Organização de Computadores Digitais - 5954008

2. Lógica Digital

Prof. Luiz Otavio Murta Jr

Local: Depto. de Computação e Matemática (FFCLRP/USP)

Principais Tópicos

2. Revisão de Lógica Digital

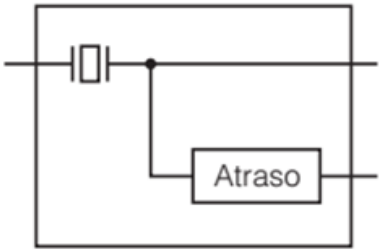
- 2.1. Álgebra Booleana
- 2.2. Portas Lógicas
- 2.3. Circuitos Combinatórios
- 2.4. Circuitos Sequenciais

2.3.4. *Clocks*

- Em geral, as frequências de pulso estão entre 100 MHz e 4 GHz, correspondendo a ciclos de *clock* de 10 nanossegundos a 250 picossegundos.
- Para conseguir alta precisão, a frequência de *clock* normalmente é controlada por um oscilador de cristal.
- Muitos eventos podem ocorrer dentro de um computador durante um único ciclo de *clock*.
- Se eles devem ocorrer em uma ordem específica, o ciclo de *clock* deve ser dividido em subciclos.

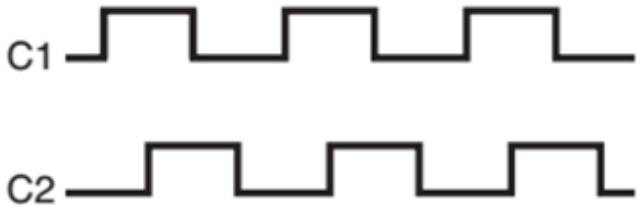
2.3.4. *Clocks*

- Uma maneira comum de prover resolução superior à do *clock* básico é aproveitar a linha de *clock* primária e inserir um circuito com um atraso conhecido,
  - gerando assim um sinal de *clock* secundário deslocado em certa fase em relação ao primeiro, conforme mostra a figura (a).



(a)

- O diagrama de temporização da figura (b) dá quatro referências de tempo para eventos discretos:
  1. Fase ascendente de C1.
  2. Fase descendente de C1.
  3. Fase ascendente de C2.
  4. Fase descendente de C2.



(b)

2.3.4. *Clocks*

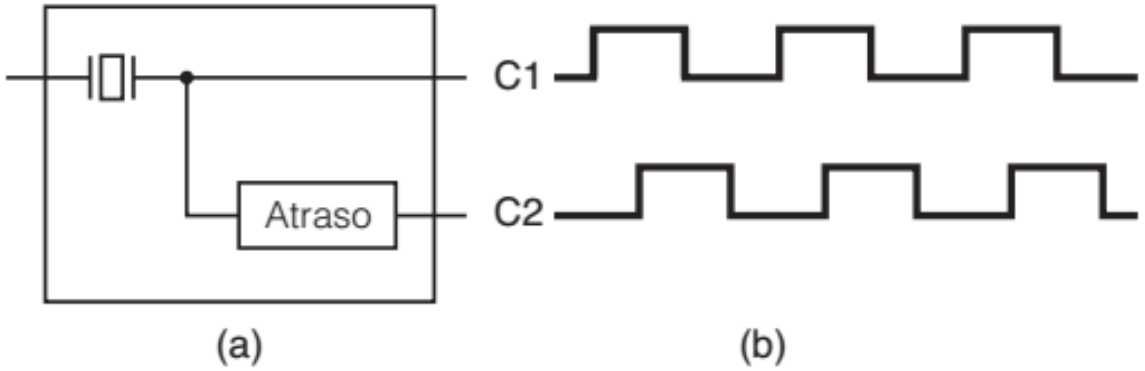
- Vinculando diferentes eventos às várias fases, pode-se conseguir a sequência requerida.
- Se forem necessárias mais do que quatro referências de tempo dentro de um ciclo de *clock*, podem-se puxar mais linhas da linha primária, com diferentes atrasos, se for preciso.
- Em alguns circuitos, estamos interessados em intervalos de tempo em vez de instantes discretos de tempo.
- Por exemplo, pode-se permitir que algum evento aconteça toda vez que C1 estiver alto, em vez de exatamente na fase ascendente.
- Outro evento só poderá acontecer quando C2 estiver alto.

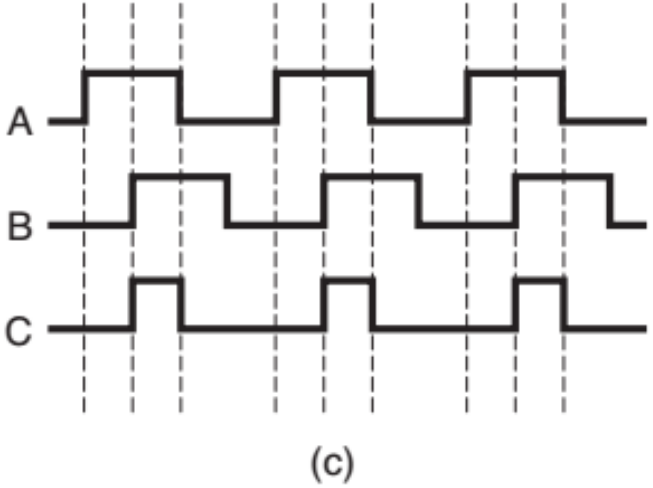
2.3.4. *Clocks*

- Se forem necessários mais de dois intervalos diferentes, podem ser instaladas mais linhas de *clock* ou pode-se fazer com que os estados altos dos dois *clocks* se sobreponham parcialmente no tempo.
- No último caso, podem-se distinguir quatro intervalos distintos: $\overline{C1}$ AND $\overline{C2}$, $\overline{C1}$ AND C2, C1 AND $\overline{C2}$ e C1 AND C2.
- A propósito, clocks são simétricos, com o tempo gasto no estado alto igual ao tempo gasto no estado baixo, como mostra a figura (b).
- Para gerar um trem de pulsos assimétrico, o *clock* básico é deslocado usando um circuito de atraso e efetuando uma operação AND com o sinal original, como mostra a figura (c) como C.

2.3.4. *Clocks*

- (a) Um *clock*. (b) Diagrama de temporização para o *clock*. (c) Geração de um *clock* assimétrico.



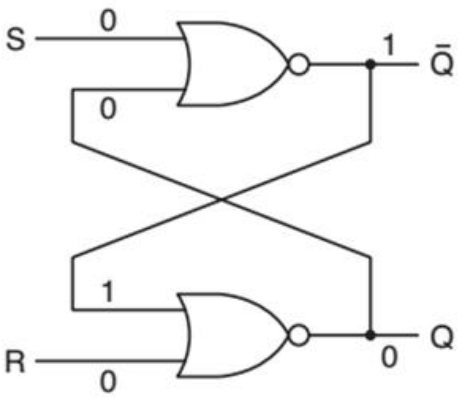


2.3.5. Memórias

- Um componente essencial de todo computador é sua memória.
- Sem ela não poderiam existir os computadores que conhecemos.
- A memória é usada para armazenar instruções a serem executadas e dados.
- Para criar uma memória de 1 bit ("*latch*"), precisamos de um circuito que "se lembre", de algum modo, de valores de entrada anteriores.
- Tal circuito pode ser construído com base em duas portas nor, como ilustrado na figura (a).

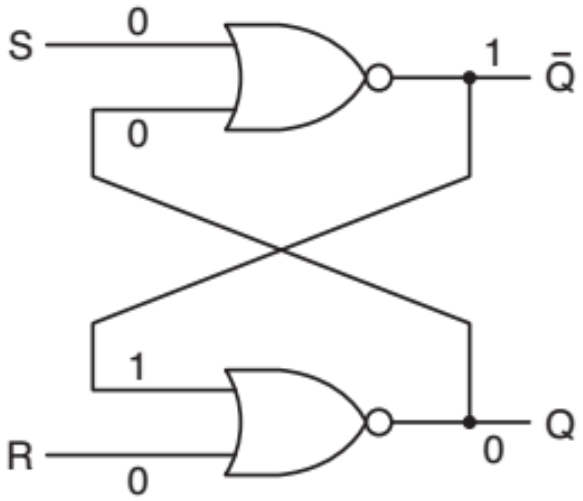
2.3.5. Memórias

- Circuitos análogos podem ser construídos com portas nand, porém, não vamos mais mencioná-los porque são conceitualmente idênticos às versões nor.
- O circuito da figura (a) é denominado *latch* SR.
- Ele tem duas entradas, S, para ativar (*setting*) o *latch*, e R, para restaurá-lo (*resetting*), isto é liberá-lo.
- O circuito também tem duas saídas, Q e Q, que são complementares, como veremos em breve.
- Ao contrário de um circuito combinacional, as saídas do *latch* não são exclusivamente determinadas pelas entradas atuais.

