1

INTRODUCTION

EXERCISE SOLUTIONS

1.2 Three definitions of “bit”:

(1) A binary digit (p. 1).

(2) Past tense of “bite” (p. 1).

(3) A small amount (pp. 6, 10).

1.3

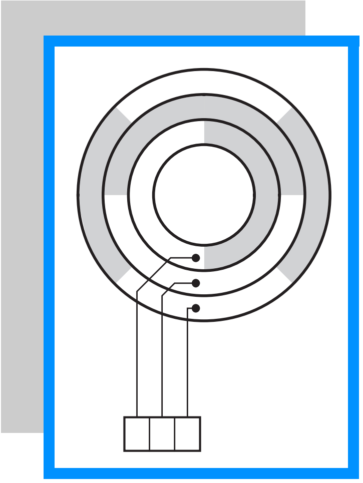
ASIC	Application-Specific Integrated Circuit
CAD	Computer-Aided Design
CD	Compact Disc
CO	Central Office
CPLD	Complex Programmable Logic Device
DAT	Digital Audio Tape
DIP	Dual In-line Pin
DVD	Digital Versatile Disc
FPGA	Field-Programmable Gate Array
HDL	Hardware Description Language
IC	Integrated Circuit
IP	Internet Protocol
LSI	Large-Scale Integration
MCM	Multichip Module

MSI	Medium-Scale Integration
NRE	Nonrecurring Engineering
OK	Although we use this word hundreds of times a week whether things are OK or not, we have probably rarely wondered about its history. That history is in fact a brief one, the word being first recorded in 1839, though it was no doubt in circulation before then. Much scholarship has been expended on the origins of OK, but Allen Walker Read has conclusively proved that OK is based on a sort of joke. Someone pronounced the phrase “all correct” as “oll (or orl) correct,” and the same person or someone else spelled it “oll korrect,” which abbreviated gives us OK. This term gained wide currency by being used as a political slogan by the 1840 Democratic candidate Martin Van Buren, who was nicknamed Old Kinderhook because he was born in Kinderhook, New York. An editorial of the same year, referring to the receipt of a pin with the slogan O.K., had this comment: “frightful letters . . . significant of the birth-place of Martin Van Buren, old Kinderhook, as also the rallying word of the Democracy of the late election, ‘all correct’ .... Those who wear them should bear in mind that it will require their most strenuous exertions ... to make all things O.K.” [From the *American Heritage Electronic Dictionary (AHED)*, copyright 1992 by Houghton Mifflin Company]
PBX	Private Branch Exchange
PCB	Printed-Circuit Board
PLD	Programmable Logic Device
PWB	Printed-Wiring Board
SMT	Surface-Mount Technology
SSI	Small-Scale Integration
VHDL	VHSIC Hardware Description Language
VLSI	Very Large-Scale Integration

1.4

ABEL	Advanced Boolean Equation Language
CMOS	Complementary Metal-Oxide Semiconductor
JPEG	Joint Photographic Experts Group
MPEG	Moving Picture Experts Group
OK	(see above)
PERL	According to some, it’s “Practical Extraction and Report Language.” But the relevant Perl FAQ entry, in perlfaq1.pod, says “never write ‘PERL’, because perl isn't really an acronym, apocryphal folklore and post-facto expansions notwithstanding.” (Thanks to Anno Siegel for enlightening me on this.)
VHDL	VHSIC Hardware Description Language

1.8 In my book, “dice” is the plural of “die.”



2

NUMBER SYSTEMS AND CODES

EXERCISE SOLUTIONS

2.1 (a) $1101011_2 = 6B_{16}$ (b) $174003_8 = 1111100000000011_2$
(c) $10110111_2 = B7_{16}$ (d) $67.24_8 = 110111.0101_2$
(e) $10100.1101_2 = 14.D_{16}$ (f) $F3A5_{16} = 1111001110100101_2$
(g) $11011001_2 = 331_8$ (h) $AB3D_{16} = 1010101100111101_2$
(i) $101111.0111_2 = 57.34_8$ (j) $15C.38_{16} = 101011100.00111_2$

2.3 (a) $1023_{16} = 1000000100011_2 = 10043_8$
(b) $7E6A_{16} = 111111001101010_2 = 77152_8$
(c) $ABCD_{16} = 1010101111001101_2 = 125715_8$
(d) $C350_{16} = 1100001101010000_2 = 141520_8$
(e) $9E36.7A_{16} = 1001111000110110.0111101_2 = 117066.364_8$
(f) $DEAD.BEEF_{16} = 1101111010101101.1011111011101111_2 = 157255.575674_8$

2.5 (a) $1101011_2 = 107_{10}$ (b) $174003_8 = 63491_{10}$
(c) $10110111_2 = 183_{10}$ (d) $67.24_8 = 55.3125_{10}$
(e) $10100.1101_2 = 20.8125_{10}$ (f) $F3A5_{16} = 62373_{10}$
(g) $12010_3 = 138_{10}$ (h) $AB3D_{16} = 43837_{10}$
(i) $7156_8 = 3694_{10}$ (j) $15C.38_{16} = 348.21875_{10}$

2.6 (a) $125_{10} = 1111101_2$ (b) $3489_{10} = 6641_8$

(c) $209_{10} = 11010001_2$ (d) $9714_{10} = 22762_8$

(e) $132_{10} = 1000100_2$ (f) $23851_{10} = 5D2B_{16}$

(g) $727_{10} = 10402_5$ (h) $57190_{10} = DF66_{16}$

(i) $1435_{10} = 2633_8$ (j) $65113_{10} = FE59_{16}$

2.7 (a)
$$\begin{array}{r} 1100010 \\ 110101 \\ +\ 11001 \\ \hline 1001110 \end{array}$$
(b)
$$\begin{array}{r} 1011000 \\ 101110 \\ -\ 100101 \\ \hline 1010011 \end{array}$$
(c)
$$\begin{array}{r} 111111110 \\ 11011101 \\ +\ \ 1100011 \\ \hline 101000000 \end{array}$$
(d)
$$\begin{array}{r} 11000000 \\ 1110010 \\ +\ 1101101 \\ \hline 11011111 \end{array}$$

2.10 (a)
$$\begin{array}{r} 1372 \\ +\ 4631 \\ \hline 59A3 \end{array}$$
(b)
$$\begin{array}{r} 4F1A5 \\ +\ B8D5 \\ \hline 5AA7A \end{array}$$
(c)
$$\begin{array}{r} F35B \\ +\ 27E6 \\ \hline 11B41 \end{array}$$
(d)
$$\begin{array}{r} 1B90F \\ +\ C44E \\ \hline 27D5D \end{array}$$

2.11

decimal	+ 18	+ 115	+79	–49	–3	–100
signed-magnitude	00010010	01110011	01001111	10110001	10000011	11100100
two’s-magnitude	00010010	01110011	01001111	11001111	11111101	10011100
one’s-complement	00010010	01110011	01001111	11001110	11111100	10011011

2.18

$$h_j = \sum_{i=0}^{3} b_{4j+i} \cdot 2^j$$

Therefore,

$$B = \sum_{i-0}^{4n-1} b_i \cdot 2^i = \sum_{i=0}^{n-1} h_i \cdot 16^i$$

$$-B = 2^{4n} - \sum_{i=0}^{4n-1} b_i \cdot 2^i = 16^n - \sum_{i=0}^{n-1} h_i \cdot 16^i$$

Suppose a $3n$-bit number B is represented by an n-digit octal number Q. Then the two’s-complement of B is represented by the 8’s-complement of Q.

2.22 Starting with the arrow pointing at any number, adding a positive number causes overflow if the arrow is advanced through the +7 to –8 transition. Adding a negative number to any number causes overflow if the arrow is not advanced through the +7 to –8 transition.

2.24 Let the binary representation of X be $x_{n-1}x_{n-2}\ldots x_1x_0$. Then we can write the binary representation of Y as $x_m x_{m-1}\ldots x_{1x_0}$, where $m = n - d$. Note that x_{m-1} is the sign bit of Y. The value of Y is

$$Y = -2^{m-1} \cdot x_{m-1} + \sum_{i=0}^{n-2} x_i \cdot 2^i$$

The value of X is

$$\begin{aligned} X &= -2^{n-1} \cdot x_{n-1} + \sum_{i=0}^{n-2} x_i \cdot 2^i \\ &= -2^{n-1} \cdot x_{n-1} + Y + 2^{m-1} \cdot x_{m-1} + \sum_{i=m-1}^{n-2} x_i \cdot 2^i \\ &= -2^{n-1} \cdot x_{n-1} + Y + 2 \cdot 2^{m-1} + \sum_{i=m}^{n-2} x_i \cdot 2^i \end{aligned}$$

Case 1 $(x_{m-1} = 0)$ In this case, $X = Y$ if and only if $-2^{n-1} \cdot x_{n-1} + \sum_{i=m}^{n-2} x_i \cdot 2^i = 0$, which is true if and only if all of the discarded bits $(x_m \ldots x_{n-1})$ are 0, the same as x_{m-1}.

Case 2 $(x_{m-1} = 1)$ In this case, $X = Y$ if and only if $-2^{n-1} \cdot x_{n-1} + 2 \cdot 2^{m-1} + \sum_{i=m}^{n-2} x_i \cdot 2^i = 0$, which is true if and only if all of the discarded bits $(x_m \ldots x_{n-1})$ are 1, the same as x_{m-1}.

2.25 If the radix point is considered to be just to the right of the leftmost bit, then the largest number is $1.11\cdots1$ and the 2’s complement of D is obtained by subtracting it from 2 (singular possessive). Regardless of the position of the radix point, the 1s’ complement is obtained by subtracting D from the largest number, which has all 1s (plural).

2.28

$$B = -b_{n-1} \cdot 2^{n-1} + \sum_{i=0}^{n-2} b_i \cdot 2^i$$

$$2B = -b_{n-1} \cdot 2^n + \sum_{i=0}^{n-2} b_i \cdot 2^{i+1}$$

Case 1 $(b_{n-1} = 0)$ First term is 0, summation terms have shifted coefficients as specified. Overflow if $b_{n-2} = 1$.

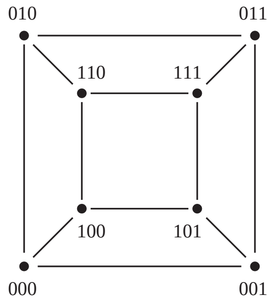
Case 2 $(b_{n-1} = 1)$ Split first term into two halves; one half is cancelled by summation term $b_{n-2} \cdot 2^{n-1}$ if $b_{n-2} = 1$. Remaining half and remaining summation terms have shifted coefficients as specified. Overflow if $b_{n-2} = 0$.

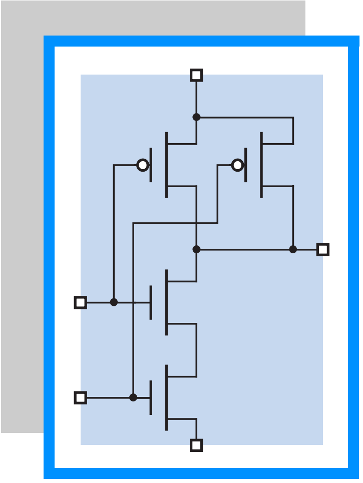
2.32 001–010, 011–100, 101–110, 111–000.

2.34 Perhaps the designers were worried about what would happen if the aircraft changed altitude in the middle of a transmission. With the Gray code, the codings of “adjacent” alitudes (at 50-foot increments) differ in only one bit. An altitude change during transmission affects only one bit, and whether the changed bit or the original is transmitted, the resulting code represents an altitude within one step (50 feet) of the original. With a binary code, larger altitude errors could result, say if a plane changed from 12,800 feet (000100000000_2) to 12,750 feet (000011111111_2) in the middle of a transmission, possibly yielding a result of 25,500 feet (000111111111_2).

2–4 DIGITAL CIRCUITS

2.37





3

DIGITAL CIRCUITS

EXERCISE SOLUTIONS

3.1 The “probably” cases may cause damage to the gate if sustained.

(a) 0	(b) 1	(c) 0	(d) undefined
(e) 1	(f) probably 1	(g) probably 0	(h) probably 0

3.3 A logic buffer is a non-linear amplifier that maps the entire set of possible analog input voltages into just two output votages, HIGH and LOW. An audio amplifier has a linear response over its specified operating range, mapping each input voltage into an output voltage that is directly proprtional to the input voltage.

3.6 From the *American Heritage Electronic Dictionary (AHED)*, copyright 1992 by Houghton Mifflin Company:

(1) A structure that can be swung, drawn, or lowered to block an entrance or a passageway.

(2) a. An opening in a wall or fence for entrance or exit. b. The structure surrounding such an opening, such as the monumental or fortified entrance to a palace or walled city.

(3) a. A means of access: the gate to riches. b. A passageway, as in an airport terminal, through which passengers proceed for embarkation.

(4) A mountain pass.

(5) The total paid attendance or admission receipts at a public event: a good gate at the football game.

(6) A device for controlling the passage of water or gas through a dam or conduit.

(7) The channel through which molten metal flows into a shaped cavity of a mold.

(8) Sports. A passage between two upright poles through which a skier must go in a slalom race.

(9) Electronics. A circuit with multiple inputs and one output that is energized only when a designated set of input pulses is received.

Well, definition (9) is closest to one of the answers that I had in mind. The other answer I was looking for is the gate of a MOS transistor.

3.14 A CMOS inverting gate has fewer transistors than a noninverting one, since an inversion comes "for free."

3.15 Simple, inverting CMOS gates generally have two transistors per input. Four examples that meet the requirements of the problem are 4-input NAND , 4-input NOR , 2-in, 2-wide AND–OR–INVERT , and 2-in, 2-wide OR–AND–INVERT.

3.18 One way is that a romance could be sparked, and the designers could end up with a lot less time to do their work. Another way is that the stray perfume in the IC production line could have contaminated the circuits used by the designers, leading to marginal operation, more debugging time by the deidicated designers, and less time for romance. By the way, the whole perfume story may be apocryphal.

3.20 Using the maximum output current ratings in both states, the HIGH-state margin is 0.69 V and the LOW-state margin is 1.02 V. With CMOS loads (output currents less than 20 μA), the margins improve to 1.349 V and 1.25 V, respectively.

3.21 The first answer for each parameter below assumes commercial operation and that the device is used with the maximum allowable (TTL) load. The number in parentheses, if any, indicates the value obtained under a lesser but specified (CMOS) load.

V_{OHmin}	3.84 V (4.4 V)
V_{IHmin}	3.15 V
V_{ILmax}	1.35 V
V_{OLmax}	0.33 V (0.1 V)
I_{Imax}	1 μA
I_{OLmax}	4 mA (20 μA)
I_{OHmax}	-4 mA (-20 μA)

3.22 Current is positive if it flows *into* a node. Therefore, an output with negative current is *sourcing* current.

3.23 The 74HC00 output drive is so weak, it's not good for driving much:

(a) Assume that in the LOW state the output pulls down to 0.33 V (the maximum V_{OL} spec). Then the output current is $(5.0\text{V})/120\Omega = 41.7\text{mA}$, which is way more than the 4-mA commercial spec.

(b) For this problem, you first have to find the Thévenin equivalent of the load, or 148.5Ω in series with 2.25V. In the HIGH state, the gate must pull the output up to 3.84V, a difference of 1.59 V across 148.5Ω, requiring 10.7 mA, which is out of spec. In the LOW state, we have a voltage drop of 2.25V – 0.33V across 148.5Ω, so the output must sink 12.9 mA, again out of spec.

3.24 (In the first printing, change "74FCT257T" to "74HC00.") The specification for the 74HC00 shows a maximum power-supply current of 10 μA when the inputs aree at 0 or 5 V, but based on the discussion in Section 3.5.3 we would expect the current to be more when the inputs are at their worst-case values (1.35 or 3.15 V). If we consider "nonlogic" input values, the maximum current will flow if the inputs are held right at the switching threshold, approximately $V_{\text{CC}}/2$.

3.26 (In the first printing, change "74FCT257T" to "74HC00.") Using the formulas on page 119, we can estimate that $R_{\text{p(on)}} = (5.0 - 3.84)/0.004 = 290\Omega$ or, using the higher value of V_{OHmin} in the spec, that $R_{\text{p(on)}} = (5.0 - 4.4)/0.00002 = 30\text{K}\Omega$. (The discrepancy shows that the output characteristic of this device is somewhat nonlinear.) We can also estimate $R_{\text{n(on)}} = 0.33/0.004 = 82.5\Omega$.

3.29 The purpose of decoupling capacitors is to provide the instantaneous power-supply current that is required during output transitions. Printed-circuit board traces have inductance, which acts as a barrier to current flow at high frequencies (fast transition rates). The farther the capacitor is from the device that needs decoupling, the larger is the instantaneous voltage drop across the connecting signal path, resulting in larger spike (up or down) in the device's power-supply voltage.

3.32 (a) 5 ns.

3.38 Smaller resistors result in shorter rise times for LOW-to-HIGH transitions but higher power consumption in the LOW state. Stated another way, larger resistors result in lower power consumption in the LOW state but longer rise times (more ooze) for LOW -to-HIGH transitions.

3.39 The resistor must drop $5.0 - 2.0 - 0.37 = 2.63\mathrm{V}$ with $5\mathrm{mA}$ of current through it. Therefore $r = 2.63 / 0.005 = 526\Omega$; a good standard value would be 510Ω.

3.41 (*The Secret of the Ooze*.) The wired output has only passive pull-up to the HIGH state. Therefore, the time for LOW-to-HIGH transitions, an important component of total delay, depends on the amount of capacitive loading and the size of the pull-up resistor. A moderate capacitive load (say, 100 pF) and even a very strong pull-up resistor (say, 150Ω) can still lead to time constants and transition times (15 ns in this example) that are longer than the delay of an additional gate with active pull-up.

3.42 The winner is 74FCT-T—48 mA in the LOW state and 15 mA in the HIGH state (see Table 3–8). TTL families don't come close.

3.46 *n* diodes are required.

3.49 For each interfacing situation, we compute the fanout in the LOW state by dividing I_{OLmax} of the driving gate by I_{ILmax} of the driven gate. Similarly, the fanout in the HIGH state is computed by dividing I_{OLmax} of the driving gate by I_{IHmax} of the driven gate. The overall fanout is the lower of these two results.

	Low-state		High-state		Overall	Excess	
Case	Ratio	Fanout	Ratio	Fanout	Fanout	State	Drive
74LS driving 74LS	$\frac{8\mathrm{mA}}{0.4\mathrm{mA}}$	20	$\frac{400\mu\mathrm{A}}{20\mu\mathrm{A}}$	20	20	none	
74LS driving 74S	$\frac{8\mathrm{mA}}{2\mathrm{mA}}$	4	$\frac{400\mu\mathrm{A}}{50\mu\mathrm{A}}$	8	4	HIGH	$200\mu\mathrm{A}$

3.50 For the pull-down, we must have at most a 0.5-V drop in order to create a V_{IL} that is no worse than a standard LS-TTL output driving the input. Since $I_{\mathrm{ILmax}} = 0.4\mathrm{mA}$, we get $R_{\mathrm{pd}} = 0.5 / 0.004 = 1250\Omega$ and $P_{pd} = V_{\mathrm{IL}}^2 / R_{\mathrm{pd}} = (0.5)^2 / 1250 = 0.2\mathrm{mW}$. (Alternatively, $P_{\mathrm{pd}} = V_{\mathrm{IL}} I_{\mathrm{IL}} = 0.5 \cdot 0.0004 = 0.2\mathrm{mW}$)

For the pull-up, we must have at most a 2.3-V drop in order to create a V_{IH} that is no worse than a standard LS-TTL output driving the input. Since $I_{\mathrm{IHmax}} = 20\mu\mathrm{A}$, we get $R_{\mathrm{pu}} = 2.3 / 0.00002 = 115\mathrm{k}\Omega$ and $P_{\mathrm{pu}} = V_{\mathrm{IH}}^2 / R_{\mathrm{pu}} = (2.3)^2 / 115000 = 0.046\mathrm{mW}$. (Alternatively, we could have calculated the result as $P_{\mathrm{pu}} = V_{\mathrm{IH}} I_{\mathrm{IH}} = 2.3 \cdot 0.00002 = 0.046\mathrm{mW}$.) The pull-up dissipates less power.

3.52 The main benefit of Schottky diodes is to prevent transistors from saturating, which allows them to switch more quickly. The main drawback is that they raise the collector-to-emitter drop across an almost-saturated transistor, which decreases LOW -state noise margin.

3.55

				LOW-state			HIGH-state		
R_{VCC} (Ω)	R_{GND} (Ω)	V_{Thev} (V)	R_{Thev} (Ω)	$V_{\mathrm{Thev}} - V_{\mathrm{OL}}$ (V)	I_{OL} (mA)	OK?	$V_{\mathrm{OH}} - V_{\mathrm{Thev}}$ (V)	I_{OH} $(\mu\mathrm{A})$	OK?
470	—	5.0	470	4.5	9.57	no	<0	—	yes
330	470	2.9375	193.875	2.4375	12.57	no	<0	—	yes

3.56

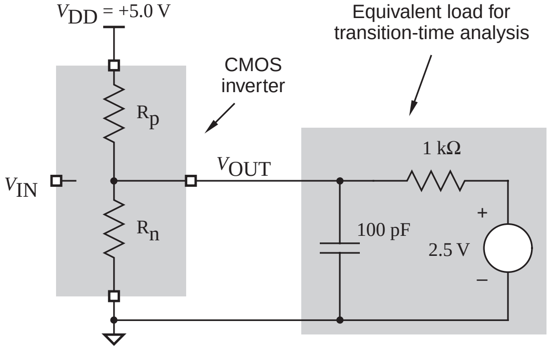
Case	LOW-state			HIGH-state		
	V_{ILmax}	$V_{\mathrm{OLmax(T)}}$	Margin	V_{IHmin}	$V_{\mathrm{OHmin(T)}}$	Margin
74HCT driving 74LS	0.8 V	0.33 V	0.47 V	2.0 V	3.84 V	1.84 V

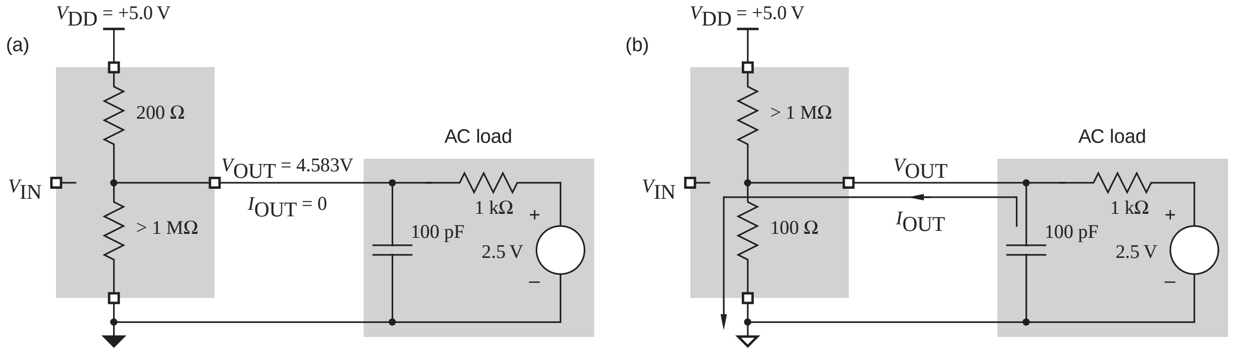
3.57 For each interfacing situation, we compute the fanout in the LOW state by dividing I_{OLmax} of the driving gate by I_{ILmax} of the driven gate. Similarly, the fanout in the HIGH state is computed by dividing I_{OHmax} of the driving gate by I_{ILmax} of the driven gate. The overall fanout is the lower of these two results.

