TECHNICAL UNIVERSITY OF KOŠICE

FACULTY OF ELECTRICAL ENGINEERING AND INFORMATICS

DEPARTMENT OF ELECTRONICS AND MULTIMEDIA COMMUNICATIONS

BASIC OF ELECTRONICS

Lecture 13

2008/09

doc. Ing. Pavol Galajda, CSc.

Ing. Mária Gamcová, Ph.D.

Applied Informatics

Contents

References and Sources

[1] Attia, J. O.: *Electronics and Circuit Analysis using MATLAB*. CRC Press, Boca Raton London New York Washington, D.C., 1999.

[2] Fonstad, C. G: *Microelectronic Devices and Circuits.* McGraw-Hill Inc., New York, 1994.

[3] Galajda, P.– Lukáč, R.: *Elektronické prvky*. Merkury-Smékal, Košice, 2001.

[4] Galajda, P.– Lukáč, R.: *Elektronické obvody*. Merkury-Smékal, Košice, 2002.

[5] Rizzoni, G.*: Principles and Applications of Electrical Engineering,* 5th Edition. Ohio State University. McGraw-Hill Higher Education, 2007.

[6] Sandige, R.S.: *The Electrical Engineering Handbook*. Ed. Richard C. Dorf. Boca Raton: CRC Press LLC, 2000.

[7] Savant, C. J.– Roden, M. R – Carpenter, G. R.: *Electronic Circuit Design - An Engineering Approach*. The Benjamin/Cummings Publishing Company Inc., Menlo Park, California, 1987.

[8] Sedra, A. S.– Smith K. C.: *Microelectronic Circuits*. Oxford University Press, Inc., Oxford. New York, 1998.

79

Logic Elements

Gregory L. Moss
Purdue University

Peter Graham
Florida Atlantic University (Retired)

Richard S. Sandige
University of Wyoming

H. S. Hinton
University of Colorado

79.1 IC Logic Family Operation and Characteristics

Gregory L. Moss

Digital logic circuits can be classified as belonging to one of two categories, either combinational (also called combinatorial) or sequential logic circuits. The output logic level of a combinatorial circuit depends only on the current logic levels present at the circuit's inputs. Sequential logic circuits, on the other hand, have a memory characteristic so the sequential circuit's output is dependent not only on the current input conditions but also on the current output state of the circuit. The primary building block in combinational circuits is the logic gate. The three simplest logic gate functions are the inverter (or NOT), AND, and OR. Other common basic logic functions are derived from these three. Table 79.1 gives **truth table** definitions of the various types of logic gates. The memory elements used to construct sequential logic circuits are called latches and flip-flops.

The integrated circuit switching logic used in modern digital systems will generally be from one of three families: transistor-transistor logic (TTL), complementary metal-oxide semiconductor logic (CMOS), or emitter-coupled logic (ECL). Each of the logic families has its advantages and disadvantages. The three major families are also divided into various subfamilies derived from performance improvements in integrated circuit (IC) design technology. Bipolar transistors provide the switching action in both TTL and ECL families, while enhancement-mode MOS transistors are the basis for the CMOS family. Recent improvements in switching circuit performance are also attained using BiCMOS technology, the merging of bipolar and CMOS technologies on a single chip. A particular logic family is usually selected by digital designers based on such criteria as

1. Switching speed
2. Power dissipation
3. PC board area requirements (levels of integration)
4. Output drive capability (**fan-out**)
5. Noise immunity characteristics
6. Product breadth
7. Sourcing of components

TABLE 79.1 Defining Truth Tables for Logic Gates

1-Input Function		2-Input Functions							
Input	Output	Inputs		Output Functions					
A	NOT	A	B	AND	OR	NAND	NOR	XOR	XNOR
0	1	0	0	0	0	1	1	0	1
1	0	0	1	0	1	1	0	1	0
		1	0	0	1	1	0	1	0
		1	1	1	1	0	0	0	1

TABLE 79.2 Logic Families and Subfamilies

Family and Subfamily	Description
TTL	Transistor-transistor logic
74xx	Standard TTL
74Lxx	Low-power TTL
74Hxx	High-speed TTL
74Sxx	Schottky TTL
74LSxx	Low-power Schottky TTL
74ASxx	Advanced Schottky TTL
74ALSxx	Advanced low-power Schottky TTL
74Fxx	Fast TTL
CMOS	Complementary metal-oxide semiconductor
4xxx	Standard CMOS
74Cxx	Standard CMOS using TTL numbering system
74HCxx	High-speed CMOS
74HCTxx	High-speed CMOS—TTL compatible
74FCTxx	Fast CMOS—TTL compatible
74ACxx	Advanced CMOS
74ACTxx	Advanced CMOS—TTL compatible
74AHCxx	Advanced high-speed CMOS
74AHCTxx	Advanced high-speed CMOS-TTL compatible
ECL (or CML)	Emitter-coupled (current-mode) logic
10xxx	Standard ECL
10Hxxx	High-speed ECL

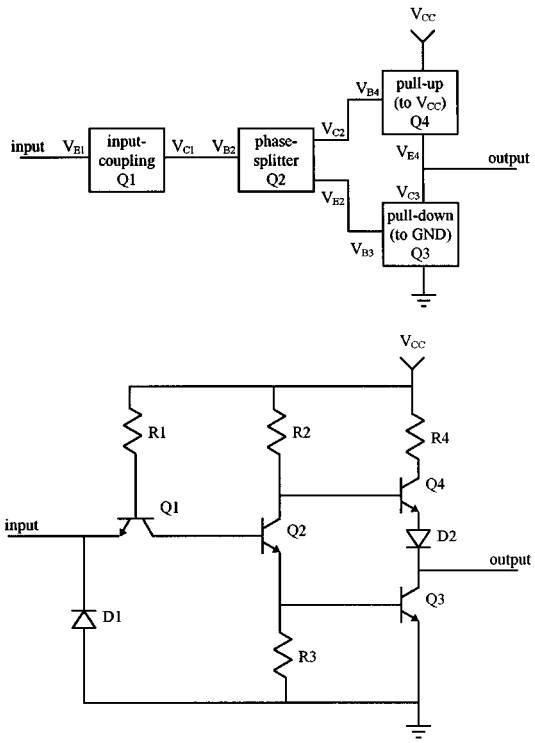
IC Logic Families and Subfamilies

The integrated circuit logic families actually consist of several subfamilies of ICs that differ in various performance characteristics. The TTL logic family has been the most widely used family type for applications that employ small-scale integration (SSI) or medium-scale integration (MSI) integrated circuits. Lower power consumption and higher levels of integration are the principal advantages of the CMOS family. The ECL family is generally used in applications that require high-speed switching logic. Today, the most common device numbering system used in the TTL and CMOS families has a prefix of 54 (generally used in military applications and having an operating temperature range of –55 to 125°C) and 74 (generally used in industrial/commercial applications and having an operating temperature range of 0 to 70°C). Table 79.2 identifies various logic families and subfamilies.

TTL Logic Family

The TTL family has been the most widely used logic family for many years in applications that use SSI and MSI. It is relatively fast and offers a great variety of standard chips.

The active switching element used in all TTL family circuits is the *npn* bipolar junction transistor (BJT). The transistor is turned on when the base is approximately 0.7 V more positive than the emitter and there is a sufficient amount of base current flowing. The turned on transistor (in non-Schottky subfamilies) is said to



input	V_{C1}	Q2	V_{C2}	V_{E2}	Q3	V_{C3}	Q4	V_{E4}	output
hi	hi	on	low	hi	on	low	off	open	low
low	low	off	hi	low	off	open	on	hi	hi

FIGURE 79.1 TTL inverter circuit block diagram and operation.

be in saturation and, ideally, acts like a closed switch between the collector and emitter terminals. The transistor is turned off when the base is not biased with a high enough voltage (with respect to the emitter). Under this condition, the transistor acts like an open switch between the collector and emitter terminals.

Figure 79.1 illustrates the transistor circuit blocks used in a standard TTL inverter. Four transistors are used to achieve the inverter function. The input to the gate connects to the emitter of transistor Q1, the input coupling transistor. A clamping diode on the input prevents negative input voltage spikes from damaging Q1. The collector voltage (and current) of Q1 controls Q2, the phase splitter transistor. Q2, in turn, controls the Q3 and Q4 transistors forming the output circuit, which is called a totem-pole arrangement. Q4 serves as a pull-up transistor to pull the output high when it is turned on. Q3 does just the opposite to the output and serves as a pull-down transistor. Q3 pulls the output low when it is turned on. Only one of the two transistors in the totem pole may be turned on at a time, which is the function of the phase splitter transistor Q2.

When a high **logic level** is applied to the inverter's input, Q1's base-emitter junction will be reverse biased and the base-collector junction will be forward biased. This circuit condition will allow Q1 collector current to flow into the base of Q2, saturating Q2 and thereby providing base current into Q3, turning it on also. The collector voltage of Q2 is too low to turn on Q4 so that it appears as an open in the top part of the totem pole. A diode between the two totem-pole transistors provides an extra voltage drop in series with the base-emitter junction of Q4 to ensure that Q4 will be turned off when Q2 is turned on. The saturated Q3 transistor brings the output near ground potential, producing a low output result for a high input into the inverter.

When a low logic level is applied to the inverter's input, Q1's base-emitter junction will be forward biased and the base-collector junction will be reverse biased. This circuit condition will turn on Q1 so that the collector terminal is shorted to the emitter and, therefore, to ground (low level). This low voltage is also on the base of Q2 and turns Q2 off. With Q2 off, there will be insufficient base current into Q3, turning it off also. Q2 leakage current is shunted to ground with a resistor to prevent the partial turning on of Q3. The collector voltage of

Q2 is pulled to a high potential with another resistor and, as a result, turns on Q4 so that it appears as a short in the top part of the totem pole. The saturated Q4 transistor provides a low resistance path from V_{CC} to the output, producing a high output result for a low input into the inverter.

A TTL NAND gate is very similar to the inverter circuit, with the exception that the input coupling transistor Q1 is constructed with multiple emitter-base junctions and each input to the NAND is connected to a separate emitter terminal. Any of the transistor's multiple emitters can be used to turn on Q1. The TTL NAND gate thus functions in the same manner as the inverter in that if any of the NAND gate inputs are low, the same circuit action will take place as with a low input to the inverter. Therefore, any time a low input is applied to the NAND gate it will produce a high ouput. Only if all of the NAND gate inputs are simultaneously high will it then produce the same circuit action as the inverter with its single input high, and the resultant output will be low. Input coupling transistors with up to eight emitter-base junctions, and therefore, eight input NAND gates, are constructed.

Storage time (the time it takes for the transistor to come out of saturation) is a major factor of propagation delay for saturated BJT transistors. A long storage time limits the switching speed of a standard TTL circuit. The propagation delay can be decreased and, therefore, the switching speed can be increased, by placing a Schottky diode between the base and collector of each transistor that might saturate. The resulting Schottky-clamped transistors do not go into saturation (effectively eliminating storage time) since the diode shunts current from the base into the collector before the transistor can achieve saturation. Today, digital circuit designs implemented with TTL logic almost exclusively use one of the Schottky subfamilies to take advantage of the significant improvement in switching speed.

CMOS Logic Family

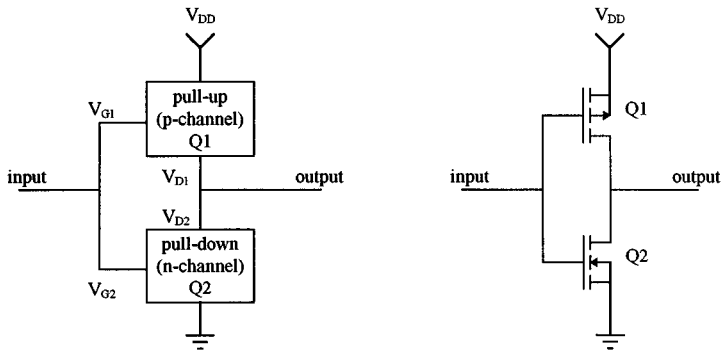
The active switching element used in all CMOS family circuits is the metal-oxide semiconductor field-effect transistor (MOSFET). CMOS stands for complementary MOS transistors and refers to the use of both types of MOSFET transistors, *n*-channel and *p*-channel, in the design of this type of switching circuit. While the physical construction and the internal physics of a MOSFET are quite different from that of the BJT, the circuit switching action of the two transistor types is quite similar. The MOSFET switch is essentially turned off and has a very high channel resistance by applying the same potential to the gate terminal as the source. An *n*-channel MOSFET is turned on and has a very low channel resistance when a high voltage with respect to the source is applied to the gate. A *p*-channel MOSFET operates in the same fashion but with opposite polarities; the gate must be more negative than the source to turn on the transistor.

A block diagram and schematic for a CMOS inverter circuit is shown in Fig. 79.2. Note that it is a simpler and much more compact circuit design than that for the TTL inverter. That fact is a major reason why MOSFET integrated circuits have a much higher circuit density than BJT integrated circuits and is one advantage that MOSFET ICs have over BJT ICs. As a result, CMOS is used in all levels of integration, from SSI through VLSI (very large scale integration).

When a high logic level is applied to the inverter's input, the *p*-channel MOSFET Q1 will be turned off and the *n*-channel MOSFET Q2 will be turned on. This will cause the output to be shorted to ground through the low resistance path of Q2's channel. The turned off Q1 has a very high channel resistance and acts nearly like an open.

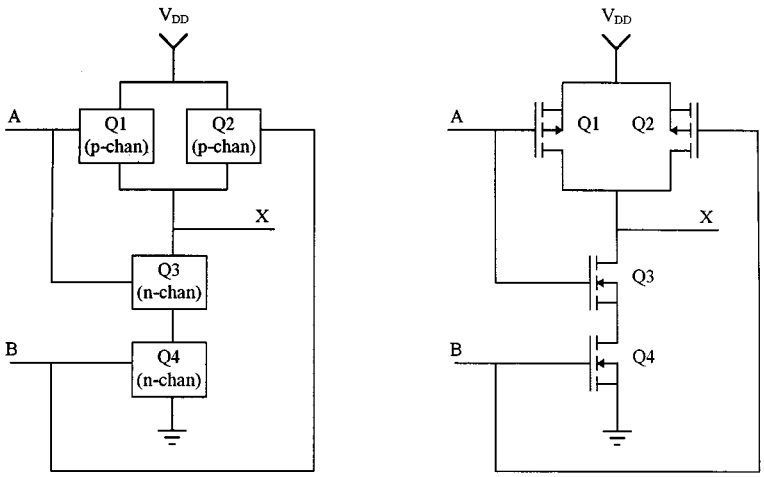
When a low logic level is applied to the inverter's input, the *p*-channel MOSFET Q1 will be turned on and the *n*-channel MOSFET Q2 will be turned off. This will cause the output to be shorted to V_{DD} through the low resistance path of Q1's channel. The turned off Q2 has a very high channel resistance and acts nearly like an open.

CMOS NAND gates are constructed by paralleling *p*-channel MOSFETs, one for each input, and putting in series an *n*-channel MOSFET for each input, as shown in the block diagram and schematic of Fig. 79.3. The NAND gate will produce a low output only when both Q3 and Q4 are turned on, creating a low resistance path from the output to ground through the two series channels. This can be accomplished by having a high on both input A and input B. This input condition will also turn off Q1 and Q2 . If either input A or input B or both is low, the respective parallel MOSFET will be turned on, providing a low resistance path for the output to V_{DD}. This will also turn off at least one of the series MOSFETs, resulting in a high resistance path for the output to ground.



input	Q1	V_{D1}	Q2	V_{D2}	output
hi	off	open	on	low	low
low	on	hi	off	open	hi

FIGURE 79.2 CMOS inverter circuit block diagram and operation.



inputs		transistors				output
A	B	Q1	Q2	Q3	Q4	X
low	low	on	on	off	off	hi
low	hi	on	off	off	on	hi
hi	low	off	on	on	off	hi
hi	hi	off	off	on	on	low

FIGURE 79.3 CMOS two-input NAND circuit block diagram and operation.

ECL Logic Family

ECL is a higher-speed logic family. While it does not offer as large a variety of IC chips as are available in the TTL family, it is quite popular for logic applications requiring high-speed switching.

The active switching element used in the ECL family circuits is also the *npn* BJT. Unlike the TTL family, however, which switches the transistors into saturation when turning them on, ECL switching is designed to prevent driving the transistors into saturation. Whenever bipolar transistors are driven into saturation, their switching speed will be limited by the charge carrier storage delay, a transistor operational characteristic. Thus, the switching speed of ECL circuits will be significantly higher than for TTL circuits. ECL operation is based

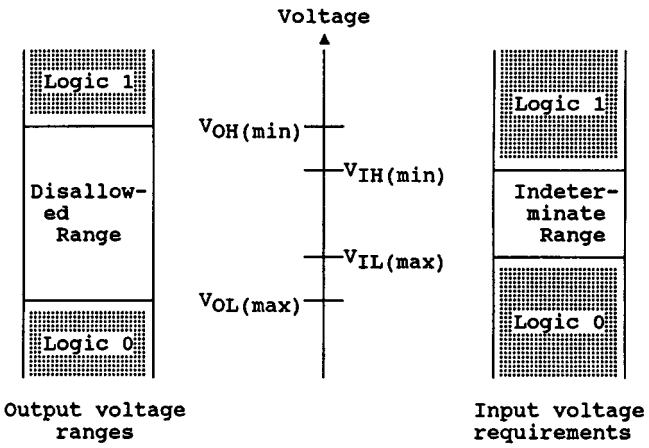


FIGURE 79.4 Switching device logic levels.

TABLE 79.3 Logic Signal Voltage Parameters for Selected Logic Subfamilies (in Volts)

Subfamily	$V_{OH(min)}$	$V_{OL(max)}$	$V_{IH(min)}$	$V_{IL(max)}$
74xx	2.4	0.4	2.0	0.8
74LSxx	2.7	0.5	2.0	0.8
74ASxx	2.5	0.5	2.0	0.8
74ALSxx	2.5	0.4	2.0	0.8
74Fxx	2.5	0.5	2.0	0.8
74HCxx	4.9	0.1	3.15	0.9
74HCTxx	4.9	0.1	2.0	0.8
74ACxx	3.8	0.4	3.15	1.35
74ACTxx	3.8	0.4	2.0	0.8
74AHCxx	4.5	0.1	3.85	1.65
74AHCTxx	3.65	0.1	2.0	0.8
10xxx	–0.96	–1.65	–1.105	–1.475
10Hxxx	–0.98	–1.63	–1.13	–1.48

on switching a fixed amount of bias current that is less than the saturation amount between two different transistors. The basic circuit found in the ECL family is the differential amplifier. One side of the differential amplifier is controlled by a bias circuit and the other is controlled by the logic inputs to the gate. This logic family is also referred to as current-mode logic (CML) because of its current switching operation.

Logic Family Circuit Parameters

Digital circuits and systems operate with only two states, logic 1 and 0, usually represented by two different voltage levels, a *high* and a *low*. The two logic levels actually consist of a range of values with the numerical quantities dependent upon the specific family that is used. Minimum high logic levels and maximum low logic levels are established by specifications for each family. Minimum device output levels for a logic high are called $V_{OH(min)}$ and minimum input levels are called $V_{IH(min)}$. The abbreviations for maximum output and input low logic levels are $V_{OL(max)}$ and $V_{IL(max)}$, respectively. Figure 79.4 shows the relationships between these parameters. Logic voltage level parameters are illustrated for selected prominent logic subfamilies in Table 79.3. As seen in this illustration, there are many operational incompatibilities between major logic family types.

