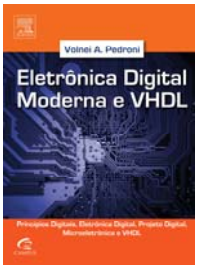


Eletrônica Digital Moderna e VHDL
Volnei A. Pedroni, Elsevier, 2010

Tradução (com revisão, atualização e ampliação) de
Digital Electronics and Design with VHDL
Elsevier / Morgan Kaufmann, USA, 2008

Soluções dos Exercícios Ímpares dos Capítulos 11-14

Capítulo 11: Circuitos Combinacionais Lógicos

Exercício 11.1. Lógica combinacional × seqüencial

Com a chave em (1): $y = a.b + b'.c$ (combinacional)

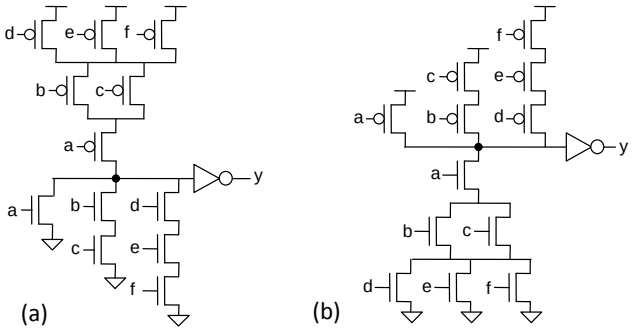
Com a chave em (2): $y = a.b + b'.y$ (sequencial)

Exercício 11.3. Porta lógica composta #2

a) Veja figura (a) abaixo.

b) Veja figura (b) abaixo.

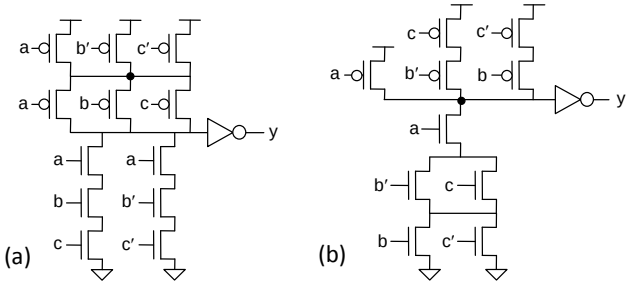
c) A parte de cima de um circuito é igual à parte de baixo do outro e vice-versa (o primeiro tem ramos AND na parte inferior e ramos OR na parte superior, enquanto que o segundo tem exatamente o oposto).



Exercício 11.5. Porta lógica composta #4

a) $y = a.b.c + a.b'.c'$. Veja figura (a) abaixo.

b) $y = a(b'+c)(b+c')$. Veja figura (b) abaixo.

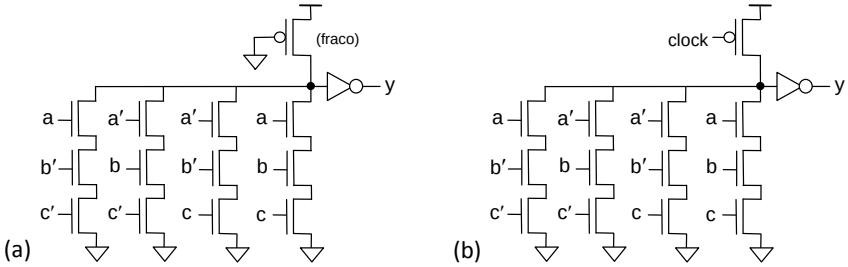


Exercício 11.7. Porta lógica composta #6

a) $y = a.b'.c' + a'.b.\ c' + a'.b'.c + a.b.c$.

b) Veja figura (a) abaixo.

c) Veja figura (b) abaixo.



Exercício 11.9. Decodificador de endereço com portas NAND

a) Equações:

$b_0 = a_2'.a_1'.a_0'$

$b_1 = a_2'.a_1'.a_0$

$b_2 = a_2'.a_1.a_0'$

$b_3 = a_2'.a_1.a_0$

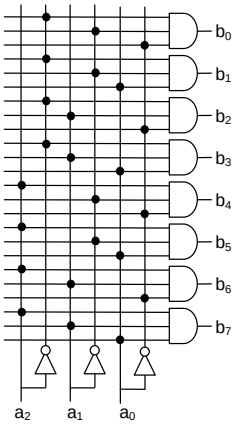
$b_4 = a_2.a_1'.a_0'$

$b_5 = a_2.a_1'.a_0$

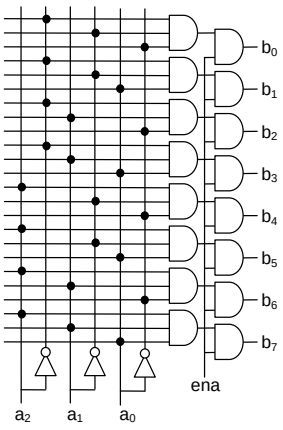
$b_6 = a_2.a_1.a_0'$

$b_7 = a_2.a_1.a_0$

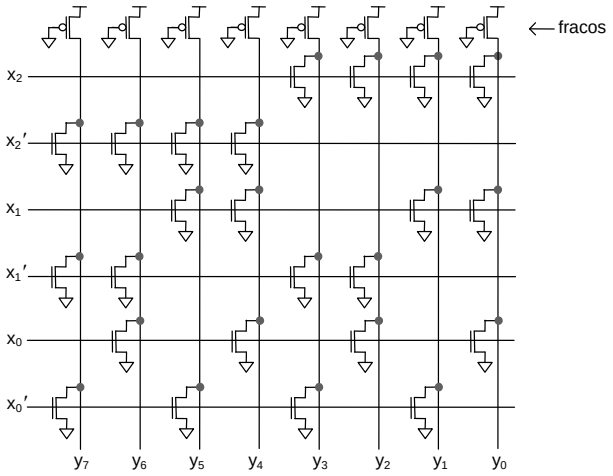
b) Circuito:



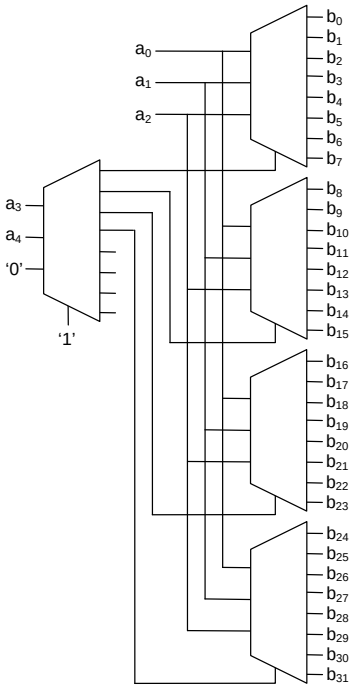
Exercício 11.11. Decodificador de endereço com enable #2



Exercício 11.13. Decodificador de endereço com lógica pseudo-nMOS

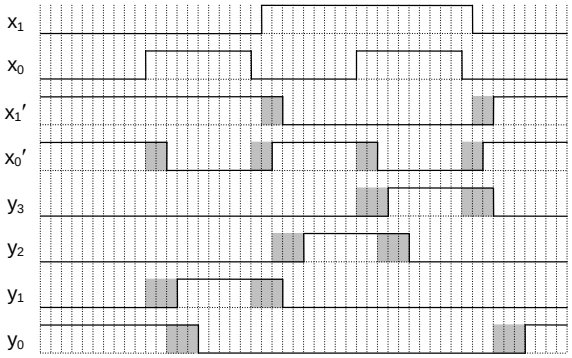


Exercício 11.15. Decodificador de endereço com mais entradas #2

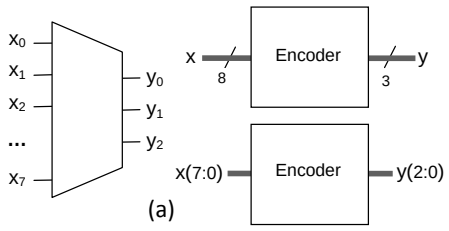


Exercício 11.17. Análise temporal de decodificador de endereço

Veja a figura abaixo, na qual os intervalos de tempo são de 1 s.