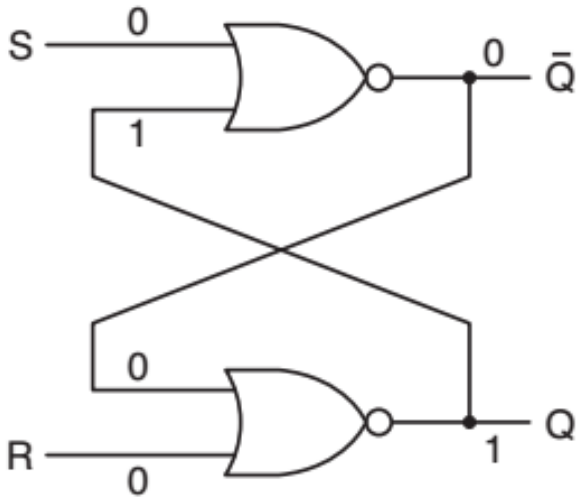


2.3.5. Memórias

- (a) *Latch* nor no estado 0. (b) *Latch* nor no estado 1. (c) Tabela verdade para nor.



(a)



(b)

A	B	NOR
0	0	1
0	1	0
1	0	0
1	1	0

(c)

2.3.5. Memórias

- Suponhamos que ambos, S e R, sejam 0, o que é verdade na maior parte do tempo.
- Apenas para polemizar, vamos supor que $\overline{Q} = 0$.
- Como $\overline{Q}$ é realimentado para a porta nor superior, ambas as suas entradas são 0,
  - portanto, sua saída, Q, é 1. O 1 é realimentado para a porta inferior que,
  - então, tem entradas 1 e 0, resultando em Q = 0.
- Esse estado é no mínimo coerente e está retratado na figura (a).
- Imaginemos que Q não seja 0, mas 1, com R e S ainda 0.

2.3.5. Memórias

- A porta superior tem entradas de 0 e 1, e uma saída, $\overline{Q}$, de 0, que é realimentada para a porta inferior.
- Esse estado, mostrado na figura (b), também é coerente.
- Um estado com as duas saídas iguais a 0 é incoerente,
  - porque força ambas as portas a ter dois 0 como entrada,
  - o que, se fosse verdade, produziria 1, não 0, como saída.
- De modo semelhante, é impossível ter ambas as saídas iguais a 1, porque isso forçaria as entradas a 0 e 1, o que resultaria 0, não 1.
- Nossa conclusão é simples:
  - para R = S = 0, o *latch* tem dois estados estáveis,
  - que denominaremos 0 e 1, dependendo de $\overline{Q}$.

2.3.5. Memórias

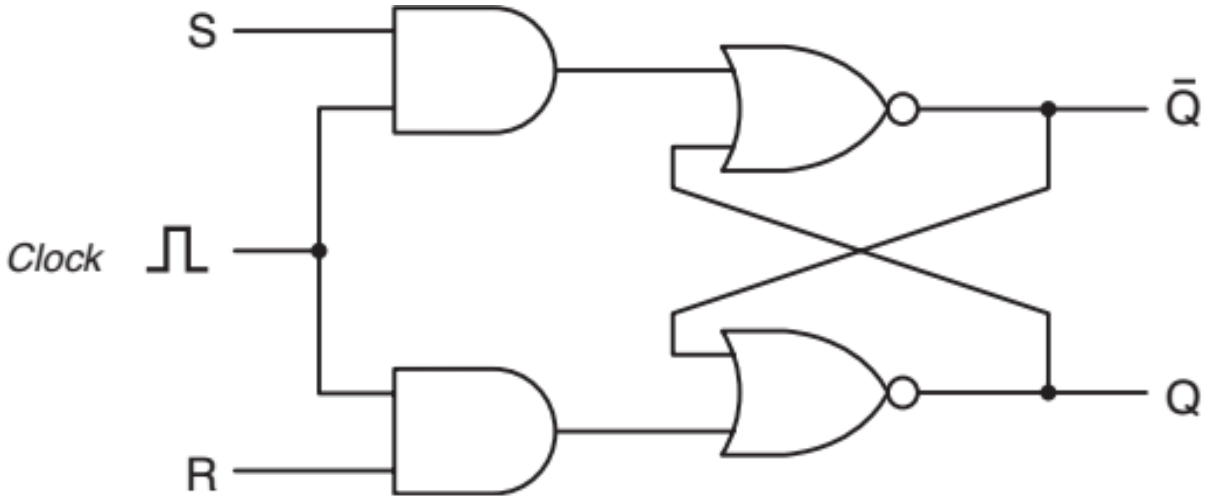
- Agora, vamos examinar o efeito das entradas sobre o estado do *latch*.
- Suponha que S se torna 1 enquanto $\overline{Q} = 0$.
- Então, as entradas para a porta superior são 1 e 0, forçando a saída $\overline{Q}$ a 0.
- Essa mudança faz ambas as entradas para a porta inferior serem 0, forçando a saída para 1.
- Portanto, ativar S (isto é, fazer com que seja 1) muda o estado de 0 para 1.
- Definir R em 1 quando o *latch* está no estado 0 não tem efeito algum porque a saída da porta NOR inferior é 0 para entradas de 10 e entradas de 11.

2.3.5. Memórias

- Usando raciocínio semelhante, é fácil ver que definir S em 1 quando em estado Q = 1 não tem efeito algum, mas definir R leva o latch ao estado $\overline{Q}$ = 0.
- Resumindo, quando S é definido em 1 momentaneamente, o latch acaba no estado $\overline{Q}$ = 1, pouco importando seu estado anterior.
- Da mesma maneira, definir R em 1 momentaneamente força o latch ao estado Q = 0.
- O circuito “se lembra” se foi S ou R definido por último. Usando essa propriedade podemos construir memórias de computadores.

2.3.5. Memórias

- Muitas vezes é conveniente impedir que o *latch* mude de estado, a não ser em certos momentos especificados.
- Para atingir esse objetivo, fazemos uma ligeira modificação no circuito básico, conforme mostra a figura, para obter um ***latch* SR com *clock***.



2.3.5. Memórias

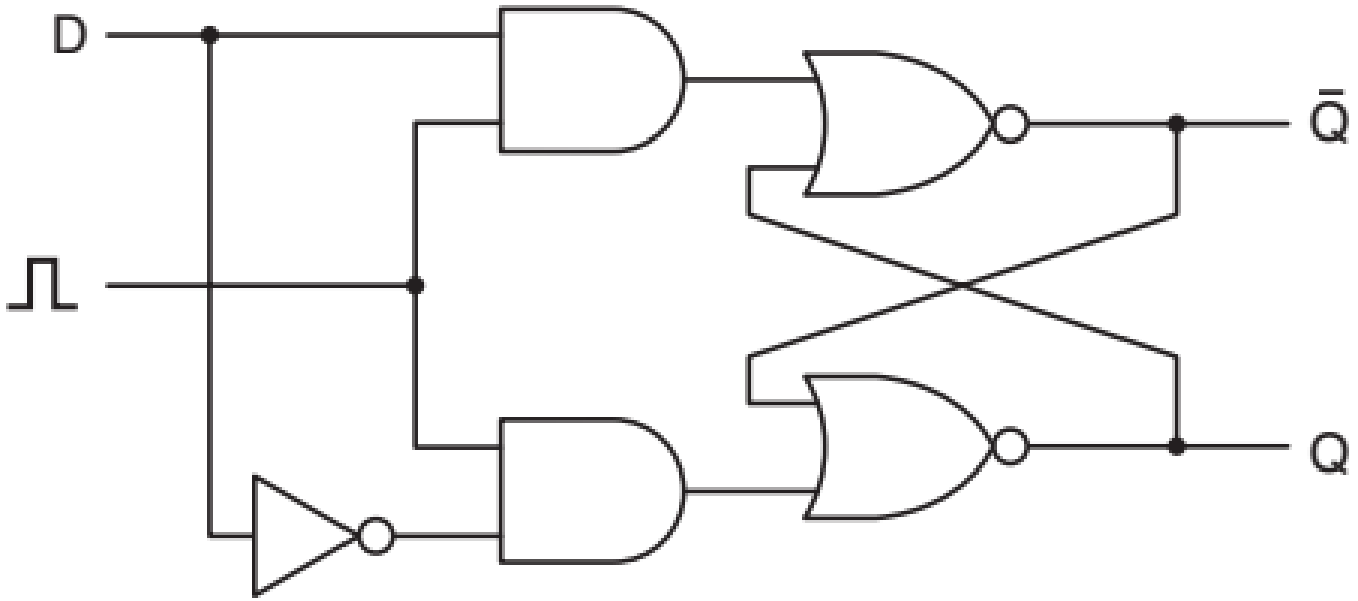
- Esse circuito tem uma entrada adicional, o *clock*, que em geral é 0.
- Com o *clock* em 0, ambas as portas and geram saída 0, independentemente de ser S e R, e o *latch* não muda de estado.
- Quando o *clock* é 1, o efeito das portas and desaparece e o *latch* se torna sensível a S e R.
- Apesar de seu nome, o sinal do *clock* não precisa ser gerado por um *clock*.
- Os termos *enable* e *strobe* também são muito usados para indicar que a entrada do *clock* é 1;
  - isto é, o circuito é sensível ao estado de S e R.