Noise margin is a quantitative measure of a device's **noise immunity.** High-level noise margin (V_{NH}) and low-level noise margin (V_{NL}) are defined in Eqs. (79.1) and (79.2).

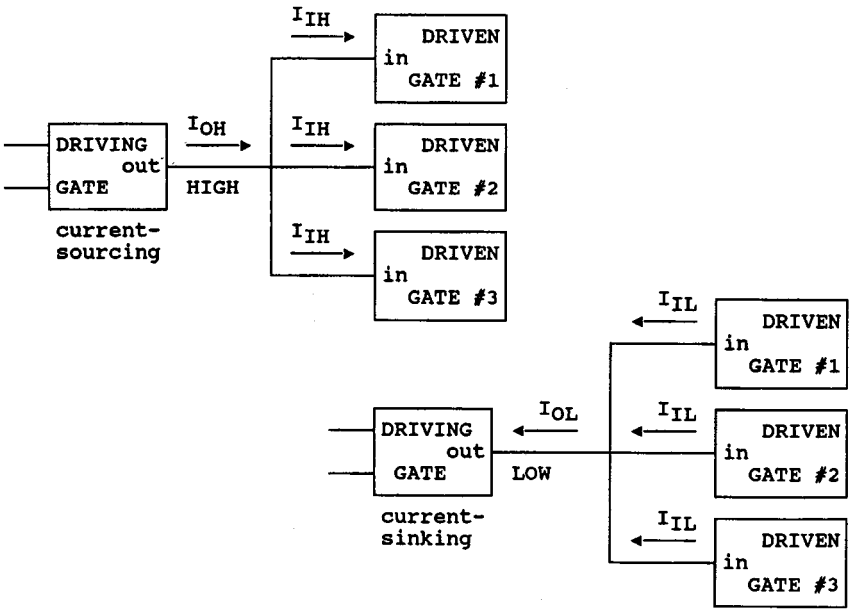


FIGURE 79.5 Current loading of driving gates.

TABLE 79.4 Worst Case Current Parameters for Selected Logic Subfamilies

Subfamily	$I_{OH(max)}$	$I_{OL(max)}$	$I_{IH(max)}$	$I_{IL(max)}$
74xx	–400 μA	16 mA	40 μA	–1.6 μA
74LSxx	–400 μA	8 mA	20 μA	–400 μA
74ASxx	–2 mA	20 mA	200 μA	–2 mA
74ALSxx	–400 μA	8 mA	20 μA	–100 μA
74Fxx	–1 mA	20 mA	20 μA	–0.6 mA
74HCxx	–4 mA	4 mA	1 μA	–1 μA
74HCTxx	–4 mA	4 mA	1 μA	–1 μA
74ACxx	–24 mA	24 mA	1 μA	–1 μA
74ACTxx	–24 mA	24 mA	1 μA	–1 μA
74AHCxx	–8 mA	8 mA	1 μA	–1 μA
74AHCTxx	–8 mA	8 mA	1 μA	–1 μA
10xxx	50 mA	–50 mA	–265 μA	500 nA
10Hxxx	50 mA	–50 mA	–265 μA	500 nA

$$V_{NH} = V_{OH(min)} - V_{IH(min)} \tag{79.1}$$

$$V_{NL} = V_{IL(max)} - V_{OL(max)} \tag{79.2}$$

Using the logic voltage values given in Table 79.3 for the selected subfamilies reveals that highest noise immunity is obtained with logic devices in the CMOS family, while lowest noise immunity is endemic to the ECL family.

Switching circuit outputs are loaded by the inputs of the devices that they are driving, as illustrated in Fig. 79.5. Worst case input loading current levels and output driving current capabilities are listed in Table 79.4 for various logic subfamilies. The fan-out of a driving device is the ratio between its output current capabilities at each logic level and the corresponding gate input current loading value. Switching circuits based on bipolar transistors have fan-out limited primarily by the current-sinking and current-sourcing capabilities of the driving device.

TABLE 79.5 Speed-Power Comparison for Selected Logic Subfamilies

Subfamily	Propagation Delay Time, ns (ave.)	Static Power Dissipation, mW (per gate)	Speed-Power Product, pJ
74xx	10	10	100
74LSxx	9.5	2	19
74ASxx	1.5	2	13
74ALSxx	4	1.2	5
74Fxx	3	6	18
74HCxx	8	0.003	24×10^{-3}
74HCTxx	14	0.003	42×10^{-3}
74ACxx	5	0.010	50×10^{-3}
74ACTxx	5	0.010	50×10^{-3}
74AHCxx	5.5	0.003	16×10^{-3}
74AHCTxx	5	0.003	14×10^{-3}
10xxx	2	25	50
10Hxxx	1	25	25

CMOS switching circuits are limited by the charging and discharging times associated with the output resistance of the driving gate and the input capacitance of the load gates. Thus, CMOS fan-out depends on the frequency of switching. With fewer (capacitive) loading inputs to drive, the maximum switching frequency of CMOS devices will increase.

The switching speed of logic devices is dependent on the device's **propagation delay time.** The propagation delay of a logic device limits the frequency at which it can be operated. There are two propagation delay times specified for logic gates: t_{PHL}, delay time for the output to change from high to low, and t_{PLH}, delay time for the output to change from low to high. Average typical propagation delay times for a single gate are listed for several logic subfamilies in Table 79.5. The ECL family has the fastest switching speed.

The amount of power required by an IC is normally specified in terms of the amount of current I_{CC} (TTL family), I_{DD} (CMOS family), or I_{EE} (ECL family) drawn from the power supply. For complex IC devices, the required supply current is given under specified test conditions. For TTL chips containing simple gates, the average power dissipation $P_{D(ave)}$ is normally calculated from two measurements, I_{CCH} (when all gate outputs are high) and I_{CCL} (when all gate outputs are low). Table 79.5 compares the static power dissipation of several logic subfamilies. The ECL family has the highest power dissipation, while the lowest is attained with the CMOS family. It should be noted that power dissipation for the CMOS family is directly proportional to the gate input signal frequency. For example, one would typically find that the power dissipation for a CMOS logic circuit would increase by a factor of 100 if the input signal frequency is increased from 1 kHz to 100 kHz.

The **speed-power product** is a relative figure of merit that is calculated by the formula given in Eq. (79.3). This performance measurement is normally expressed in picojoules (pJ).

$$\text{Speed-power product} = (t_{PHL} + t_{PLH})/2 \times P_{D(ave)} \tag{79.3}$$

A low value of speed-power product is desirable to implement high-speed (and, therefore, low propagation delay time) switching devices that consume low amounts of power. Because of the nature of transistor switching circuits, it is difficult to attain high-speed switching with low power dissipation. The continued development of new IC logic families and subfamilies is largely due to the trade-offs between these two device switching parameters. The speed-power product for various subfamilies is also compared in Table 79.5.

Interfacing Between Logic Families

The interconnection of logic chips requires that input and output specifications be satisfied. Figure 79.6 illustrates voltage and current requirements. The driving chip's $V_{OH(min)}$ must be greater than the driven circuit's $V_{IH(min)}$, and the driver's $V_{OL(max)}$ must be less than $V_{IL(max)}$ for the loading circuit. Voltage level shifters must be

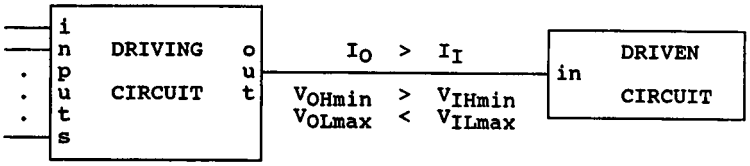


FIGURE 79.6 Circuit interfacing requirements.

used to interface the circuits together if these voltage requirements are not met. Of course, a driving circuit's output must not exceed the maximum and minimum allowable input voltages for the driven circuit. Also, the current sinking and sourcing ability of the driver circuit's output must be greater than the total current requirements for the loading circuit. Buffer gates or stages must be used if current requirements are not satisfied. All chips within a single logic family are designed to be compatible with other chips in the same family. Mixing chips from multiple subfamilies together within a single digital circuit can have adverse effects on the overall circuit's switching speed and noise immunity.

Defining Terms

Fan-out: The specification used to identify the limit to the number of loading inputs that can be reliably driven by a driving device's output.

Logic level: The high or low value of a voltage variable that is assigned to be a 1 or a 0 state.

Noise immunity: A logic device's ability to tolerate input voltage fluctuation caused by noise without changing its output state.

Propagation delay time: The time delay from when the input logic level to a device is changed until the resultant output change is produced by that device.

Speed-power product: An overall performance measurement that is used to compare the various logic families and subfamilies.

Truth table: A listing of the relationship of a circuit's output that is produced for various combinations of logic levels at the inputs.

Related Topic

25.3 Application-Specific Integrated Circuits

References

A. P. Chandrakasan and R. W. Brodersen, *Low Power Digital CMOS Design,* Boston: Kluwer Academic, 1995.

D. J. Comer, *Digital Logic and State Machine Design,* 2nd ed., Philadelphia: Saunders College Publishing, 1990.

S. H. K. Embabi, A. Bellaouar, and M. I. Elmasry, *Digital BiCMOS Integrated Circuit Design,* Boston: Kluwer Academic, 1993.

T. L. Floyd, *Digital Fundamentals,* 5th ed., Columbus, Ohio: Merrill Publishing Company, 1994.

K. Gopalan, *Introduction to Digital Microelectronic Circuits,* Chicago: Irwin, 1996.

J. D. Greenfield, *Practical Digital Design Using ICs,* 3rd ed., Englewood Cliffs, N.J.: Prentice-Hall, 1994.

R. J. Prestopnik, *Digital Electronics: Concepts and Applications for Digital Design,* Philadelphia: Saunders College Publishing, 1990.

R. S. Sandige, *Modern Digital Design,* New York: McGraw-Hill, 1990.

M. Shoji, *Theory of CMOS Digital Circuits and Circuit Failures,* Princeton, N.J.: Princeton University Press, 1992.

R. J. Tocci, *Digital Systems: Principles and Applications,* 6th ed., Englewood Cliffs, N.J.: Prentice-Hall, 1995.

S. H. Unger, *The Essence of Logic Circuits,* 2nd ed., New York: IEEE Press, 1996.

J. F. Wakerly, *Digital Design: Principles and Practices,* 2nd ed., Englewood Cliffs, N.J.: Prentice-Hall, 1994.

Further Information

Data Books and Device Index:

D. M. Howell, Ed. *IC Master,* Garden City, NY: Hearst Business Communications, annual.

Engineering Staff, *Advanced BiCMOS Technology Data Book,* Dallas: Texas Instruments, 1994.

Engineering Staff, *Advanced High-Speed CMOS Logic Data Book,* Dallas: Texas Instruments, 1996.

Engineering Staff, *ALS/AS Logic Data Book,* Dallas: Texas Instruments, 1995.

Engineering Staff, *ECLinPS Data,* Phoenix: Motorola, 1995.

Engineering Staff, *FACT Advanced CMOS Logic Databook,* Santa Clara, Calif: National Semiconductor Corporation, 1993.

Engineering Staff, *FACT Data,* Phoenix: Motorola, 1996.

Engineering Staff, *FACT & LS TTL Data,* Phoenix: Motorola, 1992.

Engineering Staff, *Low-Voltage Logic Data Book,* Dallas: Texas Instruments, 1996.

Engineering Staff, *MECL Data,* Phoenix: Motorola, 1993.

Journals and Trade Magazines:

EDN, Highlights Ranch, Colo.: Cahners Publishing.

Electronic Design, Cleveland, Ohio: Penton Publishing.

Electronic Engineering Times, Manhasset, N.Y.: CMP Publications.

IEEE Journal of Solid-State Circuits, New York: Institute of Electrical and Electronic Engineers.

IEEE Transactions on Circuits and Systems, Part I: Fundamental Theory and Applications, New York: Institute of Electrical and Electronic Engineers.

Internet Addresses for Digital Device Data Sheets:

Motorola, Inc.	http://Design-net.com
National Semiconductor Corp.	http://www.national.com/design/index.html
Texas Instruments, Inc.	http://www.ti.com/sc/docs/schome.htm

79.2 Logic Gates (IC)[1]

Peter Graham

This section introduces and analyzes the electronic circuit realizations of the basic gates of the three technologies: transistor-transistor logic (TTL), emitter-coupled logic (ECL), and complementary metal-oxide semiconductor (CMOS) logic. These circuits are commercially available on small-scale integration chips and are also the building blocks for more elaborate logic systems. The three technologies are compared with regard to speed, power consumption, and noise immunity, and parameters are defined which facilitate these comparisons. Also included are recommendations which are useful in choosing and using these technologies.

Gate Specification Parameters

Theoretically almost any logic device or system could be constructed by wiring together the appropriate configuration of the basic gates of the selected technology. In practice, however, the gates are interconnected during the fabrication process to produce a desired system on a single chip. The circuit complexity of a given chip is described by one of the following four rather broad classifications:

- **Small-Scale Integration (SSI).** The inputs and outputs of every gate are available for external connection at the chip pins (with the exception that exclusive OR and AND-OR gates are considered SSI).
- **Medium-Scale Integration (MSI).** Several gates are interconnected to perform somewhat more elaborate logic functions such as flip-flops, counters, multiplexers, etc.

[1]Based on P. Graham, "Gates," in *Handbook of Modern Electronics and Electrical Engineering,* C. Belove, Ed., New York: Wiley-Interscience, 1986, pp. 864–876. With permission.

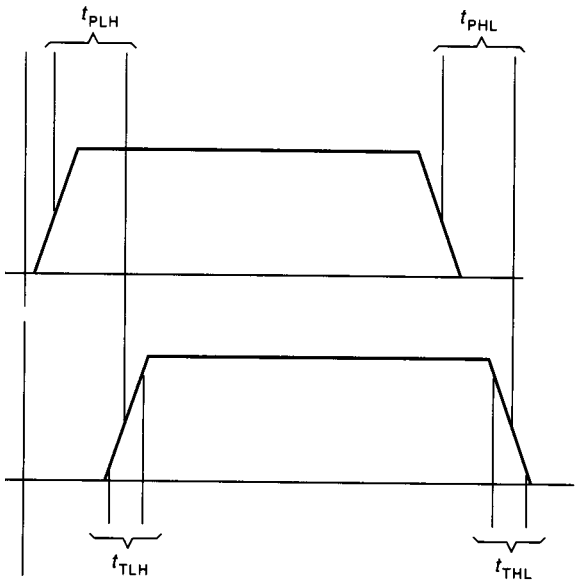


FIGURE 79.7 Definitions of switching times.

- **Large-Scale Integration (LSI).** Several of the more elaborate circuits associated with MSI are interconnected within the integrated circuit to form a logic system on a single chip. Chips such as calculators, digital clocks, and small microprocessors are examples of LSI.
- **Very-Large-Scale Integration (VLSI).** This designation is usually reserved for chips having a very high density, 1000 or more gates per chip. These include the large single-chip memories, gate arrays, and microcomputers.

Specifications of logic speed require definitions of switching times. These definitions can be found in the introductory pages of most data manuals. Four of them pertain directly to gate circuits. These are (see also Fig. 79.7):

- **LOW-to-HIGH Propagation Delay Time (t_{PLH}).** The time between specified reference points on the input and output voltage waveforms when the output is changing from low to high.
- **HIGH-to-LOW Propagation Delay Tune (t_{PHL}).** The time between specified reference points on the input and output voltage waveforms when the output is changing from high to low.
- **Propagation Delay Time (t_{PD}).** *The average of the two propagation delay times:* $t_{PD} = (t_{PD} + t_{PHL})/2$.
- **LOW-to-HIGH Transition Time (t_{TLH}).** *The rise time between specified reference points on the LOW-to-HIGH shift of the output waveform.*
- **HIGH-to-LOW Transition Time (t_{THL}).** *The fall time between specified reference points on the HIGH-to-LOW shift of the output waveform. The reference points usually are 10 and 90% of the voltage level difference in each case.*

Power consumption, driving capability, and effective loading of gates are defined in terms of currents.

- **Supply Current, Outputs High (I_{xxH}).** The current delivered to the chip by the power supply when all outputs are open and at the logical 1 level. The xx subscript depends on the technology.
- **Supply Current, Outputs Low (I_{xxL}).** The current delivered to the chip by the supply when all outputs are open and at the logical 0 level.
- **Supply Current, Worst Case (I_{xx}).** When the output level is unspecified, the input conditions are assumed to correspond to maximum supply current.

- **Input HIGH Current (I_{IH}).** The current flowing into an input when the specified HIGH voltage is applied.
- **Input LOW Current (I_{IL}).** The current flowing into an input when the specified LOW voltage is applied.
- **Output HIGH Current (I_{OH}).** The current flowing into the output when it is in the HIGH state. I_{OHmax} is the largest I_{OH} for which $V_{OH} \geq V_{OHmin}$ is guaranteed.
- **Output LOW Current (I_{OL}).** The current flowing into the output when it is in the LOW state. I_{OLmax} is the largest I_{OL} for which $V_{OL} \geq V_{OLmax}$ is guaranteed.

The most important voltage definitions are concerned with establishing ranges on the logical 1 (HIGH) and logical 0 (LOW) voltage levels.

- **Minimum High-Level Input Voltage (V_{IHmin}).** The least positive value of input voltage guaranteed to result in the output voltage level specified for a logical 1 input.
- **Maximum Low-Level Input Voltage (V_{ILmax}).** The most positive value of input voltage guaranteed to result in the output voltage level specified for a logical 0 input.
- **Minimum High-Level Output Voltage (V_{OHmin}).** The guaranteed least positive output voltage when the input is properly driven to produce a logical 1 at the output.
- **Maximum Low-Level Output Voltage (V_{OLmax}).** The guaranteed most positive output voltage when the input is properly driven to produce a logical 0 at the output.
- **Noise Margins.** $NM_H = V_{OHmin} - V_{IHmin}$ is how much larger the guaranteed least positive output logical 1 level is than the least positive input level that will be interpreted as a logical 1. It represents how large a negative-going glitch on an input 1 can be before it affects the output of the driven device. Similarly, $NM_L = V_{ILmax} - V_{OLmax}$ is the amplitude of the largest positive- going glitch on an input 0 that will not affect the output of the driven device.

Finally, three important definitions are associated with specifying the load that can be driven by a gate. Since in most cases the load on a gate output will be the sum of inputs of other gates, the first definition characterizes the relative current requirements of gate inputs.

- **Load Factor (LF).** Each logic family has a reference gate, each of whose inputs is defined to be a unit load in both the HIGH and the LOW conditions. The respective ratios of the input currents I_{IH} and I_{IL} of a given input to the corresponding I_{IH} and I_{IL} of the reference gate define the HIGH and LOW load factors of that input.
- **Drive Factor (DF).** A device output has drive factors for both the HIGH and the LOW output conditions. These factors are defined as the respective ratios of I_{OHmax} and I_{OLmax} of the gate to I_{OHmax} and I_{OLmax} of the reference gate.
- **Fan-Out.** For a given gate the fan-out is defined as the maximum number of inputs of the same type of gate that can be properly driven by that gate output. When gates of different load and drive factors are interconnected, fan-out must be adjusted accordingly.