Exercício 11.19. Codificador de endereço



(b)

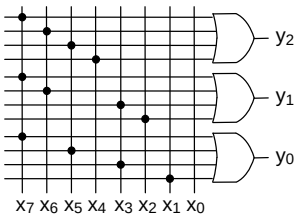
x	y
00000001	000
00000010	001
00000100	010
00001000	011
00010000	100
00100000	101
01000000	110
10000000	111

$y_2 = x_7 + x_6 + x_5 + x_4$

$y_1 = x_7 + x_6 + x_3 + x_2$

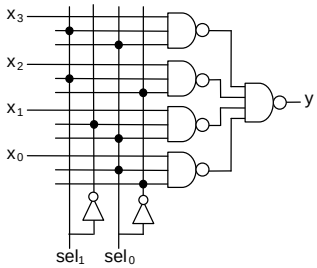
$y_0 = x_7 + x_5 + x_3 + x_1$

(c)

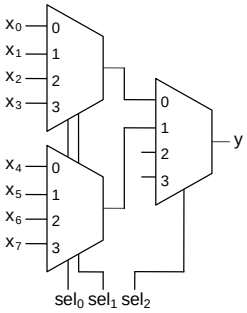


(d)

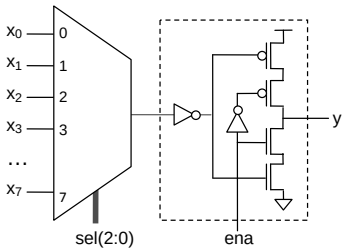
Exercício 11.21. Multiplexador com portas NAND



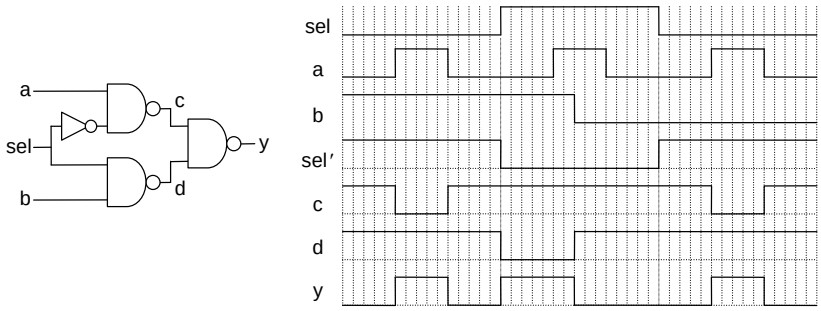
Exercício 11.23. Multiplexador com mais entradas



Exercício 11.25. Multiplexador com saída de alta impedância #1



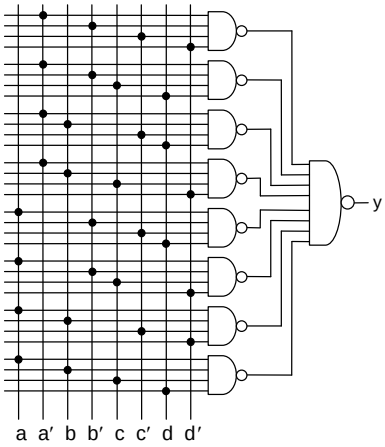
Exercício 11.27. Análise funcional de multiplexador



Exercício 11.29. Detector de paridade

a) $y = a'.b'.c'.d' + a'.b'.c.d + a'.b.c'.d + a'.b.c.d' + a.b'.c'.d + a.b'.c.d' + a.b.c.d$

b) Veja figura abaixo.



Exercício 11.31. Ordenador binário

a) $N(N–1)/2$

b) $3(N–1)(N+1)/8$ (N foi considerado ímpar)

c) Idem (b) acima

Exercício 11.33. Deslocador Lógico

Basta inverter a ordem de x e y no circuito da figura 11.26, isto é, troca-se x_i com x_{N-1-i} e y_i com y_{N-1-i} para i=0 até $N–1$.

Exercício 11.35. Schmitt triggers

Verifique manuais de fabricantes. Alguns exemplos são mostrados no capítulo 18.

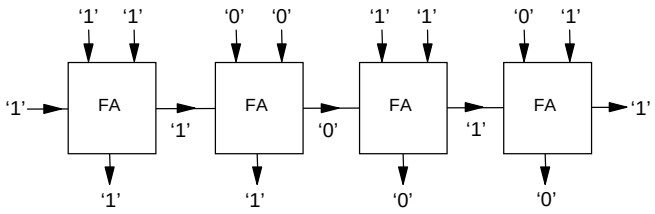
Capítulo 12: Circuitos Combinacionais Aritméticos

Exercício 12.1. Funcionamento do somador de 1 bit completo (FA)

Aplique ao circuito os sinais *a*, *b* e *cin* da tabela verdade da figura 12.1(c) e verifique se os valores corretos são produzidos nas saídas *s* e *cout*.

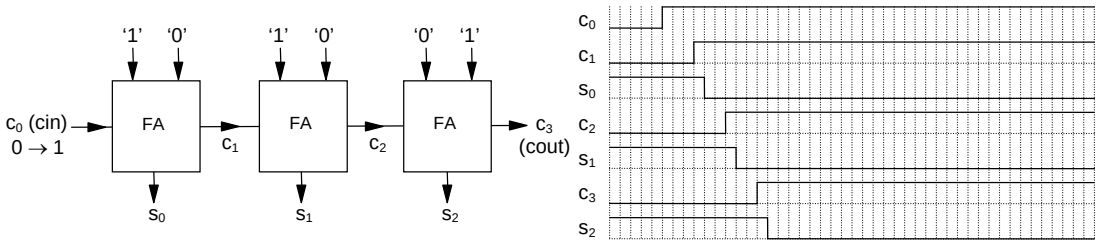
Exercício 12.3. Somador carry-ripple #2

Como mostrado na figura abaixo, as saídas são s_0 = '1', s_1 = '1', s_2 = '0', s_3 = '0' e c_4 = '1', portanto formando o vetor $c_4s_3s_2s_1s_0$ = "10011" (=19).



Exercício 12.5. Diagrama de tempo de um somador carry-ripple

Veja a figura abaixo, onde observa-se que o tempo de progagação do carry realmente se acumula nesse tipo de somador.