2.3.5. Memórias

- Até aqui evitamos falar no que acontece quando ambos, S e R, são 1, por uma boa razão:
  - o circuito se torna não determinístico quando ambos,
  - R e S, finalmente retornam a 0.
- O único estado coerente para S = R = 1 é $\overline{Q}$ = Q = 0;
  - porém, assim que ambas as entradas voltam para 0,
  - o latch deve saltar para um de seus dois estados estáveis.
- Se quaisquer das entradas cair para 0 antes da outra, a que permanecer em 1 por mais tempo vence,
  - porque, quando apenas uma entrada for 1, ela força o estado.
- Se ambas as entradas voltarem a 0 ao mesmo tempo (o que é muito improvável), o *latch* salta aleatoriamente para um de seus estados estáveis.

2.3.5. Memórias

- Uma boa maneira de resolver a instabilidade do *latch* SR (causada quando S = R = 1) é evitar que ela ocorra.
- A figura apresenta um circuito de *latch* com somente uma entrada, D.



2.3.5. Memórias

- Como a entrada para a porta and inferior é sempre o complemento da entrada para a superior, nunca ocorre o problema de ambas as entradas serem 1.
  - Quando D = 1 e o *clock* for 1, o *latch* é levado ao estado Q = 1.
  - Quando D = 0 e o *clock* for 1, ele é levado ao estado Q = 0.
  - Em outras palavras, quando o clock for 1, o valor corrente de D é lido e armazenado no *latch*.
- Esse circuito, denominado ***latch* D com *clock***, é uma verdadeira memória de 1 bit.
- O valor armazenado sempre estará disponível em Q.
- Para carregar o valor atual de D na memória, um pulso positivo é colocado na linha do *clock*.

2.3.5. Memórias

- Em muitos circuitos é necessário ler o valor em determinada linha em dado instante, e armazená-lo.
- Nessa variante, denominada *flip-flop*, a transição de estado não ocorre quando o *clock* é 1, mas durante a transição de 0 para 1 (borda ascendente), ou de 1 para 0 (borda descendente).
- Assim, o comprimento do pulso do *clock* não é importante, contanto que as transições ocorram rapidamente.
- Para dar ênfase, vamos repetir qual é a diferença entre um *flip-flop* e um *latch*.
- Um *flip-flop* é disparado pela borda, enquanto um *latch* é disparado pelo nível.

2.3.5. Memórias

- Contudo, fique atento, porque esses termos são muito confundidos na literatura.
- Muitos autores usam "*flip-flop*" quando estão se referindo a um *latch*, e vice-versa.
- Há várias formas de projetar um *flip-flop*.
- Por exemplo, se houvesse alguma maneira de gerar um pulso muito curto na borda ascendente do sinal de *clock*, esse pulso poderia ser alimentado para um *latch* D.
- Na verdade, essa maneira existe, e o circuito para ela é mostrado na figura (a).

2.3.5. Memórias

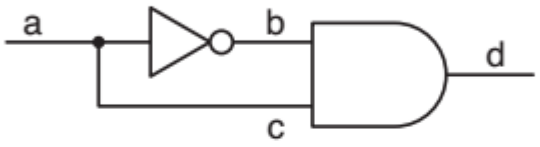
- À primeira vista, poderia parecer que a saída da porta and seria sempre zero,
  - uma vez que a operação and de qualquer sinal com seu inverso é zero,
  - mas a situação é um pouco diferente disso.
- O inversor tem um atraso de propagação pequeno, mas não zero, e é esse atraso que faz o circuito funcionar.
- Suponha que meçamos a tensão nos quatro pontos de medição a, b, c e d.
- O sinal de entrada, medido em a, é um pulso de *clock* longo, como mostrado na parte inferior da figura (b).
- O sinal em b é mostrado acima dele.

2.3.5. Memórias

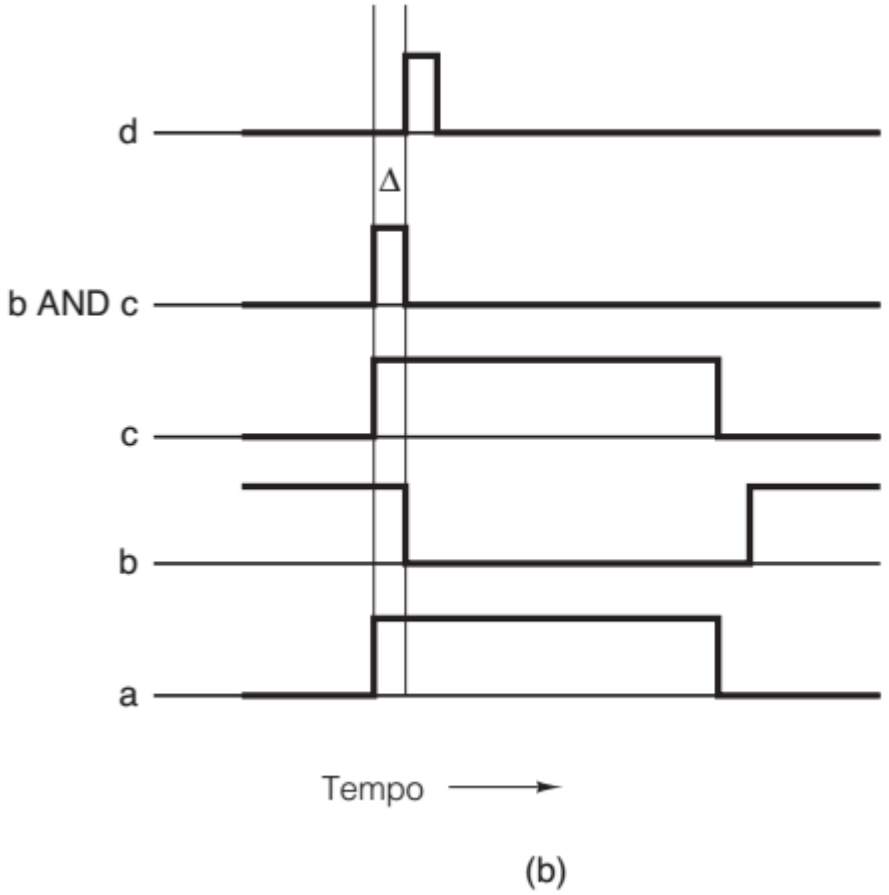
- Observe que ele está invertido e também ligeiramente atrasado, quase sempre de alguns nanossegundos, dependendo do tipo de inversor utilizado.
- O sinal em c também está atrasado, mas apenas pelo tempo correspondente à propagação (à velocidade da luz) do sinal.
- Se a distância física entre a e c for, por exemplo, 20 micra, então o atraso de propagação é 0,0001 ns, que decerto é desprezível em comparação com o tempo que o sinal leva para se propagar pelo inversor.
- Assim, para todos os efeitos e propósitos, o sinal em c é praticamente idêntico ao sinal em a.