Case	LOW-state		HIGH-state		Overall	Excess	
	Ratio	Fanout	Ratio	Fanout	Fanout	State	Drive
74HCT driving 74LS	$\frac{4\,\mathrm{mA}}{0.4\,\mathrm{mA}}$	10	$\frac{4000\ \mu\mathrm{A}}{20\ \mu\mathrm{A}}$	200	10	HIGH	3800 μA

3.64 TTL-compatible inputs have $V_{\mathrm{IHmin}} = 2.0\mathrm{V}$, and typical TTL outputs have $V_{\mathrm{OHmin}} = 2.7\mathrm{V}$. CMOS output levels are already high compared to these levels, so there’s no point in wasting silicon to make them any higher by lowering the voltage drop in the HIGH state.

3.68 Including the DC load, a CMOS output’s rise and fall times can be analyzed using the equivalent circuit shown to the right. This problem analyzes the fall time. Part (a) of the figure below shows the electrical conditions in the circuit when the output is in a steady HIGH state. Note that two resistors form a voltage divider, so the HIGH output is 4.583V, not quite 5.0 V as it was in Section 3.6.1. At time $t = 0$ the CMOS output changes to the LOW state, resulting in the situation depicted in (b). The output will eventually reach a steady LOW voltage of 0.227 V, again determined by a voltage divider.





At time $t = 0$, V_{OUT} is still 4.583 V, but the Thévenin equivalent of the voltage source and the two resistors in the LOW state is 90.9Ω in series with a 0.227-V voltage source. At time $t = \infty$, the capacitor will be discharged to the Thévenin-equivalent voltage and V_{OUT} will be 0.227 V. In between, the value of V_{OUT} is governed by an exponential law:

$$\begin{aligned} V_{\mathrm{OUT}} &= 0.227\,\mathrm{V} + (4.583 - 0.227\,\mathrm{V}) \cdot e^{-t/(R_n C_L)} \\ &= 4.356 \cdot e^{-t/(90.9 \cdot 100 \cdot 10^{-12})}\mathrm{V} \\ &= 4.356 \cdot e^{(-t)/(90.9 \cdot 10^{-9})}\mathrm{V} \end{aligned}$$

Because of the DC load resistance, the time constant is a little shorter than it was in Section 3.6.1, at 9.09 ns. To obtain the fall time, we must solve the preceding equation for $V_{\mathrm{OUT}} = 3.5$ and $V_{\mathrm{OUT}} = 1.5$, yielding

$$\begin{aligned} t &= -9.09 \cdot 10^{-9} \cdot \ln\frac{V_{\mathrm{OUT}}}{4.356} \\ t_{3.5} &= 1.99\ \mathrm{ns} \\ t_{1.5} &= 9.69\ \mathrm{ns} \end{aligned}$$

The fall time t_f is the difference between these two numbers, or 7.7 ns. This is slightly shorter than the 8.5 ns result in Section 3.6.1 because of the slightly shorter time constant.

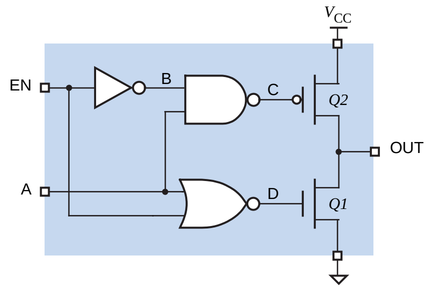
3.70 The time constant is $1\mathrm{k}\Omega \cdot 50\ \mathrm{pF} = 50\ \mathrm{ns}$. We solve the rise-time equation for the point at which V_{OUT} is 1.5 V, as on p. 118 of the text:

$$\begin{aligned} t_{1.5} &= -50 \cdot 10^{-9} \cdot \ln\frac{5.0 - 1.5}{5.0} \\ t_{1.5} &= 17.83\ \mathrm{ns} \end{aligned}$$

3.77 The LSB toggles at a rate of 16 MHz. It takes two clock ticks for the LSB to complete one cycle, so the transition frequency is 8 MHz. The MSB’s frequency is 2^7 times slower, or 62.5 KHz. The LSB’s dynamic power is the most significant, but the sum of the transitions on the higher order bits, a binary series, is equivalent to almost another 8 MHz worth of transitions on a single output bit. Including the LSB, we have almost 16 MHz, but applied to the load capacitance on just a single output. If the different ouputs actually have different load capacitances, then a weighted average would have to be used.

3.81

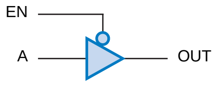
(a)



(b)

EN	A	B	C	D	*Q1*	*Q2*	OUT
L	L	H	H	H	on	off	L
L	H	H	L	L	off	on	H
H	L	L	H	L	off	off	Hi-Z
H	H	L	H	L	off	off	Hi-Z

(c)



3.84 In the situations shown in the figure, the diode with the lowest cathode voltage is forward biased, and the anode (signal C) is 0.6V higher. However, under the conditions specified in the exercise, neither diode is forward biased, no current flows through R_2, and V_{C} is 5.0V.

3.85

```
/* Transistor parameters */
#define DIODEDROP 0.6   /* volts */
#define BETA 10;
#define VCE_SAT 0.2     /* volts */
#define RCE_SAT 50      /* ohms  */
#define MAX_LEAK 0.00001 /* amperes */

main()
{
    float Vcc, Vin, R1, R2;  /* circuit parameters */
    float Ib, Ic, Vce;       /* circuit conditions */

    if (Vin < DIODEDROP) {   /* cut off */
       Ib  = 0.0;
       Ic  = Vcc/R2;  /* Tentative leakage current, limited by large R2 */
       if (Ic > MAX_LEAK) Ic = MAX_LEAK;  /* Limited by transistor */
       Vce = Vcc - (Ic * R2);
    }
    else {                   /* active or saturated */
        Ib = (Vin - DIODEDROP) / R1;
        if ((Vcc - ((BETA * Ib) * R2)) >= VCE_SAT) {   /* active */
            Ic  = BETA * Ib;
            Vce = Vcc - (Ic * R2);
        }
        else {               /* saturated */
           Vce = VCE_SAT;
           Ic  = (Vcc - Vce) / (R2 + RCE_SAT);
        }
    }
}
```

3.86 In order to turn on *Q2* fully, V_A must be 1.2 V , corresponding to the sum of the base-to-emitter drops of *Q2* and *Q5*. This could happen if both X and Y are 0.95 V or higher. In reality, a somewhat higher voltage is required, because the voltage divider consisting of *R1*, *R3*, and other components diverts current from the base of *Q2* from turning on fully until X and Y are about 1.1 V or higher (at 25°C , according to the typical characteristics graphed in the TI TTL Data Book).

3.90 When the output is HIGH, the relay coil will try to pull it to 12 volts. The high voltage will typically cause high current to flow through the output structure back into the 5-V supply and will typically blow up the output. Open-collector TTL outputs theoretically should not have this problem, but most are not designed to withstand the 12-V potential and transistor breakdown will occur. A few TTL open-collector devices are designed with “high-voltage outputs” to solve this problem.

3.92 $F = W \cdot X + Y \cdot Z$

W	X	Y	Z	G	F	W	X	Y	Z	G	F
0	0	0	0	1	0	1	0	0	0	1	0
0	0	0	1	1	0	1	0	0	1	1	0
0	0	1	0	1	0	1	0	1	0	1	0
0	0	1	1	0	1	1	0	1	1	0	1
0	1	0	0	1	0	1	1	0	0	0	1
0	1	0	1	1	0	1	1	0	1	0	1

W	X	Y	Z	G	F	W	X	Y	Z	G	F
0	1	1	0	1	0	1	1	1	0	0	1
0	1	1	1	0	1	1	1	1	1	0	1

3.96 When one module is driving HIGH and the other $n-1$ modules are output-disabled, each disabled module has a 74LS125 output sinking $20\ \mu\text{A}$ of leakage current. In addition, each of the n modules has a 74LS04 input sinking $20\ \mu\text{A}$ of leakage current. Thus, the total sink current is $(n-1+n)\cdot 20\ \mu\text{A}$. The 74LS125 can source 2.6 mA in the HIGH state, so we find

$$(n-1+n)\cdot 20\ \mu\text{A} \le 2.6\text{mA}$$

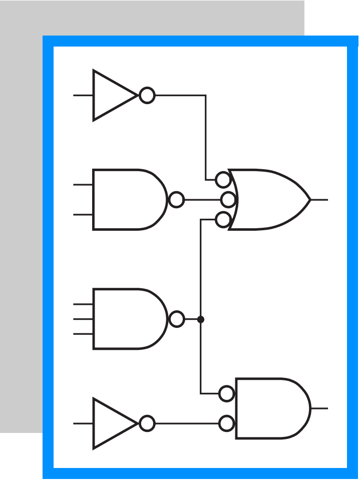
$$n \le 65$$

When one module is driving LOW and the other $n-1$ modules are output-disabled, each disabled module has a 74LS125 output sourcing $20\ \mu\text{A}$ of leakage current. In addition, each of the n modules has a 74LS04 input sourcing 0.4 mA. Thus, the total source current is $(n-1))\cdot 20\ \mu\text{A} + n\cdot 0.4\ \mu\text{A}$. The 74LS125 can sink 24 mA in the LOW state, so we find

$$(n-1)\cdot 20\mu\text{A} + n\cdot 0.4\text{mA} \le 24\text{mA}$$

$$n \le 57$$

Overall, we require $n \le 57$.



4

COMBINATIONAL LOGIC DESIGN PRINCIPLES

EXERCISE SOLUTIONS

4.2

T2:

X	X + 1	=	1
0	1	=	1
1	1	=	1

T3:

X	X + X	=	X
0	0	=	0
1	1	=	1

4.3

T3′

X	X·X	=	X
0	0	=	0
1	1	=	1

4.4

T6

X	Y	X + Y	=	Y + X
0	0	0	=	0
0	1	1	=	1
1	0	1	=	1
1	1	1	=	1

4.5 The original expression assumes precedence of $\cdot$ over $+$, that is, the expression is $X + (Y \cdot Z)$. The parenthesization must be retained for the correct result, $X' \cdot (Y' + Z')$, or the precedence must be swapped.

4.6 The answers for parts (a), (b), (c) are as follows.

$$W\cdot X\cdot Y\cdot Z\cdot(W\cdot X\cdot Y\cdot Z' + W\cdot X'\cdot Y\cdot Z + W'\cdot X\cdot Y\cdot Z + W\cdot X\cdot Y'\cdot Z)$$

$$= W\cdot X\cdot Y\cdot Z\cdot W\cdot X\cdot Y\cdot Z' + W\cdot X\cdot Y\cdot Z\cdot W\cdot X'\cdot Y\cdot Z + W\cdot X\cdot Y\cdot Z\cdot W'\cdot X\cdot Y\cdot Z + W\cdot X\cdot Y\cdot Z\cdot W\cdot X\cdot Y'\cdot Z \text{ (T8)}$$

$$= 0 + 0 + 0 + 0 \text{ (T6}', \text{T5}', \text{T2}'\text{)}$$

$$= 0 \text{ (A4}'\text{)}$$

4.7 (a)

X	Y	Z	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

(b)

W	X	Y	Z	F
0	0	0	0	1
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

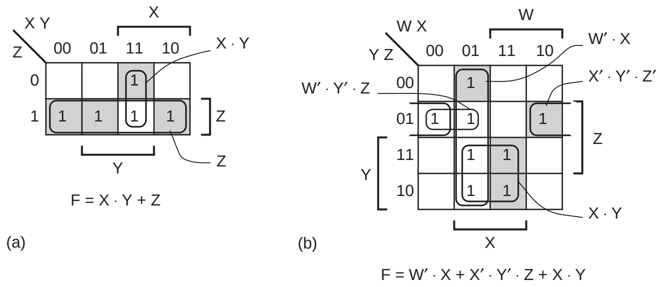
4.9

(a) $F = X'\cdot Y + Y'\cdot X = (X + Y)\cdot(X' + Y')$

(b) $F = A\cdot B = (A + B)\cdot(A + B')\cdot(A' + B)$

4.12 (1) Including inverters makes the problem too difficult. (2) In modern PLD-based designs, inverters do cost nothing and really can be ignored.

4.13



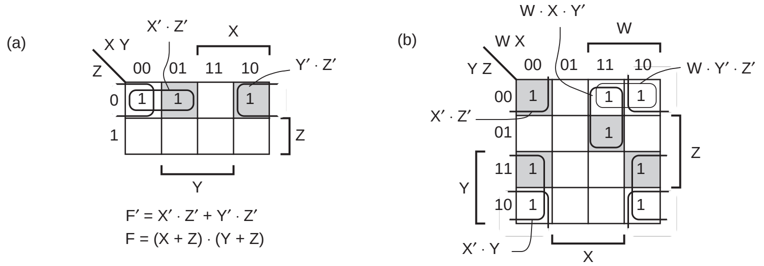
F = X · Y + Z

(a)

(b)

F = W′ · X + X′ · Y′ · Z + X · Y

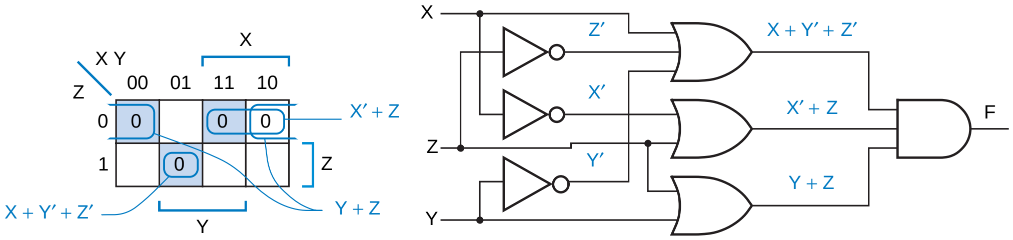
4.14



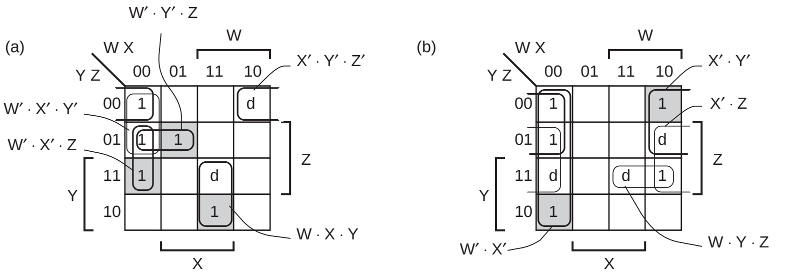
F′ = X′ · Z′ + Y′ · Z′
F = (X + Z) · (Y + Z)

F′ = X′ · Z′ + X · Y′ + W · X · Y′
F = (X + Z) · (X′ + Y) · (W′ + X′ + Y)

4.15 (a) Cost is less —one less gate input.



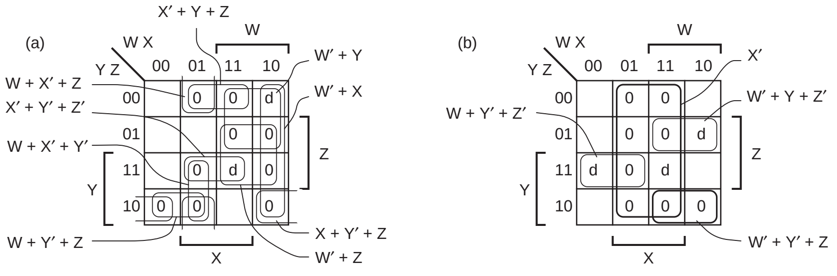
4.19



F = W′ · Y′ · Z + W′ · X′ · Z + W · X · Y
+ W′ · X′ · Y′ or X′ · Y′ · Z′

F = W′ · X′ + X′ · Y′ + X′ · Z

4.20



F = (W + X′ + Y′) · (X′ + Y + Z) · (W′ + Z′) · (X + Y′ + Z)

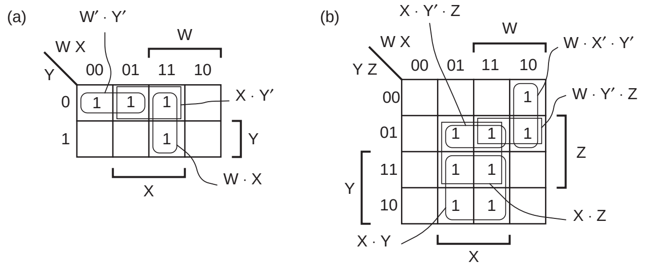
F = X′ · (W′ + Y′ + Z)

4.21 For the minimal sum-of-products expression to equal the minimal product-of-sums expression, the corresponding maps must have the opposite don’t-cares covered, so that the expressions yield the same value for the don’t-care input combinations.

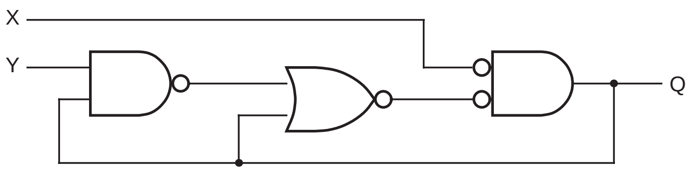
(a) Both mininal sum-of-products expressions cover cell 15; they are equal. The minimal product-of-sums expression also covers cell 15, so the expressions are not equal. The s-of-p and p-of-s expressions require the same number of gates, but the p-of-s requires one fewer input.

(b) Both mininal sum-of-products expressions cover cell 3 and 9 and not 15; they are equal. The minimal product-of-sums expression covers cell 15, and not 3 or 9, so the expressions are equal. The p-of-s expression requires fewer gates and inputs.

4.22 Consensus terms that must be added to cover the hazards are "circled" with rectangles.



4.28



Analyzing this circuit with the standard method for feedback sequential circuits (Section 5.5), we get the following excitation equation:

$$\begin{aligned} \mathsf{Q*} &= \mathsf{X}'\cdot((\mathsf{Y}\cdot\mathsf{Q})' + \mathsf{Q}) \\ &= \mathsf{X}'\cdot(\mathsf{Y}' + \mathsf{Q}' + \mathsf{Q}) \\ &= \mathsf{X}'\cdot 1 = \mathsf{X}' \end{aligned}$$

Thus, $\mathsf{Q*}$ is a function of X alone, and is independent of the circuit's previous "state."

4.29

$$\begin{aligned} \mathsf{X}\cdot 1 &= \mathsf{X} \text{ (T1}'\text{)} \\ \mathsf{X}\cdot(\mathsf{Y} + \mathsf{Y}') &= \mathsf{X} \text{ (T5)} \\ \mathsf{X}\cdot\mathsf{Y} + \mathsf{X}\cdot\mathsf{Y}' &= \mathsf{X} \text{ (T8)} \end{aligned}$$

4.31

$$\begin{aligned} (\mathsf{X} + \mathsf{Y}')\cdot\mathsf{Y} &= \mathsf{X}\cdot\mathsf{Y} + \mathsf{Y}'\cdot\mathsf{Y} \text{ (T8)} \\ &= \mathsf{X}\cdot\mathsf{Y} + \mathsf{Y}\cdot\mathsf{Y}' \text{ (T6}'\text{)} \\ &= \mathsf{X}\cdot\mathsf{Y} + 0 \text{ (T5}'\text{)} \\ &= \mathsf{X}\cdot\mathsf{Y} \text{ (T1)} \end{aligned}$$

4.35

$$\begin{aligned} \mathsf{X}_1\cdot\mathsf{X}_2\cdot\ldots\cdot\mathsf{X}_n\cdot\mathsf{X}_n &= \mathsf{X}_1\cdot\mathsf{X}_2\cdot\ldots\cdot(\mathsf{X}_n\cdot\mathsf{X}_n) \text{ (T6}', \text{T7}' \text{ as required)} \\ &= \mathsf{X}_1\cdot\mathsf{X}_2\cdot\ldots\cdot\mathsf{X}_n \text{ (T3}'\text{)} \\ \mathsf{X}_1 + \mathsf{X}_2 + \ldots + \mathsf{X}_n + \mathsf{X}_n &= \mathsf{X}_1 + \mathsf{X}_2 + \ldots + (\mathsf{X}_n + \mathsf{X}_n) \text{ (T6, T7 as required)} \\ &= (\mathsf{X}_1 + \mathsf{X}_2 + \ldots + \mathsf{X}_n) \text{ (T3)} \end{aligned}$$

4.37 Figure 3–4(d) is more appropriate, since electrically a TTL NOR gate is just the wired-AND of inverters.

4.39

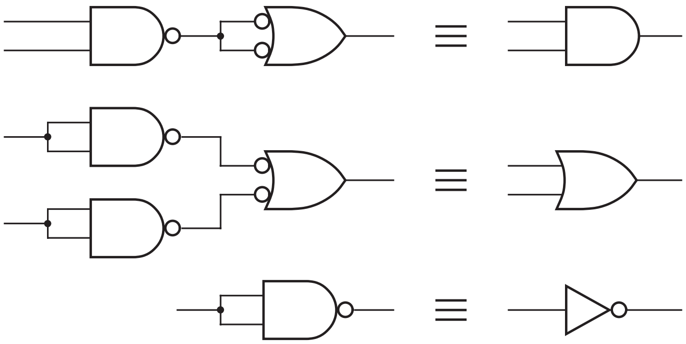
(a) True. If $\mathsf{A}\cdot\mathsf{B} = 0$ then either $\mathsf{A} = 0$ or $\mathsf{B} = 0$. If $\mathsf{A} + \mathsf{B} = 1$ then either $\mathsf{A} = 1$ or $\mathsf{B} = 1$. Therefore, $\mathsf{A}, \mathsf{B} = 0, 1$ or $1{,}0$, and $\mathsf{A} = \mathsf{B}'$.

4.41

$$
\begin{aligned}
\mathrm{F}(\mathrm{X}_1, \ldots,\mathrm{X}_i,\mathrm{X}_{i+1}, \ldots, \mathrm{X}_n) = {} & \mathrm{X}_1', \ldots, \mathrm{X}_i' \cdot \mathrm{F}(0, \ldots, 0, \mathrm{X}_{i+1}, \ldots, \mathrm{X}_n) \\
& + \mathrm{X}_1', \ldots, \mathrm{X}_i' \cdot \mathrm{F}(0, \ldots, 1, \mathrm{X}_{i+1}, \ldots, \mathrm{X}_n) \\
& \ldots \\
& + \underbrace{\mathrm{X}_1', \ldots, \mathrm{X}_i'}_{2^i \text{ minterms}} \cdot \mathrm{F}(\underbrace{1, \ldots, 1}_{2^i \text{ combs}}, \mathrm{X}_{i+1}, \ldots, \mathrm{X}_n)
\end{aligned}
$$

A dual theorem may be written based on maxterms.

4.46 Yes, 2-input NAND gates form a complete set of logic gates. We prove the result in the figure on the right by showing that these gates can be used to make 2-input AND gates, 2-input OR gates, and inverters, which form a complete set.



4.52 Take the dual, “multiply out,” and take the dual again. The result is the same as “adding out.”

4.58 (a) 16 ns. (c) 18 ns. (d) 10 ns.