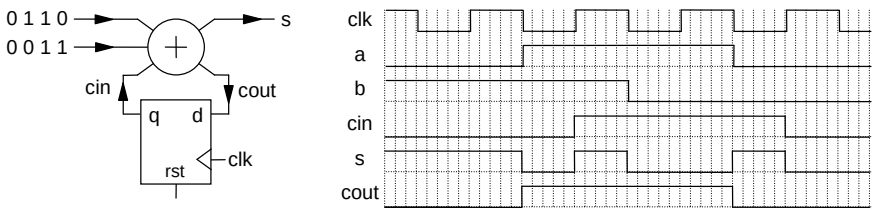
Exercício 12.7. Somador Manchester carry-chain

a) Deve-se adicionar uma porta OR para cada *P* (= *a*+*b*) e uma porta AND para cada *G* (= *a*·*b*).

b) Simplesmente aplique os valores dados para os sinais de entrada e verifique se os resultados corretos são produzidos nas saídas.

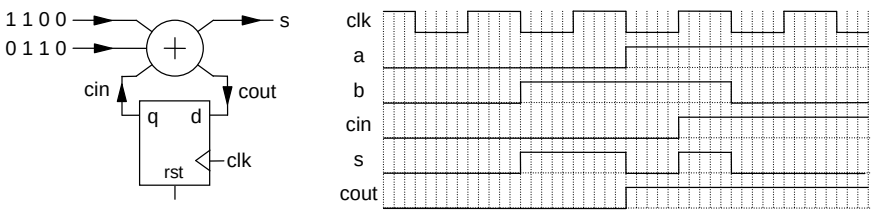
Exercício 12.9. Diagrama de tempo de um somador serial #1

Veja a figura abaixo. As sequências de soma e carry-out, tomadas nas transições positivas do clock (com delays de propagação desprezíveis), são *s* = "1001" e *cout* = "0110" (observe que no diagrama de tempo as sequências são mostradas em ordem inversa). O resultado é então $c_4s_3s_2s_1s_0$ = "01001" (=9), portanto correto.



Exercício 12.11. Diagrama de tempo de um somador serial #3

Usando o mesmo procedimento do exercício 12.9, as formas de onda abaixo são obtidas. A soma e o carry-out (tomados logo após às transições do clock, e em ordem inversa) são *s* = "0010" e *cout* = "1100", formando o vetor $c_4s_3s_2s_1s_0$ = "10010" (=18), que é o resultado correto.



Exercício 12.13. Incrementador

a) As equações do circuito (em formato SOP) são:

$b_0 = a_0'$

$b_1 = a_1 \oplus a_0 = a_1'.a_0 + a_1.a_0'$

$b_2 = a_2 \oplus (a_1.a_0) = a_2'.a_1.a_0 + a_2.a_1' + a_2.a_0'$

$b_3 = a_3 \oplus (a_2.a_1.a_0) = a_3'.a_2.a_1.a_0 + a_3.a_2' + a_3.a_1' + a_3.a_0'$

$b_4 = a_4 \oplus (a_3.a_2.a_1.a_0) = a_4'.a_3.a_2.a_1.a_0 + a_4.a_3' + a_4.a_2' + a_4.a_1' + a_4.a_0'$

b) O resultado esperado de fato ocorre, o que pode ser verificado usando o circuito ou as equações acima.

c) Devido a overflow, "00000" (= 0) é esperado, e de fato acontece.

Exercício 12.15. Complementador de dois

a) As equações do circuito (em formato SOP) são:

$b_0 = a_0$

$b_1 = a_1 \oplus a_0 = a_1'.a_0 + a_1.a_0'$

$b_2 = a_2 \oplus (a_1 + a_0) = a_2'.a_1 + a_2.a_0' + a_2.a_1'.a_0'$

$b_3 = a_3 \oplus (a_2 + a_1 + a_0) = a_3'.a_2 + a_3'.a_1 + a_3'.a_0 + a_3.a_2'.a_1'.a_0'$

$b_4 = a_4 \oplus (a_3 + a_2 + a_1 + a_0) = a_4'.a_3 + a_4'.a_2 + a_4'.a_1 + a_4'.a_0 + a_4.a_3'.a_2'.a_1'.a_0'$

b) O resultado esperado de fato ocorre, o que pode ser verificado usando o circuito ou as equações acima.

c) O resultado esperado é b = "00001" (= 1), o qual de fato ocorre (não há overflow nesse caso).

Exercício 12.17. Funcionamento de um comparador sem sinal #1

Esse circuito soma a ao complemento de dois de b; portanto, o resultado da soma será zero (invertido pela NOR) quando $a=b$, e o bit de carry-out será '1' quando $a \geq b$. Simplesmente aplique os valores dados ao circuito e veja os valores que o mesmo produz.

Exercício 12.19. Comparador completo sem sinal

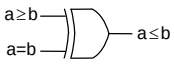
Duas soluções são apresentadas através de mapas de Karnaugh abaixo. Em (a), não é feito agrupamento de minterms, porem em (b) os estados que não podem ocorrer são considerados “don´t care”, produzindo então uma solução otimizada. As equações resultantes para a nova saída ($a \leq b$) são:

Em (a): $(a \leq b) = (a \geq b) \oplus (a = b)$

Em (b): $(a \leq b) = (a \geq b)' + (a = b)$

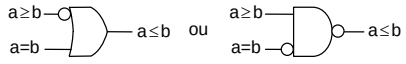
(a)

Saída a=b \ Saída a≥b	0	1
0	a<b	a>b
1	impossível	a=b



(b)

Saída a=b \ Saída a≥b	0	1
0	a<b	a>b
1	impossível	a=b

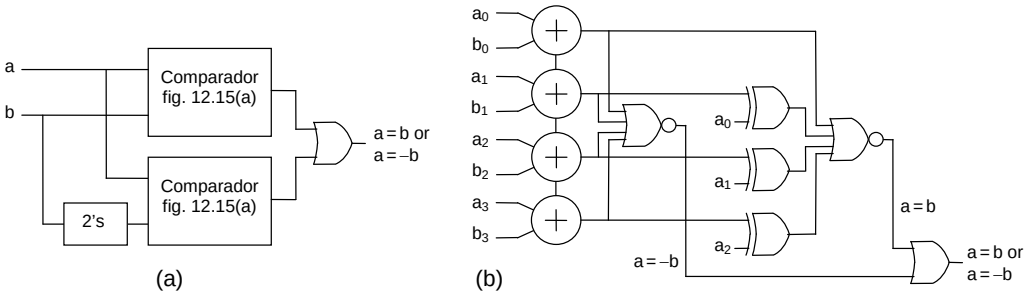


Exercício 12.21. Comparador de valor absoluto

Qualquer solução para este problema é relativamente custosa em termos de hardware. Uma solução óbvia é mostrada na figura (a) abaixo, a qual utiliza dois comparadores de igualdade, similares àquele da figura 12.15(a), um comparando a com b, outro comparando a com $-b$ (observe o complementador de dois na figura), com suas saídas indo a uma porta OR.

Outra solução é mostrada em (b), sem complementador de dois. Ao invés, um somador é empregado, o qual soma a e b diretamente. O resultado produzido pela primeira NOR será ‘1’ quando todos os bits da soma forem ‘0’, isto é,

quando $a = -b$. A segunda parte do circuito compara a soma com $2a$ (note que o vetor a foi shiftado para a esquerda uma posição). Se $a+b=2a$, então obviamente $a=b$.

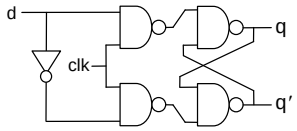


Exercício 12.23. Funcionamento de um multiplicador paralelo-serial

Simplesmente siga os passos mostrados no exemplo 12.5.