Bipolar Transistor Gates

A logic circuit using bipolar junction transistors (BJTs) can be classified either as saturated or as nonsaturated logic. A saturated logic circuit contains at least one BJT that is saturated in one of the stable modes of the circuit. In nonsaturated logic circuits none of the transistors is allowed to saturate. Since bringing a BJT out of saturation requires a few additional nanoseconds (called the storage time), nonsaturated logic is faster. The fastest circuits available at this time are emitter-coupled logic (ECL), with transistor-transistor logic (TTL) having Schottky diodes connected to prevent the transistors from saturating (Schottky TTL) being a fairly close second. Both of these families are nonsaturated logic. All TTL families other than Schottky are saturated logic.

Transistor-Transistor Logic

TTL evolved from resistor-transistor logic (RTL) through the intermediate step of diode-transistor logic (DTL). All three families are catalogued in data books published in 1968, but of the three only TTL is still available.

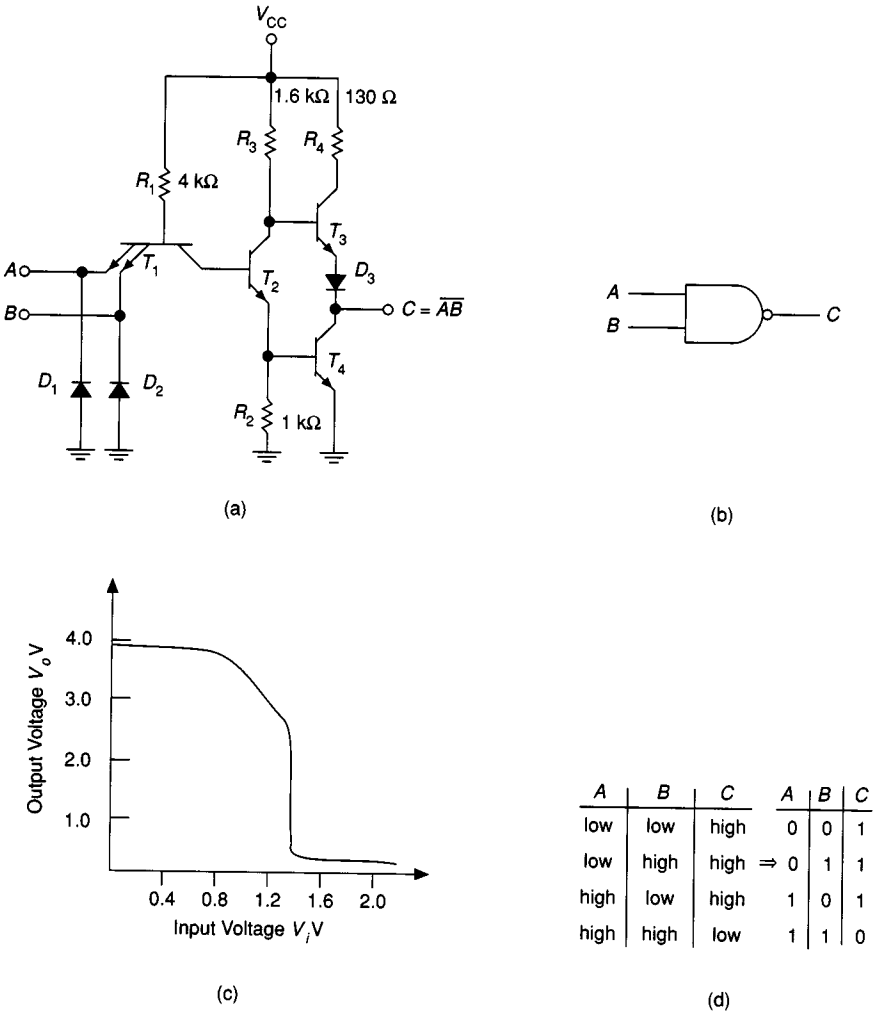


FIGURE 79.8 Two-input transistor-transistor logic (TTL) NAND gate type 7400: (a) circuit, (b) symbol, (c) voltage transfer characteristic (V_i to both inputs), (d) truth table.

The basic circuit of the standard TTL family is typified by the two-input NAND gate shown in Fig. 79.8(a). To estimate the operating levels of voltage and current in this circuit, assume that any transistor in saturation has $V_{CE} = 0.2$ and $V_{BE} = 0.75$ V. Let drops across conducting diodes also be 0.75 V and transistor current gains (when nonsaturated) be about 50. As a starting point, let the voltage levels at both inputs A and B be high enough that T_1 operates in the reversed mode. In this case the emitter currents of T_1 are negligible, and the current into the base of T_1 goes out the collector to become the base current of T_2. This current is readily calculated by observing that the base of T_1 is at $3 \times 0.75 = 2.25$ V so there is a 2.75-V drop across the 4-kΩ resistor. Thus $I_{B1} = I_{B2} = 0.7$ mA, and it follows that T_2 is saturated. With T_2 saturated, the base of T_3 is at $V_C + V_{BE4} = 0.95$ V. If T_4 is also saturated, the emitter of T_3 will be at $V_{D3} + V_{CE4} = 0.95$ V, and T_3 will be cut off. The voltage across the 1.6-kΩ resistor is $5 - 0.95 = 4.05$ V, so the collector current of T_2 is about 2.5 mA. This means the emitter current of T_2 is 3.2 mA. Of this, 0.75 mA goes through the 1-kΩ resistor, leaving 2.45 mA as the base current of T_4. Since the current gain of T_4 is about 50, it will be well into saturation for any collector current less than 100 mA, and the output at C is a logic 0. The corresponding minimum voltage levels required at the inputs are estimated from $V_{BE4} + V_{ECI}$, or about 1.7 V.

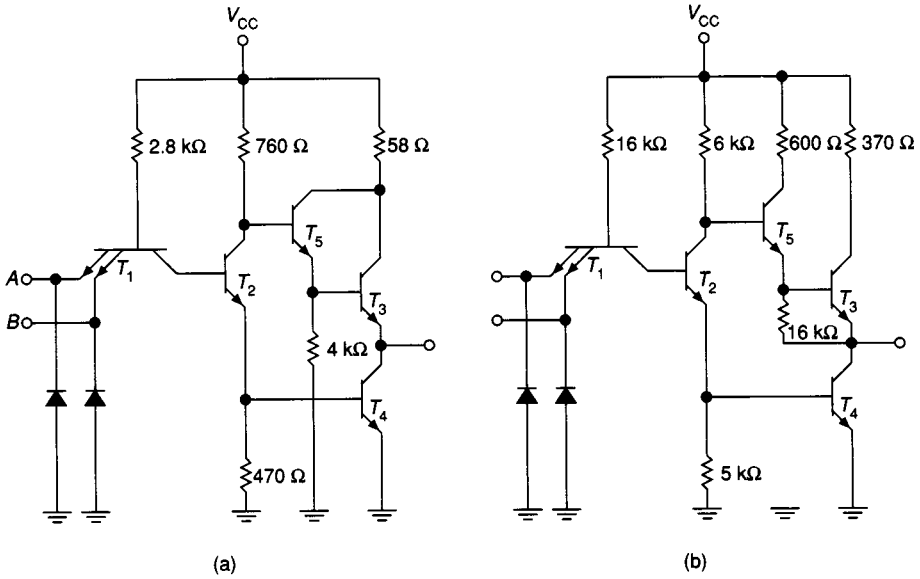


FIGURE 79.9 Modified transistor-transistor logic (TTL) two-input NAND states: (a) type 74Hxx, (b) type 74L00.

Now let either or both of the inputs be dropped to 0.2 V. T_1 is then biased to saturation in the normal mode, so the collector current of T_1 extracts the charge from the base region of T_2. With T_2 cut off, the base of T_4 is at 0 V and T_4 is cut off. T_3 will be biased by the current through the 1.6-kΩ resistor (R_3) to a degree regulated by the current demand at the output C. The drop across R_3 is quite small for light loads, so the output level at C will be $V_{CC} - V_{BE3} - V_{D3}$, which will be about 3.5 V corresponding to the logical 1.

The operation is summarized in the truth table in Fig. 79.8(d), identifying the circuit as a two-input NAND gate. The derivation of the input-output voltage transfer characteristic [Fig. 79.8(c)], where V_i is applied to inputs A and B simultaneously, can be found in most digital circuit textbooks. The sloping portion of the characteristic between $V_i = 0.55$ and 1.2 V corresponds to T_2 passing through the active region in going from cutoff to saturation.

Diodes D_1 and D_2 are present to damp out "ringing" that can occur, for example, when fast voltage level shifts are propagated down an appreciable length (20 cm or more) of microstripline formed by printed circuit board interconnections. Negative overshoots are clamped to the 0.7 V across the diode.

The series combination of the 130-Ω resistor, T_3, D_3, and T_4 in the circuit of Fig. 79.8(a), forming what is called the totem-pole output circuit, provides a low impedance drive in both the source (output $C = 1$) and sink (output $C = 0$) modes and contributes significantly to the relatively high speed of TTL. The available source and sink currents, which are well above the normal requirements for steady state, come into play during the charging and discharging of capacitive loads. Ideally T_3 should have a very large current gain and the 130-Ω resistor should be reduced to 0. The latter, however, would cause a short-circuit load current which would overheat T_3, since T_3 would be unable to saturate. All TTL families other than the standard shown in Fig. 79.8(a) use some form of Darlington connection for T_3, providing increased current gain and eliminating the need for diode D_3. The drop across D_3 is replaced by the base emitter voltage of the added transistor T_5. This connection appears in Fig. 79.9(a), an example of the 74Hxx series of TTL gates that increases speed at the expense of increased power consumption, and in Fig. 79.9(b), a gate from the 74Lxx series that sacrifices speed to lower power dissipation.

A number of TTL logic function implementations are available with open collector outputs. For example, the 7403 two-input NAND gate shown in Fig. 79.10 is the open collector version of Fig. 79.8(a). The open collector output has some useful applications. The current in an external load connected between the open collector and V_{CC} can be switched on and off in response to the input combinations. This load, for example,

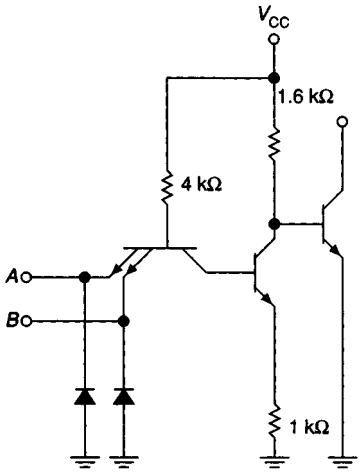


FIGURE 79.10 Open collector two-input NAND gate.

might be a relay, an indicator light, or an LED display. Also, two or more open collector gates can share a common load, resulting in the anding together of the individual gate functions. This is called a "wired-AND connection." In any application, there must be some form of load or the device will not function. There is a lower limit to the resistance of this load which is determined by the current rating of the open collector transistor. For wired-AND applications the resistance range depends on how many outputs are being wired and on the load being driven by the wired outputs. Formulas are given in the data books. Since the open collector configuration does not have the speed enhancement associated with an active pull-up, the low to high propagation delay (t_{PLH}) is about double that of the totem-pole output. It should be observed that totem-pole outputs should not be wired, since excessive currents in the active pull-up circuit could result.

Nonsaturated TTL. Two TTL families, the Schottky (74Sxx) and the low-power Schottky (74LSxx), can be classified as nonsaturating logic. The transistors in these circuits are kept out of saturation by the connection of Schottky diodes, with the anode to the base and the cathode to the collector.

Schottky diodes are formed from junctions of metal and an *n*-type semiconductor, the metal fulfilling the role of the *p*-region. Since there are thus no minority carriers in the region of the forward-biased junction, the storage time required to bring a *pn* junction out of saturation is eliminated. The forward-biased drop across a Schottky diode is around 0.3 V. This clamps the collector at 0.3 V less than the base, thus maintaining V_{CE} above the 0.3-V saturation threshold. Circuits for the two-input NAND gates 74LS00 and 74S00 are given in Fig. 79.11(a) and (b). The special transistor symbol is a short-form notation indicating the presence of the Schottky diode, as illustrated in Fig. 79.11(c).

Note that both of these circuits have an active pull-down transistor T_6 replacing the pull-down resistance connected to the emitter of T_2 in Fig. 79.9. The addition of T_6 decreases the turn-on and turn-off times of T_4. In addition, the transfer characteristic for these devices is improved by the squaring off of the sloping region between $V_i = 0.55$ and 1.2 V [see Fig. 79.8(c)]. This happens because T_2 cannot become active until T_6 turns on, which requires at least 1.2 V at the input.

The diode AND circuit of the 74LS00 in place of the multi-emitter transistor will permit maximum input levels substantially higher than the 5.5-V limit set for all other TTL families. Input leakage currents for 74LSxx are specified at $V_i = 10$ V, and input voltage levels up to 15 V are allowed. The 74LSxx has the additional feature of the Schottky diode D_1 in series with the 100-Ω output resistor. This allows the output to be pulled up to 10 V without causing a reverse breakdown of T_5. The relative characteristics of the several versions of the TTL two-input NAND gate are compared in Table 79.6. The 74F00 represents one of the new technologies that have introduced improved Schottky TTL in recent years.

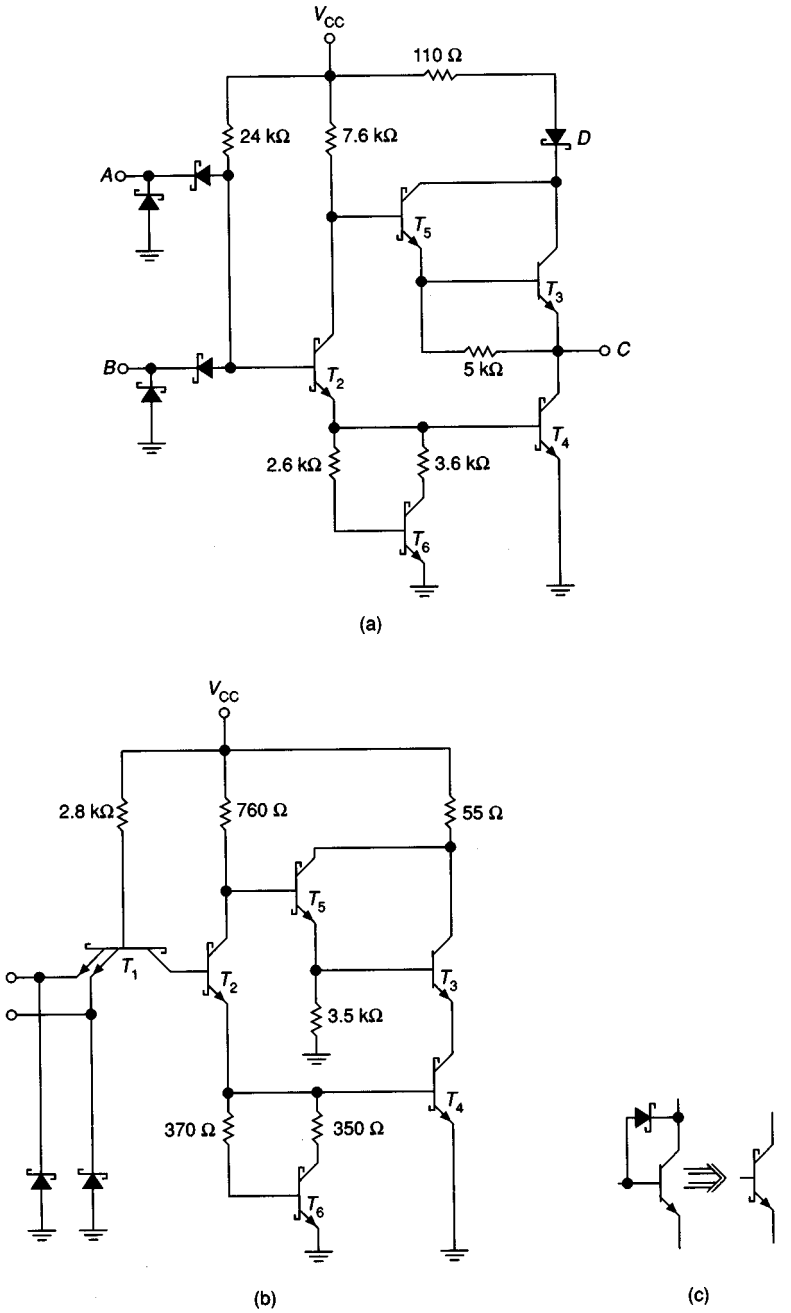


FIGURE 79.11 Transistor-transistor logic (TTL) nonsaturated logic. (a) Type 74LS00 two-input NAND gate, (b) type 74S00 two-input NAND gate, (c) significance of the Schottky transistor symbol.

TTL Design Considerations. Before undertaking construction of a logic system, the wise designer consults the information and recommendations provided in the data books of most manufacturers. Some of the more significant tips are provided here for easy reference.

1. **Power supply, decoupling, and grounding.** The power supply voltage should be 5 V with less than 5% ripple factor and better than 5% regulation. When packages on the same printed circuit board are

TABLE 79.6 Comparison of TTL Two-Input NANDGates

TTL Type	Supply Current I_{CCH}[a] (mA)	Supply Current I_{CCL} (mA)	Propagation Delay Time t_{PLH} (ns)	Propagation Delay Time t_{PHL} (ns)	Noise Margins NM_H (V)	Noise Margins NM_L (V)	Load Factor, H/L	Drive Factor, H/L	Fan-out
74F00	2.8	10.2	2.9	2.6	0.7	0.3	0.5/0.375	25/12.5	33
74S00	10	20	3	3	0.7	0.3	1.25/1.25	25/12.5	10
74H00	10	26	5.9	6.2	0.4	0.4	1.25/1.25	12.5/12.5	10
74LS00	0.8	2.4	9	10	0.7	0.3	0.5/0.25	10/5	20
7400	4	12	11	7	0.4	0.4	1/1	20/10	10
74L00	0.44	1.16	31	31	0.4	0.5	0.24/0.1125	5/2.25	20

[a]See text for explanation of abbreviations.

supplied by a bus there should be a 0.05-μF decoupling capacitor between the bus and the ground for every five to ten packages. If a ground bus is used, it should be as wide as possible, and should surround all the packages on the board. Whenever possible, use a ground plane. If a long ground bus is used, both ends must be tied to the common system ground point.

