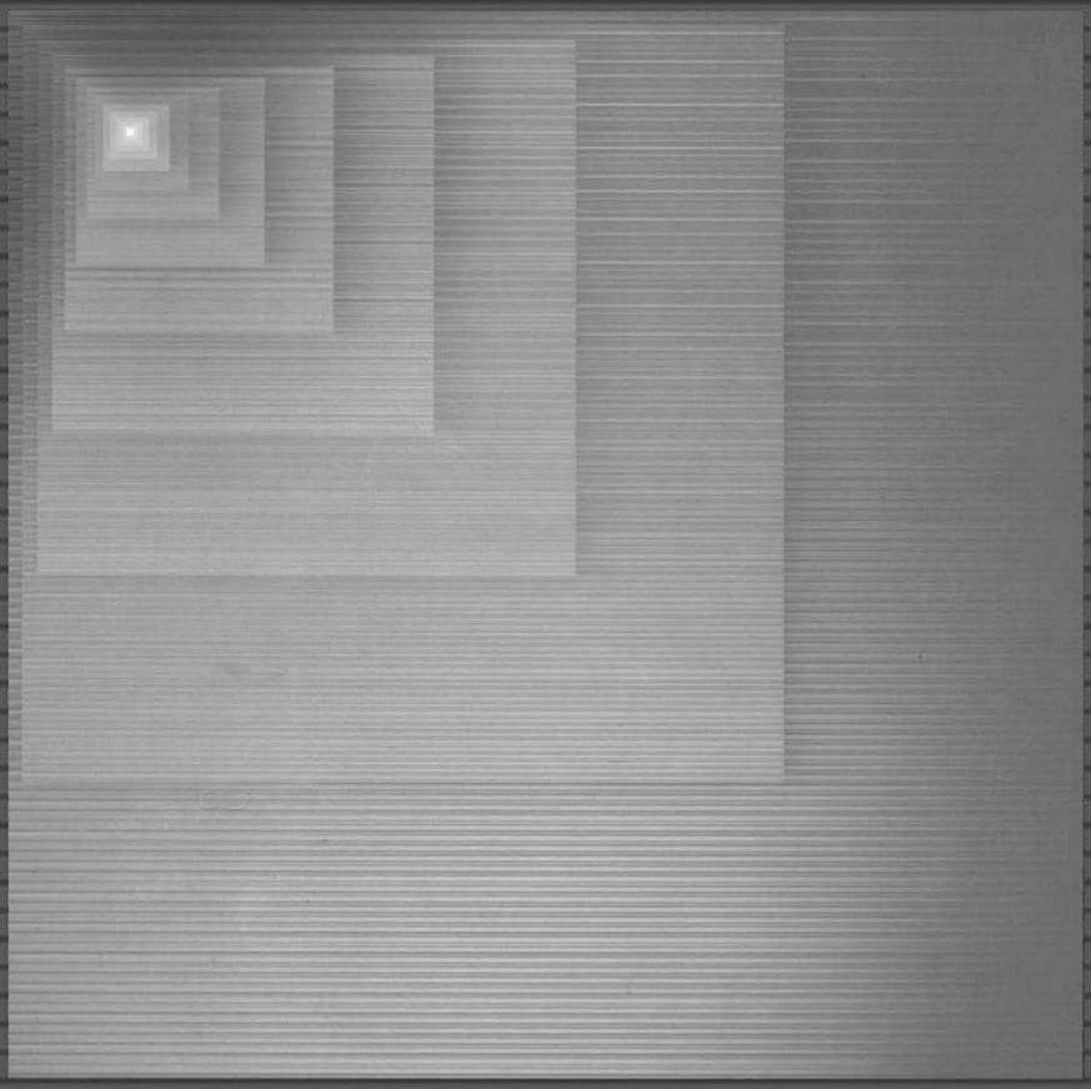
As linguagens de descrição de hardware (HDLs) permitem aos projetistas descrever seus circuitos usando uma linguagem textual ao invés de usar os desenhos dos circuitos. Esta seção, Seção 2.12, introduz o uso de HDLs para descrever a lógica combinacional. Neste livro, o conteúdo desta seção está fisicamente na Seção 9.2. Uma maneira de se usar este livro é introduzir as HDLs agora (como Seção 2.12), imediatamente após a lógica combinacional básica ter sido introduzida. Um uso alternativo é introduzir as HDLs mais tarde (como Seção 9.2), após ter obtido domínio dos projetos combinacional e seqüencial básicos e em nível de transferência entre registradores.

► 2.13 RESUMO DO CAPÍTULO

A Seção 2.1 introduziu a idéia de se usar um circuito customizado para se implementar um comportamento desejado de sistema e uma lógica combinacional. Esta última é definida na forma de um circuito digital, cujas saídas são uma função das entradas atuais do circuito. A Seção 2.2 forneceu uma breve história das chaves digitais, começando com os relés da década de 1930 e chegando aos transistores CMOS de hoje. A marca principal desse período é a velocidade surpreendente com que o tamanho e o atraso das chaves vem encolhendo continuamente por diversas décadas, chegando aos ICs capazes de alojar um bilhão ou mais de transistores. A Seção 2.3 descreveu o comportamento básico de um transistor CMOS – com apenas o suficiente de informação para elucidar o mistério de como funcionam os transistores. A Seção 2.4 introduziu três blocos construtivos fundamentais que são usados na construção de circuitos digitais – as portas AND, as portas OR e as portas NOT (inversores), com as quais é muito mais fácil de se trabalhar do que com transistores. A Seção 2.5 mostrou como a álgebra booleana poderia ser usada para representar circuitos construídos a partir de portas AND, OR e NOT, permitindo-nos construir e manipular circuitos usando a matemática – um con-

Sistemas Digitais

projeto, otimização e HDLs



FRANK VAHID

Bookman®