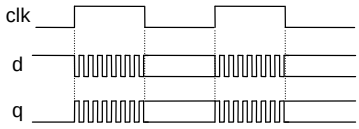
Capítulo 13: Registradores

Exercício 13.1. Do latch SR para o latch D



Exercício 13.3. Mau circuito

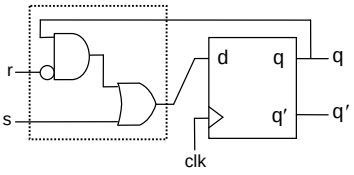
Conforme ilustra a figura abaixo, quando *clk* = '1', uma versão invertida de *q* é diretamente conectada à entrada do circuito, fazendo com que o mesmo oscile (numa frequência que depende somente do tempo de propagação através do latch). Quando *clk* retorna a zero, o valor existente naquele momento ('0' ou '1') é armazenado pelo circuito.



Exercício 13.4. Flip-flop SR

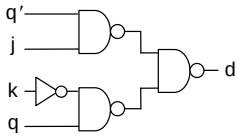
No caso abaixo, $d = s + r' \cdot q$. Portanto, se a situação proibida ($s = r =$ '1') ocorrer, *s* (set) vencerá.

clr	s	r	q+	q+'	Estado
↑	0	0	q	q'	Memória
↑	1	0	1	0	Set
↑	0	1	0	1	Reset
↑	1	1	( )	( )	Proibido



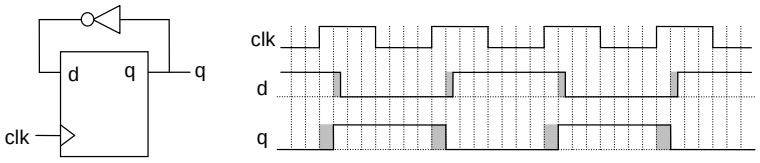
Exercício 13.5. Flip-flop JK

A função implementada pelo bloco adicional é $d = q'.j + q.k'$. Um exemplo de circuito para tal é mostrado abaixo.



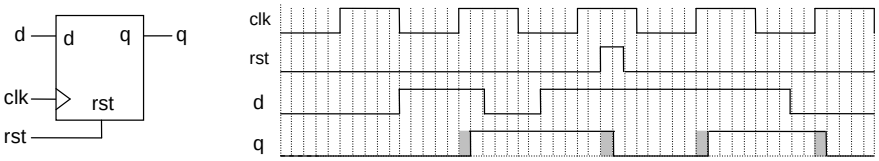
Exercício 13.7. Análise de tempo de um TFF

O resultado é mostrado abaixo. Como sempre, as áreas escuras representam os delays de propagação.



Exercício 13.9. Análise de tempo de um DFF

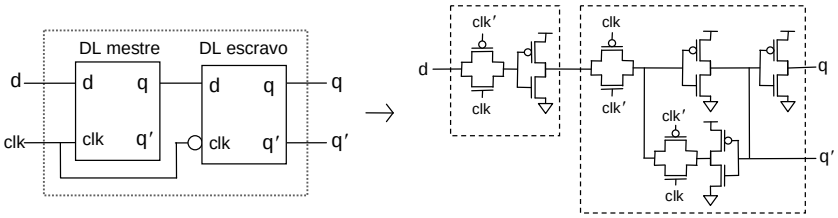
O resultado é mostrado abaixo. Como sempre, as áreas escuras representam os delays de propagação.



Exercício 13.11. DFFs mestre-escravo

a) O resultado é mostrado na figura abaixo.

b) Simplesmente siga o mesmo procedimento.



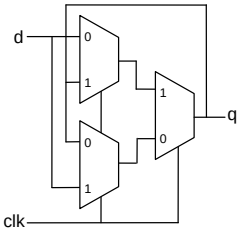
Exercício 13.13. Flip-flops de precarga e avaliação

Um DFF é considerado como sendo do tipo “precharge-evaluate” quando seus nós (todos ou alguns) são *incondicionalmente* pré-carregados (normalmente a V_{DD}) em cada ciclo do clock, independentemente do valor do dado (*d*) na entrada. Usando essa definição, constata-se o seguinte:

a) (a), (b), (c) e (f).

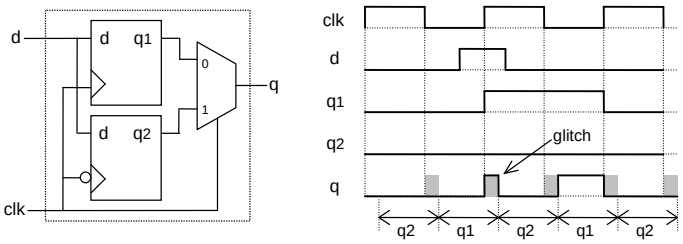
b) Somente (c).

Exercício 13.14. DFF de borda dupla com multiplexadores



Exercício 13.15. DFF de borda dupla com DFFs de borda única

O circuito em questão é mostrado na figura abaixo. Alem de desperdiçar recursos (é maior do que aquele da figura 13.20(a)), seu funcionamento é altamente dependente dos tempos de propagação (ao contrário daquele da figura 13.20(a)). Isto pode ser observado no exemplo apresentado no diagrama de tempo incluído na figura, no qual foi suposto que o mux é lento (áreas cinza no sinal *q*) em suas comutações de uma entrada à outra (observe a ocorrência de um glitch em *q*).

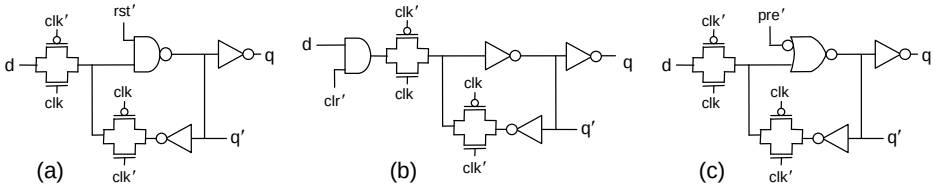


Exercício 13.17. DL com reset, clear e preset

a) Na figura (a) abaixo, *q* é zerado imediatamente quando *rst*' = '0' acontece.

b) Na figura (b), se *clr*' = '0', *q* será zerado no próximo nível alto do clock se o clock estiver baixo, ou imediatamente se o clock já estiver alto.

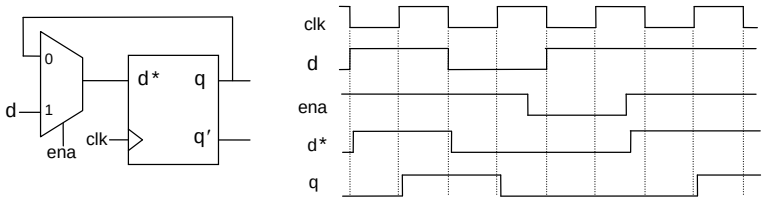
c) Na figura (c), *q* = '1' ocorre imediatamente após a aplicação de *pre*' = '0' ao circuito.



Exercício 13.19. DFF com enable #1

a) Veja figura abaixo.