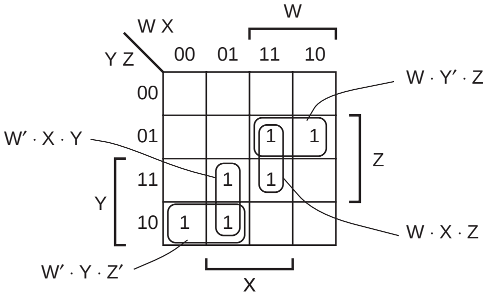
4.61 To make it easier to follow, we’ll take the dual, multiply out, and then take the dual again. Also, we’ll simplify using theorems T3′ and T6′, otherwise we’ll get a nonminimal result for sure. For Figure 4–27:

$$
\begin{aligned}
\mathrm{F} &= \mathrm{X}\cdot\mathrm{Z} + \mathrm{Y}'\cdot\mathrm{Z} + \mathrm{X}'\cdot\mathrm{Y}\cdot\mathrm{Z}' \\
\mathrm{F}^{\mathrm{D}} &= (\mathrm{X} + \mathrm{Z})\cdot(\mathrm{Y}' + \mathrm{Z})\cdot(\mathrm{X}' + \mathrm{Y} + \mathrm{Z}') \\
&= \mathrm{X}\cdot\mathrm{Y}'\cdot\mathrm{Z}' + \mathrm{X}\cdot\mathrm{Z}\cdot\mathrm{Y} + \mathrm{Z}\cdot\mathrm{Y}'\cdot\mathrm{X}' + \mathrm{Z}\cdot\mathrm{Y}'\cdot\mathrm{X}' + \mathrm{Z}\cdot\mathrm{Z}\cdot\mathrm{X}' + \mathrm{Z}\cdot\mathrm{Z}\cdot\mathrm{Y} \text{(T8, T5}', \text{T2}') \\
&= \mathrm{X}\cdot\mathrm{Y}'\cdot\mathrm{Z}' + \mathrm{X}\cdot\mathrm{Y}\cdot\mathrm{Z} + \mathrm{X}'\cdot\mathrm{Y}'\cdot\mathrm{Z}' + \mathrm{X}'\cdot\mathrm{Z} + \mathrm{Y}\cdot\mathrm{Z} \text{ (T3}', \text{T6}') \\
\mathrm{F} &= (\mathrm{X} + \mathrm{Y}' + \mathrm{Z}')\cdot(\mathrm{X} + \mathrm{Y} + \mathrm{Z})\cdot(\mathrm{X}' + \mathrm{Y}' + \mathrm{Z})\cdot(\mathrm{X}' + \mathrm{Z})\cdot(\mathrm{Y} + \mathrm{Z}) \text{ (not minimal)}
\end{aligned}
$$

For Figure 4-29:

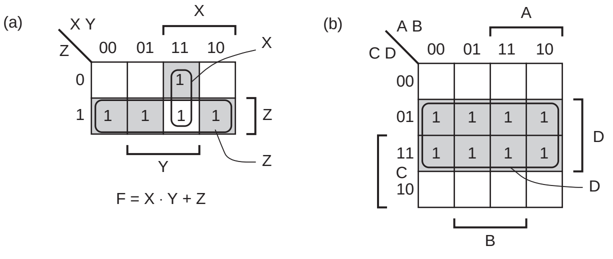
$$
\begin{aligned}
\mathrm{F} &= \mathrm{X}\cdot\mathrm{Z}' + \mathrm{Y}' \\
\mathrm{F}^{\mathrm{D}} &= (\mathrm{X} + \mathrm{Z}')\cdot\mathrm{Y}' \\
&= \mathrm{X}\cdot\mathrm{Y}' + \mathrm{Z}'\cdot\mathrm{Y}' \text{ (T8)} \\
&= \mathrm{X}\cdot\mathrm{Y}' + \mathrm{Y}'\cdot\mathrm{Z}' \text{ (T6}') \\
\mathrm{F} &= (\mathrm{X} + \mathrm{Y}')\cdot(\mathrm{Y}' + \mathrm{Z}') \text{ (minimal)}
\end{aligned}
$$

4.63



F = W′ · X · Y + W′ · Y · Z′ + W · Y′ · Z + W · X · Z

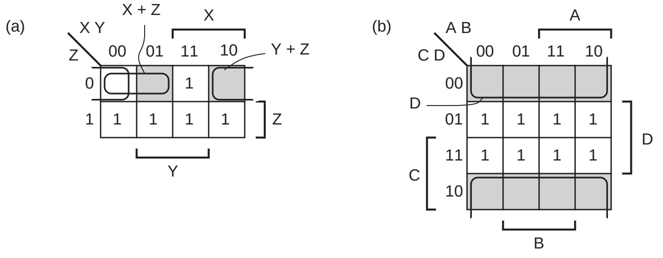
4.69 For part (d), note that it is easiest to work with the product-of-sums directly; rather than multiplying out, one simply enters the 0s on the map.



F = X · Y + Z

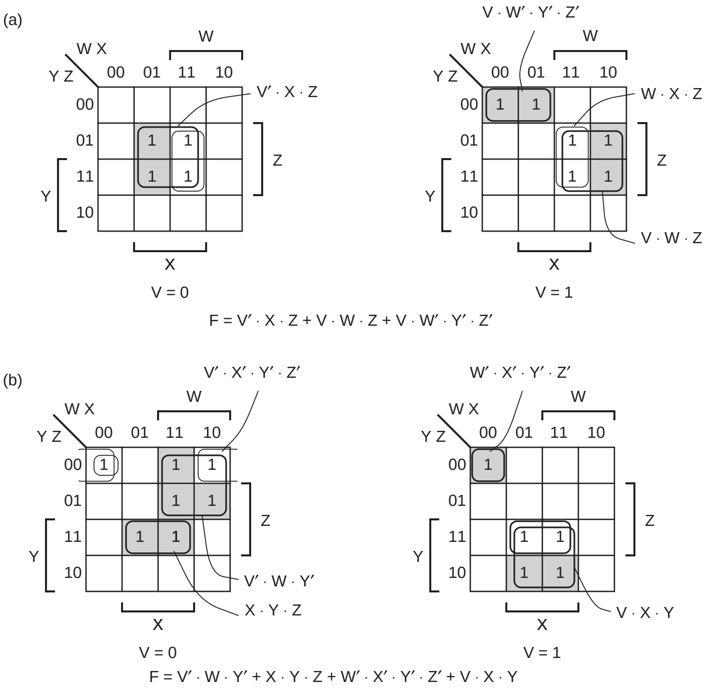
F = D

4.70 Note that in these maps are drawn for the “true” function and we’ve written sum terms for the prime implicates (the dual of prime implicants) directly, instead of using the complement method suggested in Section 4.3.6.



F = D

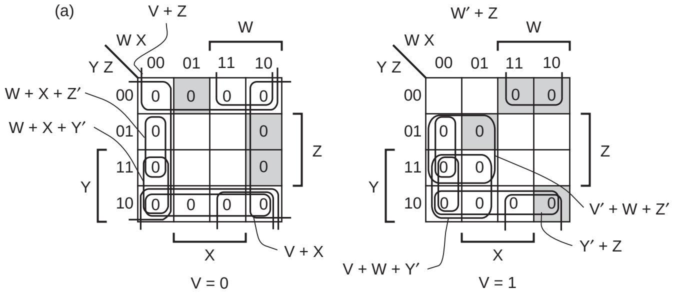
4.72



F = V′ · X · Z + V · W · Z + V · W′ · Y′ · Z′

F = V′ · W · Y′ + X · Y · Z + W′ · X′ · Y′ · Z′ + V · X · Y

4.73 Note that in these maps are drawn for the “true” function and we’ve written sum terms for the prime implicates (the dual of prime implicants) directly, instead of using the complement method suggested in Section 4.3.6.



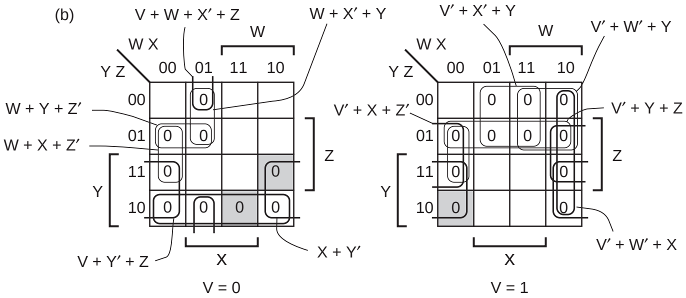
V = 0

Y Z \ W X	00	01	11	10
00	0	0	0	0
01	0			0
11	0			0
10	0	0	0	0

V = 1

Y Z \ W X	00	01	11	10
00			0	0
01	0	0		
11	0	0		
10	0	0	0	0

F = (V + Z) · (V + X) · (W′ + Z) · (V′ + W + Z′) · (Y′ + Z)



V = 0

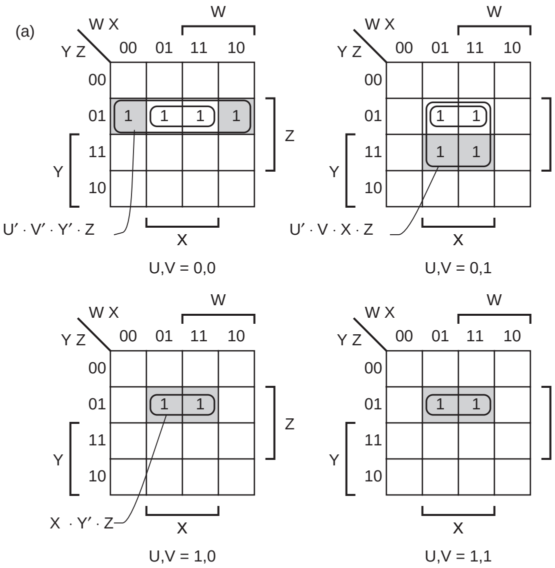
Y Z \ W X	00	01	11	10
00		0		
01	0	0		
11	0			0
10	0	0	0	0

V = 1

Y Z \ W X	00	01	11	10
00		0	0	0
01	0	0	0	0
11	0			0
10	0			0

F = (X + Y′) · (V + Y′ + Z) · (V′ + W′ + Y′) · (W + X′ + Y′) · (W + Y + Z′) or (W + X + Z′)

4.74



U,V = 0,0

Y Z \ W X	00	01	11	10
00				
01	1	1	1	1
11				
10				

U,V = 0,1

Y Z \ W X	00	01	11	10
00				
01		1	1	
11		1	1	
10				

U,V = 1,0

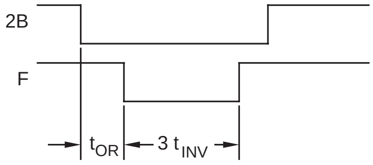
Y Z \ W X	00	01	11	10
00				
01		1	1	
11				
10				

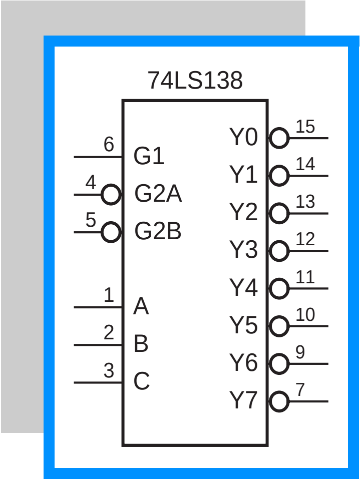
U,V = 1,1

Y Z \ W X	00	01	11	10
00				
01		1	1	
11				
10				

F = U′ · V′ · Y′ · Z + U′ · V · X · Z
+ X · Y′ · Z

4.83 The name of the circuit comes from its output equation, F = 2B OR NOT 2B.





5

COMBINATIONAL LOGIC DESIGN PRINCIPLES

EXERCISE SOLUTIONS

5.4 READY′ is an expression, with ′ being a unary operator. Use a name like READY_L or /READY instead.

5.8 Both LOW-to-HIGH and HIGH-to-LOW transitions cause positive transitions on the outputs of three gates (every second gate), and negative transitions on the other three. Thus, the total delay in either case is

$$\begin{aligned} t_p &= 3t_{\mathrm{pLH(LS00)}} + 3t_{\mathrm{pHL(LS00)}} \\ &= 3\cdot 15 + 3\cdot 15 \\ &= 90 \text{ ns} \end{aligned}$$

Since t_{pLH} and t_{pHL} for a 74LS00 are identical, the same result is obtained using a single worst-case delay of 15 ns.

5.12 The smallest typical delay through one ’LS86 for any set of conditions is 10 ns. Use the rule of thumb, “minimum equals one-fourth to one-third of typical,” we estimate 3 ns as the minimum delay through one gate. Therefore, the minimum delay through the four gates is estimated at 12 ns.

The above estimate is conservative, as it does not take into account the actual transitions in the conditions shown. For a LOW-to-HIGH input transition, the four gates have typical delays of 13, 10, 10, and 20 ns, a total of 53 ns, so the minimum is estimated at one-fourth of this or 13 ns. For a HIGH-to-LOW input transition, the four gates have typical delays of 20, 12, 12, and 13 ns, a total of 57 ns, so the minimum is estimated at 14 ns.

5.15 A decoder with active-low outputs ought to be faster, considering that this decoder structure can be implemented directly with inverting gates (which are faster than noninverting) as shown in Figures 5–35 and 5–37.

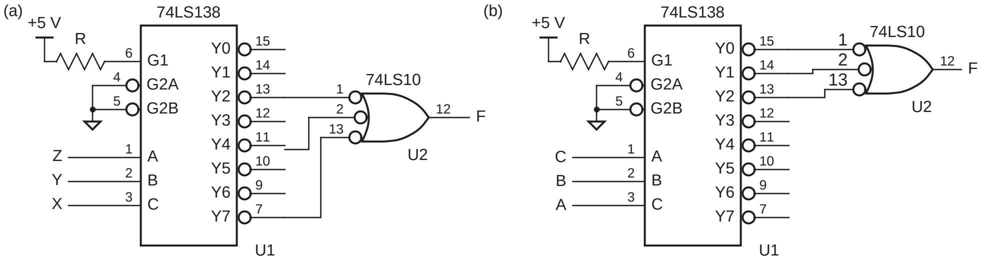
5.16 The worst-case ’138 output will have a transition in the same direction as the worst-case ’139 output, so we use t_{pHL} numbers for both, which is the worst combination. The delay through the ’139 is 38 ns, and from the active-low enable input of the ’138 is 32 ns, for a total delay of 70 ns. Using “worst-case” numbers for the parts and ignoring the structure of the circuit, an overly pessimistic result of 79 ns is obtained.

We can also work the problem with 74HCT parts. Worst-case delay through the ’139 is 43 ns, and from the active-low enable input of the ’138 is 42 ns, for a total delay of 85 ns. Ignoring the structure of the circuit, an overly pessimistic result of 88 ns is obtained.

We can also work the problem with 74FCT parts. Worst-case delay through the ’139 is 9 ns, and from the active-low enable input of the ’138 is 8 ns, for a total delay of 17 ns. Ignoring the structure of the circuit, a slightly pessimistic result of 18 ns is obtained.

Finally, we can work the problem with 74AHCT parts. Worst-case delay through the ’139 is 10.5 ns, and from the active-low enable input of the ’138 is 12 ns, for a total delay of 22.5 ns. Ignoring the structure of the circuit, a slightly pessimistic result of 23.5 ns is obtained.

5.19



5.21 Both halves of the ’139 are enabled simultaneously when EN_L is asserted. Therefore, two three-state drivers will be enabled to drive SDATA at the same time. Perhaps the designer forgot to put an extra inverter on the signal going to 1G or 2G, which would ensure that exactly one source drives SDATA at all times.

5.22 The total delay is the sum of the decoding delay through the 74LS139, enabling delay of a 74LS151, and delay through a 74LS20: $38 + 30 + 15 = 83$ ns .

5.25 The worst-case delay is the sum of the delays through an ’LS280, select-to-output through an ’LS138, and through an ’LS86: $50 + 41 + 30 = 121$ ns .

5.30 The worst-case delay is the sum of four numbers:

- In U1, the worst-case delay from any input to C4 (22 ns).
- In U2, the worst-case delay from C0 to C4 (22 ns).
- In U3, the worst-case delay from C0 to C4 (22 ns).
- In U4, the worst-case delay from C0 to any sum output (24 ns).

Thus, the total worst-case delay is 90 ns.

5.35 With the stated input combination, Y5_L is LOW and the other outputs are HIGH . We have the following cases:

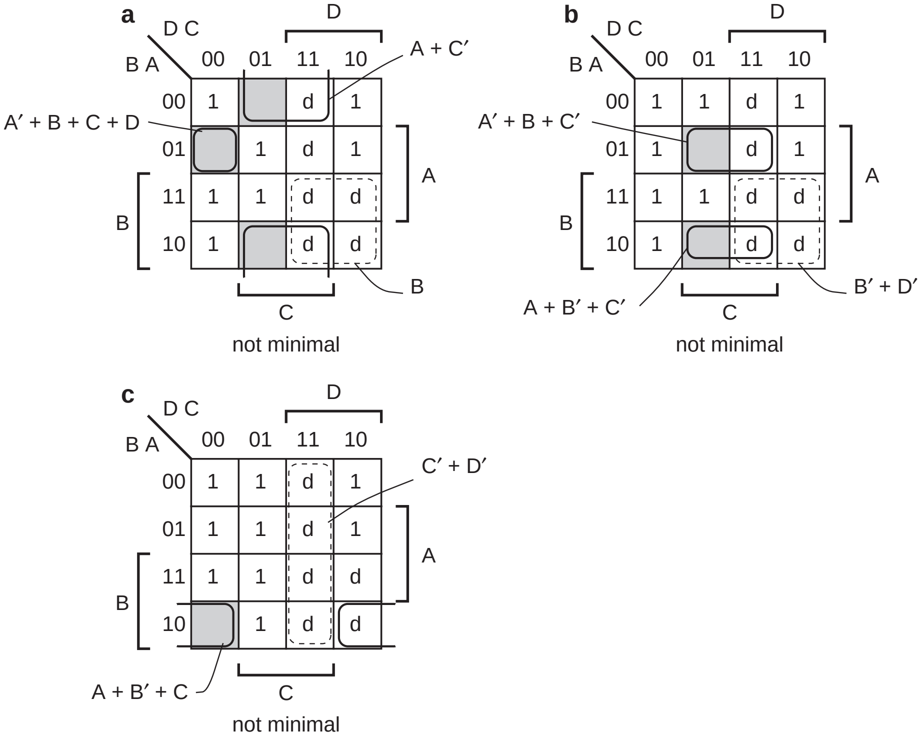
(a) Negating G2A_L or G2B_L causes Y5_L to go HIGH within 18 ns.

(b) Negating G1 causes Y5_L to go HIGH within 26 ns.

(c) Changing A or C causes Y5_L to go HIGH within 27 ns (the change propagates through 3 levels of logic internally), and causes Y4_L or Y1_L respectively to go LOW within 41 ns (2 levels).

(d) Changing B causes Y5_L to go HIGH within 20 ns (2 levels), and causes Y7_L to go LOW within 39 ns (3 levels). The delays in the ’LS138 are very strange—the worst-case t_{pHL} for 3 levels is shorter than for 2 levels!

5.39

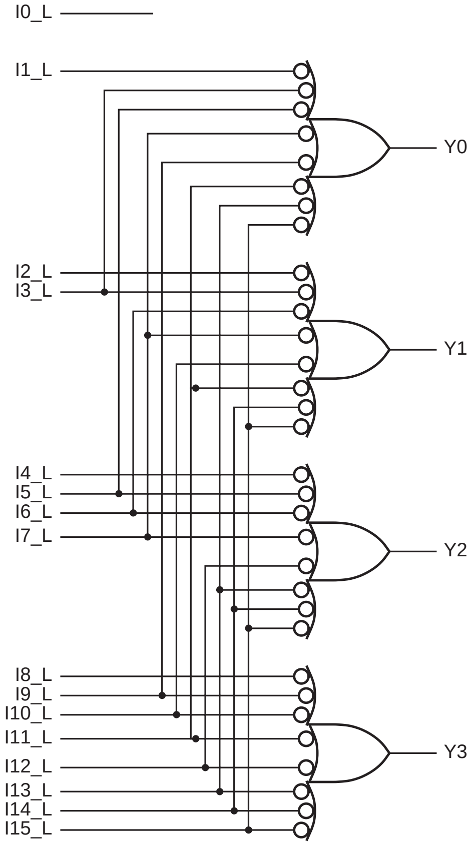


not minimal

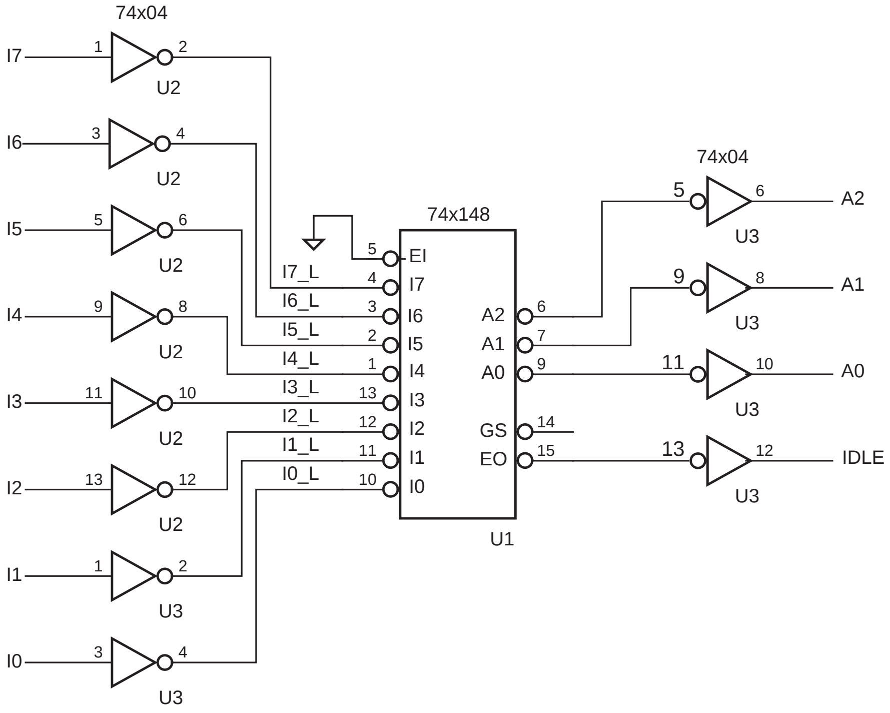
not minimal

not minimal

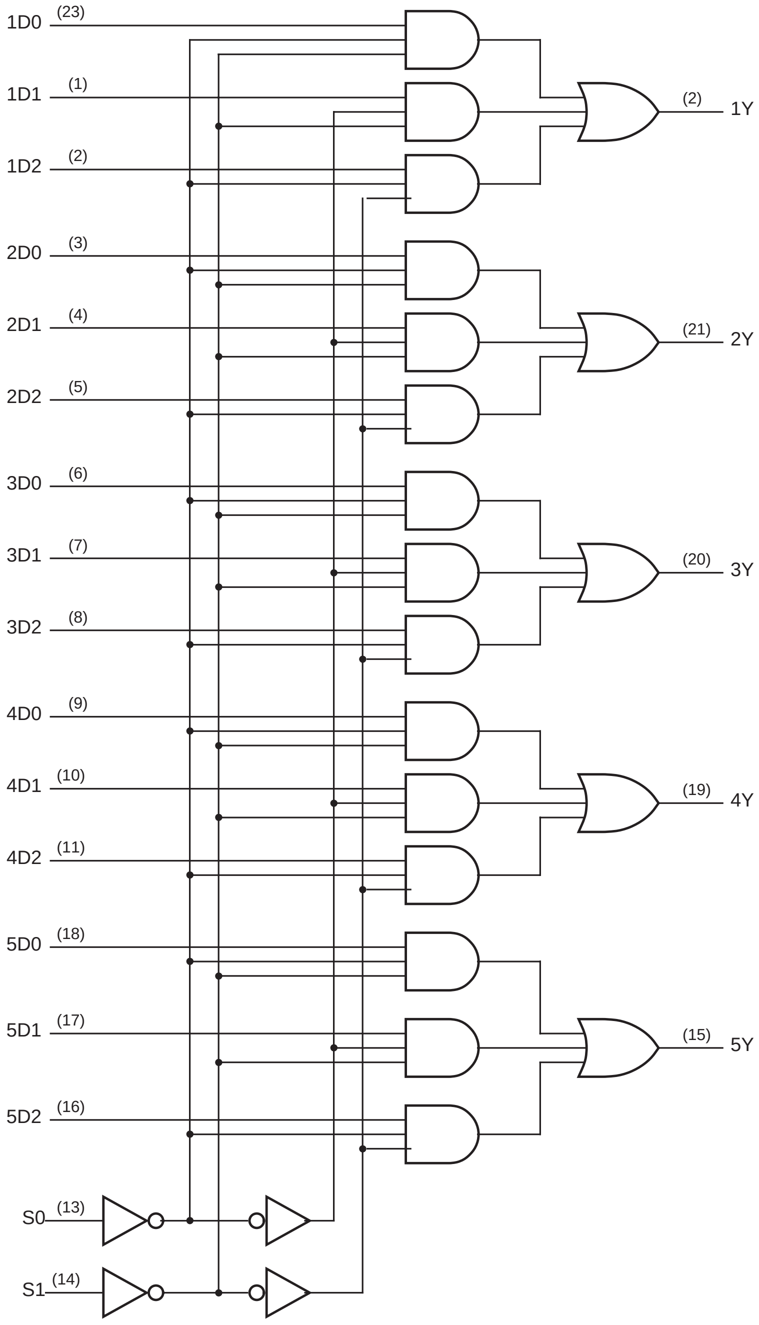
5.46 The inputs are active low and the outputs are active high in this design.