2.3.5. Memórias

- (a) Gerador de pulso. (b) Temporização em quatro pontos do circuito.



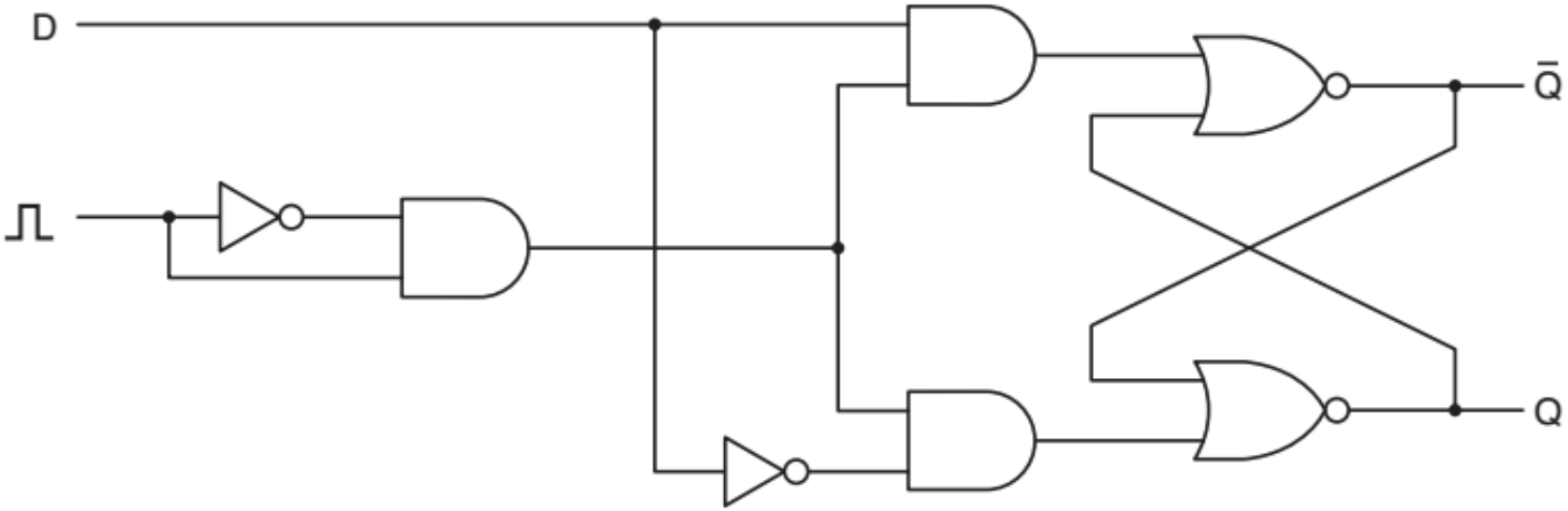
(a)



(b)

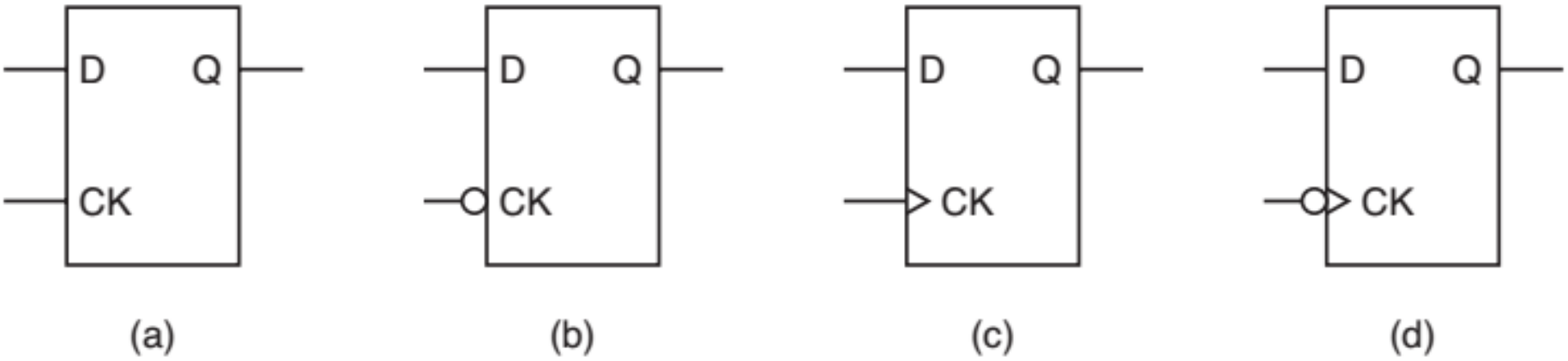
2.3.5. Memórias

- *Flip-flop* D.



2.3.5. Memórias

- *Latch* e *Flip-flop* D.



2.3.5. Memórias

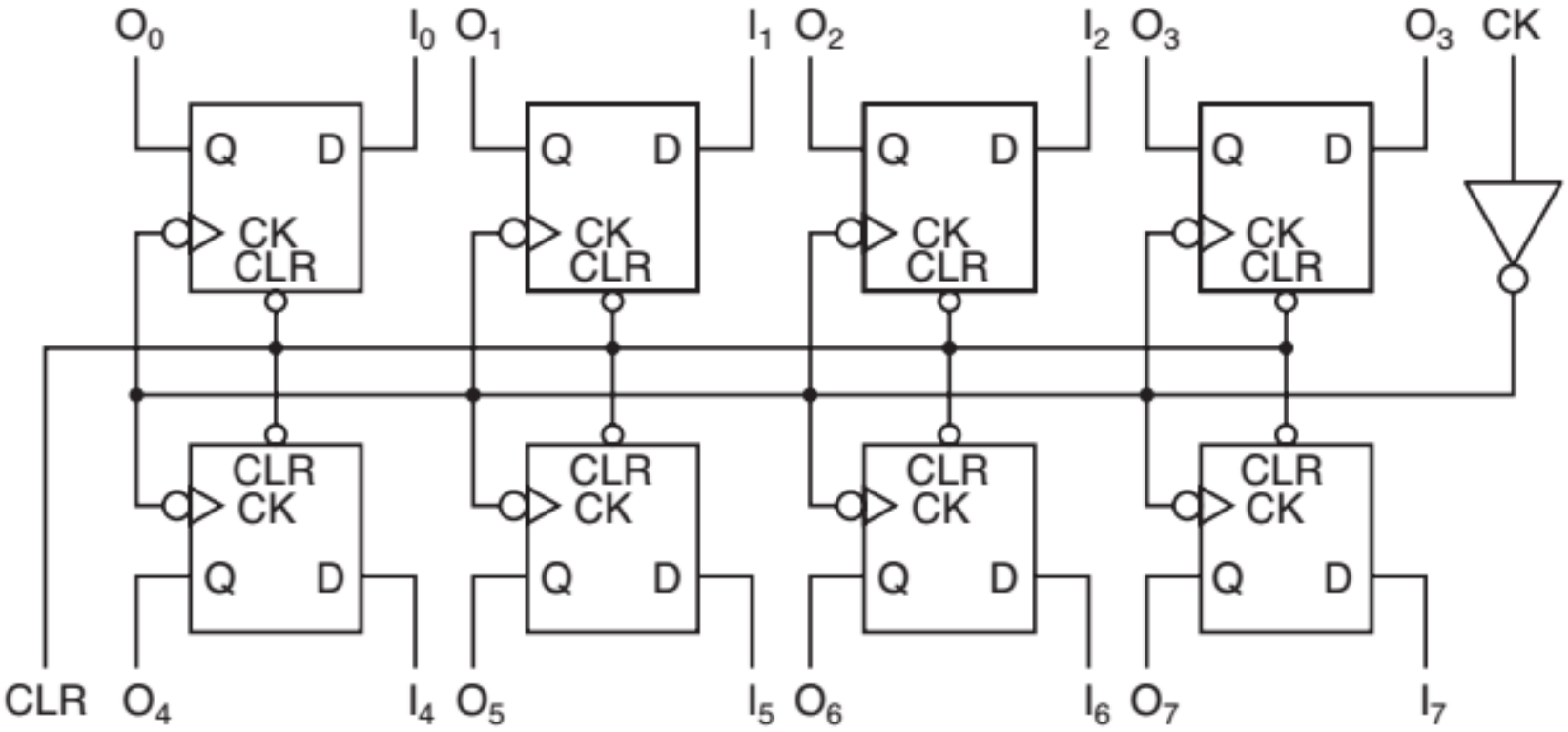
- *Flip-flops podem ser combinados em grupos para criar **registradores**, que mantêm tipos de dados com comprimentos maiores do que 1 bit.*
- *O registrador na figura próxima mostra como oito flip-flops podem ser ligados para formar um registrador armazenador de 8 bits.*
- *O registrador aceita um valor de entrada de 8 bits (I0 a I7) quando o clock CK fizer uma transição.*
- *Para implementar um registrador, todas as linhas de clock são conectadas ao mesmo sinal de entrada CK,*
  - *de modo que, quando o clock fizer uma transição,*
  - *cada registrador aceitará o novo valor de dados de 8 bits no barramento de entrada.*