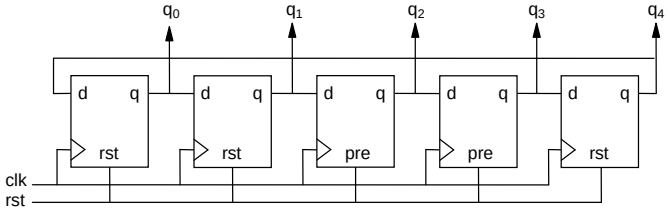
b) Sim, esse enable é verdadeiramente síncrono, pois qualquer mudança no enable só poderá afetar a saída na próxima borda (positiva, nesse exemplo) do clock.



Capítulo 14: Circuitos Sequenciais

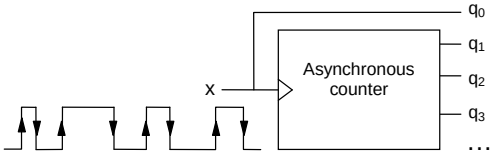
Exercício 14.1. Registrador de deslocamento circular

A solução é mostrada abaixo, a qual foi baseada na figura 14.2(c) (observe as entradas de reset e preset dos flip-flops).



Exercício 14.3. Contador de eventos

Veja a figura abaixo, na qual *x* atua como o LSB do contador.



Exercício 14.5. Contador síncrono de 0 a 31 com DFFs

a) Apenas adicione outra célula ao circuito da figura 14.5(a), após q_3, para produzir q_4. A porta AND correspondente deve ter quatro entradas (q_0, q_1, q_2, q_3).

b) Apenas adicione outra célula *padrão* ao circuito da figura 14.5(b), novamente após q_3, produzindo assim q_4.

Exercício 14.7. Contador síncrono de 0 a 255 com DFFs

Apenas adicione quatro células *padrão* ao circuito da figura 14.5(b), após q_3, para produzir q_4, q_5, q_6 e q_7.

Exercício 14.9. Contador síncrono de 0 a 4 com DFFs

Notas:

1 - O capítulo 14 apresenta dois métodos simplificados, simples e práticos, para projeto de contadores. O primeiro é ilustrado nos exemplos 14.1 (com TFFs) e 14.2 (com DFFs), ao passo que o segundo é ilustrado nos exemplos 14.3 (com TFFs) e 14.4 (com DFFs). O primeiro deles, com DFFs, será adotado na solução abaixo.

2 - Lembre da observação feita antes dos exemplos 14.1 e 14.3, que diz que esse procedimento de projeto, embora simples e prático, não garante que as expressões boleanas obtidas sejam mínimas, pois não está-se tirando vantagem dos estados que não podem ocorrer no contador. Por exemplo, para contar de 0 a 4, são necessários 3 flip-flops, os quais devem produzir os valores 0, 1, 2, 3 e 4, retornado então a 0; como os estados 5, 6 e 7 não podem ocorrer, eles poderiam ter sido incluídos como "dont´t care" na tabela verdade do contador, reduzindo assim as expressões finais (isso vai ser demonstrado no capítulo 15, exemplo 15.4). Porem, as expressões derivadas no presente método são quase ótimas, e são o melhor que pode-se fazer quando não leva-se em conta os estados "don't care".

a) O procedimento abaixo é aquele visto no exemplo 14.3.

Valor final da contagem desejado: $q_2q_1q_0$ = "100" (=4)

Próximo valor natural da contagem: $q_2q_1q_0$ = "101" (=5; este é o valor para onde o contador *iria*)

Valor inicial da contagem desejado: $q_2q_1q_0$ = "000" (=0; este é o valor para onde o contador *deve ir*)

Na lista acima, observa-se:

Para q_2: Seu próximo valor é '1', porem '0' é o valor desejado. Logo, a entrada desse DFF deve ser forçada a d_2='0' quando o valor final da contagem ($q_2q_1q_0$="100") for atingido. Lembre, porem, que os '0's do valor final não precisam ser monitorados, pois nenhum valor antes de "100" apresenta o mesmo padrão de '1's.

Para q_1: O próximo valor é '0', o qual coincide com o valor desejado. Logo, nada precisa ser feito.

Para q_0: O próximo valor é '1', mas '0' é desejado. Logo, d_0='0' é necessário quando q_2='1' acontece.

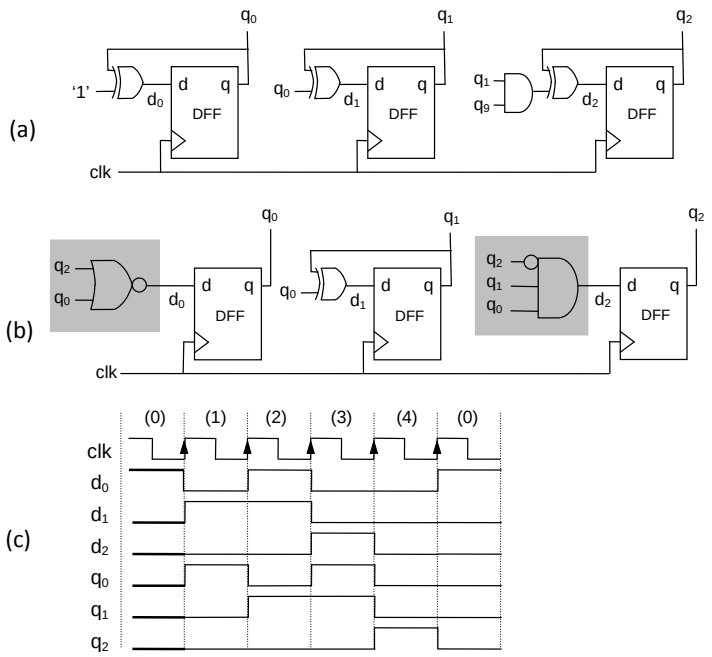
As seguintes expressões são então obtidas para d_2 e d_0, com os valores de d_{2old} e d_{0old} tomados do circuito original (módulo-2^N), mostrado na figura (a) abaixo:

$$d_{2new} = d_{2old}.condition' + \text{'0'}.condition = d_{2old}.condition' = d_{2old}.q_2' = [q_2 \oplus (q_1.q_0)].q_2' = q_2'.q_1.q_0$$

$$d_{0new} = d_{0old}.condition' + \text{'0'}.condition = d_{0old}.condition' = d_{0old}.q_2' = q_0'.q_2'$$

O circuito final, com essas expressões incluídas, consta na figura (b).

b) Veja a figura (c) abaixco. Note que, quando os delays de propagação são desprezados, é preciso lembrar que os valores armazenados pelos DFFs são aqueles imediatamente *anteriores* à borda do clock.



Exercício 14.11. Contador síncrono de 2 a 6 com DFFs

a) O procedimento abaixo é aquele visto no exemplo 14.5.

Valor final da contagem desejado: $q_2q_1q_0$ = "110" (= 6)

Próximo valor natural da contagem: $q_2q_1q_0$ = "111" (= 7)

Próximo valor desejado da contagem: $q_2q_1q_0$ = "010" (= 2)

Na lista acima, observa-se:

Para q_2: O próximo valor é '1', mas '0' é desejado. Então, essa entrada deve ser forçada a d_2 = '0' ao final da contagem.

Para q_1: O próximo valor é '1', o qual coincide com o valor desejado. Logo, nada precisa ser feito.

Para q_0: Idem a q_2. Então, precisa-se criar d_0 = '0' ao final da contagem.