5.47

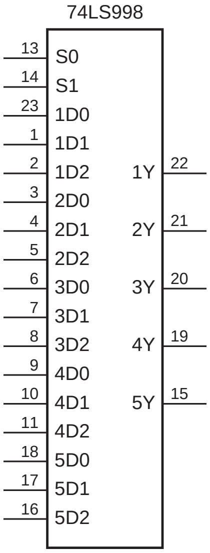

74x04
I7
1
2
U2
I6
3
4
U2
I5
5
6
U2
I4
9
8
U2
I3
11
10
U2
I2
13
12
U2
I1
1
2
U3
I0
3
4
U3
74x148
5
EI
I7_L
4
I7
I6_L
3
I6
I5_L
2
I5
I4_L
1
I4
I3_L
13
I3
I2_L
12
I2
I1_L
11
I1
I0_L
10
I0
A2
6
A1
7
A0
9
GS
14
EO
15
U1
74x04
5
6
A2
U3
9
8
A1
U3
11
10
A0
U3
13
12
IDLE
U3

5.54 An internal logic diagram for the multiplexer is shown below.



A truth table and pin assignment for the mux are shown below.

Inputs		Outputs				
S1	S0	1Y	2Y	3Y	4Y	5Y
0	0	1D0	2D0	3D0	4D0	5D0
0	1	1D1	2D1	3D1	4D1	5D1
1	0	1D2	2D2	3D2	4D2	5D2
1	1	0	0	0	0	0

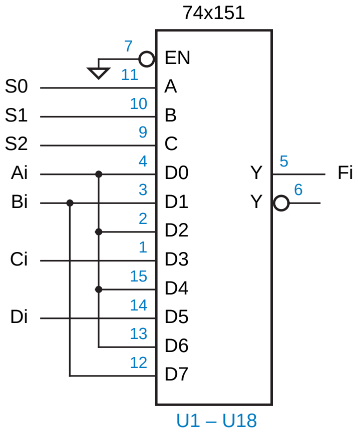


The mux can be built using a single PLD, a PAL20L8 or GAL20V8; the pin assignment shown above is based on the PLD. The corresponding ABEL program, MUX3BY5.ABL, is shown below.

```
module Mux_3x5
title '5-Bit, 3-Input Multiplexer
J. Wakerly, Marquette University'
MUX3BY5 device 'P20L8';

" Input pins
I1D0, I1D1, I1D2                     pin 23, 1, 2;
I2D0, I2D1, I2D2                     pin 3, 4, 5;
I3D0, I3D1, I3D2                     pin 6, 7, 8;
I4D0, I4D1, I4D2                     pin 9, 10, 11;
I5D0, I5D1, I5D2                     pin 18, 17, 16;
S0, S1                               pin 13, 14;
" Output pins
Y1, Y2, Y3, Y4, Y5                   pin 22, 21, 20, 19, 15;

" Set definitions
BUS0 = [I1D0,I2D0,I3D0,I4D0,I5D0];
BUS1 = [I1D1,I2D1,I3D1,I4D1,I5D1];
BUS2 = [I1D2,I2D2,I3D2,I4D2,I5D2];
OUT  = [Y1,  Y2,  Y3,  Y4,  Y5  ];

" Constants
SEL0 = ([S1,S0]==[0,0]);
SEL1 = ([S1,S0]==[0,1]);
SEL2 = ([S1,S0]==[1,0]);
IDLE = ([S1,S0]==[1,1]);

equations

OUT = SEL0 & BUS0 # SEL1 & BUS1 # SEL2 & BUS2 # IDLE & 0;

end Mux_3x5
```

5.55 This is the actual circuit of a MUX21H 2-input multiplexer cell in LSI Logic's LCA 10000 series of CMOS gate arrays. When S is 0, the output equals A; when S is 1, the output equals B.

5.60

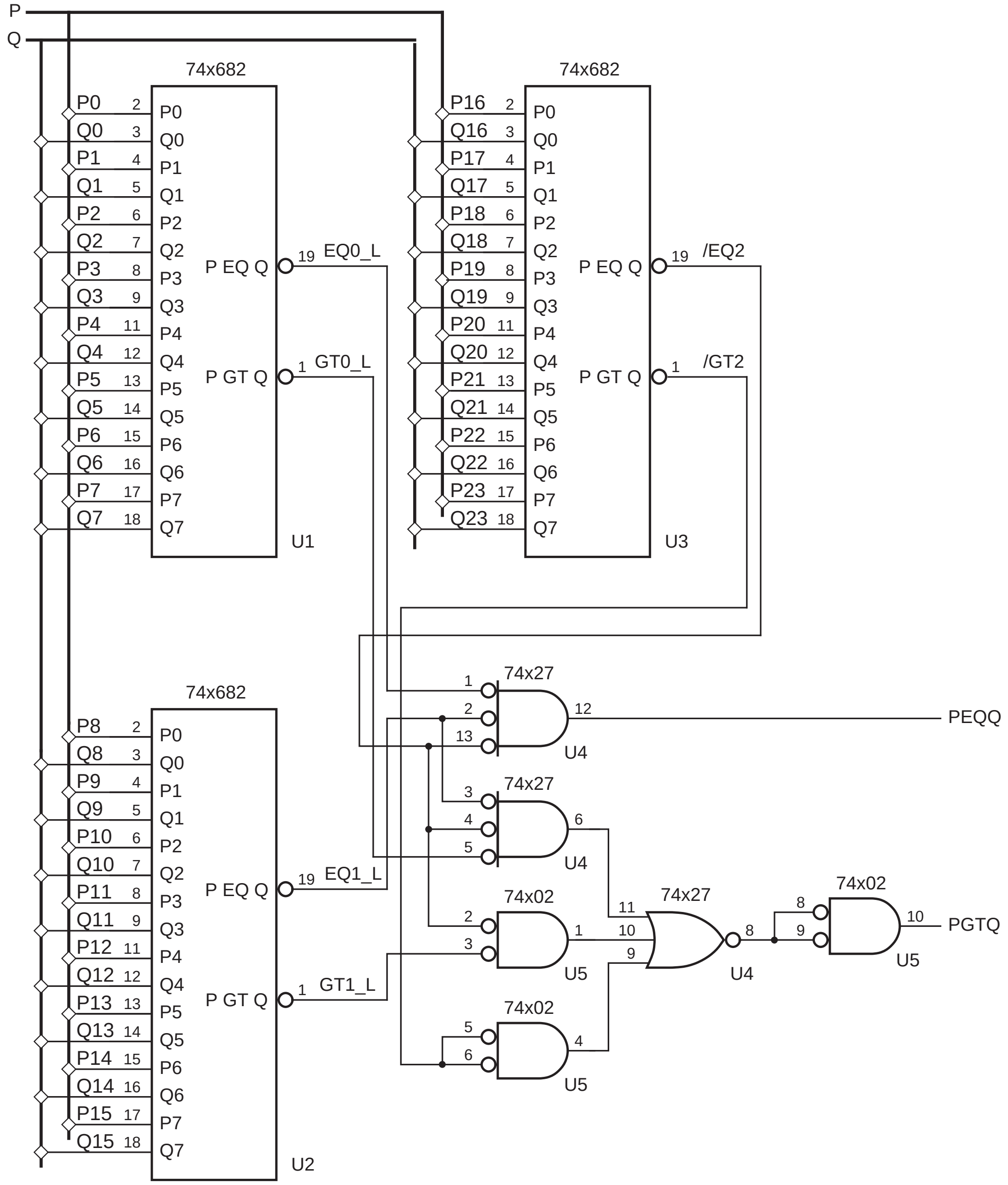


5.67 The '08 has the same pinout as the '00, but its outputs are the opposite polarity. The change in level at pin 3 of U1 is equivalent to a change at pin 4 of U2 (the input of an XOR tree), which is equivalent in turn to a change at pin 6 of U2 (the parity-generator output). Thus, the circuit simply generated and checked odd parity instead of even.

The change in level at pin 6 of U1 changed the active level of the ERROR signal.

5.69 This problem is answered in Section 5.9.3 of the text, which makes it a silly question.

5.75



5.79 The function has 65 inputs, and the worst 65-input function (a 65-input parity circuit) has 2^{65-1} terms in the minimal sum-of-products expression. Our answer can't be any worse than this, but we can do better.

The expression for c_1 has 3 product terms: $c_1 = c_0 \cdot x_0 + c_0 \cdot y_0 + x_0 \cdot y_0$

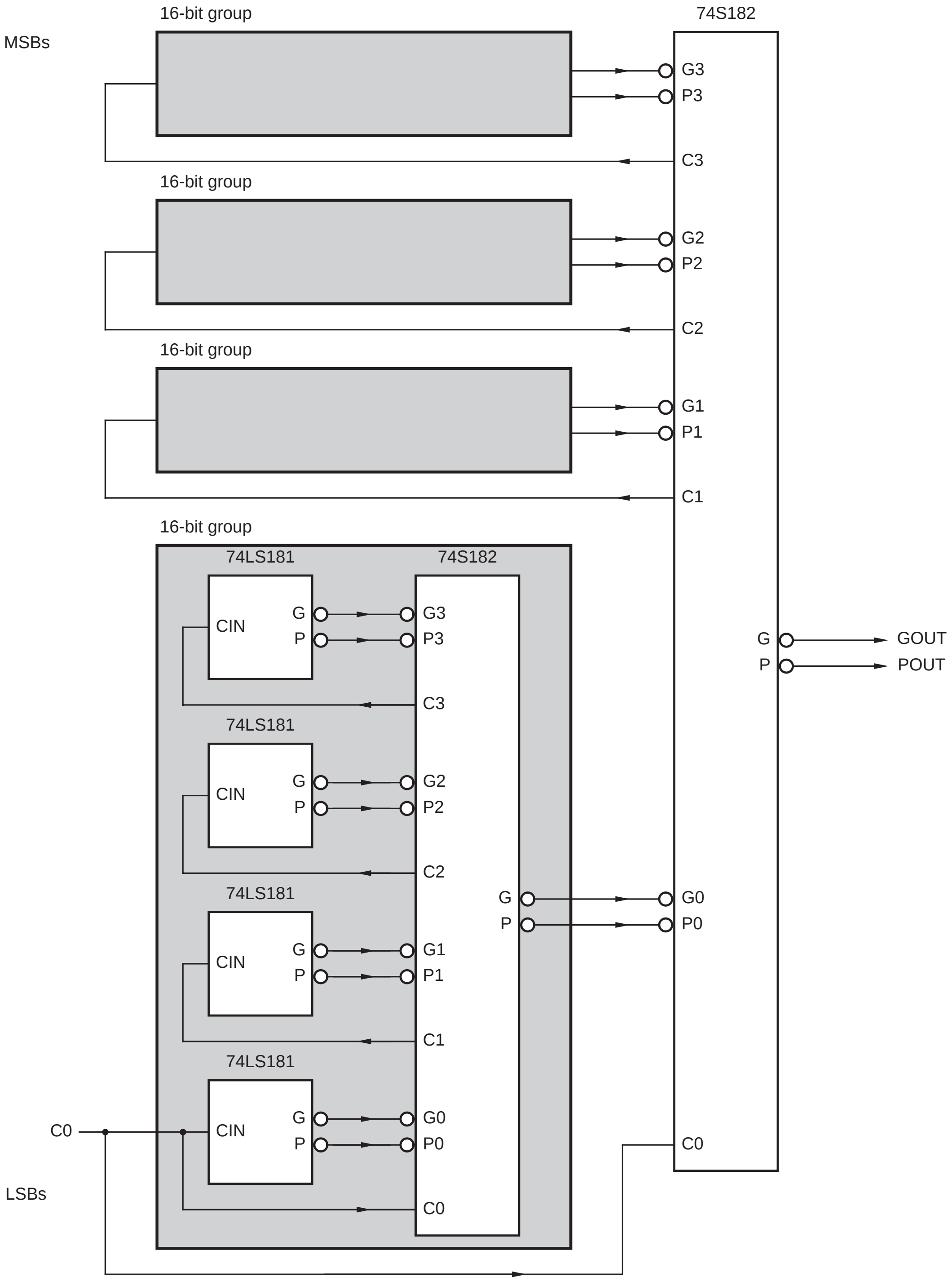
The expression for c_2 is $c_2 = c_1 \cdot x_1 + c_1 \cdot y_1 + x_1 \cdot y_1$

If we substitute our previous expression for c1 in the equation above and "multiply out," we get a result with $3 + 3 + 1 = 7$ product terms. Let us assume that no further reduction is possible.

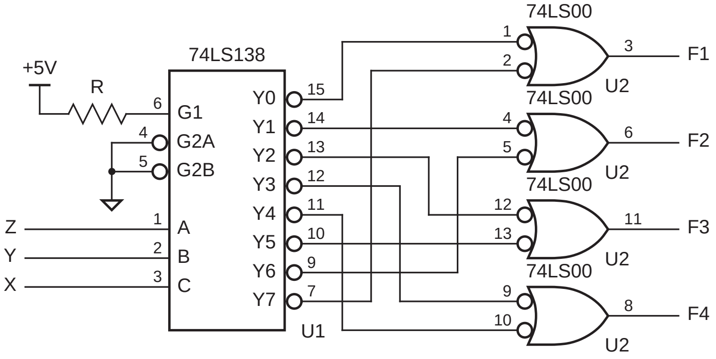
Continuing in this way, we would find that the expression for c_3 has $7 + 7 + 1 = 15$ product terms and, in general, the expression for c_i has $2^{i+1} - 1$ product terms.

Thus, the number of terms in a sum-of-products expression for c_{32} is no more than $2^{33} - 1$, fewer if minimization is possible.

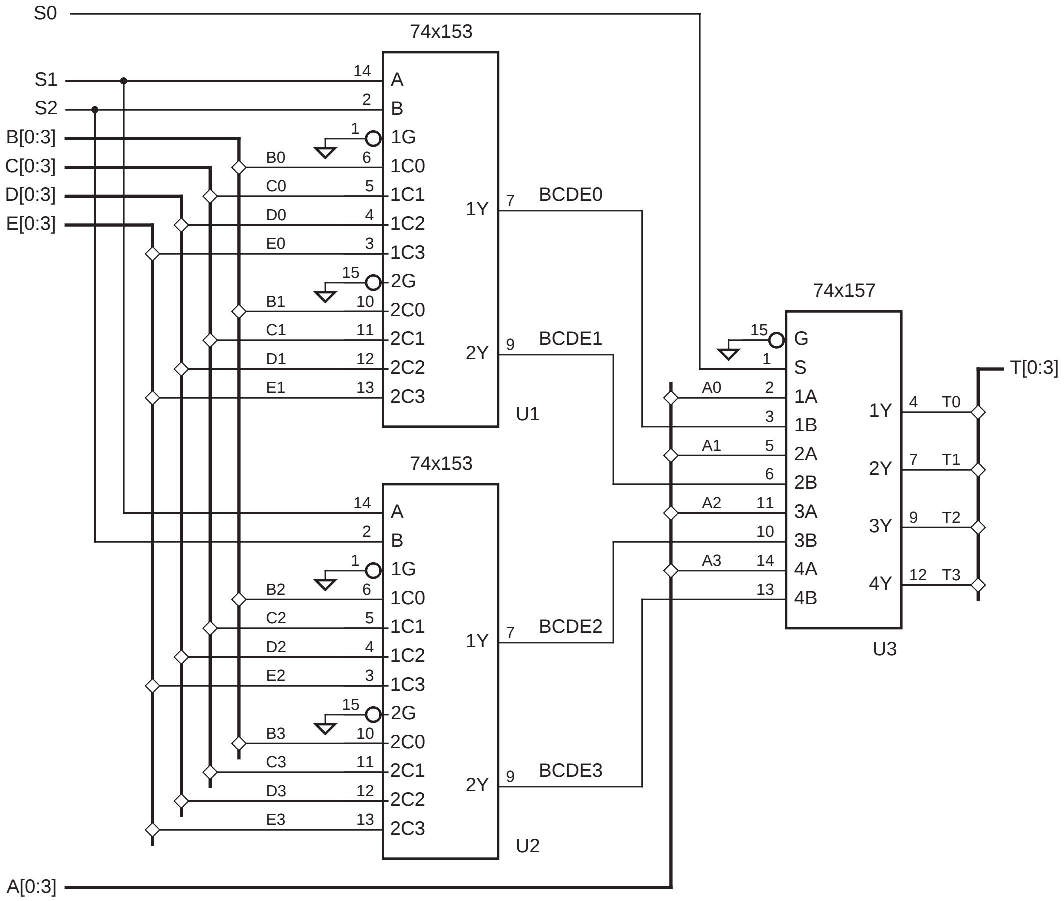
5.80


MSBs
16-bit group
74S182
G3
P3
C3
16-bit group
G2
P2
C2
16-bit group
G1
P1
C1
16-bit group
74LS181
74S182
CIN
G
P
G3
P3
C3
74LS181
CIN
G
P
G2
P2
C2
74LS181
CIN
G
P
G1
P1
C1
74LS181
CIN
G
P
G0
P0
C0
G
P
G0
P0
G
P
GOUT
POUT
C0
C0
LSBs

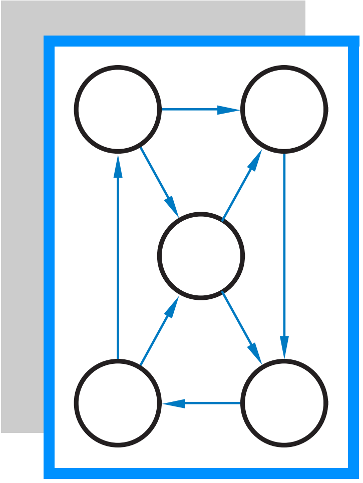
5.82



5.91



5.93 The obvious solution is to use a 74FCT682, which has a maximum delay of 11 ns to its $\overline{\mathrm{PEQQ}}$ output. However, there are faster parts in Table 5–3. In particular, the 74FCT151 has a delay of only 9 ns from any select input to Y or $\overline{\mathrm{Y}}$. To take advantage of this, we use a ’138 to decode the SLOT inputs statically and apply the resulting eight signals to the data inputs of the ’151. By applying GRANT[2–0] to the select inputs of the ’151, we obtain the MATCH_L output (as well as an active-high MATCH, if we need it) in only 9 ns!

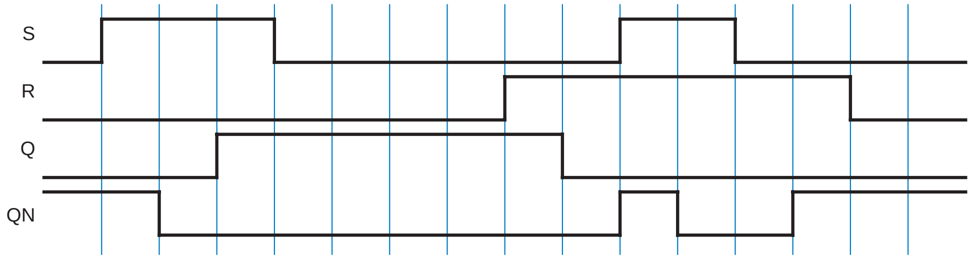


7

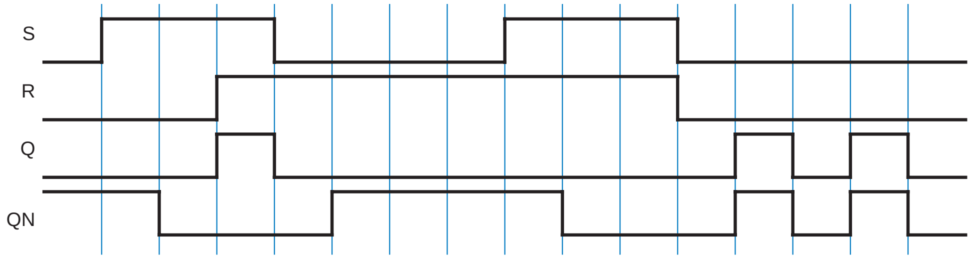
Sequential Logic Design Principles

EXERCISE SOLUTIONS

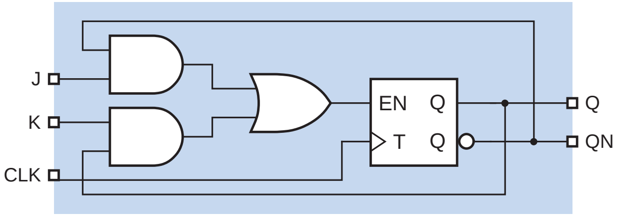
7.2



7.3 The latch oscillates if S and R are negated simultaneously. Many simulator programs will exhibit this same behavior when confronted with such input waveforms.