2. **Unused gates and inputs.** If a gate on a package is not used, its inputs should be tied either high or low, whichever results in the least supply current. For example, the 7400 draws three times the current with the output low as with the output high, so the inputs of an unused 7400 gate should be grounded. An unused input of a gate, however, must be connected so as not to affect the function of the active inputs. For a 7400 NAND gate, such an input must either be tied high or paralleled with a used input. It must be recognized that paralleled inputs count as two when determining the fan-out. Inputs that are tied high can be connected either to V_{CC} through a 1-kΩ or more resistance (for protection from supply voltage surges) or to the output of an unused gate whose input will establish a permanent output high. Several inputs can share a common protective resistance. Unused inputs of low-power Schottky TTL can be tied directly to V_{CC}, since 74LSxx inputs tolerate up to 15 V without breakdown. If inputs of low-power Schottky are connected in parallel and driven as a single input, the switching speed is decreased, in contrast to the situation with other TTL families.
3. **Interconnection.** Use of line lengths of up to 10 in. (5 in. for 74S) requires no particular precautions, except that in some critical situations lines cannot run side by side for an appreciable distance without causing cross talk due to capacitive coupling between them. For transmission line connections, a gate should drive only one line, and a line should be terminated in only one gate input. If overshoots are a problem, a 25- to 50-Ω resistor should be used in series with the driving gate input and the receiving gate input should be pulled up to 5 V through a 1-kΩ resistor. Driving and receiving gates should have their own decoupling capacitors between the V_{CC} and ground pins. Parallel lines should have a grounded line separating them to avoid cross talk.
4. **Mixing TTL subfamilies.** Even synchronous sequential systems often have asynchronous features such as reset, preset, load, and so on. Mixing high-speed 74S TTL with lower speed TTL (74LS for example) in some applications can cause timing problems resulting in anomalous behavior. Such mixing is to be avoided, with rare exceptions which must be carefully analyzed.

Emitter-Coupled Logic

ECL is a nonsaturated logic family where saturation is avoided by operating the transistors in the common collector configuration. This feature, in combination with a smaller difference between the HIGH and LOW voltage levels (less than 1 V) than other logic families, makes ECL the fastest logic available at this time. The circuit diagram of a widely used version of the basic two-input ECL gate is given in Fig. 79.12. The power supply terminals V_{CC1}, V_{CC2}, V_{EE}, and V_{TT} are available for flexibility in biasing. In normal operation, V_{CC1} and V_{CC2} are connected to a common ground, V_{EE} is biased to –5.2 V, and V_{TT} is biased to –2 V. With these values the nominal voltage for the logical 0 and 1 are, respectively, –1.75 and –0.9 V. Operation with the V_{CC} terminals grounded maximizes the immunity from noise interference.

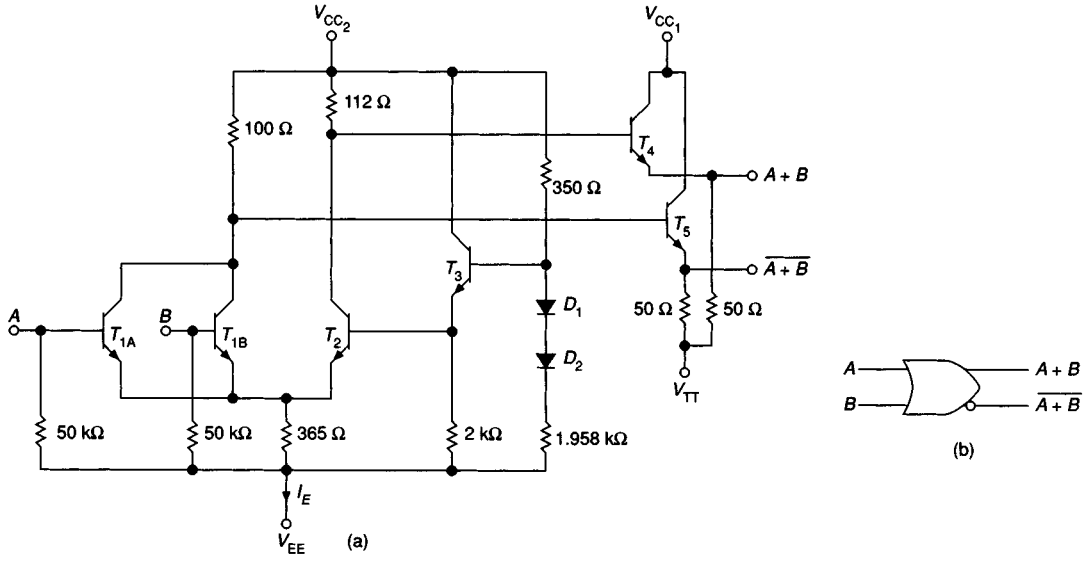


FIGURE 79.12 Emitter-coupled logic basic gate (ECL 10102): (a) circuit, (b) symbol.

A brief description of the operation of the circuit will verify that none of the transistors saturates. For the following discussion, V_{CC1} and V_{CC2} are grounded, V_{EE} is –5.2 V, and V_{TT} is –2 V. Diode drops and base-emitter voltages of active transistors are 0.8 V.

First, observe that the resistor-diode (D_1 and D_2) voltage divider establishes a reference voltage of –0.55 V at the base of T_3, which translates to –1.35 V at the base of T_2. When either or both of the inputs A and B are at the logical 1 level of –0.9 V, the emitters of T_{1A}, T_{1B}, and T_2 will be 0.8 V lower, at –1.7 V. This establishes the base-emitter voltage of T_2 at –1.35 – (–1.7) = 0.35 V, so T_2 is cut off. With T_2 off, T_4 is biased into the active region, and its emitter will be at about –0.9 V, corresponding to a logical 1 at the $(A + B)$ output. Most of the current through the 365-Ω emitter resistor, which is [–1.7 – (–5.2)]/0.365 = 9.6 mA, flows through the 100-Ω collector resistor, dropping the base voltage of T_5 to –0.96 V. Thus the voltage level at the output terminal designated $(A + B)$ is –1.76 V, corresponding to a logical 0.

When both A and B inputs are at the LOW level of –1.75 V, T_2 will be active, with its emitter voltage at –1.35 – 0.8 = –2.15 V. The current through the 365-Ω resistor becomes [–2.15 – (–5.2)]/0.365 = 8.2 mA. This current flows through the 112-Ω resistor pulling the base of T_4 down to –0.94 V, so that the $(A + B)$ output will be at the LOW level of –1.75 V. With T_{1A} and T_{1B} cut off, the base of T_5 is close to 0.0 V, and the $(A + B)$ output will therefore be at the nominal HIGH level of –0.9 V.

Observe that the output transistors T_4 and T_5 are always active and function as emitter followers, providing the low-output impedances required for driving capacitive loads. As T_{1A} and/or T_{1B} turn on, and T_2 turns off as a consequence, the transition is accomplished with very little current change in the 365-Ω emitter resistor. It follows that the supply current from V_{EE} does not undergo the sudden increases and decreases prevalent in TTL, thus eliminating the need for decoupling capacitors. This is a major reason why ECL can be operated successfully with the low noise margins which are inherent in logic having a relatively small voltage difference between the HIGH and LOW voltage levels (see Table 79.7). The small level shifts between LOW and HIGH also permit low propagation times without excessively fast rise and fall times. This reduces the effects of residual capacitive coupling between gates, thereby lessening the required noise margin. For this reason the faster ECL (100xxx) should not be used where the speed of the 10xxx series is sufficient. A comparison of three ECL series is given in Table 79.7. The propagation times t_{PLH} and t_{PHL} and transition times t_{TLH} and t_{THL} are defined in Fig. 79.7. Transitions are between the 20 and 80% levels.

TABLE 79.7 Comparison of ECL Quad Two-Input NOR Gates ($V_{TT} = V_{EE} = 5.2$ V, $V_{CC1} = 0$ V)

ECL Type	Power Supply Terminal V_{EE} (V)	Power Supply Current I_E (mA)	Propagation Delay Time t_{PLH}[a] (ns)	Propagation Delay Time t_{PHL} (ns)	Transition Time t_{TLH}[b] (ns)	Noise Margins t_{THL}[b] (ns)	Noise Margins NM_H (V)	Noise Margins NM_L (V)	Test Load
ECL II									
1012	–5.2	18[c]	5	4.5	4	6	0.175	0.175	Fan-out of 3
95102	–5.2	11	2	2	2	2	0.14	0.145	50 Ω
10102	–5.2	20	2	2	2.2	2.2	0.135	0.175	50 Ω
ECLIII									
1662	–5.2	56[c]	1	1.1	1.4	1.2	0.125	0.125	50 Ω
100102[d]	–4.5	55	0.75	0.75	0.7	0.7	0.14	0.145	50 Ω
11001[e]	–5.2	24	0.7	0.7	0.7	0.7	0.145	0.175	50 Ω

[a] See text for explanation of abbreviations.
[b] 20 to 80% levels.
[c] Maximum value (all other typical).
[d] Quint 2-input NOR/OR gate.
[e] Dual 5/4-input NOR/OR gate.

The 50-Ω pull-down resistors shown in Fig. 79.12 are connected externally. The outputs of several gates can therefore share a common pull-down resistor to form a wired-OR connection. The open emitter outputs also provide flexibility for driving transmission lines, the use of which in most cases is mandatory for interconnecting this high-speed logic. A twisted pair interconnection can be driven using the complementary outputs $(A + B)$ and $(A + B)$ as a differential output. Such a line should be terminated in an ECL line receiver (10114).

Since ECL is used in high-speed applications, special techniques must be applied in the layout and interconnection of chips on circuit boards. Users should consult design handbooks published by the suppliers before undertaking the construction of an ECL logic system.

While ECL is not compatible with any other logic family, interfacing buffers, called translators, are available. In particular, the 10124 converts TTL output levels to ECL complementary levels, and the 10125 converts either single-ended or differential ECL outputs to TTL levels. Among other applications of these translators, they allow the use of ECL for the highest speed requirements of a system while the rest of the system uses the more rugged TTL. Another translator is the 10177, which converts the ECL output levels to *n*-channel metal-oxide semiconductor (NMOS) levels. This is designed for interfacing ECL with *n*-channel memory systems.

Complementary Metal-Oxide Semiconductor (CMOS) Logic

Metal-oxide semiconductor (MOS) technology is prevalent in LSI systems due to the high circuit densities possible with these devices. *p*-Channel MOS was used in the first LSI systems, and it still is the cheapest to produce because of the higher yields achieved due to the longer experience with PMOS technology. PMOS, however, is largely being replaced by NMOS (*n*-channel MOS), which has the advantages of being faster (since electrons have greater mobility than holes) and having TTL compatibility. In addition, NMOS has a higher function/chip area density than PMOS, the highest density in fact of any of the current technologies. Use of NMOS and PMOS, however, is limited to LSI and VLSI fabrications. The only MOS logic available as SSI and MSI is CMOS (complementary MOS).

CMOS is faster than NMOS and PMOS, and it uses less power per function than any other logic. While it is suitable for LSI, it is more expensive and requires somewhat more chip area than NMOS or PMOS. In many respects it is unsurpassed for SSI and MSI applications. Standard CMOS (the 4000 series) is as fast as low-power TTL (74Lxx) and has the largest noise margin of any logic type.

A unique advantage of CMOS is that for all input combinations the steady-state current from V_{DD} to V_{SS} is almost zero because at least one of the series FETs is open. Since CMOS circuits of any complexity are interconnections of the basic gates, the quiescent currents for these circuits are extremely small, an obvious advantage which becomes a necessity for the practicality of digital watches, for example, and one which alleviates

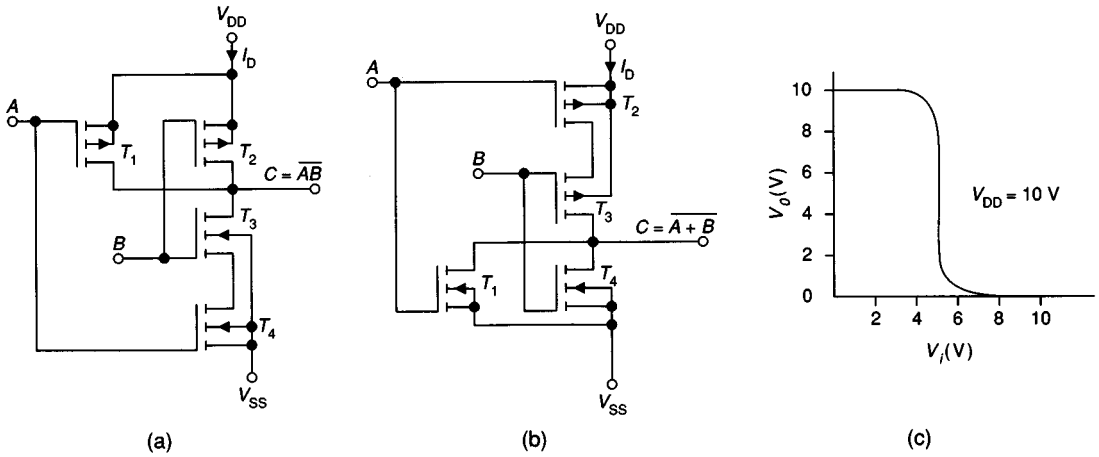


FIGURE 79.13 (a) Complementary metal-oxide semiconductor (CMOS) NAND gate, (b) NOR gate, and (C) inverter transfer characteristic.

heat dissipation problems in high-density chips. Also a noteworthy feature of CMOS digital circuits is the absence of components other than FETs. This attribute, which is shared by PMOS and NMOS, accounts for the much higher function/chip area density than is possible with TTL or ECL. During the time the output of a CMOS gate is switching there will be current flow from V_{DD} to V_{SS}, partly due to the charging of junction capacitances and partly because the path between V_{DD} and V_{SS} closes momentarily as the FETs turn on and off. This causes the dc supply current to increase in proportion to the switching frequency in a CMOS circuit. Manufacturers specify that the supply voltage for standard CMOS can range over $3\ V \le V_{DD} - V_{SS} \le 18\ V$, but switching speeds are slower at the lower voltages, mainly due to the increased resistances of the "on" transistors. The output switches between low and high when the input is midway between V_{DD} and V_{SS}, and the output logical 1 level will be V_{DD} and the logical 0 level V_{SS} [Fig. 79.13(c)]. If CMOS is operated with $V_{DD} = 5$ V and $V_{SS} = 0$ V, the V_{DD} and V_{SS} levels will be almost compatible with TTL except that the TTL totem-pole output high of 3.4 V is marginal as a logical 1 for CMOS. To alleviate this, when CMOS is driven with TTL a 3.3-kΩ pull-up resistor between the TTL output and the common V_{CC}, V_{DD} supply terminal should be used. This raises V_{OH} of the TTL output to 5 V.

All CMOS inputs are diode protected to prevent static charge from accumulating on the FET gates and causing punch-through of the oxide insulating layer. A typical configuration is illustrated in Fig. 79.14. Diodes D_1 and D_2 clamp the transistor gates between V_{DD} and V_{SS}. Care must be taken to avoid input voltages that would cause excessive diode currents. For this reason manufacturers specify an input voltage constraint from $V_{SS} - 0.5$ V to $V_{DD} + 0.5$ V. The resistance R_s helps protect the diodes from excessive currents but is introduced at the expense of switching speed, which is deteriorated by the time constant of this resistance and the junction capacitances.

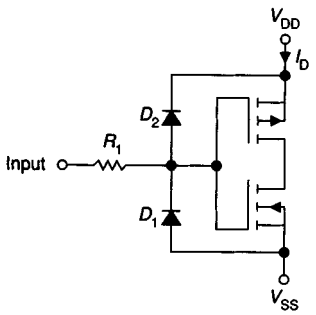


FIGURE 79.14 Diode protection of input transistor gates. $200\ \Omega < R_s < 1.5\ k\Omega$.

Advanced versions of CMOS have been developed which are faster than standard CMOS. The first of these to appear were designated 74HCxx and 74HCTxx. The supply voltage range for this series is limited to $2\ V \le V_{DD} - V_{SS} \le 6\ V$. The pin numbering of a given chip is the same as its correspondingly numbered TTL device. Furthermore, gates with the HCT code have skewed transfer characteristics which match those of its TTL cousin, so that these chips can be directly interchanged with low-power Schottky TTL.

TABLE 79.8 Comparison of Standard, High-Speed, and Advanced High-Speed CMOS

			Standard CMOS NORGates		High-Speed CMOS Inverter		Advanced CMOS Inverter	
Parameter	Symbol	Unit	4001B	4011UB	74HC04	74HCT04	74AC04	74ACT04
Supply voltage	V_{DD}-V_{SS}	V	15	15	6	5.5	5.5	5.5
Input voltage	V_{IHmin}	V	11	12.5	4.2	2	3.85	2
thresholds	V_{ILmax}	V	4	2.5	1.8	0.8	1.65	0.8
Guaranteed output	V_{OHmin}	V	13.5	13.5	5.9	4.5	4.86	4.76
levels at	V_{OLmax}	V	1.5	1.5	0.1	0.26	0.32	0.37
maximum IO								
Maximum	I_{OH}	mA	–8.8	–3.5	–4	–4	–24	–24
output currents	I_{OL}	mA	8.8	8.8	4	4	24	24
Noise	NM_L	V	2.5	2.5	1.7	0.54	1.33	.43
margins	NM_H	V	2.5	2.5	1.7	2.5	1.01	1.24
Propagation	t_{PLH}	ns	40	40	16	15	4	4.3
times	t_{PHL}	ns	40	40	16	17	3.5	3.9
Max input	I_{INmax}	μA	0.1	0.1	0.1	0.1	0.1	0.1
current leakage								
D-flip-flop			4013B		74HC374	74HCT374A	74AC374	74ACT374
max frequency	f_{max}	MHz	7.0	N.A.	35	30	100	100
(guaranteed								
minimum)								

More recently, a much faster CMOS has appeared and carries the designations 74ACxx and 74ACTxx. These operate in the same supply voltage range and bear the same relationship with TTL as the HCMOS. The driving capabilities (characterized by I_{OH} and I_{OL}) of this series are much greater, such that they can be fanned out to 10 low-power Schottky inputs.

The three types of CMOS are compared in Table 79.8. The relative speeds of these technologies are best illustrated by including in the table the maximum clock frequencies for *D* flip-flops. In each case, the frequency given is the maximum for which the device is guaranteed to work. It is worth noting that a typical maximum clocking of 160 MHz is claimed for the 74ACT374 *D* flip-flop.

CMOS Design Considerations

Design and handling recommendations for CMOS, which are included in several of the data books, should be consulted by the designer using this technology. A few selected recommendations are included here to illustrate the importance of such information.

1. All unused CMOS inputs should be tied either to V_{DD} or V_{SS}, whichever is appropriate for proper operation of the gate. This rule applies even to inputs of unused gates, not only to protect the inputs from possible static charge buildup, but to avoid unnecessary supply current drain. Floating gate inputs will cause all the FETs to be conducting, wasting power and heating the chip unnecessarily.
2. CMOS inputs should never be driven when the supply voltage V_{DD} is off, since damage to the input-protecting diodes could result. Inputs wired to edge connectors should be shunted by resistors to V_{DD} or V_{SS} to guard against this possibility.
3. Slowly changing inputs should be conditioned using Schmitt trigger buffers to avoid oscillations that can arise when a gate input voltage is in the transition region.
4. Wired-AND configurations cannot be used with CMOS gates, since wiring an output HIGH to an output LOW would place two series FETs in the "on" condition directly across the chip supply.
5. Capacitive loads greater than 5000 pF across CMOS gate outputs act as short circuits and can overheat the output FETs at higher frequencies.
6. Designs should be used that avoid the possibility of having low impedances (such as generator outputs) connected to CMOS inputs prior to power-up of the CMOS chip. The resulting current surge when V_{DD} is turned on can damage the input diodes.

While this list of recommendations is incomplete, it should alert the CMOS designer to the value of the information supplied by the manufacturers.