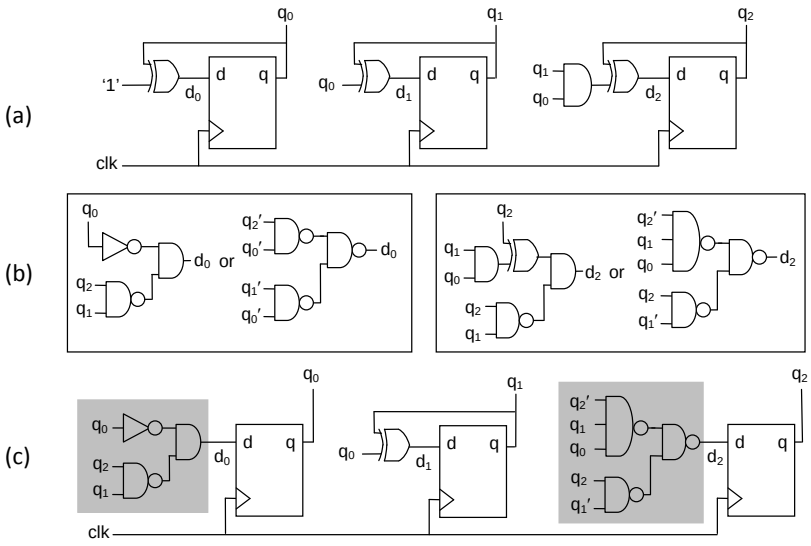
Consequentemente:

$$d_{2new} = d_{2old}.condition' + \text{'0'}.condition = d_{2old}.condition' = [q_2 \oplus (q_1.q_0)].(q_2.q_1)' = q_2.q_1' + q_2'.q_1.q_0$$

$$d_{0new} = d_{0old}.condition' + \text{'0'}.condition = d_{0old}.condition' = q_0'.(q_2.q_1)' = q_2'.q_0' + q_1'.q_0'$$

O resultado é mostrado na figura abaixo. O circuito original consta em (a), as modificações, em (b), e o circuito final, em (c).

b) Siga o mesmo procedimento do exercício 14.9.



Exercício 14.13. Contador síncrono de 8 a 15 com quatro DFFs
Siga o mesmo raciocínio do exemplo 14.5.

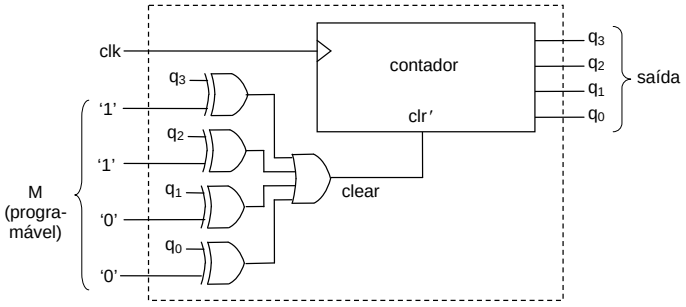
Exercício 14.15. Contador síncrono de 20 a 25 com cinco DFFs
Siga o mesmo raciocínio do exemplo 14.5.

Exercício 14.17. Contador síncrono de 0 a 1023 com enable serial
Apenas adicione seis células *padrão* ao circuito da figura 14.5(b), após q_3, para produzir q_4 a q_9. O circuito irá contar então de 0 até $2^N–1 = 2^{10}–1 = 1023$.

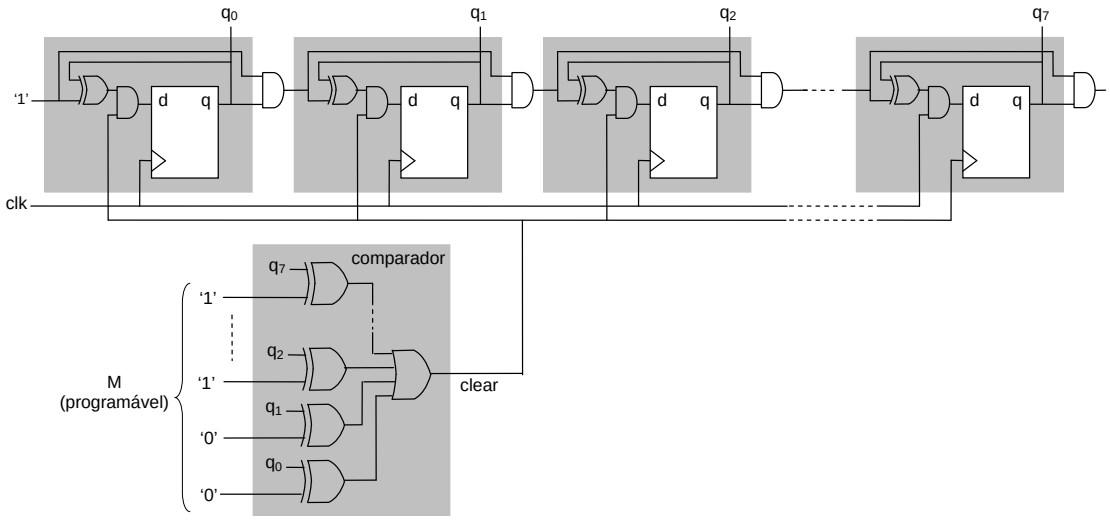
Exercício 14.19. Contador síncrono com enable #1
a) Quando *ena* = '0', o contador deve parar, tornando a contar somente quando *ena* voltar a '1'. No circuito proposto, o sinal de enable interrompe o primeiro flip-flop, porem não interrompe necessariamente os demais. Por exemplo, suponha que quando *ena* = '0' acontece tenha-se q_0 = '1'; como q_0 é o único habilitador do segundo DFF, esse DFF continuará contando normalmente, apesar do primeiro estar parado. Portanto, essa não é uma opção correta de enable.

b) Deve ser feito como na figura 14.14, utilizando dois blocos de quatro bits.

Exercício 14.21. Contador programável de 4 bits #1
Uma solução é mostrada abaixo, a qual contem um comparador de igualdade (veja figura 12.15(a)) na entrada. Quando a saída do contador coincide com o valor programado, clear = '0' é gerado, o qual causa o retorno do contador a zero na próxima borda do clock. O contador pode ser, por exemplo, aquele da figura 14.11(b).



Exercício 14.23. Contador programável de 8 bits #1
Essa solução é similar àquela do exercício 14.21, com apenas duas diferenças: 8 bits ao invés de 4, e enable serial ao invés de paralelo. O circuito completo é mostrado abaixo.

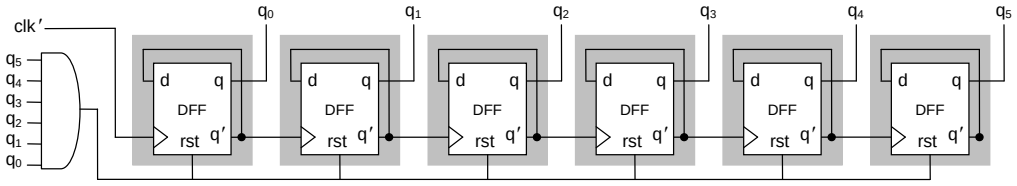


Exercício 14.25. Contador assíncrono de 0 a 63 com DFFs

Trata-se de um contador assíncrono de módulo 2^N, onde N=6. Portanto, basta adicional duas unidades ao circuito da figura 14.16(c) ou 14.16(d), para produzir q_4 e q_5.