7.5



7.8 Just tie the J and $\overline{\text{K}}$ inputs together and use as the D input.

7.9 Excitation and output equations:

$$\begin{aligned} D1 &= Q1' + Q2 \\ D2 &= Q2' \cdot X \\ Z &= Q1 + Q2' \end{aligned}$$

Excitation/transition table; state/output table:

	EN	
Q1 Q2	0	1
00	10	11
01	10	10
10	00	01
11	10	10
	Q1* Q2*	

	EN		
S	0	1	Z
A	C	D	1
B	C	C	0
C	A	B	1
D	C	C	1
	S*		

7.15 Excitation equations:

$$\begin{aligned} D2 &= (Q1 \oplus Q0) \oplus (Q1' \cdot Q2') \\ D1 &= Q2 \\ D0 &= Q1 \end{aligned}$$

Excitation/transition table; state table:

Q2 Q1 Q0	Q2* Q1* Q0*
000	100
001	000
010	101
011	001
100	010
101	110
110	111
111	011

S	S*
A	E
B	A
C	F
D	B
E	C
F	G
G	H
H	D

7.18 Excitation and output equations:

$$\begin{aligned} J0 &= K0 = EN \\ J1 &= K1 = Q0 \cdot EN \\ MAX &= EN \cdot Q1 \cdot Q0 \end{aligned}$$

Note that the characteristic equation for a J-K flip-flop is $Q* = J \cdot Q' + K' \cdot Q$. Thus, we obtain the following transition equations:

$$Q0* = EN' \cdot Q0 + EN \cdot Q0'$$

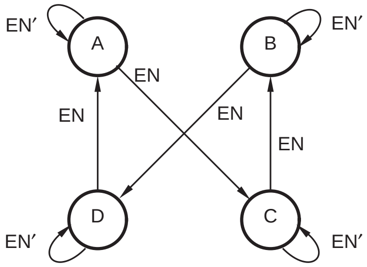
$$Q1* = EN' \cdot Q1 + EN \cdot Q0 \cdot Q1' + EN \cdot Q0' \cdot Q1$$

Transition/output table; state/output table:

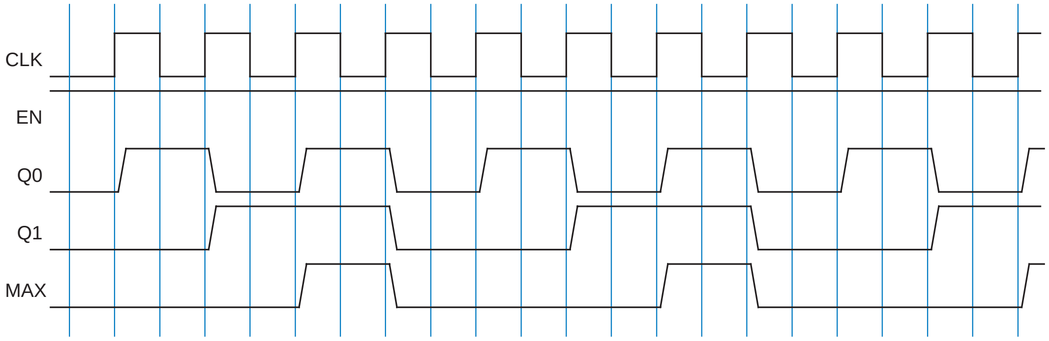
	EN	
Q1 Q2	0	1
00	00,0	01,0
01	01,0	10,0
10	10,0	11,0
11	11,0	00,1
	Q1* Q2*, MAX	

	EN	
S	0	1
A	A,0	B,0
B	B,0	C,0
C	C,0	D,0
D	D,0	A,1
	S*, MAX	

State diagram:



Timing diagram:



7.20 This can be done algebraically. If all of the input combinations are covered, the logical sum of the expressions on all the transitions leaving a state must be 1. If the sum is not 1, it is 0 for all input combinations that are uncovered. For double-covered input combinations, we look at all possible pairs of transitions leaving a state. The product of a pair of transition equations is 1 for any double-covered input combinations.

(a) State D, $Y = 0$ is uncovered.

(b) State A, $(X{+}Z') = 0$ is uncovered. State B, $W = 1$ is double-covered; $(W{+}X) = 0$ is uncovered. State C, $(W{+}X{+}Y{+}Z) = 0$ is uncovered; $(W \cdot X + W \cdot Y + Z \cdot Y + Z \cdot X) = 1$ is double covered. State D, $(X \cdot Y + \cdot X' \cdot Z + W \cdot Z) = 0$ is uncovered; $(W \cdot X' \cdot Z + W \cdot X \cdot Y \cdot Z) = 1$ is double-covered;

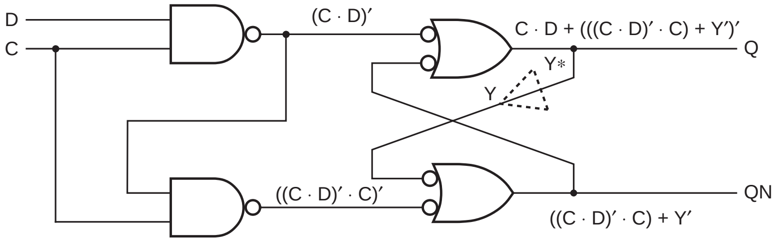
7.21 Table 9–4 on page 804 shows an output-coded state assignment. Here is a corresponding transition list:

S	L3Z	L2Z	L1Z	R1Z	R2Z	R3Z	Transition expression	S*	L3Z*	L2Z*	L1Z*	R1Z*	R2Z*	R3Z*
IDLE	0	0	0	0	0	0	(LEFT + RIGHT + HAZ)′	IDLE	0	0	0	0	0	0
IDLE	0	0	0	0	0	0	LEFT · HAZ′ · RIGHT′	L1	0	0	1	0	0	0
IDLE	0	0	0	0	0	0	HAZ + LEFT · RIGHT	LR3	1	1	1	1	1	1
IDLE	0	0	0	0	0	0	RIGHT · HAZ′ · LEFT′	R1	0	0	0	1	0	0
L1	0	0	1	0	0	0	HAZ′	L2	0	1	1	0	0	0
L1	0	0	1	0	0	0	HAZ	LR3	1	1	1	1	1	1
L2	0	1	1	0	0	0	HAZ′	L3	1	1	1	0	0	0
L2	0	1	1	0	0	0	HAZ	LR3	1	1	1	1	1	1
L3	1	1	1	0	0	0	1	IDLE	0	0	0	0	0	0
R1	0	0	0	1	0	0	HAZ′	R2	0	0	0	1	1	0
R1	0	0	0	1	0	0	HAZ	LR3	1	1	1	1	1	1
R2	0	0	0	1	1	0	HAZ′	R3	0	0	0	1	1	1
R2	0	0	0	1	1	0	HAZ	LR3	1	1	1	1	1	1
R3	0	0	0	1	1	1	1	IDLE	0	0	0	0	0	0
LR3	1	1	1	1	1	1	1	IDLE	0	0	0	0	0	0

The excitation equations and circuit diagram follow directly from this transition list.

7.25 The minimum setup time is the clock period times the duty cycle. That is, the minimum setup time is the time that the clock is 1.

7.27 As shown in Section 7.9.1, the excitation equation for the latch of Figure 7–72 is $Y* = C \cdot D + C' \cdot Y + D \cdot Y$ Below, we analyze Figure X7.27 in the same way:



The feedback equation is

$$\begin{aligned} Y* &= C \cdot D + (((C \cdot D)' \cdot C) + Y')' \\ &= (C \cdot D) + ((C \cdot D)' \cdot C)' \cdot Y \\ &= C \cdot D + ((C \cdot D) + C') \cdot Y \\ &= C \cdot D + (D + C') \cdot Y \\ &= C \cdot D + D \cdot Y + C' \cdot Y \end{aligned}$$

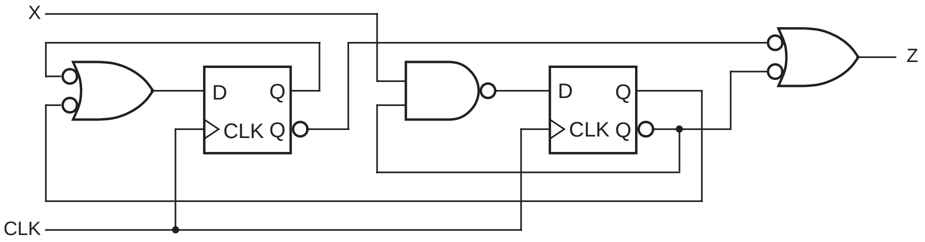
The feedback equations are the same, and so the circuits have identical steady-state behavior.

The circuit in Figure X7.27 is better in two ways. It uses one less gate, and it has one less load on the D input.

7.29 The AND gate in the original circuit is replaced with a NAND gate. As a result, the second flip-flop stores the opposite of the value stored in the original circuit; to compensate, swap connections to its Q and QN outputs.

The OR gates in the original circuit are also replaced with NAND gates. As a result, each input must be connected to a signal of the opposite polarity as before, that is, to the complementary flip-flop output. In the case of connections to the second flip-flop, we swapped outputs twice, so the connections remain the same.

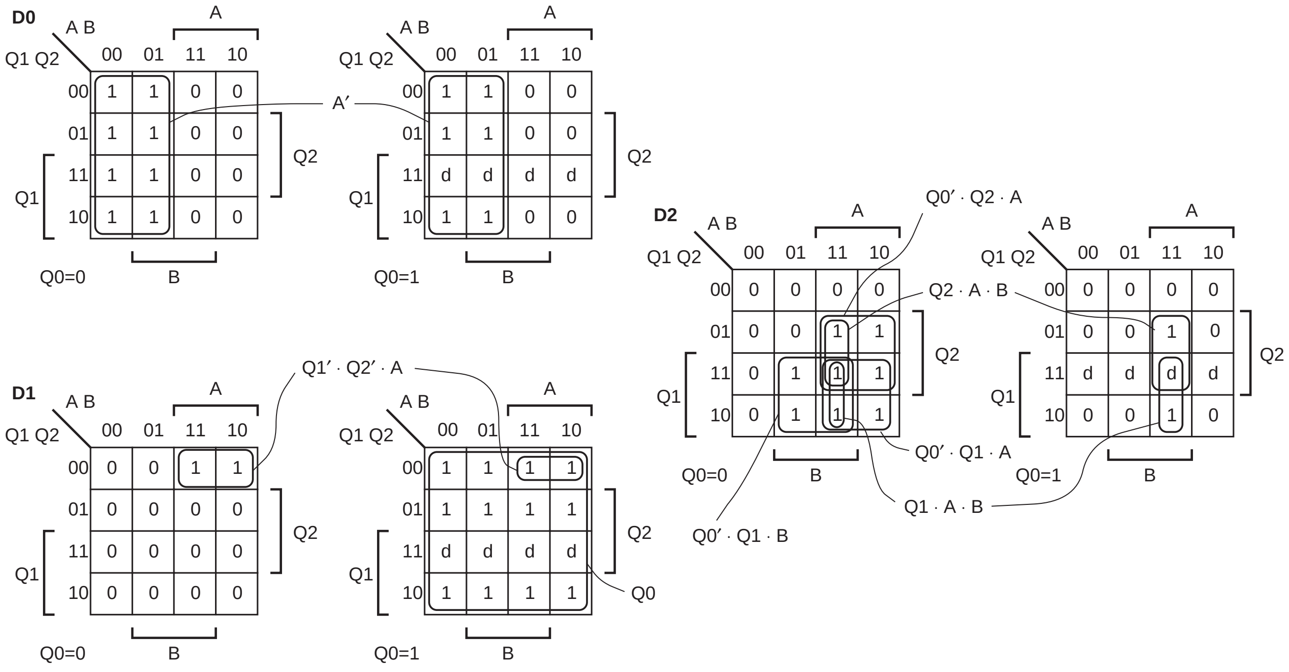
The final circuit below uses three 2-input NAND gates.



7.45 A transition table corresponding to the state table is shown below:

	A B				
Q2 Q1 Q0	00	01	11	10	Z
000	001	001	010	010	0
001	011	011	010	010	0
010	001	001	100	100	0
011	011	011	110	010	1
100	001	101	100	100	1
101	011	011	110	010	1
110	001	101	100	100	1
	Q2* Q1* Q0*				

This table leads to the following Karnaugh maps for the excitation logic, assuming a "minimal cost" treatment of unused states.



D0, Q0=0

Q1 Q2 \ A B	00	01	11	10
00	1	1	0	0
01	1	1	0	0
11	1	1	0	0
10	1	1	0	0

D0, Q0=1

Q1 Q2 \ A B	00	01	11	10
00	1	1	0	0
01	1	1	0	0
11	d	d	d	d
10	1	1	0	0

D1, Q0=0

Q1 Q2 \ A B	00	01	11	10
00	0	0	1	1
01	0	0	0	0
11	0	0	0	0
10	0	0	0	0

D1, Q0=1

Q1 Q2 \ A B	00	01	11	10
00	1	1	1	1
01	1	1	1	1
11	d	d	d	d
10	1	1	1	1

D2, Q0=0

Q1 Q2 \ A B	00	01	11	10
00	0	0	0	0
01	0	0	1	1
11	0	1	1	1
10	0	1	1	1

D2, Q0=1

Q1 Q2 \ A B	00	01	11	10
00	0	0	0	0
01	0	0	1	0
11	d	d	d	d
10	0	0	1	0

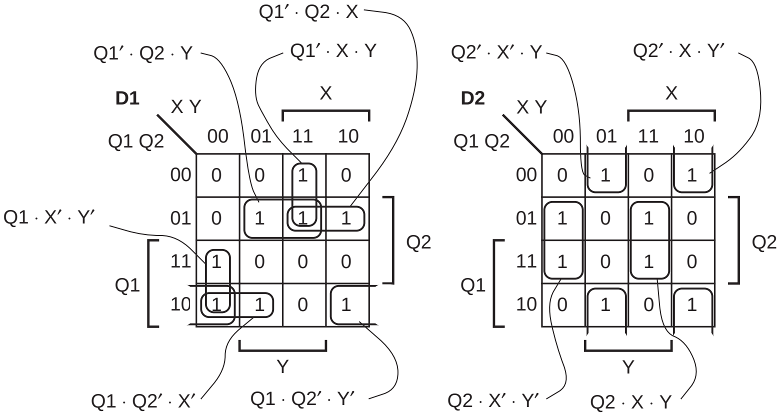
The resulting excitation equations are

$$\begin{aligned} D0 &= A' \\ D1 &= Q1' \cdot Q2' \cdot A + Q0 \\ D2 &= Q2 \cdot A \cdot B + Q0' \cdot Q2 \cdot A + Q0' \cdot Q1 \cdot A + Q1 \cdot A \cdot B + Q0' \cdot Q1 \cdot B \end{aligned}$$

Ignoring inverters, a circuit realization with the new equations requires one 2-input gate, six 3-input gates, and one 5-input gate. This is more expensive than Figure 7–54, by four gates.

7.49 The new state assignment yields the following transition/excitation table and Karnaugh maps:

	X Y				
Q1 Q0	00	01	11	10	Z
00	00	01	11	01	1
01	01	11	10	11	0
11	11	10	00	10	0
10	10	00	01	00	0
	Q2* Q1* or D1 D2				



This yields the following excitation equations:

$$\begin{aligned} D1 &= Q1' \cdot Q2 \cdot X + Q1' \cdot Q2 \cdot Y + Q1' \cdot X \cdot Y + Q1 \cdot Q2' \cdot X' + Q1 \cdot Q2' \cdot Y' + Q1 \cdot X' \cdot Y' \\ D2 &= Q2 \cdot X \cdot Y + Q2' \cdot X \cdot Y' + Q2' \cdot X' \cdot Y + Q2 \cdot X' \cdot Y' \end{aligned}$$

Compared with the results of original state assigment, these equations require two more 3-input AND gates, plus a 6-input OR gate inplace of a 4-input one. However, if we are not restricted to a sum-of-products realization, using the fact that $D2 = Q2 \oplus X \oplus Y$ might make this realization less expensive when discrete gates are used.

7.57 Here is the transition list:

S	Q2	Q1	Q0	Transition expression	S*	Q2*	Q1*	Q0*
IDLE	0	0	0	(LEFT+RIGHT+HAZ)′	IDLE	0	0	0
IDLE	0	0	0	LEFT	L1	0	0	1
IDLE	0	0	0	HAZ	LR3	1	0	0
IDLE	0	0	0	RIGHT	R1	1	0	1
L1	0	0	1	1	L2	0	1	1
L2	0	1	1	1	L3	0	1	0
L3	0	1	0	1	IDLE	0	0	0
R1	1	0	1	1	R2	1	1	1
R2	1	1	1	1	R3	1	1	0
R3	1	1	0	1	IDLE	0	0	0
LR3	1	0	0	1	IDLE	0	0	0

The transition/excitation and output equations below follow directly from the transition list.

$$\begin{aligned} \mathrm{D2} = \mathrm{Q2*} &= \mathrm{Q2'}\cdot\mathrm{Q1'}\cdot\mathrm{Q0'}\cdot\mathrm{HAZ} \\ &\quad + \mathrm{Q2'}\cdot\mathrm{Q1'}\cdot\mathrm{Q0'}\cdot\mathrm{RIGHT} \\ &\quad + \mathrm{Q2}\cdot\mathrm{Q1'}\cdot\mathrm{Q0} \\ &\quad + \mathrm{Q2}\cdot\mathrm{Q1}\cdot\mathrm{Q0} \\ &= \mathrm{Q2'}\cdot\mathrm{Q1'}\cdot\mathrm{Q0'}\cdot(\mathrm{HAZ}+\mathrm{RIGHT}) + \mathrm{Q2}\cdot\mathrm{Q0} \end{aligned}$$

$$\begin{aligned} \mathrm{D1} = \mathrm{Q1*} &= \mathrm{Q2'}\cdot\mathrm{Q1'}\cdot\mathrm{Q0} \\ &\quad + \mathrm{Q2'}\cdot\mathrm{Q1}\cdot\mathrm{Q0} \\ &\quad + \mathrm{Q2}\cdot\mathrm{Q1'}\cdot\mathrm{Q0} \\ &\quad + \mathrm{Q2}\cdot\mathrm{Q1}\cdot\mathrm{Q0} \\ &= \mathrm{Q0} \end{aligned}$$

$$\begin{aligned} \mathrm{D0} = \mathrm{Q0*} &= \mathrm{Q2'}\cdot\mathrm{Q1'}\cdot\mathrm{Q0'}\cdot\mathrm{LEFT} \\ &\quad + \mathrm{Q2'}\cdot\mathrm{Q1'}\cdot\mathrm{Q0'}\cdot\mathrm{RIGHT} \\ &\quad + \mathrm{Q2'}\cdot\mathrm{Q1'}\cdot\mathrm{Q0} \\ &\quad + \mathrm{Q2}\cdot\mathrm{Q1'}\cdot\mathrm{Q0} \\ &= \mathrm{Q2'}\cdot\mathrm{Q1'}\cdot\mathrm{Q0'}\cdot(\mathrm{LEFT}+\mathrm{RIGHT}) + \mathrm{Q1'}\cdot\mathrm{Q0} \end{aligned}$$

Starting from the IDLE state, the following transitions may be observed:

S	Q2	Q1	Q0	LEFT	RIGHT	HAZ	Q2*	Q1*	Q0*	S*
IDLE	0	0	0	1	0	1	1	0	1	R1
IDLE	0	0	0	0	1	1	1	0	1	R1
IDLE	0	0	0	1	1	0	1	0	1	R1
IDLE	0	0	0	1	1	1	1	0	1	R1

For each input combination, the machine goes to the R1 state, because R1’s encoding is the logical OR of the encodings of the two or three next states that are specified by the ambiguous state diagram.

The behavior aboveis not so good and is a result of synthesis choices—state encoding and logic synthesis method. If a different state encoding were used for R1, or if a different synthesis method were used (e.g., product-of-s-terms), then the results could be different. For example, starting with the transition list given earlier, we can obtain the following set of transition equations using the product-of-s-terms method:

$$\begin{aligned}
\mathrm{D2} = \mathrm{Q2}^* &= (\mathrm{Q2} + \mathrm{Q1} + \mathrm{Q0} + \mathrm{LEFT} + \mathrm{RIGHT} + \mathrm{HAZ}) \\
&\quad \cdot(\mathrm{Q2} + \mathrm{Q1} + \mathrm{Q0} + \mathrm{LEFT}') \\
&\quad \cdot(\mathrm{Q2} + \mathrm{Q1} + \mathrm{Q0}') \\
&\quad \cdot(\mathrm{Q2} + \mathrm{Q1}' + \mathrm{Q0}') \\
&\quad \cdot(\mathrm{Q2} + \mathrm{Q1}' + \mathrm{Q0}) \\
&\quad \cdot(\mathrm{Q2}' + \mathrm{Q1}' + \mathrm{Q0}) \\
&\quad \cdot(\mathrm{Q2}' + \mathrm{Q1} + \mathrm{Q0}) \\
&= (\mathrm{Q2} + \mathrm{Q1} + \mathrm{RIGHT} + \mathrm{HAZ})\cdot(\mathrm{Q2} + \mathrm{Q1} + \mathrm{LEFT}')\cdot(\mathrm{Q2} + \mathrm{Q0}')\cdot(\mathrm{Q1}' + \mathrm{Q0})\cdot(\mathrm{Q2}' + \mathrm{Q0})
\end{aligned}$$

$$\begin{aligned}
\mathrm{D1} = \mathrm{Q1}^* &= (\mathrm{Q2} + \mathrm{Q1} + \mathrm{Q0} + \mathrm{LEFT} + \mathrm{RIGHT} + \mathrm{HAZ}) \\
&\quad \cdot(\mathrm{Q2} + \mathrm{Q1} + \mathrm{Q0} + \mathrm{LEFT}') \\
&\quad \cdot(\mathrm{Q2} + \mathrm{Q1} + \mathrm{Q0} + \mathrm{HAZ}') \\
&\quad \cdot(\mathrm{Q2} + \mathrm{Q1} + \mathrm{Q0} + \mathrm{RIGHT}') \\
&\quad \cdot(\mathrm{Q2} + \mathrm{Q1}' + \mathrm{Q0}) \\
&\quad \cdot(\mathrm{Q2}' + \mathrm{Q1}' + \mathrm{Q0}) \\
&\quad \cdot(\mathrm{Q2}' + \mathrm{Q1} + \mathrm{Q0}) \\
&= (\mathrm{Q2} + \mathrm{Q1} + \mathrm{Q0})\cdot(\mathrm{Q1}' + \mathrm{Q0})\cdot(\mathrm{Q2}' + \mathrm{Q0})
\end{aligned}$$

$$\begin{aligned}
\mathrm{D0} = \mathrm{Q0}^* &= (\mathrm{Q2} + \mathrm{Q1} + \mathrm{Q0} + \mathrm{LEFT} + \mathrm{RIGHT} + \mathrm{HAZ}) \\
&\quad \cdot(\mathrm{Q2} + \mathrm{Q1} + \mathrm{Q0} + \mathrm{HAZ}') \\
&\quad \cdot(\mathrm{Q2} + \mathrm{Q1}' + \mathrm{Q0}') \\
&\quad \cdot(\mathrm{Q2} + \mathrm{Q1}' + \mathrm{Q0}) \\
&\quad \cdot(\mathrm{Q2}' + \mathrm{Q1}' + \mathrm{Q0}') \\
&\quad \cdot(\mathrm{Q2}' + \mathrm{Q1}' + \mathrm{Q0}) \\
&\quad \cdot(\mathrm{Q2}' + \mathrm{Q1} + \mathrm{Q0}) \\
&= (\mathrm{Q2} + \mathrm{Q0} + \mathrm{LEFT} + \mathrm{RIGHT})\cdot(\mathrm{Q2} + \mathrm{Q0} + \mathrm{HAZ}')\cdot(\mathrm{Q1}')\cdot(\mathrm{Q2}' + \mathrm{Q0})
\end{aligned}$$

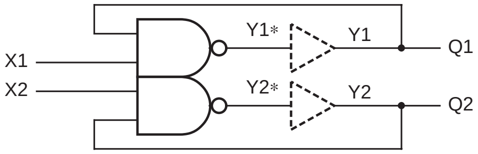
These equations yield the following transitions:

S	Q2	Q1	Q0	LEFT	RIGHT	HAZ	Q2*	Q1*	Q0*	S*
IDLE	0	0	0	0	0	0	0	0	0	IDLE
IDLE	0	0	0	0	1	1	1	0	0	LR3
IDLE	0	0	0	1	1	0	0	0	1	L1
IDLE	0	0	0	1	1	1	0	0	0	IDLE

This is obviously different and still not particularly good behavior.