Choosing a Logic Family

A logic designer planning a system using SSI and MSI chips will find that an extensive variety of circuits is available in all three technologies: TTL, ECL, and CMOS. The choice of which technology will dominate the system is governed by what are often conflicting needs, namely, speed, power consumption, noise immunity, cost, availability, and the ease of interfacing. Sometimes the decision is easy. If the need for a low static power drain is paramount, CMOS is the only choice. It used to be the case that speed would dictate the selection; ECL was high speed, TTL was moderate, and CMOS low. With the advent of advanced TTL and, especially, advanced CMOS the choice is no longer clear-cut. All three will work at 100 MHz or more. ECL might be used since it generates the least noise because the transitions are small, yet for that same reason it is more susceptible to externally generated noise. Perhaps TTL might be the best compromise between noise generation and susceptibility. Advanced CMOS is the noisiest because of its rapid rise and fall times, but the designer might opt to cope with the noise problems to take advantage of the low standby power requirements.

A good rule is to use devices which are no faster than the application requires and which consume the least power consistent with the needed driving capability. The information published in the manufacturers' data books and designer handbooks is very helpful when choice is in doubt.

Defining Term

Logic gate: Basic building block for logic systems that controls the flow of pulses.

Related Topics

25.3 Application-Specific Integrated Circuits • 81.2 Logic Circuits

References

Advanced CMOS Logic Designers Handbook, Dallas: Texas Instruments, Inc., 1987.
C. Belove and D. Schilling, *Electronic Circuits, Discrete and Integrated,* 2nd ed., New York: McGraw-Hill, 1979.
FACT Data, Phoenix: Motorola Semiconductor Products, Inc., 1989.
Fairchild Advanced Schottky TTL, California: Fairchild Camera and Instrument Corporation, 1980.
W. I. Fletcher, *An Engineering Approach to Digital Design,* Englewood Cliffs, N.J.: Prentice-Hall, 1980.
High Speed CMOS Logic Data, Phoenix: Motorola Semiconductor Products, Inc., 1989.
P. Horowitz and W. Hill, *The Art of Electronics,* 2nd ed., New York: Cambridge University Press, 1990.
MECL System Design Handbook, Phoenix: Motorola Semiconductor Products, Inc., 1988.
H. Taub and D. Schilling, *Digital Integrated Electronics,* New York: McGraw-Hill, 1977.
The TTL Data Book for Design Engineers, Dallas: Texas Instruments, Inc., 1990.

Further Information

An excellent presentation of the practical design of logic systems using SSI and MSI devices is developed in the referenced book *An Engineering Approach to Digital Design* by William I. Fletcher. The author pays particular attention to the importance of device speed and timing.

The Art of Electronics by Horowitz and Hill is particularly helpful for its practical approach to interfacing digital with analog.

Everything one needs to know about digital devices and their interconnection can be found somewhere in the data manuals, design handbooks, and application notes published by the device manufacturers. Unfortunately, no single publication has it all, so the serious user should acquire as large a collection of these sources as possible.

CHAPTER

13

Digital Logic Circuits

Digital computers have taken a prominent place in engineering and science over the last two decades, performing a number of essential functions such as numerical computations and data acquisition. It is not necessary to further stress the importance of these electronic systems in this book, since you are already familiar with personal computers and programming languages. The objective of the chapter is to discuss the essential features of digital logic circuits, which are at the heart of digital computers, by presenting an introduction to *combinational logic circuits.*

The chapter starts with a discussion of the binary number system, and continues with an introduction to Boolean algebra. The self-contained treatment of Boolean algebra will enable you to design simple logic functions using the techniques of combinational logic, and several practical examples are provided to demonstrate that even simple combinations of logic gates can serve to implement useful circuits in engineering practice. In a later section, we introduce a number of logic modules which can be described using simple logic gates but which provide more advanced functions. Among these, we discuss read-only memories, multiplexers, and decoders. Throughout the chapter, simple examples are given to demonstrate the usefulness of digital logic circuits in various engineering applications.

Chapter 13 provides the background needed to address the study of digital systems, which will be undertaken in Chapter 14. Upon completion of the chapter, you should be able to:

- Perform operations using the binary number system.
- Design simple combinational logic circuits using logic gates.
- Use Karnaugh maps to realize logical expressions.
- Interpret data sheets for multiplexers, decoders, and memory ICs.

13.1 ANALOG AND DIGITAL SIGNALS

One of the fundamental distinctions in the study of electronic circuits (and in the analysis of any signals derived from physical measurements) is that between analog and digital signals. As discussed in the preceding chapter, an **analog signal** is an electrical signal whose value varies in analogy with a physical quantity (e.g., temperature, force, or acceleration). For example, a voltage proportional to a measured variable pressure or to a vibration naturally varies in an analog fashion. Figure 13.1 depicts an analog function of time, $f(t)$. We note immediately that for each value of time, t, $f(t)$ can take one value among any of the values in a given range. For example, in the case of the output voltage of an op-amp, we expect the signal to take any value between $+V_{\text{sat}}$ and $-V_{\text{sat}}$, where V_{sat} is the supply-imposed saturation voltage.

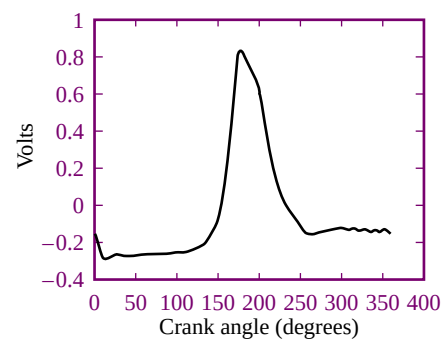


Figure 13.1 Voltage analog of internal combustion engine in-cylinder pressure

A **digital signal,** on the other hand, can take only a *finite number of values*. This is an extremely important distinction, as will be shown shortly. An example of a digital signal is a signal that allows display of a temperature measurement on a digital readout. Let us hypothesize that the digital readout is three digits long and can display numbers from 0 to 100, and let us assume that the temperature sensor is correctly calibrated to measure temperatures from 0 to 100°F. Further, the output of the sensor ranges from 0 to 5 volts, where 0 V corresponds to 0°F and 5 V to 100°F. Therefore, the calibration constant of the sensor is

$$k_T = \frac{100^\circ - 0^\circ}{5 - 0} = 20^\circ \text{ V}$$

Clearly, the output of the sensor is an analog signal; however, the display can show only a finite number of readouts (101, to be precise). Because the display itself can only take a value out of a discrete set of states—the integers from 0 to 100—we call it a digital display, indicating that the variable displayed is expressed in digital form.

Now, each temperature on the display corresponds to a *range of voltages:* each digit on the display represents one hundredth of the 5-volt range of the sensor, or 0.05 V = 50 mV. Thus, the display will read 0 if the sensor voltage is between 0 and 49 mV, 1 if it is between 50 and 99 mV, and so on. Figure 13.2 depicts the staircase function relationship between the analog voltage and the digital readout. This **quantization** of the sensor output voltage is in effect an approximation. If one wished to know the temperature with greater precision, a greater number of display digits could be employed.

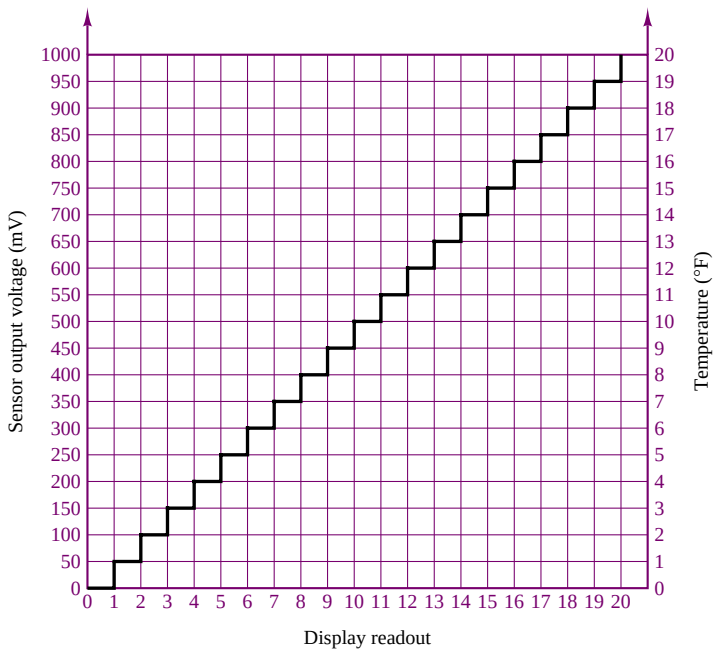


Figure 13.2 Digital representation of an analog signal

The most common digital signals are binary signals. A **binary signal** is a signal that can take only one of two discrete values and is therefore characterized by transitions between two states. Figure 13.3 displays a typical binary signal. In binary arithmetic (which we discuss in the next section), the two discrete values f_1 and f_0 are represented by the numbers 1 and 0. In binary voltage waveforms, these values are represented by two voltage levels. For example, in the TTL convention (see Chapter 10), these values are (nominally) 5 V and 0 V, respectively; in CMOS circuits, these values can vary substantially. Other conventions are also used, including reversing the assignment—for example, by letting a 0-V level represent a logic 1 and a 5-V level represent a logic 0. Note that in a binary waveform, knowledge of the transition between one state and another (e.g., from f_0 to f_1 at

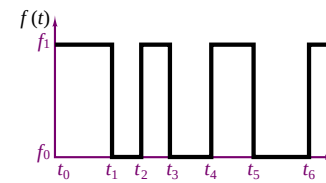


Figure 13.3 A binary signal

$t = t_2$) is equivalent to knowledge of the state. Thus, digital logic circuits can operate by detecting transitions between voltage levels. The transitions are often called **edges** and can be positive (f_0 to f_1) or negative (f_1 to f_0). Virtually all of the signals handled by a computer are binary. From here on, whenever we speak of digital signals, you may assume that the text is referring to signals of the binary type, unless otherwise indicated.

13.2 THE BINARY NUMBER SYSTEM

The binary number system is a natural choice for representing the behavior of circuits that operate in one of two states (on or off, 1 or 0, or the like). The diode and transistor gates and switches studied in Chapter 10 fall in this category. Table 13.1 shows the correspondence between decimal and binary number systems for decimal numbers up to 16.

Table 13.1 Conversion from decimal to binary

Decimal number, n_{10}	**Binary number, n_2**
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111
16	10000

Binary numbers are based on powers of 2, whereas the decimal system is based on powers of 10. For example, the number 372 in the decimal system can be expressed as

$$372 = (3 \times 10^2) + (7 \times 10^1) + (2 \times 10^0)$$

while the binary number 10110 corresponds to the following combination of powers of 2:

$$10110 = (1 \times 2^4) + (0 \times 2^3) + (1 \times 2^2) + (1 \times 2^1) + (0 \times 2^0)$$

It is relatively simple to see the correspondence between the two number systems if we add the terms on the right-hand side of the previous expression. Let n_2 represent the number n **base 2** (i.e., in the binary system) and n_{10} the same number **base 10.** Then, our notation will be as follows:

$$10110_2 = 16 + 0 + 4 + 2 + 0 = 22_{10}$$

Note that a fractional number can also be similarly represented. For example, the number 3.25 in the decimal system may be represented as

$$3.25_{10} = 3 \times 10^0 + 2 \times 10^{-1} + 5 \times 10^{-2}$$

while in the binary system the number 10.011 corresponds to

$$\begin{aligned} 10.011_2 &= 1 \times 2^1 + 0 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3} \\ &= 2 + 0 + 0 + \tfrac{1}{4} + \tfrac{1}{8} = 2.375_{10} \end{aligned}$$

Table 13.1 shows that it takes four binary digits, also called **bits,** to represent the decimal numbers up to 15. Usually, the rightmost bit is called the **least significant bit,** or **LSB,** and the leftmost bit is called the **most significant bit,** or **MSB.** Since binary numbers clearly require a larger number of digits than decimal numbers, the digits are usually grouped in sets of four, eight, or sixteen. Four bits are usually termed a **nibble,** eight bits are called a **byte,** and sixteen bits (or two bytes) form a **word.**

Table 13.2 Rules for addition

$0 + 0 = 0$
$0 + 1 = 1$
$1 + 0 = 1$
$1 + 1 = 0$ (with a carry of 1)

Addition and Subtraction

The operations of addition and subtraction are based on the simple rules shown in Table 13.2. Note that, just as is done in the decimal system, a carry is generated

whenever the sum of two digits exceeds the largest single-digit number in the given number system, which is 1 in the binary system. The carry is treated exactly as in the decimal system. A few examples of binary addition are shown in Figure 13.4, with their decimal counterparts.

Decimal	Binary	Decimal	Binary	Decimal	Binary
5	101	15	1111	3.25	11.01
+6	+110	+20	+10100	+5.75	+101.11
11	1011	35	100011	9.00	1001.00

(Note that in this example, $3.25 = 3\frac{1}{4}$ and $5.75 = 5\frac{3}{4}$.)

Figure 13.4 Examples of binary addition

The procedure for subtracting binary numbers is based on the rules of Table 13.3. A few examples of binary subtraction are given in Figure 13.5, with their decimal counterparts.

Table 13.3 Rules for subtraction

$0 - 0 = 0$
$1 - 0 = 1$
$1 - 1 = 0$
$0 - 1 = 1$ (with a borrow of 1)

Decimal	Binary	Decimal	Binary	Decimal	Binary
9	1001	16	10000	6.25	110.01
–5	–101	–3	–11	–4.50	–100.10
4	0100	13	01101	1.75	001.11

Figure 13.5 Examples of binary subtraction

Multiplication and Division

Whereas in the decimal system the multiplication table consists of $10^2 = 100$ entries, in the binary system we only have $2^2 = 4$ entries. Table 13.4 represents the complete multiplication table for the binary number system.

Table 13.4 Rules for multiplication

$0 \times 0 = 0$
$0 \times 1 = 0$
$1 \times 0 = 0$
$1 \times 1 = 1$

Division in the binary system is also based on rules analogous to those of the decimal system, with the two basic laws given in Table 13.5. Once again, we need be concerned with only two cases, and just as in the decimal system, division by zero is not contemplated.

Table 13.5 Rules for division

$0 \div 1 = 0$
$1 \div 1 = 1$

Conversion from Decimal to Binary

The conversion of a decimal number to its binary equivalent is performed by successive division of the decimal number by 2, checking for the remainder each time. Figure 13.6 illustrates this idea with an example. The result obtained in Figure 13.6 may be easily verified by performing the opposite conversion, from binary to decimal:

$$110001 = 2^5 + 2^4 + 2^0 = 32 + 16 + 1 = 49$$

The same technique can be used for converting decimal fractional numbers to their binary form, provided that the whole number is separated from the fractional part and each is converted to binary form (separately), with the results added at the

Remainder
$49 \div 2 = 24 + 1$
$24 \div 2 = 12 + 0$
$12 \div 2 = 6 + 0$
$6 \div 2 = 3 + 0$
$3 \div 2 = 1 + 1$
$1 \div 2 = 0 + 1$
$49_2 = 110001_2$

Figure 13.6 Example of conversion from decimal to binary

Remainder
$37 \div 2 = 18 + 1$
$18 \div 2 = 9 + 0$
$9 \div 2 = 4 + 1$
$4 \div 2 = 2 + 0$
$2 \div 2 = 1 + 0$
$1 \div 2 = 0 + 1$
$37_{10} = 100101_2$

$2 \times 0.53 = 1.06 \rightarrow 1$
$2 \times 0.06 = 0.12 \rightarrow 0$
$2 \times 0.12 = 0.24 \rightarrow 0$
$2 \times 0.24 = 0.48 \rightarrow 0$
$2 \times 0.48 = 0.96 \rightarrow 0$
$2 \times 0.96 = 1.92 \rightarrow 1$
$2 \times 0.92 = 1.84 \rightarrow 1$
$2 \times 0.84 = 1.68 \rightarrow 1$
$2 \times 0.68 = 1.36 \rightarrow 1$
$2 \times 0.36 = 0.72 \rightarrow 0$
$2 \times 0.72 = 1.44 \rightarrow 1$
$0.53_{10} = 0.10000111101$

Figure 13.7 Conversion from decimal to binary

end. Figure 13.7 outlines this procedure by converting the number 37.53 to binary form. The procedure is outlined in two steps. First, the integer part is converted; then, to convert the fractional part, one simple technique consists of multiplying the decimal fraction by 2 in successive stages. If the result exceeds 1, a 1 is needed to the right of the binary fraction being formed (100101 . . . , in our example). Otherwise, a 0 is added. This procedure is continued until no fractional terms are left. In this case, the decimal part is 0.53_{10}, and Figure 13.7 illustrates the succession of calculations. Stopping the procedure outlined in Figure 13.7 after 11 digits results in the following approximation:

$$37.53_{10} = 100101.10000111101$$

Greater precision could be attained by continuing to add binary digits, at the expense of added complexity. Note that an infinite number of binary digits may be required to represent a decimal number *exactly.*

Complements and Negative Numbers

To simplify the operation of subtraction in digital computers, **complements** are used almost exclusively. In practice, this corresponds to replacing the operation $X - Y$ with the operation $X + (-Y)$. This procedure results in considerable simplification, since the computer hardware need include only adding circuitry. Two types of complements are used with binary numbers: the **one's complement** and the **two's complement.**

The one's complement of an n-bit binary number is obtained by subtracting the number itself from $(2^n - 1)$. Two examples are as follows:

$$\begin{aligned} a &= 0101 \\ \text{One's complement of } a &= (2^4 - 1) - a \\ &= (1111) - (0101) \\ &= 1010 \\ b &= 101101 \\ \text{One's complement of } b &= (2^6 - 1) - b \\ &= (111111) - (101101) \\ &= 010010 \end{aligned}$$

The two's complement of an n-bit binary number is obtained by subtracting the number itself from 2^n. Two's complements of the same numbers a and b used in the preceding illustration are computed as follows:

$$\begin{aligned} a &= 0101 \\ \text{Two's complement of } a &= 2^4 - a \\ &= (10000) - (0101) \\ &= 1011 \\ b &= 101101 \\ \text{Two's complement of } b &= 2^6 - b \\ &= (1000000) - (101101) \\ &= 010011 \end{aligned}$$

A simple rule that may be used to obtain the two's complement directly from a binary number is the following: Starting at the least significant (rightmost) bit,

copy each bit *until the first 1 has been copied,* and then replace each successive 1 by a 0 and each 0 by a 1. You may wish to try this rule on the two previous examples to verify that it is much easier to use than the subtraction from 2^n.