Exercício 14.27. Contador assíncrono de 0 a 62 com DFFs

Este problema é similar àquele no exemplo 14.7. O valor a ser monitorado (porta AND do reset) é "estado final + 1 = 63", isto é, $q_5q_4q_3q_2q_1q_0$= "111111". O circuito correspondente consta abaixo. Lembre que esta arquitetura só é válida se a ordem temporal de variação dos sinais for aquela real de um circuito assíncrono, ou seja, q_0, depois q_1, depois q_2, etc. (delays no roteamento e cargas excessivas podem prejudicar essa sequência).



Exercício 14.29. Contador assíncrono de 0 a 254 com DFFs

Simplesmente adote o mesmo procedimento do exemplo 14.7 ou do exercício 14.27.

Exercício 14.31. Registrador de deslocamento com saídas sincronizadas

Este circuito deve produzir dois vetores na saída, denominados $x = x_3\,x_2\,x_1\,x_0$ e $y = y_3\,y_2\,y_1\,y_0$, com $y(t) = x(t–T)$, onde T é o período do clock. Basta tomar o SR da figura 14.2(c) e fazer nele as seguintes conexões: $x_3\,x_2\,x_1\,x_0 = q_3\,q_0\,q_1\,q_2$ e $y_3\,y_2\,y_1\,y_0 = q_0\,q_1\,q_2\,q_3$.

Exercício 14.33. Gerador de sinal de duas janelas #2

Esse exercício é similar àquele do exemplo 14.8 (figura 14.9), com a única diferença que agora o valor de x é 19 ao invés de 9. Portanto, $q_4\,q_3'\,q_2'\,q_1\,q_0$ deve ser monitorado pela porta AND, pois x = "10011" (=19).

Exercício 14.35. Gerador de sinal de quatro janelas

Esse exercício é similar àquele do exemplo 14.9 (figura 14.10), apenas com modificações nos valores de x_1, y_1, x_2 e y_2. Um contador de 50 estados é agora necessário, porque $5T_0 + 10T_0 + 15T_0 + 20T_0 = 50T_0$. Pode ser um contador de 0 a 49, por exemplo, com parâmetros x_1 = 4, y_1 = 14, x_2 = 29 e y_2 = 49; nesse caso:

x_1 = "000100" (=4) $\rightarrow q_5'\,q_4'\,q_3'\,q_2\,q_1'\,q_0'$

y_1 = "001110" (=14) $\rightarrow q_5'\,q_4'\,q_3\,q_2\,q_1\,q_0'$

x_2 = "011101" (=29) $\rightarrow q_5'\,q_4\,q_3\,q_2\,q_1'\,q_0$

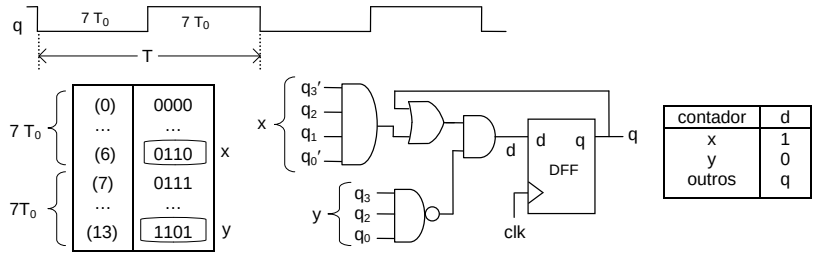
y_2 = "110001" (=49) → $q_5\, q_4\, q_3'\, q_2'\, q_1'\, q_0$

Exercício 14.37. Circuito divisor por 8

Esse é um contador de módulo 2^N, com N=3. Basta então um contador binário convencional, com 3 flip-flops, o qual se auto-resetará após o último estado ("111" → "000"). O MSB (q_2) terá fase simétrica automaticamente (veja o diagrama de tempo da figura 14.3(c)).

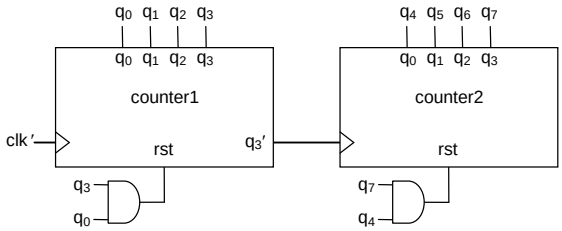
Exercício 14.39. Circuito divisor por 14 com fase simétrica

Este é apenas um gerador de sinal de duas janelas (seção 14.4) de borda única. A solução apresentada abaixo foi obtida diretamente da figura 14.9 (as únicas mudanças foram nos valores de *x* e *y*). É mais simples do que para valores ímpares de *M* porque não requer o DFF e a AND extras do exemplo 14.10 (figura 14.23).



Exercício 14.41. Contador BCD de dois dígitos #2

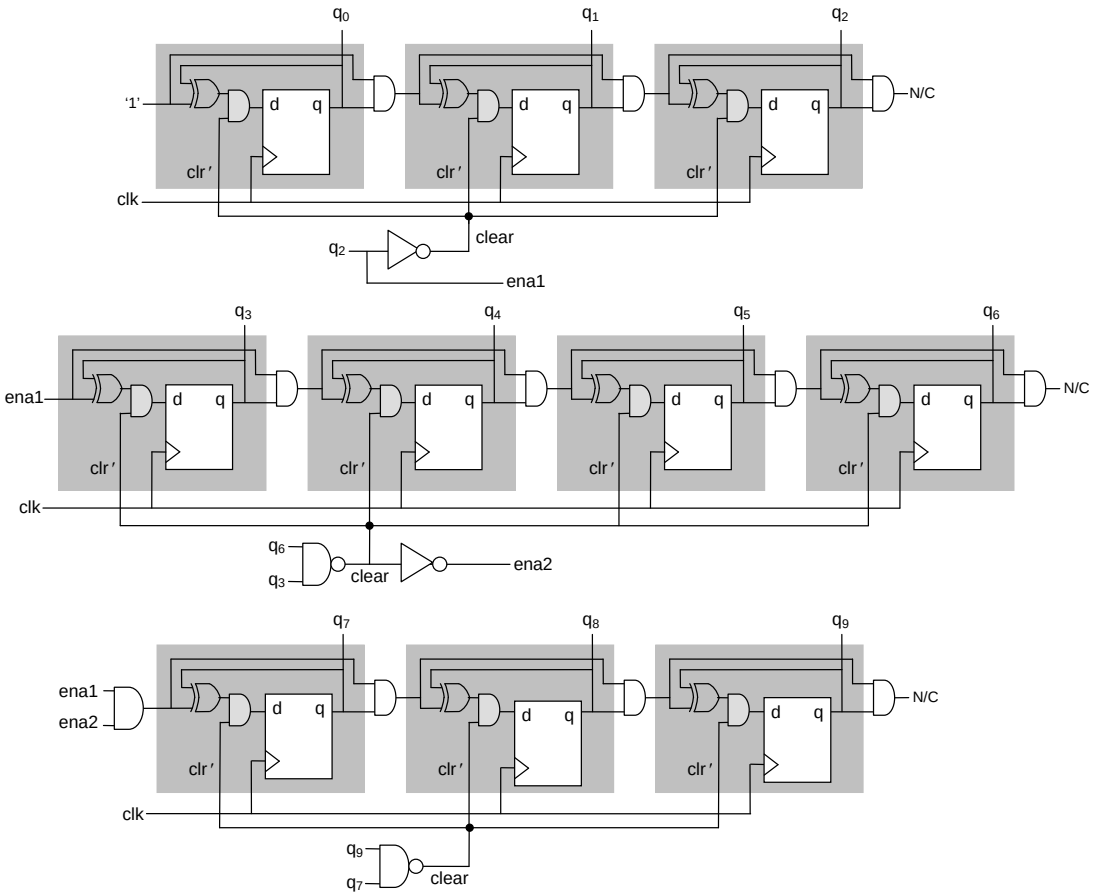
A figura abaixo mostra dois contadores assíncronos de quatro bits (cada um desses circuitos é similar àquele do exercício 14.27; logo, a observação feita lá sobre a sequência temporal dos sinais é indispensável também aqui). O bit q_3' do primeiro contador serve como clock ao segundo. Como a frequência de q_3' é *fclk*/10, o segundo contador será incrementado a cada dez ciclos de clock. Ambos são resetados após "1001" (=9) ocorrer. Novamente, os '0's não precisam ser monitorados, só os '1's, daí que as portas AND só necessitam de q_3 e q_0 no primeiro contador e q_7 e q_4 no segundo.