7.58 Let E(SB), E(SC), and E(SD) be the binary encodings of states SB, SC, and SD respectively. Then $\mathrm{E(SD)} = \mathrm{E(SB)} + \mathrm{E(SC)}$, the bit-by-bit logical OR of E(SB) and E(SC). This is true because the synthesis method uses the logical OR of the next values for each state variable and, by extension, the logical OR of the encoded states.

7.68 As far as I know, I was the first person to propose BUT-flops, and Glenn Trewitt was the first person to analyze them, in 1982. To analyze, we break the feedback loops as shown in the figure to the right.



The excitation and output equations are

$$\begin{aligned}
Y1 &= [(X1 \cdot Y1) \cdot (X2 \cdot Y2)']' \\
&= X1' + Y1' + X2 \cdot Y2 \\
Y2 &= [(X2 \cdot Y2) \cdot (X1 \cdot Y1)']' \\
&= X2' + Y2' + X1 \cdot Y1 \\
Q1 &= Y1 \\
Q2 &= Y2
\end{aligned}$$

The corresponding transition/state table is

	X1 X2			
Y1 Y2	00	01	11	10
00	11	11	11	11
01	11	10	10	11
11	11	10	11	01
10	11	11	01	01
	Y1* Y2*			

The two stable total states are circled. Notice that state 00 is unreachable.

When X1 X2 = 00 or 11, the circuit generally goes to stable state 11, with Q1 Q2 = 11. The apparent oscillation between states 01 and 10 when X1 X2 = 11 may not occur in practice, because it contains a critical race that tends to force the circuit into stable state 11.

When X1 X2 = 01 or 10, the Q output corresponding to the HIGH input will oscillate, while the other output remains HIGH .

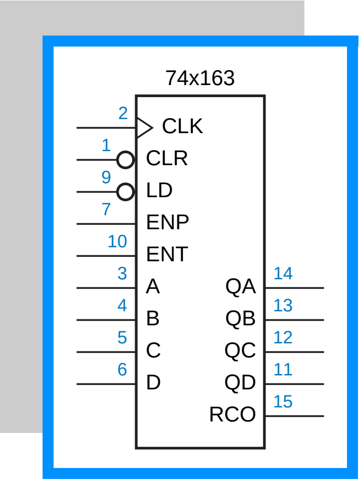
Whether this circuit is useful is a matter of opinion.

7.71 When X=1, the circuit was supposed to "count" through its eight states in Gray-code order. When X=0, it remains in the current state. If this were the case, I suppose it could be used as a 3-bit random number generator. However, I messed up on the logic diagram and the circuit actually does something quite different and completely useless, compared to what I intended when I wrote the problem. Someday I'll fix this problem. Also, metastability may occur when X is changed from 1 to 0.

7.79 Figure X5.59 requires two "hops" for each input change. Figure 7–66 is faster, requiring only one hop for each input change. On the other hand, Figure 7–66 cannot be generalized for *n*>2.

7.90 Either this exercise is a joke, or a correct answer is much too dangerous to publish. Nevertheless, Earl Levine offers two possible answers:

(Stable output)	Was the last answer to this question "yes"?
(Oscillating output)	Was the last answer to this question "no"?



8

Sequential Logic Design Practices

EXERCISE SOLUTIONS

8.1 In the first three printings, change “`RAMBANK0`” to “`RAMBANK1`” in the third line of the exercise. The results are the same. The new expression describes exactly the input combinations in which the 8 high-order bits of `ABUS` are 00000001_2, the same as the original expression using don’t-cares.

8.2 The 16-series devices have 64×32 or 2048 fuses (see Figure 10–2). The 20-series devices have 64×40 or 2560 fuses.

8.3 There are $64 \times 32 = 2048$ fuses in the AND array (see Figure 10–4). Each of the eight macrocells has one fuse to control the output polarity and one fuse to select registered vs. combinational configuration in the 16V8R, or to assert the output-enable in the 16V8S. There are also two global fuses to select the overall configuration (16V8C, 16V8R, or 16V8S). The total number of fuses is therefore $2048 + 16 + 2 = 2066$.

A real 16V8 (depending on the manufacturer) has at least 64 additional fuses to disable individual product terms, 64 user-programmable fuses that do nothing but store a user code, and a security fuse. (Once the security fuse is programmed, the rest of the fuse pattern can no longer be read.)

8.5 The f_{maxE} column below gives the answers in MHz.

Part numbers	Suffix	t_{PD}	t_{CO}	t_{CF}	t_{SU}	t_{H}	f_{maxE}	f_{maxI}
PAL16L8, PAL16Rx, PAL20L8, PAL20Rx	-5	5	4	–	4.5	0	117.7	117.7
PAL16L8, PAL16Rx, PAL20L8, PAL20Rx	-7	7.5	6.5	–	7	0	74.1	74.1
PAL16L8, PAL16Rx, PAL20L8, PAL20Rx	-10	10	8	–	10	0	55.6	55.6
GAL22V10	-25	25	15	13	15	0	33.3	35.7

8.6 The f_{maxI} column above gives the answers in MHz.

8.7

```
module Eight_Bit_Reg
title ’8-bit Edge-Triggered Register’
Z74X374 device ’P16V8R’;

" Input pins
CLK, !OE                          pin 1, 11;
D1, D2, D3, D4, D5, D6, D7, D8    pin 2, 3, 4, 5, 6, 7, 8, 9;

" Output pins
Q1, Q2, Q3, Q4, Q5, Q6, Q7, Q8    pin 19, 18, 17, 16, 15, 14, 13, 12;

" Set definitions
D = [D1,D2,D3,D4,D5,D6,D7,D8];
Q = [Q1,Q2,Q3,Q4,Q5,Q6,Q7,Q8];

equations

Q := D;

end Eight_Bit_Reg
```

8.10 If EN or CLK is 0, the output will be stable. If both are 1, the results are unpredictable, since they depend on circuit timing. It is certain that the circuit’s output will be unstable as long as this condition is true.

8.11 The counter is modified to return to a count of 0 when count 9 is reached.

```
module Z74x162
title '4-bit Decade Counter'
"Z74X162 device 'P16V8R';
" Input pins
CLK, !OE                          pin 1, 11;
A, B, C, D                        pin 2, 3, 4, 5;
!LD, !CLR, ENP, ENT               pin 6, 7, 8, 9;
" Output pins
QA, QB, QC, QD                    pin 19, 18, 17, 16 istype 'reg';
RCO                               pin 15;
" Set definitions
INPUT = [ D,  C,  B,  A ];
COUNT = [QD, QC, QB, QA ];
equations
COUNT.CLK = CLK;
COUNT := !CLR & (  LD & INPUT
                # !LD &  (ENT & ENP) & (COUNT < 9) & (COUNT + 1)
                # !LD &  (ENT & ENP) & (COUNT == 9) & 0
                # !LD & !(ENT & ENP) &  COUNT);
RCO = (COUNT == 9) & ENT;
end Z74x162
```

8.13 The counting direction is controlled by QD: count up when QD=1, count down when QD=0. A load occurs when the counter is in the terminal state, 1111 when counting up, 0000 when counting down. The MSB is complemented during a load and the other bits are unchanged.

Let us assume that the counter is initially in one of the states 0000–0111. Then the counter counts down (QD=0). Upon reaching state 0000, it loads 1000 and subsequently counts up (QD=1). Upon reaching state 1111, the counter loads 0111, and subsequently counts down, repeating the cycle.

If the counter is initially in one of the states 1000–1111, the same cyclic behavior is observed. The counting sequence has a period of 16 and is, in decimal,

8, 9, 10, 11, 12, 13, 14, 15, 7, 6, 5, 4, 3, 2, 1, 0, 8, 9, ...

If only the three LSBs are observed, the sequence is

0, 1, 2, 3, 4, 5, 6, 7, 7, 6, 5, 4, 3, 2, 1, 0, 0, 1, ...

8.14 The only difference between a ’163 and a ’161 is that the CLR_L input of ’161 is asynchronous. Thus, the counter will go from state 1010 to state 0000 immediately, before the next clock tick, and go from state 0000 to state 0001 at the clock tick. Observing the state just before each clock tick, it is therefore a modulo-10 counter, with the counting sequence 0, 1, ..., 9, 0, 1, ...

Note that this type of operation is not recommended, because the width of the CLR_L pulse is not well controlled. That is, the NAND gate will negate the CLR_L pulse as soon as either one of its inputs goes to 0. If, say, the counter’s QB output clears quickly, and its QD output clears slowly, it is possible for CLR_L to be asserted long enough to clear QB but not QD, resulting in an unexpected next state of 1000, or possibly metastability of the QD output.

8.17 The path from the Q1 counter output (B decoder input) to the Y2_L output has 10 ns more delay than the Q2 and Q0 (C and A) paths. Let us examine the possible Y2_L glitches in Figure 8–43 with this in mind:

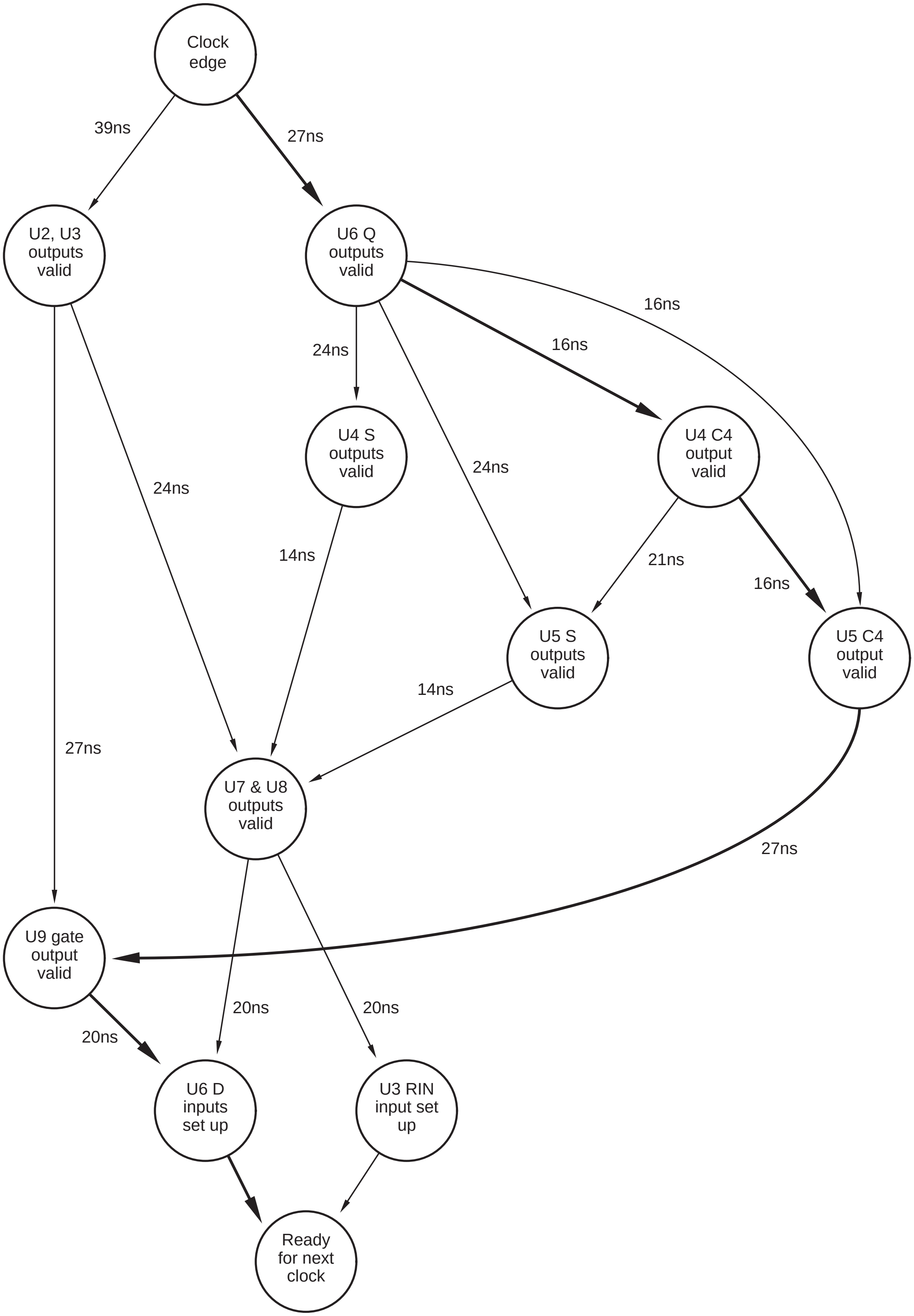
3→4 (011→100) Because of the delay in the Q1 path, this transition will actually look like 011→110→100. The Y6_L output will have a 10-ns glitch, but Y2_L will not.

7→0 (111→000) Because of the delay in the Q1 path, this transition will actually look like 111→010→000. The Y2_L output will have a 10-ns glitch, but the others will not.

8.19 The delay calculation is different, depending on the starting state.

In the INIT state, U7 and U8 take 21 ns to propagate the CLEAR signal to the outputs. Then U6 requires 20 ns setup time on its D inputs while U3 requires 20 ns setup time on its RIN input. This implies a minimum clock period of 41 ns, assuming zero delay from the control unit.

In states M1–M8, the minimum clock period depends on the delay of the combinational logic in the control unit that asserts SELSUM when MPY0 is asserted. However, the most obvious way to do this is to connect MPY0 directly to SELSUM, creating a delay of 0 ns. This assumption is made below. Also, it is assumed that MPY0 is 1 to find the worst case. The figure on the next page shows the worst-case path, in heavy lines, to be 106 ns. Since we would like to use the same clock for all states, the minimum clock period is 106 ns.


Clock edge
39ns
27ns
U2, U3 outputs valid
U6 Q outputs valid
16ns
24ns
16ns
U4 S outputs valid
24ns
U4 C4 output valid
24ns
14ns
21ns
16ns
U5 S outputs valid
U5 C4 output valid
14ns
27ns
U7 & U8 outputs valid
27ns
U9 gate output valid
20ns
20ns
20ns
U6 D inputs set up
U3 RIN input set up
Ready for next clock

8.20 The synchronizer fails if META has not settled by the beginning of the setup-time window for FF2, which is 5 ns before the clock edge. Since the clock period is 40 ns, the available metastability resolution time is 35 ns. The MTBF formula is

$$\mathrm{MTBF}(t_r) = \frac{\exp(t_r / \tau)}{T_0 \cdot f \cdot a}$$

Substituting the proper values of τ and T_0 for the 'F74, and of f and a for the problem, we calculate

$$\mathrm{MTBF}(35\mathrm{ns}) = \frac{\exp(35/0.4)}{2.0 \cdot 10^{-4} \cdot 10^{6} \cdot 10^{6}} \approx 2 \cdot 10^{28}\mathrm{s}$$

8.22 Refer to the sample data sheet on page 169 of the text: “Not more than one output should be shorted at a time; duration of short-circuit should not exceed one second.” In the switch debounce circuit, the short lasts only for a few tens of *nanoseconds*, so it’s OK.

8.23 CMOS outputs can “latch up” under certain conditions. According to the Motorola *High-Speed CMOS Logic Data* book (1988 edition, pp. 4–10), a 74HCT output can latch up if a voltage outside the range $-0.5 \le V_{\mathrm{out}} \le V_{\mathrm{CC}} + 0.5\mathrm{V}$ is forced on the output by an external source. In a switch debounce circuit using 74HCT04s, the switch connection to ground is an external source, but the voltage (0 V) is within the acceptable range and should not be a problem.

Another potential problem is excessive short-circuit current, but again the data book indicates that shorting the output briefly is not a problem, as long as “the maximum package power dissipation is not violated” (i.e., the short is not maintained for a long time).

Similar considerations apply to 74AC and 74ACT devices, but in the balance, such devices are *not* recommended in the switch-debounce application, as we’ll explain. On one hand, typical 74AC/74ACT devices are even less susceptible to latch-up than 74HCT devices. (For example, see the Motorola *FACT Data* book, 1988 edition, pp. 2–9.) On the other hand, 74AC/74ACT’s high performance may create noise problems for *other* devices in a system. In particular, when the 74AC/74ACT HIGH output is shorted to ground, it may momentarily drag the local 5 V power-supply rail down with it, especially if the decoupling capacitors are small, far away, or missing. This will in turn cause incorrect operation of the other, nearby logic devices.

8.25 TTL inputs require significant current, especially in the LOW state. The bus holder cannot supply enough current unless the series resistor is made much smaller, which then creates a significant load on the bus.

8.26 .

```
library IEEE;
use IEEE.std_logic_1164.all;
--
-- Exercise 8-26
-- This code combines the address latch and
-- and the decoder and its latch
entity latch_decode is
port (
       abus : in std_logic_vector ( 31 downto 0);
       avalid : in std_logic;
       la : out std_logic_vector ( 19 downto 0);
       romcs, ramcs0, ramcs1, ramcs2 : out std_logic
);
end entity latch_decode;

architecture behave of latch_decode is


begin
process (avalid, abus)
begin
if (avalid = '1') then
       la <= abus (19 downto 0);
end if;
end process;

process (abus, avalid)
variable rom, ram1, ram2, ram0 : std_logic_vector (11 downto 0);
begin
rom  := "111111111111";
ram0 := "000000000000";
ram1 := "000000010000";
ram2 := "000000100000";

If (avalid= '1') then
    if (abus (31 downto 20) = rom ) then romcs  <= '1'; else romcs  <= '0'; end if;
    if (abus (31 downto 20) = ram0) then ramcs0 <= '1'; else ramcs0 <= '0'; end if;
    if (abus (31 downto 20) = ram1) then ramcs1 <= '1'; else ramcs1 <= '0'; end if;
    if (abus (31 downto 20) = ram2) then ramcs2 <= '1'; else ramcs2 <= '0'; end if;
end if;
end process;

end behave;
```

8.28 The maximum delay from clock to output of a 74HCT74 flip-flop is 44 ns. For a 4-bit ripple counter, the delay ripples through four such stages for a total maximum delay of 176 ns. Similarly, the maximum delays using 74AHCT and 74LS74 flip-flops are 40 ns and 160 ns, respectively.

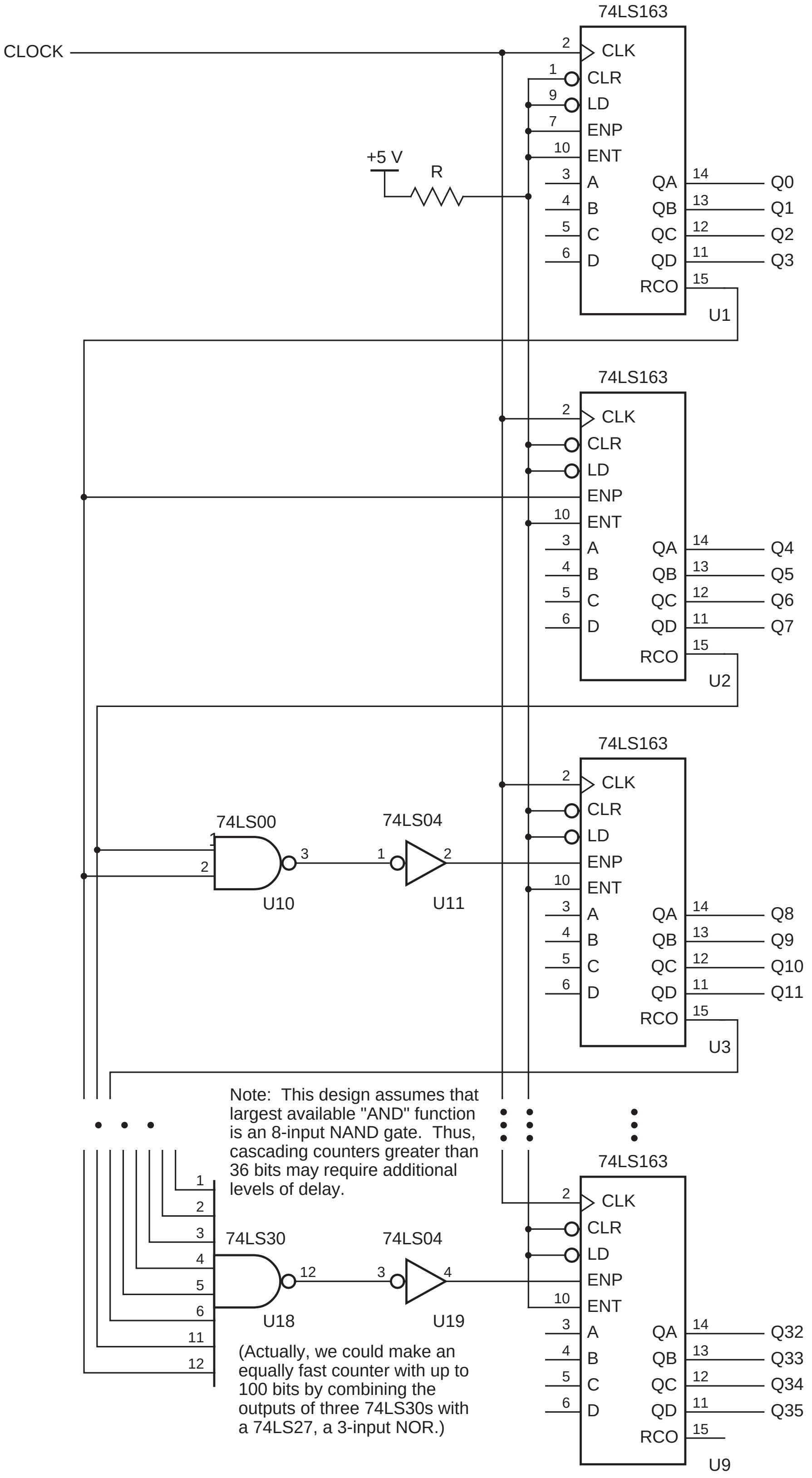
8.32

$$t_{\text{period(min)}} = t_{\text{pTQ}} + 3t_{\text{AND}} + t_{\text{setup}}$$
$$f_{\text{max}} = 1/t_{\text{period(min)}}$$

8.37

Inputs				Current state				Next state			
CLR_L	LD_L	ENT	ENP	QD	QC	QB	QA	QD*	QC*	QB*	QA*
0	x	x	x	x	x	x	x	0	0	0	0
1	0	x	x	x	x	x	x	D	C	B	A
1	1	0	x	x	x	x	x	QD	QC	QB	QA
1	1	x	0	x	x	x	x	QD	QC	QB	QA
1	1	1	1	0	0	0	0	0	0	0	1
1	1	1	1	0	0	0	1	0	0	1	0
1	1	1	1	0	0	1	0	0	0	1	1
1	1	1	1	0	0	1	1	0	1	0	0
1	1	1	1	0	1	0	0	0	1	0	1
1	1	1	1	0	1	0	1	0	1	1	0
1	1	1	1	0	1	1	0	0	1	1	1
1	1	1	1	0	1	1	1	1	0	0	0
1	1	1	1	1	0	0	0	1	0	0	1
1	1	1	1	1	0	0	1	0	0	0	0
1	1	1	1	1	0	1	0	1	0	1	1
1	1	1	1	1	0	1	1	0	1	0	0
1	1	1	1	1	1	0	0	1	1	0	1
1	1	1	1	1	1	0	1	0	1	0	0
1	1	1	1	1	1	1	0	1	1	1	1
1	1	1	1	1	1	1	1	0	0	0	0

8.38



Note: This design assumes that largest available "AND" function is an 8-input NAND gate. Thus, cascading counters greater than 36 bits may require additional levels of delay.