Different conventions exist in the binary system to represent whether a number is negative or positive. One convention, called the **sign-magnitude convention,** makes use of a *sign bit,* usually positioned at the beginning of the number, for which a value of 1 represents a minus sign and a value of 0, a plus sign. Thus, an eight-bit binary number would consist of a sign bit followed by seven *magnitude bits,* as shown in Figure 13.8(a). In a digital system that uses eight-bit signed integer words, we could represent integer numbers (decimal) in the range

$$-(2^7 - 1) \leq N \leq +(2^7 - 1)$$

or

$$-127 \leq N \leq +127$$

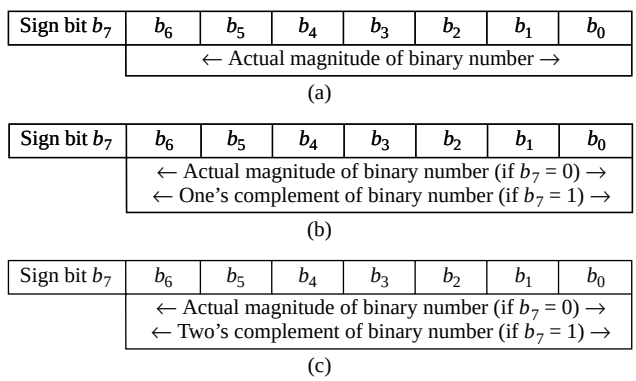


Figure 13.8 (a) Eight-bit sign-magnitude binary number; (b) Eight-bit one's complement binary number; (c) Eight-bit two's complement binary number

A second convention uses the one's complement notation. In this convention, a sign bit is also used to indicate whether the number is positive (sign bit = 0) or negative (sign bit = 1). However, the magnitude of the binary number is represented by the true magnitude if the number is positive, and by its *one's complement* if the number is negative. Figure 13.8(b) illustrates the convention. For example, the number $(91)_{10}$ would be represented by the seven-bit binary number $(1011011)_2$ with a leading 0 (the sign bit): $(\mathbf{0}1011011)_2$. On the other hand, the number $(-91)_{10}$ would be represented by the seven-bit one's complement binary number $(0100100)_2$ with a leading 1 (the sign bit): $(\mathbf{1}0100100)_2$.

Most digital computers use the two's complement convention in performing integer arithmetic operations. The two's complement convention represents positive numbers by a sign bit of 0, followed by the true binary magnitude; negative numbers are represented by a sign bit of 1, *followed by the two's complement of the binary number,* as shown in Figure 13.8(c). The advantage of the two's complement convention is that the algebraic sum of two's complement binary numbers is carried out very simply by adding the two numbers *including the sign bit.* Example 13.1 illustrates two's complement addition.

EXAMPLE 13.1 Two's Complement Operations

Problem

Perform the following subtractions using two's complement arithmetic:

1. $X - Y = 1011100 - 1110010$
2. $X - Y = 10101111 - 01110011$

Solution

Analysis: The two's complement subtractions are performed by replacing the operation $X - Y$ with the operation $X + (-Y)$. Thus, we first find the two's complement of Y and add the result to X in each of the two cases:

$$X - Y = 1011100 - 1110010 = 1011100 + (2^7 - 1110010)$$
$$= 1011100 + 0001110 = 1101010$$

Next, we add the *sign bit* (in boldface type) in front of each number (1 in first case since the difference $X - Y$ is a negative number):

$$X - Y = \mathbf{1}1101010$$

Repeating for the second subtraction gives:

$$X - Y = 10101111 - 01110011 = 10101111 + (2^8 - 01110011) = 10101111$$
$$+10001101 = 00111100$$
$$= \mathbf{0}00111100$$

where the first digit is a 0 because $X - Y$ is a positive number.

The Hexadecimal System

Table 13.6 Hexadecimal code

0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

It should be apparent by now that representing numbers in base 2 and base 10 systems is purely a matter of convenience, given a specific application. Another base frequently used is the **hexadecimal system,** a direct derivation of the binary number system. In the hexadecimal (or hex) code, the bits in a binary number are subdivided into groups of four. Since there are 16 possible combinations for a four-bit number, the natural digits in the decimal system (0 through 9) are insufficient to represent a hex digit. To solve this problem, the first six letters of the alphabet are used, as shown in Table 13.6. Thus, in hex code, an eight-bit word corresponds to just two digits; for example:

$$1010\ 0111_2 = A7_{16}$$
$$0010\ 1001_2 = 29_{16}$$

Binary Codes

In this subsection, we describe two common binary codes that are often used for practical reasons. The first is a method of representing decimal numbers in digital logic circuits that is referred to as **binary-coded decimal,** or **BCD, representation.** In effect, the simplest BCD representation is just a sequence of four-bit binary numbers that stops after the first 10 entries, as shown in Table 13.7. There are

Table 13.7 BCD code

0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Table 13.8 Three-bit Gray code

Binary	Gray
000	000
001	001
010	011
011	010
100	110
101	111
110	101
111	100

also other BCD codes, all reflecting the same principle: that each decimal digit is represented by a fixed-length binary word. One should realize that although this method is attractive because of its direct correspondence with the decimal system, it is not efficient. Consider, for example, the decimal number 68. Its binary representation by direct conversion is the seven-bit number 1000100. On the other hand, the corresponding BCD representation would require eight bits:

$$68_{10} = 01101000_{\text{BCD}}$$

Another code that finds many applications is the **Gray code.** This is simply a reshuffling of the binary code with the property that any two consecutive numbers differ only by one bit. Table 13.8 illustrates the three-bit Gray code. The Gray code can be very useful in practical applications, because in counting up or down according to this code, the binary representation of a number changes only one bit at a time. The next example illustrates an application of the Gray code to a practical engineering problem.

Digital Position Encoders

Position encoders are devices that output a digital signal proportional to their (linear or angular) position. These devices are very useful in measuring instantaneous position in *motion control* applications. Motion control is a technique that is used when it is necessary to accurately control the motion of a moving object; examples are found in robotics, machine tools, and servomechanisms. For example, in positioning the arm of a robot to pick up an object, it is very important to know its exact position at all times. Since one is usually interested in both rotational and translational motion, two types of encoders are discussed in this example: *linear* and *angular* position encoders.

An optical position encoder consists of an *encoder pad*, which is either a strip (for translational motion) or a disk (for rotational motion) with alternating black and white areas. These areas are arranged to reproduce some binary code, as shown in Figure 13.9, where both the conventional binary and Gray codes are depicted for a four-bit linear encoder pad. A fixed array of photodiodes (see Chapter 8) senses the reflected light from each of the cells across a row of the encoder path; depending on the amount of light

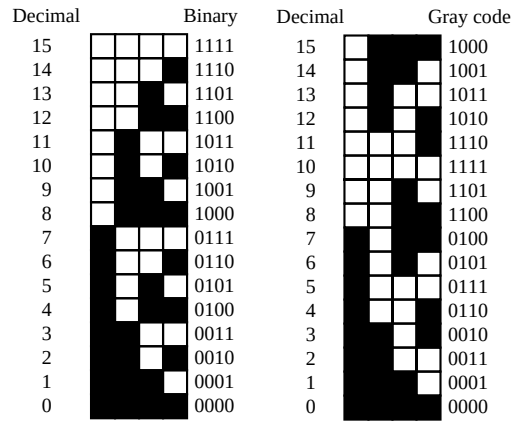


Figure 13.9 Binary and Gray code patterns for linear position encoders

reflected, each photodiode circuit will output a voltage corresponding to a binary 1 or 0. Thus, a different four-bit word is generated for each row of the encoder.

Suppose the encoder pad is 100 mm in length. Then its resolution can be computed as follows. The pad will be divided into $2^4 = 16$ segments, and each segment corresponds to an increment of $100/16$ mm $= 6.25$ mm. If greater resolution were necessary, more bits could be employed: an eight-bit pad of the same length would attain a resolution of $100/256$ mm $= 0.39$ mm.

A similar construction can be employed for the five-bit angular encoder of Figure 13.10. In this case, the angular resolution can be expressed in degrees of rotation, where $2^5 = 32$ sections correspond to 360°. Thus, the resolution would be $360^\circ/32 = 11.25^\circ$. Once again, greater angular resolution could be obtained by employing a larger number of bits.

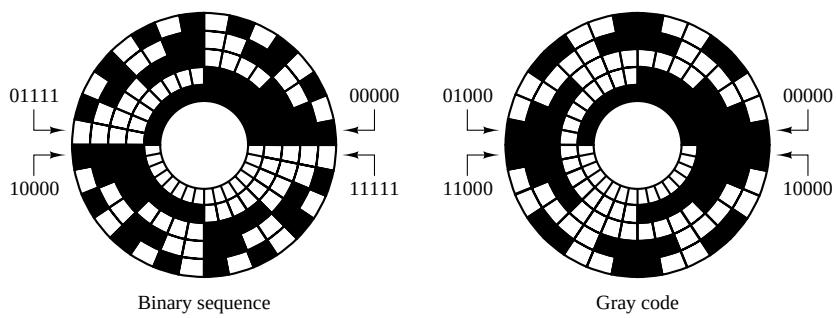


Figure 13.10 Binary and Gray code patterns for angular position encoders

EXAMPLE 13.2 Conversion from Binary to Hexadecimal

Problem

Convert the following binary numbers to hexadecimal form.

1. 100111
2. 1011101
3. 11001101
4. 101101111001
5. 100110110
6. 1101011011

Solution

Analysis: A simple method for binary to hexadecimal conversion consists of grouping each binary number into four-bit groups, and then performing the conversion for each four-bit word following Table 13.6:

1. $100111_2 = 0010_2 0111_2 = 27_{16}$
2. $1011101_2 = 0101_2 1101_2 = 5D_{16}$
3. $11001101_2 = 1100_2 1101_2 = CD_{16}$
4. $101101111001_2 = 1011_2 0111_2 1001_2 = B79_{16}$
5. $100110110_2 = 0001_2 0011_2 0100_2 = 136_{16}$
6. $1101011011_2 = 0011_2 0101_2 1011_2 = 35B_{16}$

Comments: Note that we start grouping always from the right-hand side. The reverse process is equally easy: To convert from hexadecimal to binary, replace each hexadecimal number with the equivalent four-bit binary word.

Check Your Understanding

13.1 Convert the following decimal numbers to binary form:

a. 39 b. 59
c. 512 d. 0.4475
e. $\frac{25}{32}$ f. 0.796875
g. 256.75 h. 129.5625
i. 4,096.90625

13.2 Convert the following binary numbers to decimal:

a. 1101 b. 11011
c. 10111 d. 0.1011
e. 0.001101 f. 0.001101101
g. 111011.1011 h. 1011011.001101
i. 10110.0101011101

13.3 Perform the following additions and subtractions. Express the answer in decimal form for problems (a)–(d) and in binary form for problems (e)–(h).

a. $1001.1_2 + 1011.01_2$ b. $100101_2 + 100101_2$
c. $0.1011_2 + 0.1101_2$ d. $1011.01_2 + 1001.11_2$
e. $64_{10} - 32_{10}$ f. $127_{10} - 63_{10}$
g. $93.5_{10} - 42.75_{10}$ h. $(84\frac{9}{32})_{10} - (48\frac{5}{16})_{10}$

13.4 How many possible numbers can be represented in a 12-bit word?

13.5 If we use an eight-bit word with a sign bit (seven magnitude bits plus one sign bit) to represent voltages -5 V and $+5$ V, what is the smallest increment of voltage that can be represented?

13.6 Convert the following numbers from hex to binary or from binary to hex:

a. F83 b. 3C9
c. A6 d. 110101110_2
e. 10111001_2 f. 11011101101_2

13.7 Find the two's complement of the following binary numbers:

a. 11101001 b. 10010111 c. 1011110

13.8 Convert the following numbers from hex to binary, and find their two's complements:

a. F43 b. 2B9 c. A6

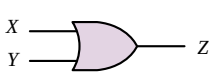
13.3 BOOLEAN ALGEBRA

The mathematics associated with the binary number system (and with the more general field of logic) is called *Boolean*, in honor of the English mathematician George Boole, who published a treatise in 1854 entitled *An Investigation of the Laws of Thought, on Which Are Founded the Mathematical Theories of Logic and Probabilities.* The development of a *logical algebra*, as Boole called it, is one of the results of his investigations. The variables in a Boolean, or logic, expression can take only one of two values, usually represented by the numbers 0 and 1. These variables are sometimes referred to as true (1) and false (0). This convention is normally referred to as **positive logic.** There is also a **negative logic** convention in which the roles of logic 1 and logic 0 are reversed. In this book we shall employ only positive logic.

Analysis of **logic functions,** that is, functions of logical (Boolean) variables, can be carried out in terms of truth tables. A truth table is a listing of all the possible values each of the Boolean variables can take, and of the corresponding value of the desired function. In the following paragraphs we shall define the basic logic functions upon which Boolean algebra is founded, and we shall describe each in terms of a set of rules and a truth table; in addition, we shall also introduce **logic gates**. Logic gates are physical devices (see Chapter 10) that can be used to implement logic functions.

Table 13.9 Rules for logical addition (OR)

$0 + 0 = 0$
$0 + 1 = 1$
$1 + 0 = 1$
$1 + 1 = 1$



OR gate

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

Truth table

Figure 13.11 Logical addition and the OR gate

AND and OR Gates

The basis of **Boolean algebra** lies in the operations of **logical addition,** or the **OR** operation; and **logical multiplication,** or the **AND** operation. Both of these find a correspondence in simple logic gates, as we shall presently illustrate. Logical addition, although represented by the symbol $+$, differs from conventional algebraic addition, as shown in the last rule listed in Table 13.9. Note that this rule also differs from the last rule of binary addition studied in the previous section. Logical addition can be represented by the logic gate called an **OR gate,** whose symbol and whose inputs and outputs are shown in Figure 13.11. The OR gate represents the following logical statement:

If either X or Y is true (1), then Z is true(1). **(13.1)**

This rule is embodied in the electronic gates discussed in Chapter 9, in which a logic 1 corresponds, say, to a 5-V signal and a logic 0 to a 0-V signal.

Logical multiplication is denoted by the center dot ($\cdot$) and is defined by the rules of Table 13.10. Figure 13.12 depicts the **AND gate**, which corresponds to this operation. The AND gate corresponds to the following logical statement:

If both X and Y are true (1), then Z is true (1). **(13.2)**

One can easily envision logic gates (AND and OR) with an arbitrary number of inputs; three- and four-input gates are not uncommon.

The rules that define a logic function are often represented in tabular form by means of a **truth table.** Truth tables for the AND and OR gates are shown in Figures 13.11 and 13.12. A truth table is nothing more than a tabular summary of all of the possible outputs of a logic gate, given all the possible input values. If the number of inputs is 3, the number of possible combinations grows from 4 to 8, but the basic idea is unchanged. Truth tables are very useful in defining logic functions. A typical logic design problem might specify requirements such as "the output Z shall be logic 1 only when the condition ($X = 1$ AND $Y = 1$) OR ($W = 1$) occurs, and shall be logic 0 otherwise." The truth table for this particular logic function is shown in Figure 13.13 as an illustration. The design consists, then, of determining the combination of logic gates that exactly implements the required logic function. Truth tables can greatly simplify this procedure.

The AND and OR gates form the basis of all logic design in conjunction with the **NOT gate.** The NOT gate is essentially an inverter (which can be constructed using bipolar or field-effect transistors, as discussed in Chapter 10), and it provides the complement of the logic variable connected to its input. The complement of a logic variable X is denoted by $\overline{X}$. The NOT gate has only one input, as shown in Figure 13.14.

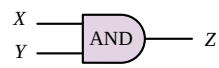
To illustrate the use of the NOT gate, or inverter, we return to the design example of Figure 13.13, where we required that the output of a logic circuit be $Z = 1$ only if $X = 0$ AND $Y = 1$ OR if $W = 1$. We recognize that except for the requirement $X = 0$, this problem would be identical if we stated it as follows: "The output Z shall be logic 1 only when the condition ($\overline{X} = 1$ AND $Y = 1$) OR ($W = 1$) occurs, and shall be logic 0 otherwise." If we use an inverter to convert X to $\overline{X}$, we see that the required condition becomes ($\overline{X} = 1$ AND $Y = 1$) OR ($W = 1$). The formal solution to this elementary design exercise is illustrated in Figure 13.15.

In the course of the discussion of logic gates, extensive use will be made of truth tables to evaluate logic expressions. A set of basic rules will facilitate this task. Table 13.11 lists some of the rules of Boolean algebra; each of these can be proven by using a truth table, as will be shown in examples and exercises. An example proof for rule 16 is given in Figure 13.16 in the form of a truth table. This technique can be employed to prove any of the laws of Table 13.11. From the simple truth table in Figure 13.16, which was obtained step by step, we can clearly see that indeed $X \cdot (X + Y) = X$. This methodology for proving the validity of logical equations is called **proof by perfect induction.** The 19 rules of Table 13.11 can be used to simplify logic expressions.

To complete the introductory material on Boolean algebra, a few paragraphs need to be devoted to two very important theorems, called **De Morgan's theorems.**

Table 13.10 Rules for logical multiplication (AND)

$0 \cdot 0 = 0$
$0 \cdot 1 = 0$
$1 \cdot 0 = 0$
$1 \cdot 1 = 1$



AND gate

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

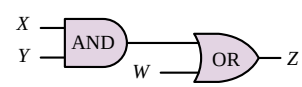
Truth table

Figure 13.12 Logical multiplication and the AND gate

Logic gate realization of the statement "the output Z shall be logic 1 only when the condition ($X = 1$ AND $Y = 1$) OR ($W = 1$) occurs, and shall be logic 0 otherwise."

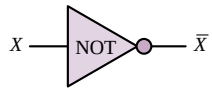
X	Y	W	Z
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Truth table



Solution using logic gates

Figure 13.13 Example of logic function implementation with logic gates



NOT gate

X	$\overline{X}$
1	0
0	1

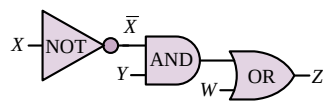
Truth table for NOT gate

Figure 13.14 Complements and the NOT gate

X	$\overline{X}$	Y	W	Z
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1

(Required logic function)

Truth table



Solution using logic gates

Figure 13.15 Solution of a logic problem using logic gates

X	Y	$(X+Y)$	$X\cdot(X+Y)$
0	0	0	0
0	1	1	0
1	0	1	1
1	1	1	1

Figure 13.16 Proof of rule 16 by perfect induction

Table 13.11 Rules of Boolean algebra

1. $0 + X = X$	
2. $1 + X = 1$	
3. $X + X = X$	
4. $X + \overline{X} = 1$	
5. $0 \cdot X = 0$	
6. $1 \cdot X = X$	
7. $X \cdot X = X$	
8. $X \cdot \overline{X} = 0$	
9. $\overline{\overline{X}} = X$	
10. $X + Y = Y + X$	Commutative law
11. $X \cdot Y = Y \cdot X$	Commutative law
12. $X + (Y + Z) = (X + Y) + Z$	Associative law
13. $X \cdot (Y \cdot Z) = (X \cdot Y) \cdot Z$	Associative law
14. $X \cdot (Y + Z) = X \cdot Y + X \cdot Z$	Distributive law
15. $X + X \cdot Z = X$	Absorption law
16. $X \cdot (X + Y) = X$	
17. $(X + Y) \cdot (X + Z) = X + Y \cdot Z$	
18. $X + \overline{X} \cdot Y = X + Y$	
19. $X \cdot Y + Y \cdot Z + \overline{X} \cdot Z = X \cdot Y + \overline{X} \cdot Z$	

These are stated here in the form of logic functions:

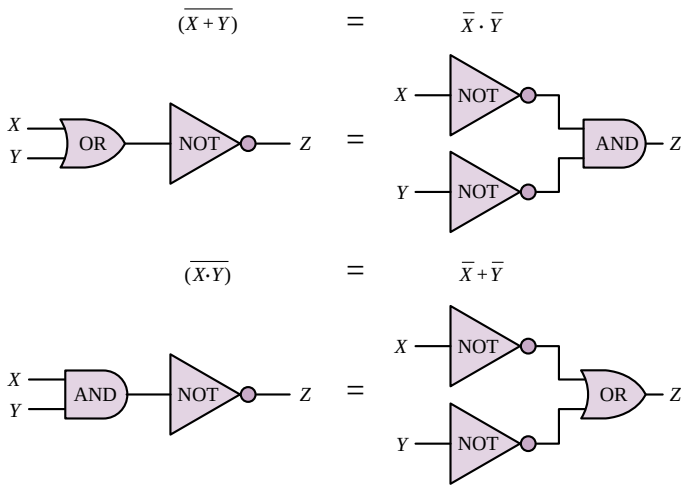
$$\overline{(X + Y)} = \overline{X} \cdot \overline{Y} \tag{13.3}$$

$$\overline{(X \cdot Y)} = \overline{X} + \overline{Y} \tag{13.4}$$

These two laws state a very important property of logic functions:

Any logic function can be implemented using only OR and NOT gates, or using only AND and NOT gates.