2.3.5. Memórias

- *Os próprios flip-flops são do tipo da figura (d) anterior,*
  - *mas as bolhas de inversão nos flip-flops são canceladas pelo inversor ligado ao sinal de clock CK,*
  - *de modo que os flip-flops são carregados na transição ascendente do clock.*
- *Todos os oito sinais clear também são ligados, de modo que, quando o sinal clear CLR passar para 0,*
  - *todos os flip-flops serão forçados a passar para o seu estado 0.*
- *Caso você queira saber por que o sinal de clock CK é invertido na entrada e depois invertido novamente em cada flip-flop,*
  - *um sinal de entrada pode não ter corrente suficiente para alimentar todos os oito flip-flops;*
  - *o inversor da entrada, na realidade, está sendo usado como um amplificador.*

2.3.5. Memórias

- *Um registrador de 8 bits construído a partir de flip-flops de único bit.*



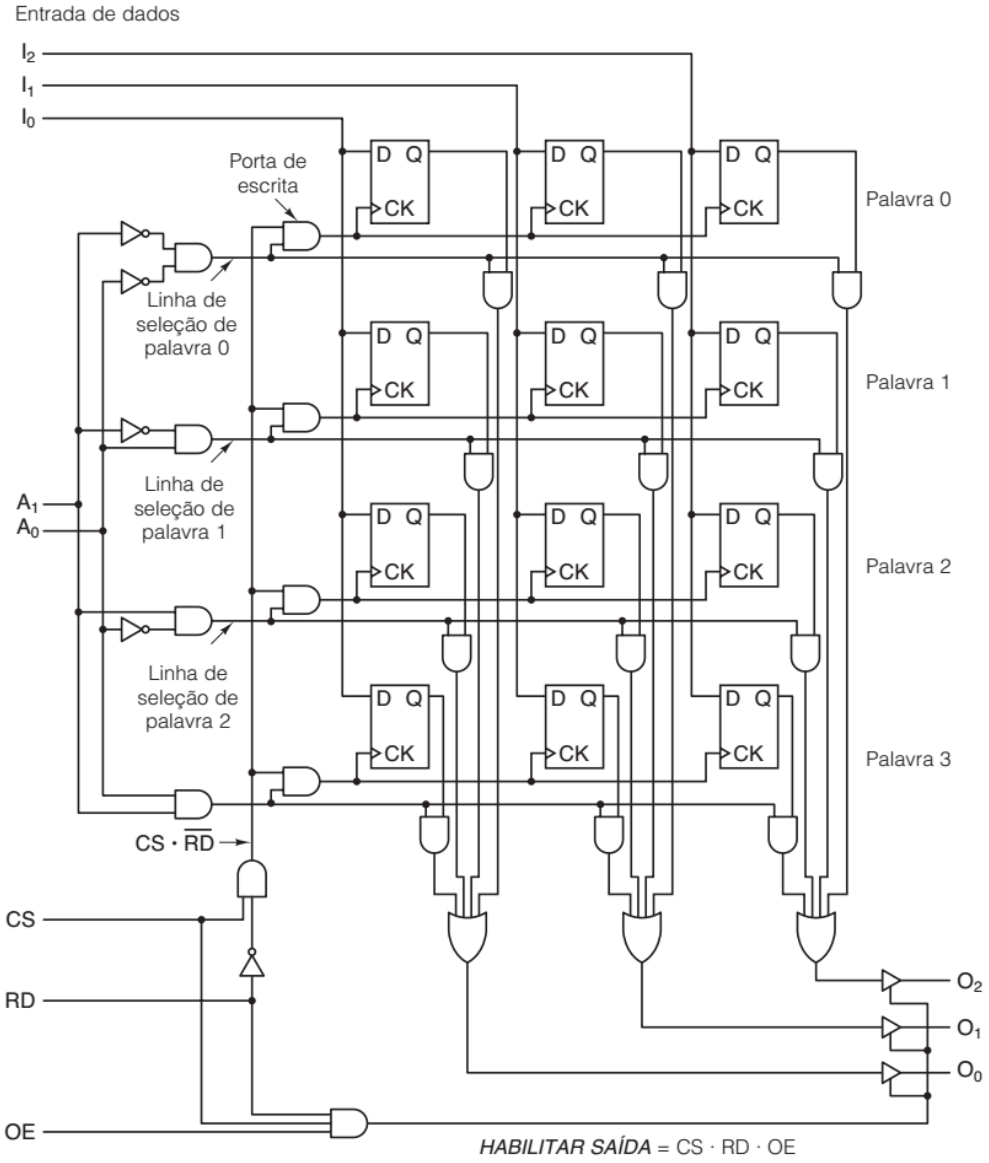
2.3.5. Memórias

- *Embora agora tenhamos progredido de uma simples memória de 1 bit da figura apresentada antes para a de 8 bits da figura anterior,*
  - *para construir memórias grandes é preciso uma organização diferente,*
  - *na qual palavras individuais podem ser endereçadas.*
- *Uma organização de memória muito utilizada e que obedece a esse critério é mostrada na próxima figura.*
- *Esse exemplo ilustra uma memória com quatro palavras de 3 bits.*

2.3.5. Memórias

- *Embora uma capacidade total de memória de 12 bits seja pouco mais do que nosso flip-flop octal,*
  - *ela requer um número menor de pinos e, mais importante,*
  - *o projeto pode ser estendido com facilidade para memórias grandes.*
- *Observe que o número de palavras é sempre uma potência de 2.*

2.3.5. Memórias



- Diagrama lógico para uma memória 4x3. Palavras de 4 bits em cada linha.

2.3.5. Memórias

- Embora à primeira vista talvez pareça complicada,
  - a memória da figura na verdade é bastante simples devido à sua estrutura regular.
- Ela tem oito linhas de entrada e três de saída.
  - Três entradas são de dados: I_0, I_1 e I_2;
  - duas são para o endereço: A_0 e A_1;
  - e três são para controle:
    - ✓ cs para chip *select* (selecionar chip),
    - ✓ rd para distinguir entre ler e escrever,
    - ✓ e oe para *output enable* (habilitar saída).

2.3.5. Memórias

- As três saídas são para dados: O_0, O_1 e O_2.
  - É interessante notar que essa memória de 12 bits requer menos sinais que o registrador de 8 bits anterior.
  - Este requer 20 sinais, incluindo alimentação e terra, enquanto a memória de 12 bits requer apenas 13 sinais.
- O bloco de memória requer menos sinais porque, diferente do registrador, os bits de memória compartilham um sinal de saída.
  - Nessa memória, cada um dos 4 bits de memória compartilha um sinal de saída.
  - O valor das linhas de endereço determina quais dos 4 bits de memória pode receber ou enviar um valor.

2.3.5. Memórias

- Para selecionar esse bloco de memória, a lógica externa deve estabelecer cs alto e também rd alto (1 lógico) para leitura e baixo (0 lógico) para escrita.
  - As duas linhas de endereço devem ser ajustadas para indicar qual das quatro palavras de 3 bits deve ser lida ou escrita.
  - Para uma operação de leitura, as linhas de entrada de dados não são usadas, mas a palavra selecionada é colocada nas linhas de saída de dados.
  - Para uma operação de escrita, os bits presentes nas linhas de entrada de dados são carregados na palavra de memória selecionada; as linhas de saída de dados não são usadas.