(Actually, we could make an equally fast counter with up to 100 bits by combining the outputs of three 74LS30s with a 74LS27, a 3-input NOR.)

The minimum clock period is the sum of:

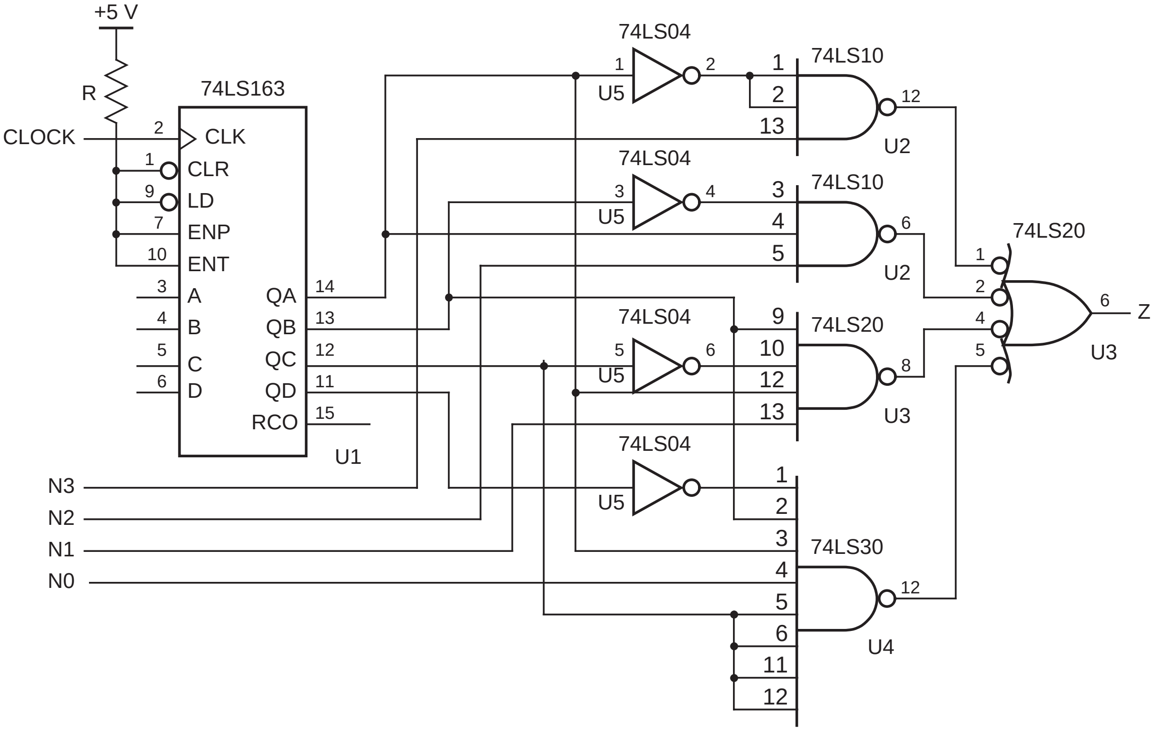
(a) The delay from the clock edge to any RCO output (35 ns).

(b) The delay from any RCO output to any ENP input, that is, two gate delays ($2 \cdot 15 = 30\,\text{ns}$).

(c) The setup time to the next clock edge required by the ENP inputs (20 ns).

Thus, the minimum clock period is 85 ns, and the corresponding maximum clock frequency is 11.76 MHz.

8.41 To get even spacing, the strategy is for the MSB (N3) to select half the states, the ones where QA is 0. The next bit down (N2) selects one-fourth of the states, the ones where QB is 0 and the less significant counter bits (i.e., QA) are all 1. Likewise, N1 selects the one-eighth of the states where QC is 0 and QB and QA are 1, and N0 selects the state where QD is 0 and QC, QB, and QA are all 1. In this way, each non-1111 counter state is assigned to one input bit.



8.53

```
--Chris Dunlap
--Xilinx Applications

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity v74x163s is
generic(size : integer := 8);  --added generic
port   (clk,clr_l,ld_l,enp,ent : in std_logic;
       d : in std_logic_vector (size-1 downto 0); --changes range of input
       q : out std_logic_vector (size-1 downto 0); --changes range of output
       rco : out std_logic);
end v74x163s;

architecture v74x163_arch of v74x163s is

component synsercell is
  port (clk, ldnoclr, di, coclrorld,cntenp,cnteni : in std_logic;
       qi,cnteni1 : out std_logic);
end component;

signal ldnoclr,noclrorld : std_logic;
signal scnten : std_logic_vector (size downto 0); --creates a ranged temp with overflow room

begin
ldnoclr <= (not ld_l) and clr_l;
noclrorld <= ld_l and clr_l;
scnten(0) <= ent;
rco <= scnten(size);
gi: for i in 0 to size-1 generate --counts for size of counter
  U1: synsercell port map (clk, ldnoclr, noclrorld, enp, d(i), scnten(i), scnten(i+1), q(i));
  end generate;
end v74x163_arch;
```

8.54

```
--**********************************
-- PROBLEM : WAKERLY - 8.54
-- FILES :
--   8_54_top.vhd : top level file
--   8_54_par2ser.vhd : parallel to serial converter
--   8_54_control.vhd : control module
--   8_54_shift_synch.vhd : 8 bit shift register
--
-- DESCRIPTION :
--   Creates a parallel to serial converter.
--   Data in is described as 8 x 8bit modules,
--   with a single 8 bit data bus that carries
--   data of the format given in Figure 8-55.
--   Each serial link has its own SYNCH(i) line;
--   the pulses should be staggered so SYNCH(i+1)
--   occurs 1 clock cycle after SYNCH(i).
--
--   Because of this, the load_synch line should
--   also be staggered so the data transmitted
--   over the serial link will correspond to its
--   associated SYNCH line.
--**********************************

-- library declarations
library IEEE;
use IEEE.std_logic_1164.all;

-- top level entity declaration
entity wak_8_54_top is
    port (
        data: in STD_LOGIC_VECTOR (63 downto 0);
        clock: in STD_LOGIC;
        synch: buffer STD_LOGIC_VECTOR (7 downto 0);
        sdata: out STD_LOGIC_VECTOR (7 downto 0)
    );
end wak_8_54_top;

architecture wak_8_54_arch of wak_8_54_top is

signal load_shift_master: std_logic;
signal synch_master: std_logic;
signal load_shift: std_logic_vector (7 downto 0);

--component declarations
component par2ser is
    port (
        clock: in STD_LOGIC;
        data: in STD_LOGIC_VECTOR (7 downto 0);
        load_shift: in STD_LOGIC;
        sdata: out STD_LOGIC
    );
end component;
```

```
component control is
    port (
        clock: in STD_LOGIC;
        load_shift: out STD_LOGIC;
        synch: out STD_LOGIC
    );
end component;

component shift_synch is
    port (
        clock: in STD_LOGIC;
        synch_in: in STD_LOGIC;
        synch: buffer STD_LOGIC_VECTOR (7 downto 0)
    );
end component;

begin

--component instantiations
S1: shift_synch port map (clock=>clock, synch_in=>synch_master, synch=>synch);
S2: shift_synch port map (clock=>clock, synch_in=>load_shift_master,
synch=>load_shift);

U1: par2ser port map (clock=>clock, data=>data(7 downto 0), load_shift=>load_shift(0),
sdata=>sdata(0));
U2: par2ser port map (clock=>clock, data=>data(15 downto 8),
load_shift=>load_shift(1), sdata=>sdata(1));
U3: par2ser port map (clock=>clock, data=>data(23 downto 16),
load_shift=>load_shift(2), sdata=>sdata(2));
U4: par2ser port map (clock=>clock, data=>data(31 downto 24),
load_shift=>load_shift(3), sdata=>sdata(3));
U5: par2ser port map (clock=>clock, data=>data(39 downto 32),
load_shift=>load_shift(4), sdata=>sdata(4));
U6: par2ser port map (clock=>clock, data=>data(47 downto 40),
load_shift=>load_shift(5), sdata=>sdata(5));
U7: par2ser port map (clock=>clock, data=>data(55 downto 48),
load_shift=>load_shift(6), sdata=>sdata(6));
U8: par2ser port map (clock=>clock, data=>data(63 downto 56),
load_shift=>load_shift(7), sdata=>sdata(7));

U9: control port map (clock=>clock, load_shift=>load_shift_master,
synch=>synch_master);

end wak_8_54_arch;
```

```
--*************************************
-- Basically an 8-bit shift register
--  takes the synch_in signal as an
--  input, and outputs an 8 bit signal,
--  each consecutive bit delayed by one
--  from the previous bit.

-- library declaration
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

-- top level entity declaration
entity shift_synch is
    port (
        clock: in STD_LOGIC;
        synch_in: in STD_LOGIC;
        synch: buffer STD_LOGIC_VECTOR (7 downto 0)
    );
end shift_synch;

architecture shift_synch_arch of shift_synch is

begin

-- low order synch signal is simply passed through
-- to output.  all others are delayed.
  synch(0) <= synch_in;

  process(clock)
  begin
    if clock'event and clock='1' then

      for I in 0 to 6 loop
        synch(I+1) <= synch(I);
      end loop;
    end if;
  end process;

end shift_synch_arch;
```

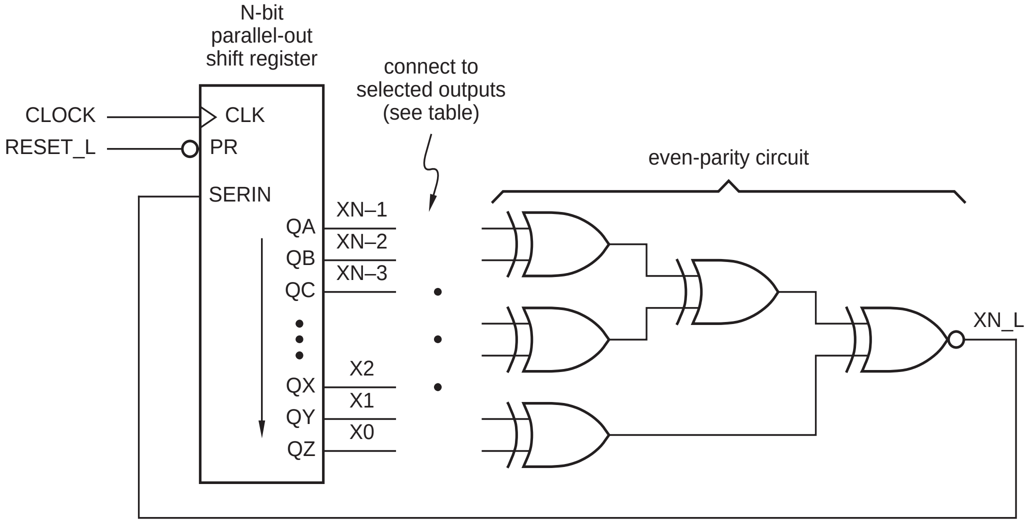
```
--***************************************
-- Parallel to serial converter
--  Data is entered through 8 bit DATA bus
--  It is loaded into the register when
--  load_shift is low.  If load_shift is
--  high, shift data serially out through sdata

-- library declarations
library IEEE;
use IEEE.std_logic_1164.all;

-- top level entity declaration
entity par2ser is
    port (
        clock: in STD_LOGIC;
        data: in STD_LOGIC_VECTOR (7 downto 0);
        load_shift: in STD_LOGIC;
        sdata: out STD_LOGIC
    );
end par2ser;

architecture par2ser_arch of par2ser is

-- internal signal declaration
signal REG: STD_LOGIC_VECTOR(7 downto 0);
signal DIN: std_logic;

begin

-- DIN <= 0 will set the high order bit to be
-- zero once data is loaded in.
DIN <= '0';

-- process to create shift register
--accomplished by simply taking the DIN signal
--and concatenating on the end the previous
--6 high order bits.
process (clock)
  begin
    if clock'event and clock='1' then
      if load_shift = '0' then
       REG <= data;
      else
        REG <= DIN & REG(7 downto 1);
      end if;
    end if;
  sdata <= REG(0);
end process;

end par2ser_arch;
```

```
--******************************
-- Control logic
--  controls the loading of the
--  parallel to serial shift register
--  through the load_shift signal.
--  also, controls the synch word.
--  this occurs every 256 clock cycles.

--library declaration
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

--top level entity declaration
entity control is
    port (
        clock: in STD_LOGIC;
        load_shift: out STD_LOGIC;
        synch: out STD_LOGIC
    );
end control;

architecture control_arch of control is
--internal signal declaration
signal COUNT: STD_LOGIC_VECTOR(7 downto 0);
signal load: STD_LOGIC;
begin
load <= '0';  --define constant

process (clock)
  begin
    if clock'event and clock='1' then

      count <= count + 1;

      if count(2 downto 0) = "110" then
        load_shift <= load;
      else
        load_shift <= not load;
      end if;

      if count = 254 then
        synch <= '1';
      else
        synch <= '0';
      end if;

    end if;
end process;

end control_arch;
```

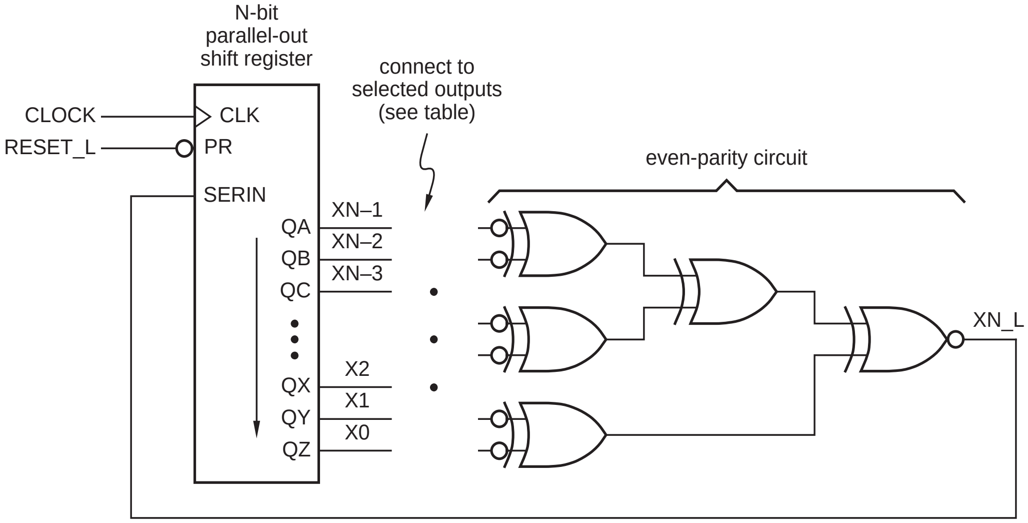
8.55 This is really the second paragraph of Exercise 8.54.

8.62 Regardless of the number of shift-register outputs connected to the odd-parity circuit, its output in state 00...00 is 0, and the 00...00 state persists forever. However, suppose that an odd number of shift-register outputs are connected. Then the output of the odd-parity circuit in state 11...11 is 1, and the 11...11 state also persists forever. In this case, the number of states in the "maximum-length" sequence can be no more than $2^n - 2$, since two of the states persist forever. Therefore, if an LFSR counter generates a sequence of length $2^n - 1$, it must have an even number of shift-register outputs connected to the odd-parity circuit.

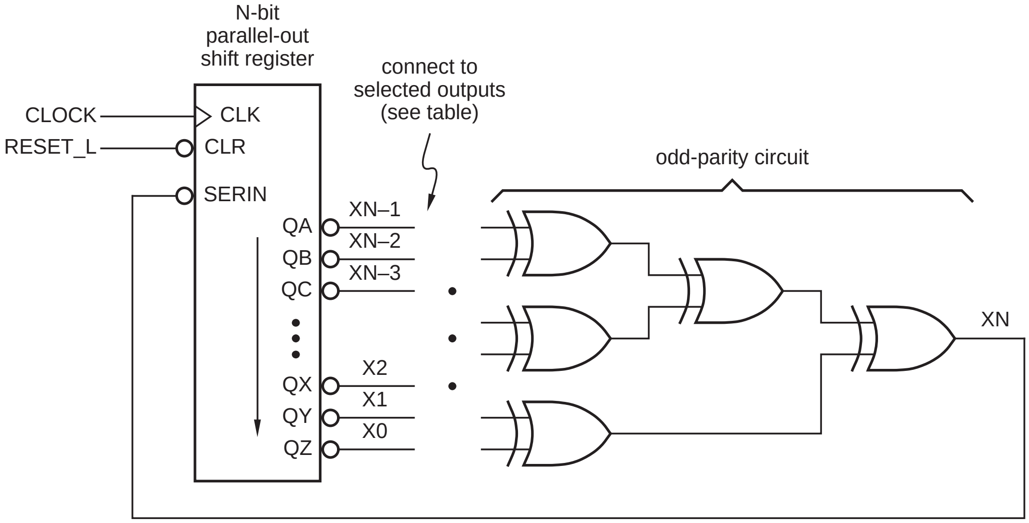
8.64 The figure below shows the effect of physically changing the odd-parity circuit to an even-parity circuit (i.e., inverting its output).



From Exercise 9.32, we know that an even number of shift-register outputs are connected to the parity circuit, and we also know that complementing two inputs of an XOR gate does not change its output. Therefore, we can redraw the logic diagram as shown below.



Finally, we can move the inversion bubbles as shown below.



This circuit has exactly the same structure as Figure 8–68, except that the shift register stores complemented data. When we look at the external pins of the shift register in the first figure in this solution, we are looking at that complemented data. Therefore, each state in the counting sequence of the even-parity version (our first figure) is the complement of the corresponding state in the odd-parity version (Figure 8–68). The odd-parity version visits all states except 00...00, so the even-parity version visits all states except 11...11.

8.68

```
--Johnny West
--Xilinx Applications
--8.68 : Design an iterative circuit for checking the parity of a 16-bit data
--       word with a single even-parity bit.

------------------------------------------------------------------------------------
------------
--This portion of the code constructs the iterative module that will cascaded 16
--times in order to check the parity of a 16 bit word.
------------------------------------------------------------------------------------
------------
library IEEE;
use IEEE.std_logic_1164.all;

--Generic Iterative Module
entity Iterative_Module is
  port (
        carry_in, primary_in: in STD_LOGIC;
              carry_out: out STD_LOGIC
          );
end Iterative_Module;

--An XOR is performed on the current parity of a word (carry_in) and the
--next bit in the word (boundary_in)
--Even parity produces a 0 and odd parity produces a 1
architecture Parity_Check of Iterative_Module is
begin
  carry_out <= carry_in xor primary_in;
end Parity_Check;
```

```
------------------------------------------------------------------------------------
--This portion of the code cascades the interative module 16 times and connects the
--modules together.
------------------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;

--Top level entity for checking the parity of a 16bit word
entity Parity_16bit is
  port (data_word: in STD_LOGIC_VECTOR (15 downto 0);
        parity: out STD_LOGIC                         );
end Parity_16bit;

--Architecture consists of 16 cascaded iterative modules
architecture Parity_16bit_arch of parity_16bit is

component Iterative_Module
  port (carry_in, primary_in: in STD_LOGIC;
              carry_out: out STD_LOGIC    );
end component;

signal carry: STD_LOGIC_VECTOR (15 downto 0);
signal cin0: STD_LOGIC;

begin
  cin0 <= '0';
  P0: Iterative_Module port map (cin0, data_word(0), carry(0));
  P1: Iterative_Module port map (carry(0), data_word(1), carry(1));
  P2: Iterative_Module port map (carry(1), data_word(2), carry(2));
  P3: Iterative_Module port map (carry(2), data_word(3), carry(3));
  P4: Iterative_Module port map (carry(3), data_word(4), carry(4));
  P5: Iterative_Module port map (carry(4), data_word(5), carry(5));
  P6: Iterative_Module port map (carry(5), data_word(6), carry(6));
  P7: Iterative_Module port map (carry(6), data_word(7), carry(7));
  P8: Iterative_Module port map (carry(7), data_word(8), carry(8));
  P9: Iterative_Module port map (carry(8), data_word(9), carry(9));
  P10: Iterative_Module port map (carry(9), data_word(10), carry(10));
  P11: Iterative_Module port map (carry(10), data_word(11), carry(11));
  P12: Iterative_Module port map (carry(11), data_word(12), carry(12));
  P13: Iterative_Module port map (carry(12), data_word(13), carry(13));
  P14: Iterative_Module port map (carry(13), data_word(14), carry(14));
  P15: Iterative_Module port map (carry(14), data_word(15), carry(15));
  parity <= carry(15); --parity = 0 is even parity and 1 if odd parity
end parity_16bit_arch;
```

8.75 In the following design, RESET is not recognized until the end of phase 6. RESTART is still recognized at the end of any phase; otherwise it would have no real use (i.e., only going back to phase 1 after the end of phase 6, which happens anyway.) Presumably, RESTART would be used only with great care or in unusual circumstances (e.g., during debugging).

```
module TIMEGEN6
title 'Six-phase Master Timing Generator'

" Input and Output pins
MCLK, RESET, RUN, RESTART                  pin;
T1, P1_L, P2_L, P3_L, P4_L, P5_L, P6_L     pin istype 'reg';

" State definitions
PHASES = [P1_L, P2_L, P3_L, P4_L, P5_L, P6_L];
NEXTPH = [P6_L, P1_L, P2_L, P3_L, P4_L, P5_L];
SRESET = [1, 1, 1, 1, 1, 1];
P1 =     [0, 1, 1, 1, 1, 1];
P6 =     [1, 1, 1, 1, 1, 0];

equations
T1.CLK = MCLK; PHASES.CLK = MCLK;

WHEN (RESET & PHASES==P6 & !T1) THEN {T1 := 1; PHASES := SRESET;}
ELSE WHEN (PHASES==SRESET) # RESTART THEN {T1 := 1; PHASES := P1;}
ELSE WHEN RUN & T1 THEN {T1 := 0; PHASES := PHASES;}
ELSE WHEN RUN & !T1 THEN {T1 := 1; PHASES := NEXTPH;}
ELSE {T1 := T1; PHASES := PHASES;}

end TIMEGEN6
```

8.81 This problem can be a bit confusing since the states in Table 7–14 have the same names as the ’163 data inputs. Therefore, we shall use the names SA, SB, and so on for the states.

The idea is to normally allow the counter to count to the next state, but to force it to go to SA or SB when the wrong input is received. The CLR_L input is used to go to SA (0000), and the LD_L input is used to go to SB (0001; the counter’s A–D data inputs are tied LOW and HIGH accordingly).