De Morgan's laws can easily be visualized in terms of logic gates, as shown in Figure 13.17. The associated truth tables are proof of these theorems.



Truth table

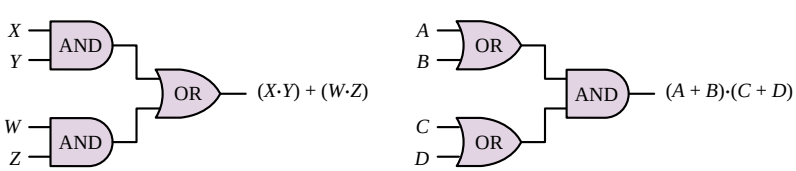
X	Y	$Z = \overline{(X+Y)} = \overline{X} \cdot \overline{Y}$
0	0	1
0	1	0
1	0	0
1	1	0

Truth table

X	Y	$Z = \overline{(X+Y)} = \overline{X} + \overline{Y}$
0	0	1
0	1	1
1	0	1
1	1	0

Figure 13.17 De Morgan's laws

The importance of De Morgan's laws is in the statement of the **duality** that exists between AND and OR operations: any function can be realized by just one of the two basic operations, plus the complement operation. This gives rise to two families of logic functions: **sums of products** and **product of sums,** as shown in Figure 13.18. Any logical expression can be reduced to either one of these two forms. Although the two forms are equivalent, it may well be true that one of the two has a simpler implementation (fewer gates). Example 13.3 illustrates this point.



Sum of products expression $(X \cdot Y) + (W \cdot Z)$

Product of sums expression $(A + B) \cdot (C + D)$

Figure 13.18 Sum-of-products and product-of-sums logic functions

EXAMPLE 13.3 Simplification of Logical Expression

Problem

Using the rules of Table 13.11, simplify the following function using the rules of Boolean algebra.

$$f(A, B, C, D) = \overline{A} \cdot \overline{B} \cdot D + \overline{A} \cdot B \cdot D + B \cdot C \cdot D + A \cdot C \cdot D$$

Solution

Find: Simplified expression for logical function of four variables.

Analysis:

$$\begin{aligned} f &= \overline{A} \cdot \overline{B} \cdot D + \overline{A} \cdot B \cdot D + B \cdot C \cdot D + A \cdot C \cdot D \\ &= \overline{A} \cdot D \cdot \left(\overline{B} + B\right) + B \cdot C \cdot D + A \cdot C \cdot D && \text{Rule 14} \\ &= \overline{A} \cdot D + B \cdot C \cdot D + A \cdot C \cdot D && \text{Rule 4} \\ &= \left(\overline{A} + A \cdot C\right) \cdot D + B \cdot C \cdot D && \text{Rule 14} \\ &= \left(\overline{A} + C\right) \cdot D + B \cdot C \cdot D && \text{Rule 18} \\ &= \overline{A} \cdot D + C \cdot D + B \cdot C \cdot D && \text{Rule 14} \\ &= \overline{A} \cdot D + C \cdot D \cdot (1 + B) && \text{Rule 14} \\ &= \overline{A} \cdot D + C \cdot D = \left(\overline{A} + C\right) \cdot D && \text{Rules 2 and 6} \end{aligned}$$

Fail-Safe Autopilot Logic

This example aims to illustrate the significance of De Morgan's laws and of the duality of the sum-of-products and product-of-sums forms. Suppose that a fail-safe autopilot system in a commercial aircraft requires that, prior to initiating a takeoff or landing maneuver, the following check must be passed: two of three possible pilots must be available. The three possibilities are the pilot, the co-pilot, and the autopilot. Imagine further that there exist switches in the pilot and co-pilot seats that are turned on by the weight of the crew, and that a self-check circuit exists to verify the proper operation of the autopilot system. Let the variable X denote the pilot state (1 if the pilot is sitting at the controls), Y denote the same condition for the co-pilot, and Z denote the state of the autopilot, where $Z = 1$ indicates that the autopilot is functioning. Then, since we wish two of these conditions to be active before the maneuver can be initiated, the logic function corresponding to "system ready" is:

$$f = X \cdot Y + X \cdot Z + Y \cdot Z$$

This can also be verified by the truth table shown below.

Pilot	Co-pilot	Autopilot	System ready
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

The function f defined above is based on the notion of a *positive check;* that is, it indicates when the system is ready. Let us now apply De Morgan's laws to the function f, which is in sum-of-products form:

$$\overline{f} = g = \overline{X \cdot Y + X \cdot Z + Y \cdot Z} = (\overline{X} + \overline{Y}) \cdot (\overline{X} + \overline{Z}) \cdot (\overline{Y} + \overline{Z})$$

The function g, in product-of-sums form, conveys exactly the same information as the function f, but it performs a negative check; in other words, g verifies the *system not ready condition.* You see then that whether one chooses to implement the function in one form or another is simply a matter of choice; the two forms give exactly the same information.

EXAMPLE 13.4 Realizing Logic Functions from Truth Tables

Problem

Realize the logic function described by the truth table below.

A	*B*	*C*	*y*
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Solution

Known Quantities: Value of function $y(A, B, C)$ for each possible combination of logical variables A, B, C.

Find: Logical expression realizing the function y.

Analysis: To determine a logical expression for the function y, we first need to convert the truth table into a logical expression. We do so by expressing y as the sum of the products of the three variables for each combination that yields $y = 1$. If the value of a variable is 1, we use the uncomplemented variable. If it's 0, we use the complemented variable. For example, the second row (first instance of $y = 1$) would yield the term $\overline{A} \cdot \overline{B} \cdot C$. Thus,

$$\begin{aligned} y &= \overline{A} \cdot \overline{B} \cdot C + \overline{A} \cdot B \cdot C + A \cdot \overline{B} \cdot \overline{C} + A \cdot \overline{B} \cdot C + A \cdot B \cdot \overline{C} + A \cdot B \cdot C \\ &= \overline{A} \cdot C(\overline{B} + B) + A \cdot \overline{B} \cdot (\overline{C} + C) + A \cdot B \cdot (\overline{C} + C) \\ &= \overline{A} \cdot C + A \cdot \overline{B} + A \cdot B = \overline{A} \cdot C + A \cdot (\overline{B} + B) = \overline{A} \cdot C + A = A + C. \end{aligned}$$

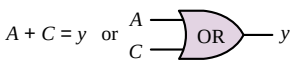


Figure 13.19

Thus, the function is a two-input OR gate, as shown in Figure 13.19.

Comments: The derivation above has made use of two rules from Table 13.11: rules 4 and 18. Could you have predicted that the variable B would not be used in the final realization? Why?

EXAMPLE 13.5 DeMorgan's Theorem and Product-of-Sums Expressions

Problem

Realize the logic function $y = A + B \cdot C$ in product-of-sums form. Implement the solution using AND, OR, and NOT gates.

Solution

Known Quantities: Logical expression for the function $y(A, B, C)$.

Find: Physical realization using AND, OR, and NOT gates.

Analysis: We use the fact that $\overline{\overline{y}} = y$ and apply DeMorgan's theorem as follows:

$$\overline{y} = \overline{A + (B \cdot C)} = \overline{A} \cdot \overline{(B \cdot C)} = \overline{A} \cdot \left(\overline{B} + \overline{C}\right)$$

$$\overline{\overline{y}} = y = \overline{\overline{A} \cdot \left(\overline{B} + \overline{C}\right)}.$$

The above sum-of-products function is realized using complements of each variable (obtained using NOT gates) and is finally complemented as shown in Figure 13.20.

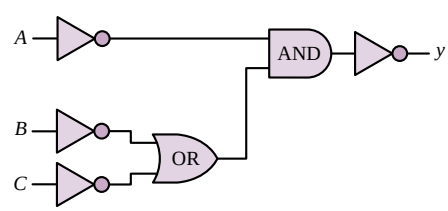


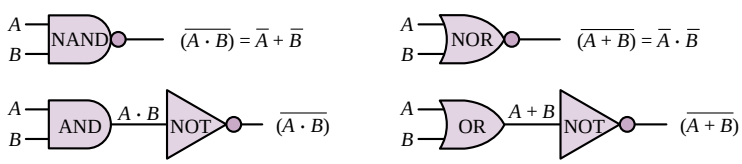
Figure 13.20

Comments: It should be evident that the original sum-of-products expression, which could be implemented with just one AND and one OR gate has a much more efficient realization. In the next section we show a systematic approach to function minimization.

Focus on Computer-Aided Solutions: An *Electronics Workbench*™ simulation of the logic circuit of Figure 13.20 may be found in the accompanying CD-ROM.

NAND and NOR Gates

In addition to the AND and OR gates we have just analyzed, the complementary forms of these gates, called NAND and NOR, are very commonly used in practice. In fact, NAND and NOR gates form the basis of most practical logic circuits. Figure 13.21 depicts these two gates, and illustrates how they can be easily interpreted in terms of AND, OR, and NOT gates by virtue of De Morgan's laws. You can readily verify that the logic function implemented by the NAND and NOR gates corresponds, respectively, to AND and OR gates followed by an inverter. It is very important to note that, by De Morgan's laws, the NAND gate performs a *logical addition* on the *complements* of the inputs, while the NOR gate performs a *logical multiplication* on the *complements* of the inputs. Functionally, then, any logic function could be implemented with either NOR or NAND gates only.



A	B	$\overline{A}$	$\overline{B}$	$\overline{(A \cdot B)}$
0	0	1	1	1
0	1	1	0	1
1	0	0	1	1
1	1	0	0	0

NAND gate

A	B	$\overline{A}$	$\overline{B}$	$\overline{(A + B)}$
0	0	1	1	1
0	1	1	0	0
1	0	0	1	0
1	1	0	0	0

NOR gate

Figure 13.21 Equivalence of NAND and NOR gates with AND and OR gates

In the next section we shall learn how to systematically approach the design of logic functions. First, we provide a few examples to illustrate logic design with NAND and NOR gates.

EXAMPLE 13.6 Realizing the AND Function with NAND Gates

Problem

Use a truth table to show that the AND function can be realized using only NAND gates, and show the physical realization.

Solution

Known Quantities: AND and NAND truth tables.

A	$B(=A)$	$A{\cdot}B$	$\overline{(A{\cdot}B)}$
0	0	0	1
1	1	1	0

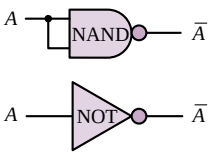


Figure 13.22 NAND gate as an inverters

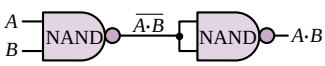


Figure 13.23

Find: AND realization using NAND gates.

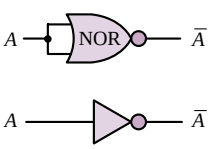
Assumptions: Consider two-input functions and gates.

Analysis: The truth table below summarizes the two functions:

A	B	NAND $\overline{A \cdot B}$	AND $A \cdot B$
0	0	1	0
0	1	1	0
1	0	1	0
1	1	0	1

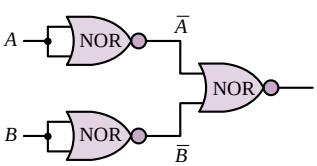
Clearly, to realize the AND function we need to simply invert the output of a NAND gate. This is easily accomplished if we observe that a NAND gate with its inputs tied together acts as an inverter; you can verify this in the above truth table by looking at the NAND output for the input combinations 0-0 and 1-1, or by referring to Figure 13.22. The final realization is shown in Figure 13.23.

Comments: NAND gates naturally implement functions that contain complemented products. Gates that employ negative logic are a natural consequence of the inverting characteristics of transistor switches (refer to Section 10.5). Thus, one should expect that NAND (and NOR) gates are very commonly employed in practice.



A	$B(=A)$	$(A+B)$	$\overline{(A+B)}$
0	0	0	1
1	1	1	0

Figure 13.24 NOR gate as an inverter



$(\overline{\overline{A} + \overline{B}}) = A{\cdot}B$

Figure 13.25

EXAMPLE 13.7 Realizing the AND Function with NOR Gates

Problem

Show analytically that the AND function can be realized using only NOR gates, and determine the physical realization.

Solution

Known Quantities: AND and NOR functions.

Find: AND realization using NOR gates.

Assumptions: Consider two-input functions and gates.

Analysis: We can solve this problem using De Morgan's theorem. The output of an AND gate can be expressed as $f = A \cdot B$. Using De Morgan's theorem we write:

$$f = \overline{\overline{f}} = \overline{\overline{A \cdot B}} = \overline{\overline{A} + \overline{B}}$$

The above function is implemented very easily if we see that a NOR gate with its input tied together acts as a NOT gate (see Figure 13.24). Thus, the logic circuit of Figure 13.25 provides the desired answer.

Comments: NOR gates naturally implement functions that contain complemented sums. Gates that employ negative logic are a natural consequence of the inverting characteristics of transistor switches (refer to Section 10.5). Thus, one should expect that NOR (and NAND) gates are very commonly employed in practice.

EXAMPLE 13.8 Realizing a Function with NAND and NOR Gates

Problem

Realize the following function using only NAND and NOR gates:

$$y = \overline{\overline{(A \cdot B)} + C}$$

Solution

Known Quantities: Logical expression for y.

Find: Realization of y using only NAND and NOR gates.

Assumptions: Consider two-input functions and gates.

Analysis: On the basis of the two preceding examples, we see that we can realize the term $Z = \overline{(A \cdot B)}$ using a two-input NAND gate, and the term $\overline{Z + C}$ using a two-input NOR gate. The solution is shown in Figure 13.26.

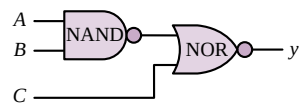


Figure 13.26

The XOR (Exclusive OR) Gate

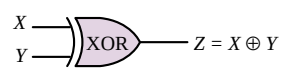
It is rather common practice for a manufacturer of integrated circuits to provide common combinations of logic circuits in a single integrated circuit package. We review many of these common **logic modules** in Section 13.5. An example of this idea is provided by the **exclusive OR (XOR) gate,** which provides a logic function similar, but not identical, to the OR gate we have already studied. The XOR gate acts as an OR gate, except when its inputs are all logic 1s; in this case, the output is a logic 0 (thus the term *exclusive*). Figure 13.27 shows the logic circuit symbol adopted for this gate, and the corresponding truth table. The logic function implemented by the XOR gate is the following: "either X or Y, but not both." This description can be extended to an arbitrary number of inputs.

The symbol adopted for the exclusive OR operation is $\oplus$, and so we shall write

$$Z = X \oplus Y$$

to denote this logic operation. The XOR gate can be obtained by a combination of the basic gates we are already familiar with. For example, if we observe that the XOR function corresponds to $Z = X \oplus Y = (X + Y) \cdot (\overline{X \cdot Y})$, we can realize the XOR gate by means of the circuit shown in Figure 13.28.

Common IC logic gate configurations, device numbers, and data sheets are included in the CD-ROM that accompanies this book. These devices are typically available in both of the two more common device families, TTL and CMOS. The devices listed in the CD-ROM are available in CMOS technology under the numbers SN74AHXX. The same logic gate ICs are also available as TTL devices.



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

Truth table

Figure 13.27 XOR gate

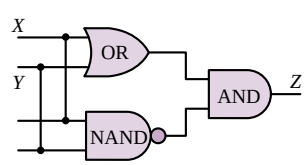


Figure 13.28 Realization of an XOR gate

Check Your Understanding

13.9 Show that one can obtain an OR gate using NAND gates only. [*Hint:* Use three NAND gates.]

13.10 Show that one can obtain an AND gate using NOR gates only. [*Hint:* Use three NOR gates.]

13.11 Prepare a step-by-step truth table for the following logic expressions:

a. $\overline{(X + Y + Z)} + (X \cdot Y \cdot Z) \cdot \overline{X}$

b. $\overline{X} \cdot Y \cdot Z + Y \cdot (Z + W)$

c. $(X \cdot \overline{Y} + Z \cdot \overline{W}) \cdot (W \cdot X + \overline{Z} \cdot Y)$

[*Hint:* Your truth table must have 2^n entries, where n is the number of logic variables.]

13.12 Implement the logic functions of Check Your Understanding Exercise 13.11 using NAND and NOR gates only. [*Hint:* Use De Morgan's theorems and the fact that $\overline{\overline{f}} = f$.]

13.13 Implement the logic functions of Check Your Understanding Exercise 13.11 using AND, OR, and NOT gates only.

13.14 Show that the XOR function can also be expressed as $Z = X \cdot \overline{Y} + Y \cdot \overline{X}$. Realize the corresponding function using NOT, AND, and OR gates. [*Hint:* Use truth tables for the logic function Z (as defined in the exercise) and for the XOR function.]

13.4 KARNAUGH MAPS AND LOGIC DESIGN

In examining the design of logic functions by means of logic gates, we have discovered that more than one solution is usually available for the implementation of a given logic expression. It should also be clear by now that some combinations of gates can implement a given function more efficiently than others. How can we be assured of having chosen the most efficient realization? Fortunately, there is a procedure that utilizes a map describing all possible combinations of the variables present in the logic function of interest. This map is called a **Karnaugh map,** after its inventor. Figure 13.29 depicts the appearance of Karnaugh maps for two-, three-, and four-variable expressions in two different forms. As can be seen, the row and column assignments for two or more variables are arranged so that all adjacent terms change by only one bit. For example, in the two-variable map, the columns next to column 01 are columns 00 and 11. Also note that each map consists of 2^N **cells,** where N is the number of logic variables.