2.3.5. Memórias

- Agora, vamos examinar atentamente a figura para ver como isso funciona.
  - As quatro portas *and* de seleção de palavras à esquerda da memória formam um decodificador.
  - Os inversores de entrada foram instalados de modo que cada porta é habilitada (saída é alta) por um endereço diferente.
  - Cada porta comanda uma linha de seleção de palavra, de cima para baixo, para as palavras 0, 1, 2 e 3.
- Quando o chip é selecionado para uma escrita, a linha vertical rotulada cs · rd estará alta,
  - habilitando uma das quatro portas de escrita,
  - dependendo de qual linha de seleção de palavra esteja alta.

2.3.5. Memórias

- A saída da porta de escrita comanda todos os sinais ck para a palavra selecionada,
  - carregando os dados de entrada nos flip-flops para aquela palavra.
- Uma escrita é efetuada apenas se cs estiver alto e rd estiver baixo, e,
  - ainda assim, somente a palavra selecionada por A0 e A1 é escrita; as outras palavras não são alteradas.
- Ler é semelhante a escrever.
- A decodificação de endereço é idêntica à da escrita.
- Mas agora a linha cs · rd está baixa,
  - portanto, todas as portas de escrita estão desabilitadas e nenhum dos flip-flops é modificado.

2.3.5. Memórias

- Em vez disso, a linha de seleção de palavra que for escolhida habilita as portas and vinculadas aos Q bits da palavra selecionada.
  - Portanto, a palavra selecionada entrega seus dados às portas or de quatro entradas na parte inferior da figura, enquanto as outras três palavras produzem 0s.
- Em consequência, a saída das portas or é idêntica ao valor armazenado na palavra selecionada.
- As três palavras não selecionadas não dão nenhuma contribuição à saída.

2.3.5. Memórias

- As memórias que estudamos até aqui podem ser escritas e lidas.
- Elas são denominadas memórias RAM (Random Access Memory – memória de acesso aleatório),
  - nome suspeito porque todos os chips de memória têm acesso aleatório.
- No entanto, o termo já é muito utilizado para que o mudemos agora.
- RAMs podem ser de duas variedades, estáticas e dinâmicas.
- Nas estáticas (Static RAMs – SRAMs), a construção interna usa circuitos similares ao nosso flip-flop D básico.

2.3.5. Memórias

- Uma das propriedades dessas memórias é que seus conteúdos são conservados enquanto houver fornecimento de energia.
- As RAMs estáticas são muito rápidas.
- Um tempo de acesso típico é da ordem de um nanossegundo ou menos.
- Por essa razão, elas são muito usadas como memória cache.
- RAMS dinâmicas (Dynamic RAMs – DRAMs), ao contrário, não usam flip-flops.
- Em vez disso, uma RAM dinâmica é um arranjo de células, cada uma contendo um transistor e um pequenino capacitor.

2.3.5. Memórias

- Os capacitores podem ser carregados ou descarregados, permitindo que 0s e 1s sejam armazenados.
- Como a carga elétrica tende a “vazar”,
  - cada bit em uma RAM dinâmica deve ser renovado (recarregado) com alguns milissegundos de intervalo para evitar que os dados desapareçam.
- Como a lógica externa é que tem de cuidar da renovação,
  - as RAMs dinâmicas precisam de uma interface mais complexa do que as estáticas,
  - embora em muitas aplicações essa desvantagem seja compensada por suas maiores capacidades.

2.3.5. Memórias

- Sendo as RAMs dinâmicas compostas de apenas um transistor e um capacitor por bit,
  - em comparação com os seis transistores por bit para a melhor RAM estática,
  - elas têm densidade muito alta (muitos bits por chip).
- As memórias principais quase sempre são construídas com RAMs dinâmicas.
- Essa grande capacidade tem um preço: são lentas (dezenas de nanossegundos).
  - Dessa maneira, a combinação de uma cache de RAM estática e uma memória principal de RAM dinâmica tenta combinar as boas propriedades de cada uma.
- Existem diversos tipos de RAMs dinâmicas.

2.3.5. Memórias

- A mais antiga ainda existente (em computadores antigos) é a DRAM FPM (Fast Page Mode – modo de página rápida).
- Ela é organizada internamente como uma matriz de bits e funciona da seguinte maneira:
  - o hardware escolhe um endereço de linha e então seleciona endereços de coluna um a um,
  - como descrevemos para o ras e o cas no contexto da figura.
- Sinais explícitos informam à memória quando é hora de responder,
  - de modo que ela funciona de forma assíncrona com o clock do sistema principal.

2.3.5. Memórias

- A DRAM FPM foi substituída pela EDO (Extended Data Output – saída de dados ampliada),
  - que permite iniciar uma segunda referência à memória antes de ser concluída a referência à memória precedente.
- Esse paralelismo simples não acelerava uma referência individual à memória,
  - mas melhorava a largura de banda da memória,
  - resultando em mais palavras por segundo.
- FPM e EDO funcionavam bastante bem quando os tempos de ciclo de chips de memória eram de 12 nanossegundos ou mais lentos.

2.3.5. Memórias

- Quando os processadores ficaram tão rápidos que era mesmo preciso ter memórias mais rápidas,
  - a FPM e a EDO foram substituídas pela SDRAM (Synchronous DRAM – DRAM síncrona),
  - que é uma híbrida de RAM estática e dinâmica, comandada pelo clock do sistema principal.
- A grande vantagem da SDRAM é que o clock elimina a necessidade de sinais de controle para informar ao chip de memória quando responder.
  - Em vez disso, a CPU informa à memória por quantos ciclos ela deve funcionar e então a inicia.
- Em cada ciclo subsequente,
  - a memória entrega 4, 8 ou 16 bits, dependendo de quantas linhas de saída ela tem.

2.3.5. Memórias

- A melhoria seguinte em relação à SDRAM foi a SDRAM DDR (Double Data Rate – dupla taxa de dados).
  - Com essa memória, o chip de memória produz saída na borda ascendente do clock e também na borda descendente, dobrando a taxa de dados.
  - Portanto, um chip DDR de 8 bits de largura funcionando a 200 MHz entrega dois valores de 8 bits 200 milhões de vezes por segundo (por um curto intervalo, é claro),
  - O que dá uma taxa de saída (burst) teórica de 3,2 Gbps.
- As interfaces de memória DDR2 e DDR3 oferecem desempenho adicional em relação à DDR,
  - aumentando as velocidades do barramento de memória para 533 MHz e 1.067 MHz, respectivamente.
- No momento, os chips DDR3 mais velozes poderiam enviar dados a 17,067 GB/s.

2.3.5. Memórias

- Em muitas aplicações, o programa e alguns dos dados devem permanecer armazenados mesmo quando o fornecimento de energia for interrompido.
- Além do mais, uma vez instalados, nem o programa nem os dados são alterados.
- Esses requisitos levaram ao desenvolvimento de ROMs (Read-Only Memories – memórias somente de leitura),
  - que não podem ser alteradas nem apagadas,
  - seja intencionalmente ou não.
- Os dados de uma ROM são inseridos durante sua fabricação por um processo que expõe um material fotossensível por meio de uma máscara que contém o padrão de bits desejado e então grava o padrão sobre a superfície exposta (ou não exposta).
- A única maneira de mudar o programa em uma ROM é substituir o chip inteiro.