Exercício 14.43. Timer #1

Todos os contadores da figura 4.24(b) são mostrados abaixo. O primeiro conta de 0 a 4; o segundo, de 0 a 9; e o terceiro, de 0 a 5. Todos são contadores síncronos com enable serial.

Pergunta: No terceiro contador, seria suficiente ter *ena2* ao invés de *ena1* e *ena2* (porta AND) para controlar o sinal de enable deste contador?



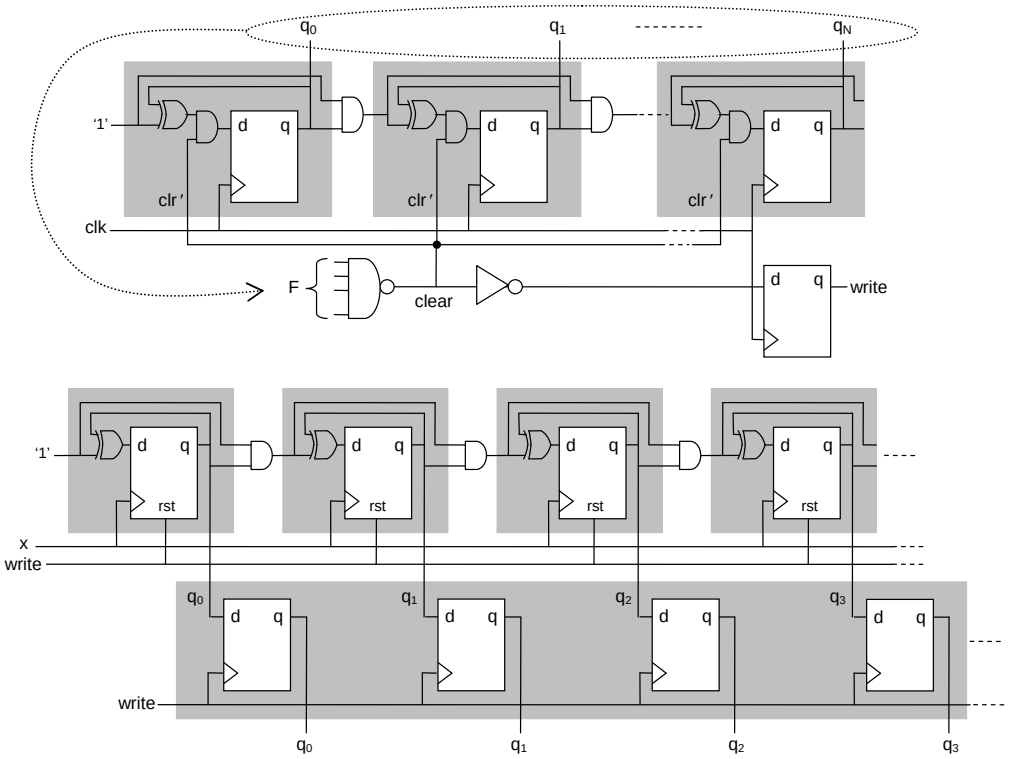
Exercício 14.45. Medidor de freqüência #1

a) Neste caso, uma janela de 1 s é usada como base de tempo. Como a precisão da medida cresce à medida que mais eventos ocorrem dentro dessa janela, essa arquitetura é mais adequada para altas frequências.

b) 1/(*F*+1) (≈0)

c) O circuito (sem sincronizador) é mostrado abaixo. A parte superior contem o contador que converte o sinal de clock em um sinal de duas janelas, sendo uma (*write*='0') com duração de *F* períodos de clock, e outra (*write*='1') com duração de apenas um período. Tal sinal é o próprio sinal de clear; porem, como ele está sujeito a glitches, uma versão registrada do mesmo é empregada, daí a necessidade do DFF extra que consta na saída desse contador.

O segundo circuito contem, na parte superior, o contador de eventos em *x*, enquanto que a parte inferior contem o registrador de uso geral necessário para armazenar o resultado da contagem (f_x). *x* opera como clock para o contador, de forma que ele conta as transições positivas de *x*. Note que quando *write* vai a '1' duas coisas acontecem: (i) a saída do contador é armazenada no registrador e (ii) o contador é resetado.



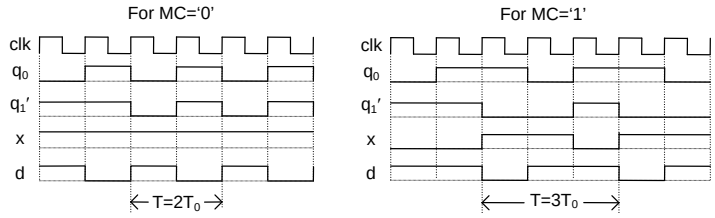
Exercício 14.47. Funcionamento de um PLL

Disponível na seção 14.6.1.

Exercício 14.49. Prescaler de módulo duplo #1

a) *M*=2 para *MC*='0', *M*=3 para *MC*='1'.

b-c) Veja figuras abaixo.



Exercício 14.51. Prescaler de módulo duplo #3

a) *M*=4 para *MC*='0', *M*=5 para *MC*='1'.

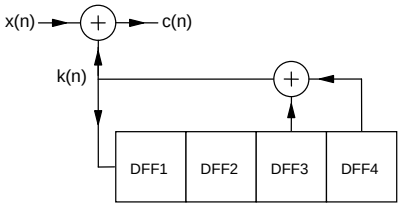
b-c) Essas partes são deixadas para o leitor.

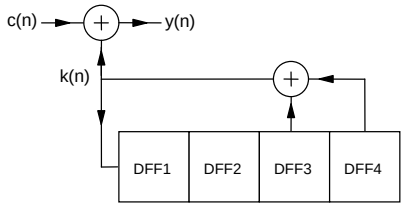
Exercício 14.53. Scrambler aditivo

a) Veja figuras abaixo.

b) Entrando *x* = "10101010..." (a partir da ponta esquerda), com o scrambler inicializado em "1111", obtém-se *c* = "101110011111010...".

c) Obtém-se *y* = "101010101010 . . .".


x(n)
c(n)
k(n)
DFF1
DFF2
DFF3
DFF4


c(n)
y(n)
k(n)
DFF1
DFF2
DFF3
DFF4