Inspecting the state table on page 582 of the text, we see that state A should be loaded when X=1 and the machine is in state SA, SD, or SH. Thus,

$$\begin{aligned} \mathrm{CLR\_L} &= [\mathrm{X}\cdot(\mathrm{QC}'\cdot\mathrm{QB}'\cdot\mathrm{QA}' + \mathrm{QC}'\cdot\mathrm{QB}\cdot\mathrm{QA} + \mathrm{QC}\cdot\mathrm{QB}\cdot\mathrm{QA})]' \\ &= [\mathrm{X}\cdot(\mathrm{QC}'\cdot\mathrm{QB}'\cdot\mathrm{QA} + \mathrm{QB}\cdot\mathrm{QA})]' \end{aligned}$$

All of the other next-states when X=1 are the natural successors obtained by counting.

Similarly, state SB should be loaded when X=0 and the machine is in state SB, SC, SE, SF, or SH. In addition, notice that in state SG, the next state SE is required when X=0; the load input must be used in this case too, but a different value must be loaded. Thus, LD_L will be asserted in six of the eight states when X=0.

Optionally, LD_L could be easily asserted in the remaining two states as well, since the required next states (SB and SE) are ones that we must generate in other six cases anyway. Thus, we can connect X to the LD_L input, so we always load when X=0. Then, we load either SB (0010) or SE (0100) depending on the current state. Therefore, we can write the following equations for data inputs A–D:

$$\begin{aligned} \mathrm{A} &= 0 \\ \mathrm{B} &= \mathrm{SA} \boldsymbol{+} \mathrm{SB} \boldsymbol{+} \mathrm{SC} \boldsymbol{+} \mathrm{SE} \boldsymbol{+} \mathrm{SF} \boldsymbol{+} \mathrm{SH} \\ &= \mathrm{QB}' + \mathrm{QC}'\cdot\mathrm{QA}' + \mathrm{QC}\cdot\mathrm{QA} \\ \mathrm{C} &= \mathrm{B}' \\ \mathrm{D} &= 0 \end{aligned}$$

Alternatively, we could realize C as an AND-OR circuit and complement to get B:

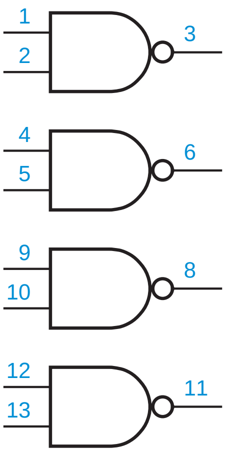
$$\begin{aligned} \mathrm{C} &= \mathrm{QC}'\cdot\mathrm{QB}\cdot\mathrm{QA} + \mathrm{QC}\cdot\mathrm{QB}\cdot\mathrm{QA}' \\ \mathrm{B} &= \mathrm{C}' \end{aligned}$$

The logic diagram follows directly from these equations. The output logic can be realized using the equations on page 584 of the text.

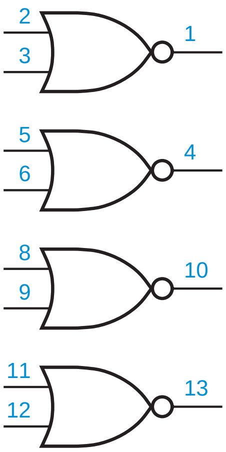
8.90 Transitions on SYNCIN occur a maximum of 20 ns after the rising edge of CLOCK. Given a 40-ns clock period and a 10-ns setup-time requirement for the other ’ALS74s, 10 ns is the maximum propagation delay of the combinational logic.

Combinational SSI Devices

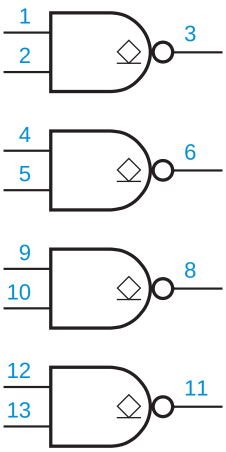
74x00



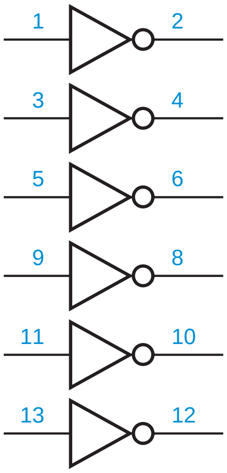
74x02



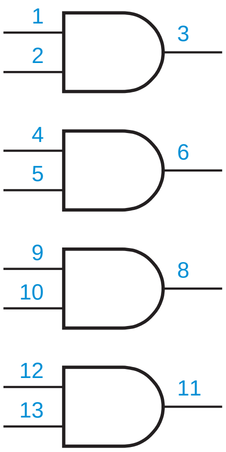
74x03



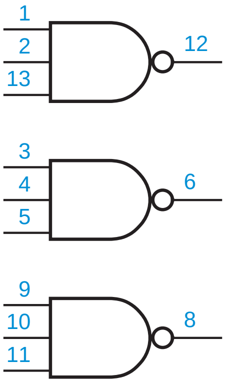
74x04



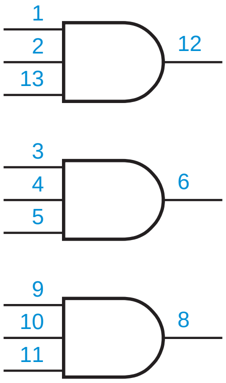
74x08



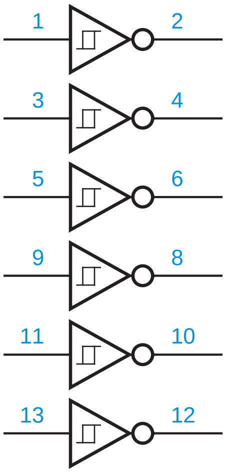
74x10



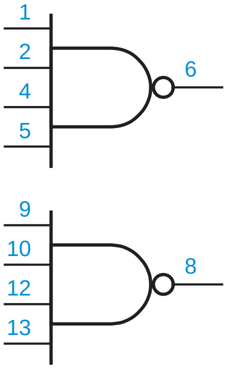
74x11



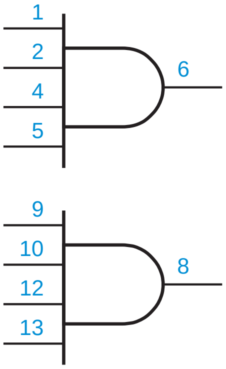
74x14



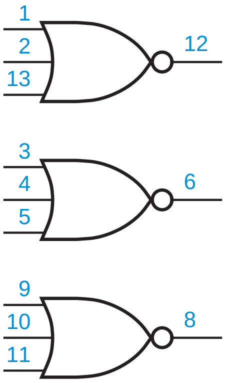
74x20



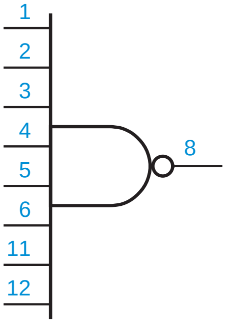
74x21



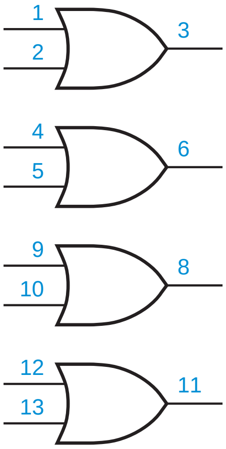
74x27



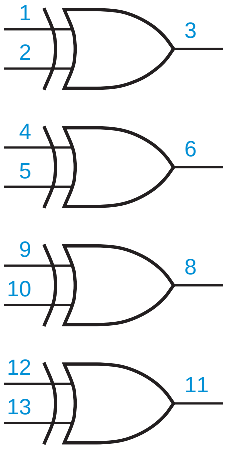
74x30



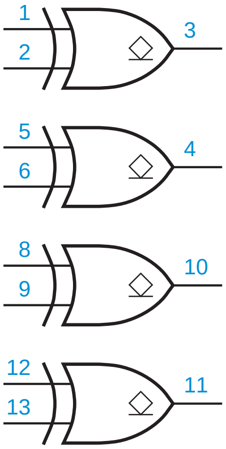
74x32



74x86

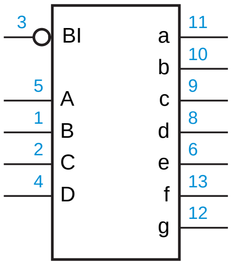


74x266

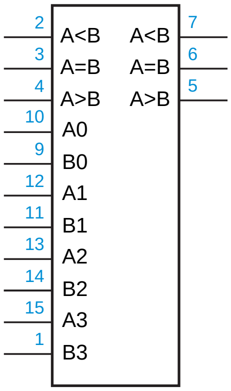


Combinational MSI Devices

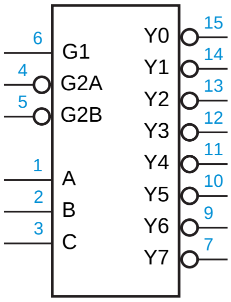
74x49



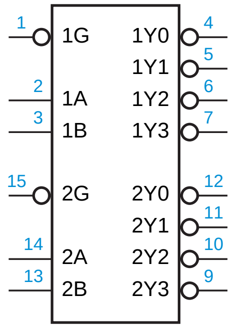
74x85



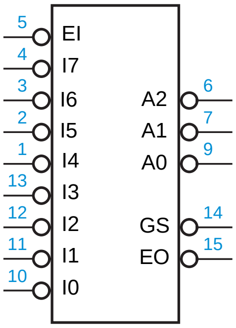
74x138



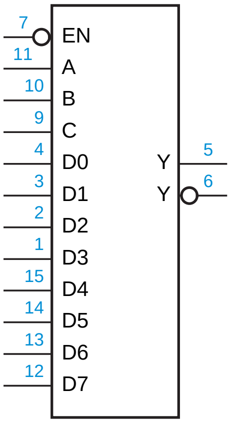
74x139



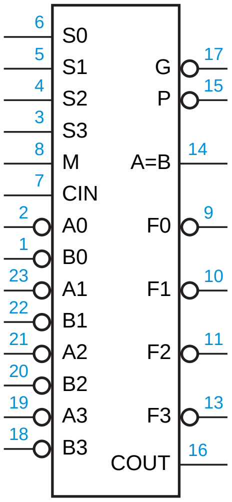
74x148



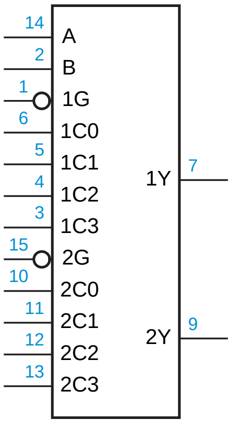
74x151



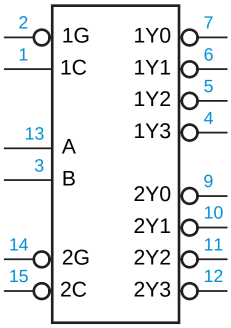
74x181



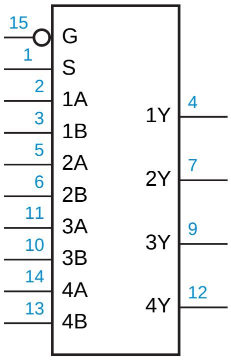
74x153



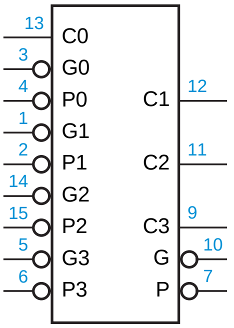
74x155



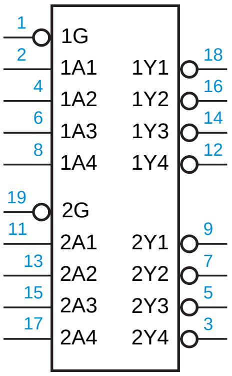
74x157



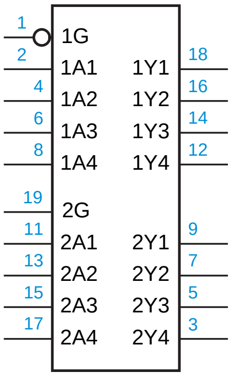
74x182



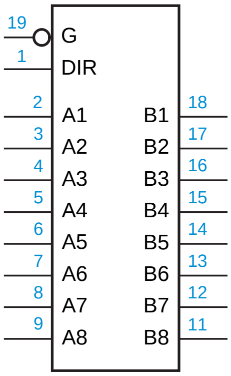
74x240



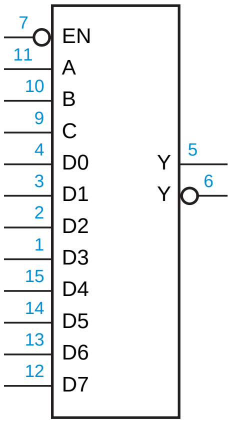
74x241



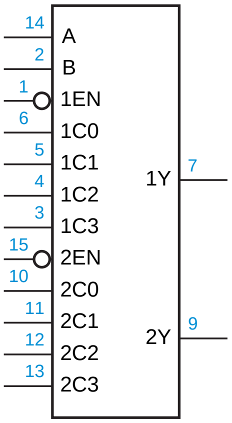
74x245



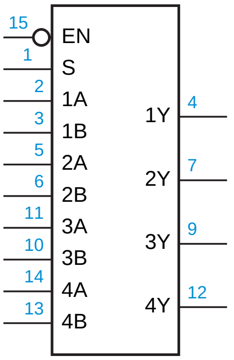
74x251



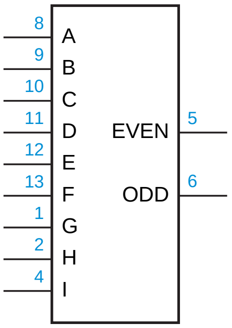
74x253



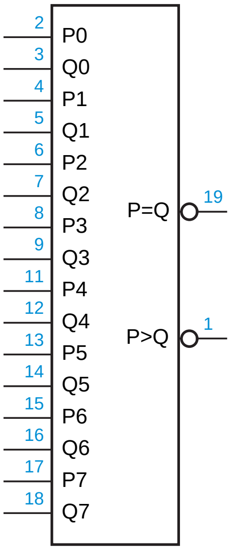
74x257



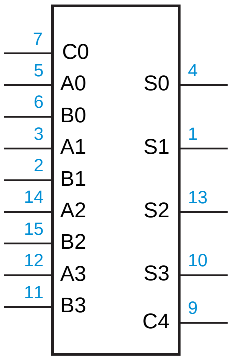
74x280



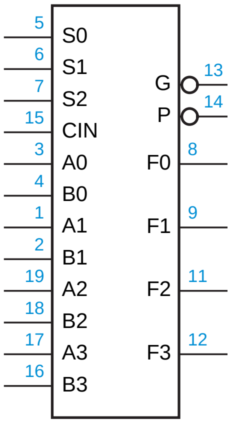
74x682



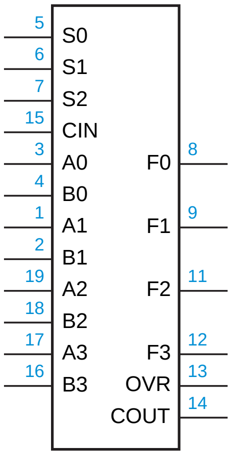
74x283



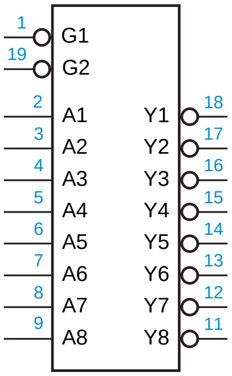
74x381



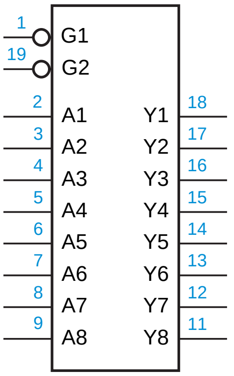
74x382



74x540

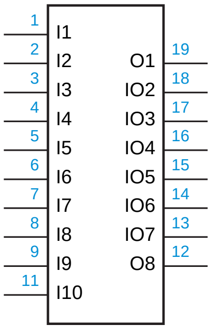


74x541

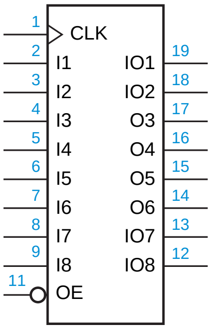


16R-Series 20-pin Bipolar PLDs

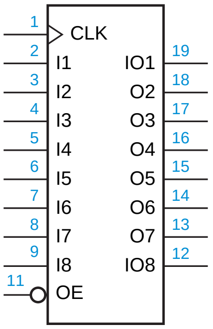
PAL16L8



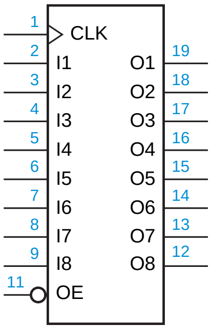
PAL16R4



PAL16R6

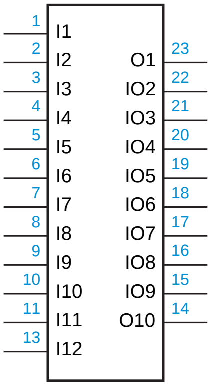


PAL16R8

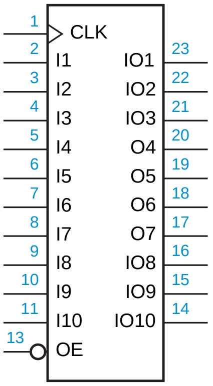


20X-Series 24-pin Bipolar PLDs

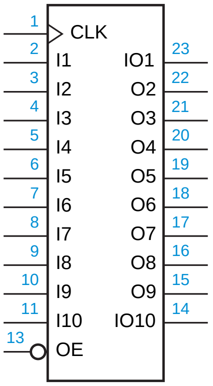
PAL20L10



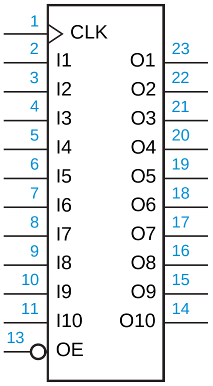
PAL20X4



PAL20X8

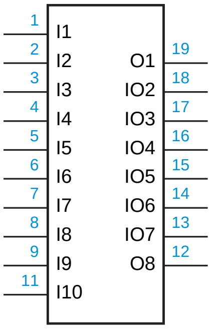


PAL20X10

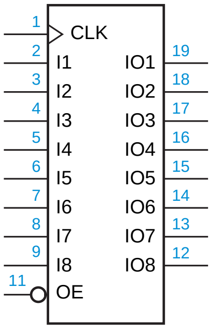


CMOS GAL Devices

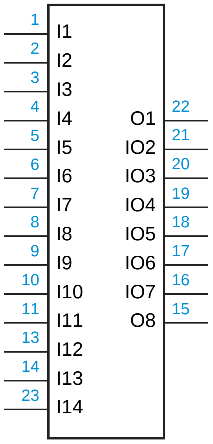
GAL16V8C



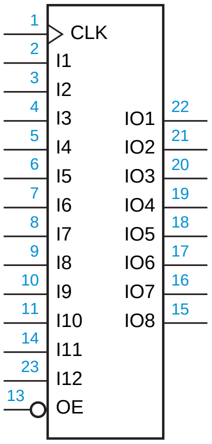
GAL16V8R



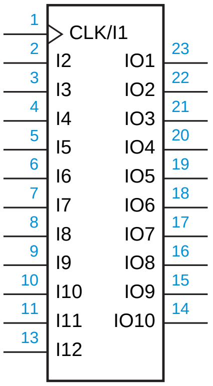
GAL20V8C



GAL20V8R

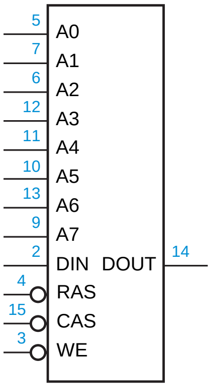


GAL22V10

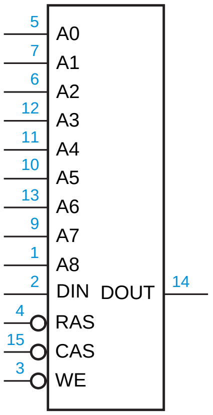


Dynamic RAMs

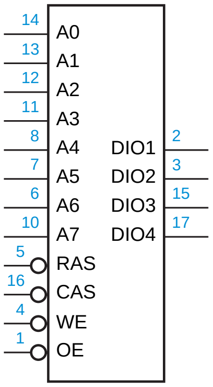
4164



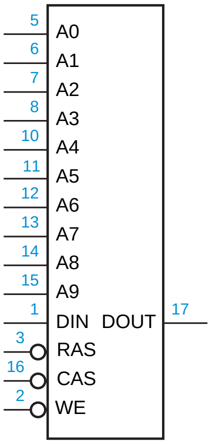
4256



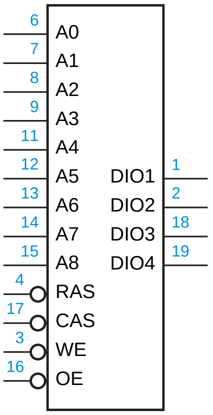
4464



41000

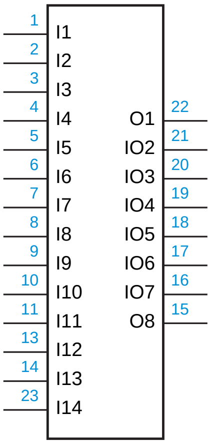


44256

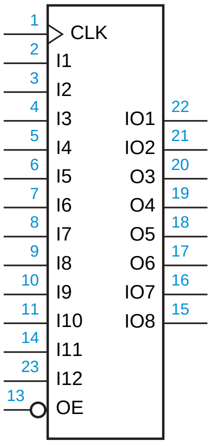


20R-Series 24-pin Bipolar PLDs

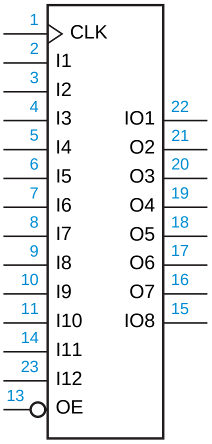
PAL20L8



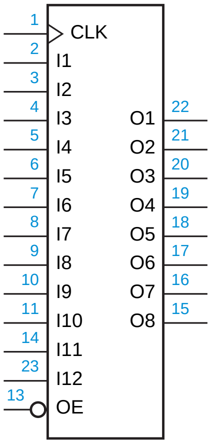
PAL20R4



PAL20R6

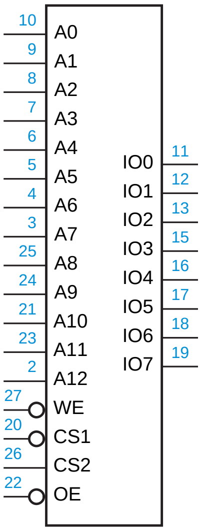


PAL20R8

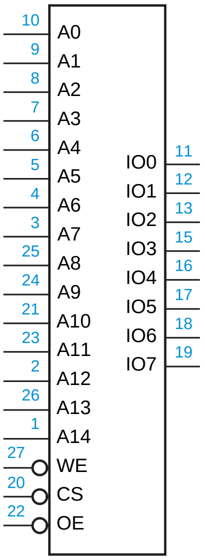


Static RAMs

HM6264



HM62256


10 A0
9 A1
8 A2
7 A3
6 A4
5 A5
4 A6
3 A7
25 A8
24 A9
21 A10
23 A11
2 A12
26 A13
1 A14
27 WE
20 CS
22 OE
IO0 11
IO1 12
IO2 13
IO3 15
IO4 16
IO5 17
IO6 18
IO7 19