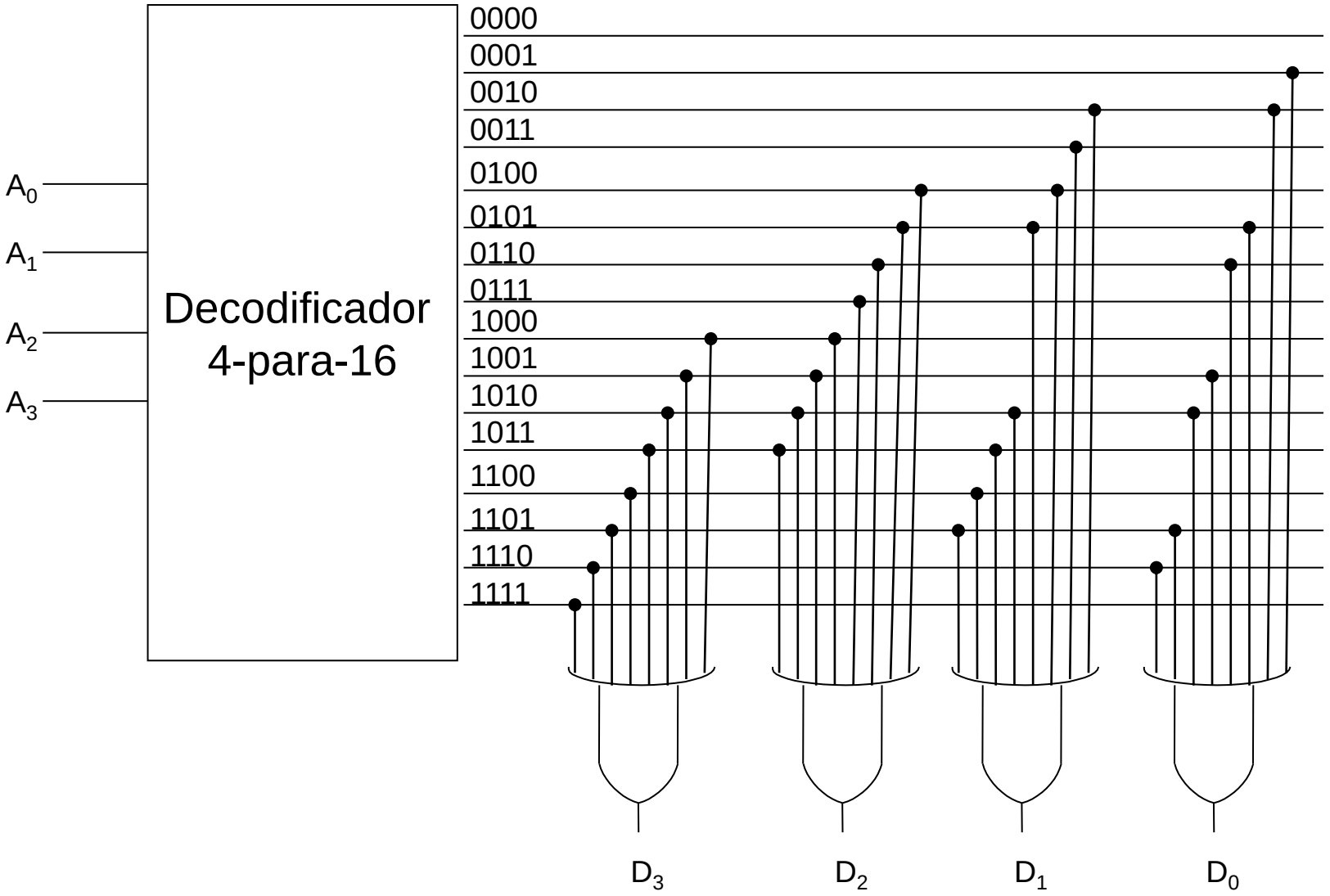
2.3.5. Memória Apenas de Leitura

Exemplo 2.3.5. Memória ROM de 64 bits, consistindo de 16 palavras de 4 bits cada, cuja tabela verdade é dada por

Entrada (Endereço)				**Saída (Dado Armazenado)**			
A_3	A_2	A_1	A_0	D_3	D_2	D_1	D_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

2.3.5. Memória Apenas de Leitura

Exercício 2.3.6.



2.3.5. Memórias

- ROMs são muito mais baratas que RAMs quando fabricadas em volumes grandes o bastante para cobrir o custo da fabricação da máscara.
- Todavia, são inflexíveis porque não podem ser alteradas após a manufatura,
  - o tempo decorrido entre fazer o pedido e receber as ROMs pode chegar a semanas.
- Para facilitar o desenvolvimento pelas empresas de novos produtos com ROM,
  - foi inventada a PROM (Programmable ROM – ROM programável).
- Uma PROM é como uma ROM,
  - exceto que ela pode ser programada (uma vez) em campo,
  - eliminando o tempo de espera entre produção e entrega.
- Muitas PROMs contêm um arranjo de minúsculos fusíveis em seu interior.

2.3.5. Memórias

- Um fusível específico pode ser queimado selecionando sua linha e coluna e então aplicando alta tensão a um pino especial no chip.
- O desenvolvimento seguinte nessa linha foi a EPROM (Erasable PROM – PROM apagável),
  - que não só pode ser programada, mas também apagada em campo.
- Quando a janela de quartzo de uma EPROM é exposta a uma forte luz ultravioleta durante 15 minutos, todos os bits são definidos em 1.
- Se a expectativa é ter muitas alterações durante o ciclo de projeto,
  - as EPROMs são muito mais econômicas do que as PROMs, porque podem ser reutilizadas.
- As EPROMS costumam ter a mesma organização que as RAMs estáticas.
- Ainda melhor do que a EPROM é a EEPROM, que pode ser apagada aplicando-se pulsos em vez de ser exposta à luz ultravioleta dentro de uma câmara especial.

2.3.5. Memórias

- Além disso, uma EEPROM pode ser reprogramada no local,
  - enquanto uma EPROM tem de ser inserida em um dispositivo especial de programação de EPROM para ser programada.
- Uma desvantagem é que a capacidade das maiores EEPROMs é em geral somente 1/64 da capacidade das EPROMs comuns,
  - e sua velocidade é a metade.
- EEPROMs não podem competir com DRAMs ou SRAMs porque são 10 vezes mais lentas, sua capacidade é 100 vezes menor e são muito mais caras.
  - Elas são usadas somente em situações em que sua não volatilidade for crucial.
- Um tipo mais recente de EEPROM é a memória flash.
  - Diferente da EPROM, que é apagada pela exposição à luz ultravioleta, e da EEPROM, cujos bytes podem ser apagados, os blocos da memória flash podem ser apagados e reescritos.
  - Como a EEPROM, a memória flash pode ser apagada sem ser removida do circuito.

2.3.5. Memórias

- Vários fabricantes produzem pequenas placas de circuito impresso com até 64 GB de memória flash que são utilizadas como um "filme" para armazenar fotos em câmeras digitais e muitas outras finalidades.
- Como já vimos, a memória flash agora está começando a substituir os discos mecânicos.
- Assim como um disco, a memória flash oferece tempos de acesso menores com menor consumo de energia, mas com um custo por bit muito mais alto.
- Um resumo dos diversos tipos de memória pode ser visto na tabela.

2.3.5. Memórias

- Comparação entre vários tipos de memórias (Arranjo de portas programável em campo).

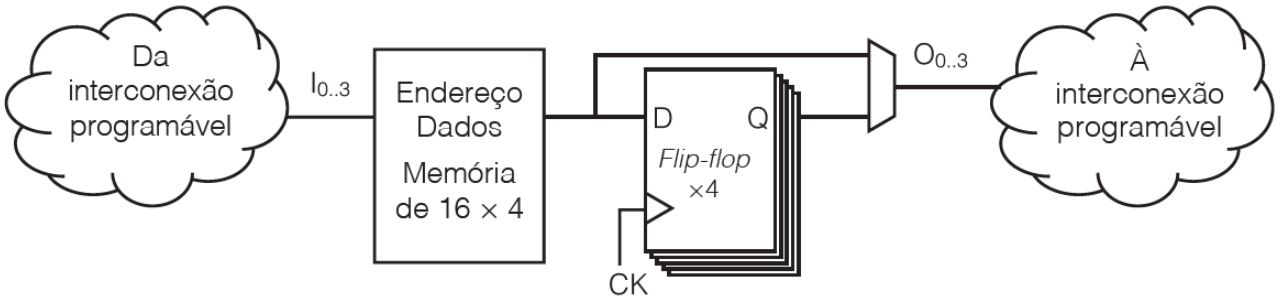
Tipo	Categoria	Modo de apagar	Byte alterável	Volátil	Utilização típica
SRAM	Leitura/escrita	Elétrico	Sim	Sim	*Cache* de nível 2
DRAM	Leitura/escrita	Elétrico	Sim	Sim	Memória principal (antiga)
SDRAM	Leitura/escrita	Elétrico	Sim	Sim	Memória principal (nova)
ROM	Somente de leitura	Não é possível	Não	Não	Equipamentos de grande volume
PROM	Somente de leitura	Não é possível	Não	Não	Equipamentos de pequeno volume
EPROM	Principalmente leitura	Luz UV	Não	Não	Prototipagem de dispositivos
EEPROM	Principalmente leitura	Elétrico	Sim	Não	Prototipagem de dispositivos
Flash	Leitura/escrita	Elétrico	Não	Não	Filme para câmera digital

2.3.6. FPGAs

- Field-Programmable Gate Arrays (FPGAs) são chips que contêm lógica programável, de modo que podem formar um circuito lógico qualquer simplesmente carregando o FPGA com dados de configuração apropriados.
- A principal vantagem dos FPGAs é que novos circuitos de hardware podem ser construídos em horas, em vez dos meses necessários para fabricar ICs.
- Porém, os circuitos integrados não serão extintos, pois ainda possuem uma vantagem de custo significativa em relação aos FPGAs para aplicações de alto volume, e também são mais rápidos e usam muito menos energia.
- Contudo, com suas vantagens de tempo de projeto, os FPGAs são usados constantemente para protótipo de projeto e aplicações com baixo volume.

2.3.6. FPGAs

- Agora, vejamos o interior de um FPGA para entender como ele pode ser usado para executar uma grande gama de circuitos lógicos.
- O chip FPGA contém dois componentes principais que são replicados muitas vezes:
  - LUTs (*LookUp Tables* – tabelas de pesquisa), e
  - interconexões programáveis.
- Vejamos agora como estes são utilizados.
- Uma LUT, mostrada na Figura é uma pequena memória programável que produz um sinal de saída opcionalmente para um registrador, que é então enviada para a interconexão programável.



2.3.6. FPGAs

- A memória programável é usada para criar uma função lógica qualquer.
- A LUT na figura tem uma memória de 16 × 4, que pode simular qualquer circuito lógico com 4 bits de entrada e 4 bits de saída.
- A programação da LUT requer a carga da memória com as respostas apropriadas da lógica combinatória sendo simulada.
- Em outras palavras, se a lógica combinatória produz o valor Y quando recebe a entrada X, o valor Y é escrito na LUT no índice X.
- O projeto de exemplo na Figura mostra como uma única LUT de 4 entradas poderia executar um contador de 3 bits com reset.

Designação de sinal

FPGA	Contador
I_3	CLR
$O_{2..0}$	$O_{2..0}$
CK	CK

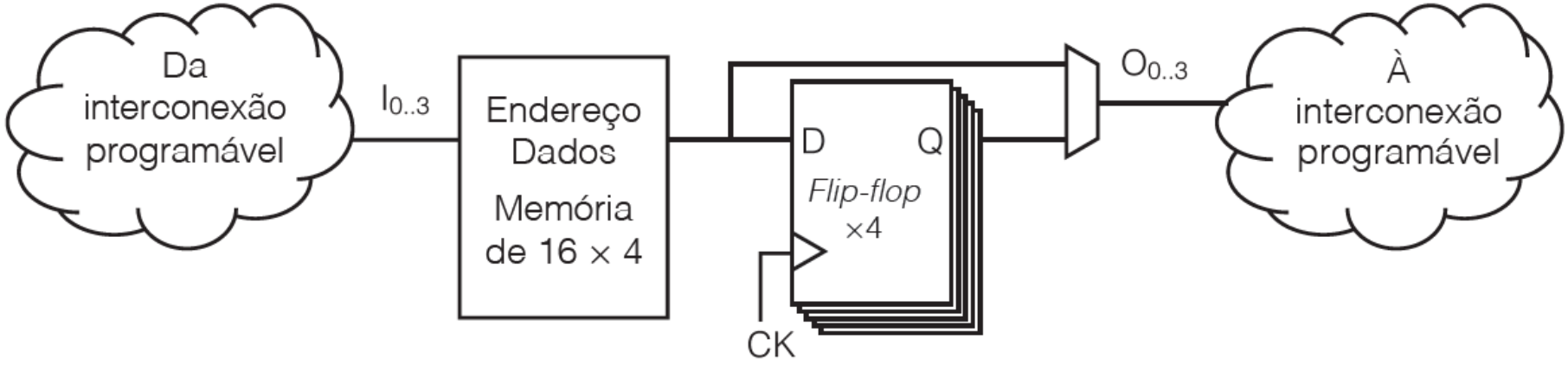
Endereço	Dados
0	1
1	2
2	3
3	0

Endereço	Dados
4	0
5	0
6	0
7	0

2.3.6. FPGAs

- O exemplo de contador conta de modo contínuo somando um (módulo 4) ao valor atual, a menos que um sinal de reset CLR seja afirmado, que nesse caso retorna o valor do contador a zero.
- Para pôr em prática o contador do exemplo, as quatro entradas superiores da LUT são todas zero.
- Essas entradas enviam o valor zero quando o contador é reiniciado.
- Assim, o bit mais significativo da entrada da LUT (I_3) representa a entrada de reset (CLR) que é ativada com uma lógica 1.
- Para as entradas restantes da LUT, o valor no índice $I_{0..3}$ da LUT contém o valor (I + 1) módulo 4.
- Para concluir o projeto, o sinal de saída $O_{0..3}$ deve ser conectado, usando a interconexão programável para o sinal de entrada interno $I_{0..3}$.

2.3.6. FPGAs



(a)

Designação de sinal

FPGA	Contador
I_3	CLR
$O_{2..0}$	$O_{2..0}$
CK	CK

Endereço	Dados
0	1
1	2
2	3
3	0

Endereço	Dados
4	0
5	0
6	0
7	0

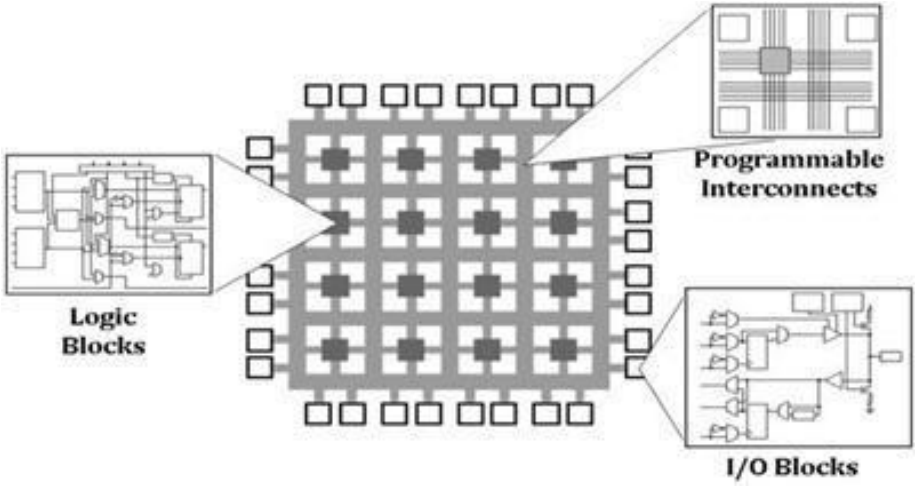
(b)

2.3.6. FPGAs

- Para entender melhor o contador baseado em FPGA com reset, vamos considerar sua operação.
- Se, por exemplo, o estado atual do contador for 2 e o sinal de reset (CLR) não for afirmado, o endereço de entrada da LUT será 2, que produzirá uma saída de 3 nos *flip-flops*.
- Se o sinal de reset (CLR) fosse afirmado para o mesmo estado, a entrada na LUT seria 6, que produziria o próximo estado de 0.
- Apesar de tudo, esse pode parecer um modo arcaico de se construir um contador com reset e, de fato, um projeto totalmente personalizado, com um circuito incrementador e sinais de reset para os *flip-flops*, seria menor, mais rápido e usaria menos energia.
- A principal vantagem do projeto baseado em FPGA é que você pode ajustá-lo em uma hora em casa, enquanto o projeto totalmente personalizado, mais eficiente, deve ser fabricado com base no silício, o que poderia levar pelo menos um mês.

2.3.6. FPGAs

- Para usar um FPGA, o projeto precisa ser descrito usando uma descrição de circuito ou uma linguagem de descrição de hardware (ou seja, uma linguagem de programação usada para descrever estruturas de hardware).
- O projeto é então processado por um sintetizador, que mapeia o circuito para uma arquitetura FPGA específica.
- Um desafio do uso de FPGAs é que o projeto que você quer mapear nunca parece ser o suficiente.



2.3.6. FPGAs

- Os FPGAs são fabricados com uma quantidade variável de LUTs, com quantidades maiores custando mais.
- Em geral, se o seu projeto não for suficiente, você terá que simplificar ou abrir mão de alguma funcionalidade, ou então comprar um FPGA maior (e mais caro).
- Projetos muito grandes podem não caber nos maiores FPGAs, exigindo que o projetista mapeie o projeto em vários FPGAs;
  - essa tarefa é definitivamente mais difícil,
  - porém, ainda muito mais fácil do que projetar um circuito integrado personalizado completo.