Each cell in a Karnaugh map contains a **minterm,** that is, a product of the N variables that appear in our logic expression (in either uncomplemented or complemented form). For example, for the case of three variables ($N = 3$), there are $2^3 = 8$ such combinations, or minterms: $\overline{X} \cdot \overline{Y} \cdot \overline{Z}$, $\overline{X} \cdot \overline{Y} \cdot Z$, $\overline{X} \cdot Y \cdot \overline{Z}$, $\overline{X} \cdot Y \cdot Z$, $X \cdot \overline{Y} \cdot \overline{Z}$, $X \cdot \overline{Y} \cdot Z$, $X \cdot Y \cdot \overline{Z}$, and $X \cdot Y \cdot Z$. The content of each cell—that is, the minterm—is the product of the variables appearing at the corresponding vertical and horizontal coordinates. For example, in the three-variable map, $X \cdot Y \cdot \overline{Z}$ appears at the intersection of $X \cdot Y$ and $\overline{Z}$. The map is filled by placing a value of 1 for any combination of variables for which the desired output is a 1. For example, consider the function of three variables for which we desire to have an output of 1 whenever the variables X, Y, and Z have the following values:

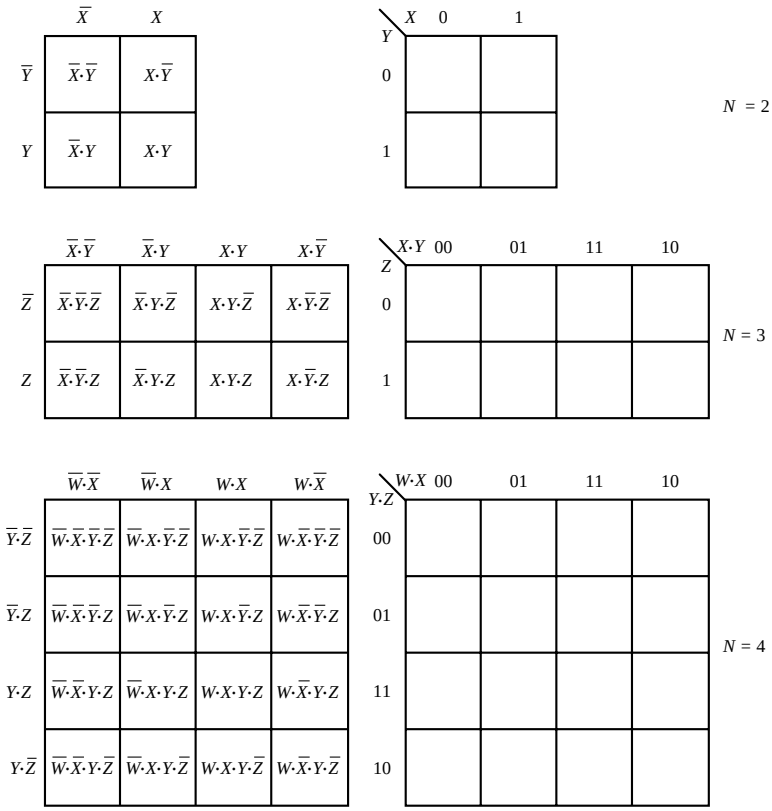


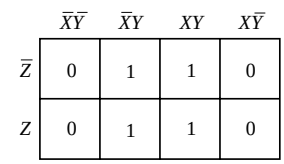
Figure 13.29 Two-, three-, and four-variable Karnaugh maps

$X = 0$	$Y = 1$	$Z = 0$
$X = 0$	$Y = 1$	$Z = 1$
$X = 1$	$Y = 1$	$Z = 0$
$X = 1$	$Y = 1$	$Z = 1$

The same truth table is shown in Figure 13.30 together with the corresponding Karnaugh map.

The Karnaugh map provides an immediate view of the values of the function in graphical form. Further, the arrangement of the cells in the Karnaugh map is such that any two adjacent cells contain minterms that vary in only one variable. This property, as will be verified shortly, is quite useful in the design of logic functions by means of logic gates, especially if we consider the map to be continuously wrapping around itself, as if the top and bottom, and right and left, edges were touching each other. For the three-variable map given in Figure 13.29, for example, the cell $X \cdot \overline{Y} \cdot \overline{Z}$ is adjacent to $\overline{X} \cdot \overline{Y} \cdot \overline{Z}$ if we "roll" the map so that the right edge touches the left. Note that these two cells differ only in the variable X, a property we earlier claimed adjacent cells have.[1]

[1] A useful rule to remember is that in a two-variable map there are two minterms adjacent to any given minterm; in a three-variable map, three minterms are adjacent to any given minterm; in a four-variable map, the number is four, and so on.

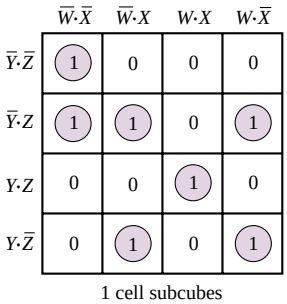


Karnaugh map

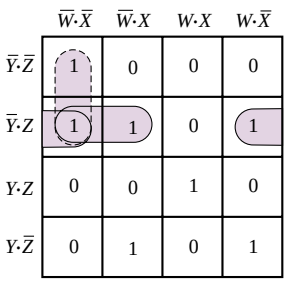
X	Y	Z	Desired Function
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Truth table

Figure 13.30 Truth table and Karnaugh map representations of a logic function

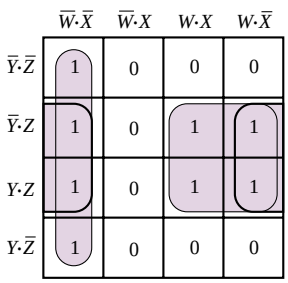


1 cell subcubes

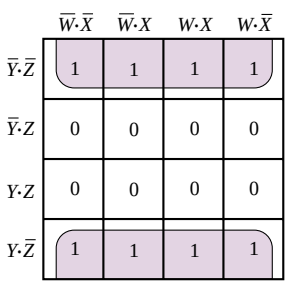


2 cell subcubes

Figure 13.32 One- and two-cell subcubes for the Karnaugh map of Figure 13.31



(a)



(b)

Figure 13.33 Four- and eight-cell subcubes for an arbitrary logic function

Shown in Figure 13.31 is a more complex, four-variable logic function, which will serve as an example in explaining how Karnaugh maps can be used directly to implement a logic function. First, we define a subcube as a set of 2^m adjacent cells with logical value 1, for $m = 1, 2, 3, \ldots, N$. Thus, a subcube can consist of 1, 2, 4, 8, 16, 32,... cells. All possible subcubes for the four-variable map of Figure 13.31 are shown in Figure 13.32. Note that there are no four-cell subcubes in this particular case. Note also that there is some overlap between subcubes. Examples of four-cell and eight-cell subcubes are shown in Figure 13.33 for an arbitrary expression.

X	Y	Y	Z	Desired Function
0	0	0	0	1
0	0	0	1	1
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

Truth table for four-variable expression

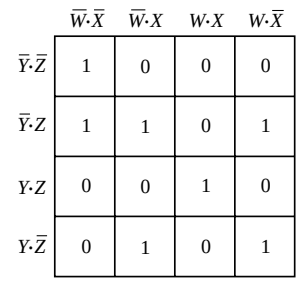


Figure 13.31 Karnaugh map for a four-variable expression

In general, one tries to find the largest possible subcubes to cover all of the "1" entries in the map. How do maps and subcubes help in the realization of logic functions, then? The use of maps and subcubes in minimizing logic expressions is best explained by considering the following rule of Boolean algebra:

$$Y \cdot X + Y \cdot \overline{X} = Y$$

where the variable Y could represent a product of logic variables (for example, we could similarly write $(Z \cdot W) \cdot X + (Z \cdot W) \cdot \overline{X} = Z \cdot W$ with $Y = Z \cdot W$). This rule is easily proven by factoring Y:

$$Y \cdot (X + \overline{X})$$

and observing that $X + \overline{X} = 1$, always. Then it should be clear that the variable X need not appear in the expression at all. Let us apply this rule to a more complex logic expression, to verify that it can also apply to this case. Consider the logic expression

$$\overline{W} \cdot X \cdot \overline{Y} \cdot Z + \overline{W} \cdot \overline{X} \cdot \overline{Y} \cdot Z + W \cdot \overline{X} \cdot \overline{Y} \cdot Z + W \cdot X \cdot \overline{Y} \cdot Z$$

and factor it as follows:

$$\begin{aligned}\overline{W} \cdot Z \cdot \overline{Y} \cdot (X + \overline{X}) + W \cdot \overline{Y} \cdot Z \cdot (\overline{X} + X) &= \overline{W} \cdot Z \cdot \overline{Y} + W \cdot \overline{Y} \cdot Z \\ &= \overline{Y} \cdot Z \cdot (\overline{W} + W) = \overline{Y} \cdot Z\end{aligned}$$

Quite a simplification! If we consider, now, a map in which we place a 1 in the cells corresponding to the minterms $\overline{W} \cdot X \cdot \overline{Y} \cdot Z$, $\overline{W} \cdot \overline{X} \cdot \overline{Y} \cdot Z$, $W \cdot \overline{X} \cdot \overline{Y} \cdot Z$, and $W \cdot X \cdot \overline{Y} \cdot Z$, forming the previous expression, we obtain the Karnaugh map of Figure 13.34. It can easily be verified that the map of Figure 13.34 shows a single four-cell subcube corresponding to the term $\overline{Y} \cdot Z$.

We have not established formal rules yet, but it definitely appears that the map method for simplifying Boolean expressions is a convenient tool. In effect, the map has performed the algebraic simplification automatically! We can see that in any subcube, one or more of the variables present will appear in both complemented *and* uncomplemented form in all their combinations with the other variables. These variables can be eliminated. As an illustration, in the *eight-cell* subcube case of Figure 13.35, the full-blown expression would be:

$$\overline{W} \cdot \overline{X} \cdot \overline{Y} \cdot \overline{Z} + \overline{W} \cdot X \cdot \overline{Y} \cdot \overline{Z} + W \cdot X \cdot \overline{Y} \cdot \overline{Z} + W \cdot \overline{X} \cdot \overline{Y} \cdot \overline{Z}$$

$$+\overline{W} \cdot \overline{X} \cdot Y \cdot \overline{Z} + \overline{W} \cdot X \cdot Y \cdot \overline{Z} + W \cdot X \cdot Y \cdot \overline{Z} + W \cdot \overline{X} \cdot Y \cdot \overline{Z}$$

However, if we consider the eight-cell subcube, we note that the three variables X, W, and Z appear both in complemented and uncomplemented form in all their combinations with the other variables and thus can be removed from the expression. This reduces the seemingly unwieldy expression simply to $\overline{Y}$! In logic design terms, a simple inverter is sufficient to implement the expression.

The example just shown is a particularly simple one, but it illustrates how simple it can be to determine the minimal expression for a logic function. It should be apparent that the larger a subcube, the greater the simplification that will result. For subcubes that do not intersect, as in the previous example, the solution can be found easily, and is unique.

	$\overline{W}\cdot\overline{X}$	$\overline{W}\cdot X$	$W\cdot X$	$W\cdot\overline{X}$
$\overline{Y}\cdot\overline{Z}$	0	0	0	0
$\overline{Y}\cdot Z$	1	1	1	1
$Y\cdot Z$	0	0	0	0
$Y\cdot\overline{Z}$	0	0	0	0

Figure 13.34 Karnaugh map for the function $\overline{W} \cdot X \cdot \overline{Y} \cdot Z + \overline{W} \cdot \overline{X} \cdot \overline{Y} \cdot Z + W \cdot \overline{X} \cdot \overline{Y} \cdot Z + W \cdot X \cdot \overline{Y} \cdot Z$

	$\overline{W}\cdot\overline{X}$	$\overline{W}\cdot X$	$W\cdot X$	$W\cdot\overline{X}$
$\overline{Y}\cdot\overline{Z}$	1	1	1	1
$\overline{Y}\cdot Z$	1	1	1	1
$Y\cdot Z$	0	0	0	0
$Y\cdot\overline{Z}$	0	0	0	0

Figure 13.35

Sum-of-Products Realizations

Although not explicitly stated, the logic functions of the preceding section were all in sum-of-products form. As you know, it is also possible to realize logic functions in product-of-sums form. This section discusses the implementation of logic functions in sum-of-products form and gives a set of design rules. The next section will do the same for product-of-sums form logical expressions. The following rules are a useful aid in determining the minimal sum-of-products expression:

FOCUS ON METHODOLOGY

Sum-of-Products Realizations

1. Begin with isolated cells. These must be used as they are, since no simplification is possible.
2. Find all cells that are adjacent to only one other cell, forming two-cell subcubes.
3. Find cells that form four-cell subcubes, eight-cell subcubes, and so forth.
4. The minimal expression is formed by the collection of the *smallest number of maximal subcubes.*

The following examples illustrate the application of these principles to a variety of problems.

EXAMPLE 13.9 Logic Circuit Design Using Karnaugh Maps

Problem

Design a logic circuit that implements the truth table of Figure 13.36.

A	B	C	D	y
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	1

Figure 13.36

Solution

Known Quantities: Truth table for $y(A, B, C, D)$.

Find: Realization of y.

Assumptions: Two-, three-, and four-input gates are available.

Analysis: We use the Karnaugh map of Figure 13.37, which is shown with values of 1 and 0 already in place. We recognize four subcubes in the map; three are four-cell subcubes, and one is a two-cell subcube. The expressions for the subcubes are: $\overline{A} \cdot \overline{B} \cdot \overline{D}$ for the two-cell subcube; $\overline{B} \cdot \overline{C}$ for the subcube that wraps around the map; $\overline{C} \cdot D$ for the four-by-one subcube; and $A \cdot D$ for the square subcube at the bottom of the map. Thus, the expression for y is:

$$y = \overline{A} \cdot \overline{B} \cdot \overline{D} + \overline{B} \cdot \overline{C} + \overline{C}D + AD.$$

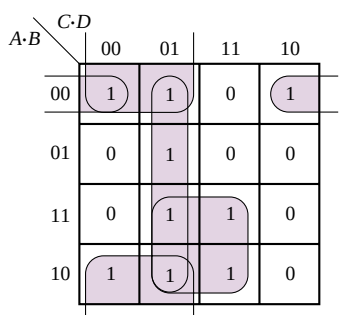


Figure 13.37 Karnaugh map for Example 13.9

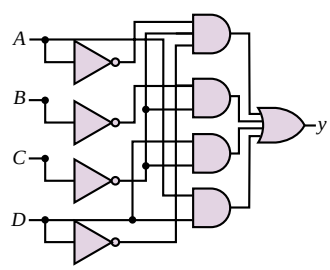


Figure 13.38 Logic circuit realization of Karnaugh map of Figure 13.37

Ewb

The implementation of the above function with logic gates is shown in Figure 13.38.

Comments: The Karnaugh map covering of Figure 13.37 is a sum-of-products expression because we covered the map using the ones.

EXAMPLE 13.10 Deriving a Sum-of-Products Expression from a Logic Circuit

Problem

Derive the truth table and minimum sum-of-products expression for the circuit of Figure 13.39.

Solution

Known Quantities: Logic circuit representing $f(x, y, z)$.

Find: Expression for f and corresponding truth table.

Analysis: To determine the truth table, we write the expression corresponding to the logic circuit of Figure 13.39:

$$f = \overline{x} \cdot \overline{y} + y \cdot z$$

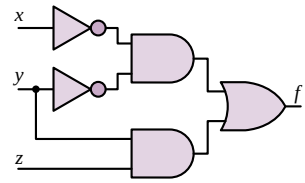
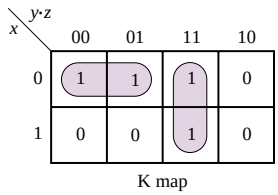


Figure 13.39

The truth table corresponding to this expression and the corresponding Karnaugh map with sum-of-products covering are shown in Figure 13.40.

x	y	z	f
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1



K map

Figure 13.40

Comments: If we used zeros in covering the Karnaugh map for this example, the resulting expression would be a product-of-sums. You may verify that, in the case of this example, the complexity of the circuit would be unchanged. Note also that there exists a third subcube ($x = 0$, $yz = 01, 11$) that is not used because it does not help minimize the solution.

EXAMPLE 13.11 Realizing a Product-of-Sums Using Only NAND Gates

Problem

Realize the following function in sum-of-products form, using only two-input NAND gates.

$$f = (\overline{x} + \overline{y}) \cdot (y + \overline{z})$$

Solution

Known quantities: $f(x, y, z)$.

Find: Logic circuit for f using only NAND gates.

Analysis: The first step is to convert the expression for f into an expression that can be easily implemented with NAND gates. We observe that direct application of De Morgan's theorem yields:

$$\overline{x} + \overline{y} = \overline{x \cdot y}$$

$$y + \overline{z} = \overline{\overline{z} \cdot \overline{y}}$$

Thus, we can write the function as follows:

$$f = (\overline{x \cdot y}) \cdot \left(\overline{\overline{z} \cdot \overline{y}}\right)$$

and implement it with five NAND gates, as shown in Figure 13.41

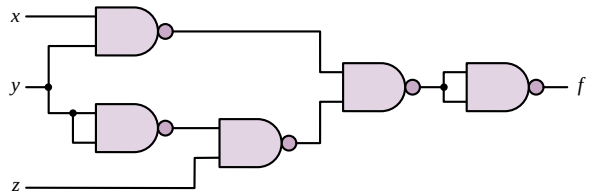


Figure 13.41

Comments: Note that we used two NAND gates as inverters—one to obtain $\overline{y}$, the other to invert the output of the fourth NAND gate, equal to $\overline{(\overline{x \cdot y})} \cdot \overline{(\overline{z \cdot \overline{y}})}$.

EXAMPLE 13.12 Simplifying Expressions by Using Karnaugh Maps

Problem

Simplify the following expression by using a Karnaugh map.

$$f = x \cdot y + \overline{x} \cdot z + y \cdot z$$

Solution

Known Quantities: $f(x, y, z)$.

Find: Minimal expression for f.

Analysis: We cover a three-term Karnaugh map to reflect the expression give above. The result is shown in Figure 13.42. It is clear that the Karnaugh map can be covered by using just two terms (subcubes): $f = x \cdot y + \overline{x} \cdot z$. Thus, the term $y \cdot z$ is redundant.

Comments: The Karnaugh map covering clearly shows that the term $y \cdot z$ corresponds to covering a third two-cell subcube vertically intersecting the two horizontal two-cell subcubes already shown. Clearly, the third subcube is redundant.

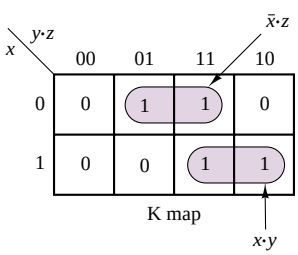


Figure 13.42

EXAMPLE 13.13 Simplifying a Logic Circuit by Using the Karnaugh Map

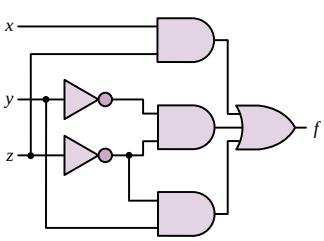


Figure 13.43

Problem

Derive the Karnaugh map corresponding to the circuit of Figure 13.43 and use the resulting map to simplify the expression.

Solution

Known Quantities: Logic circuit.

Find: Simplified logic circuit.

Analysis: We first determine the expression $f(x, y, z)$ from the logic circuit:

$$f = (x \cdot z) + (\overline{x} \cdot \overline{z}) + (y \cdot \overline{z})$$

This expression leads to the Karnaugh map shown in Figure 13.44. Inspection of the Karnaugh map reveals that the map could have been covered more efficiently by using four-cell subcubes. The improved map covering, corresponding to the simpler function $f = x + \overline{z}$, and the resulting logic circuit are shown in Figure 13.45.

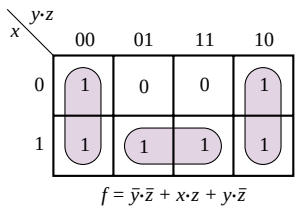


Figure 13.44

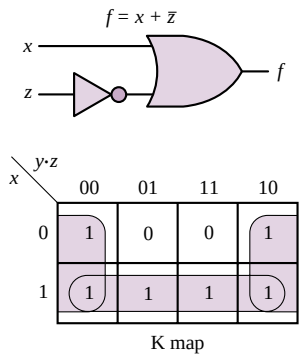


Figure 13.45

Comments: In general, one wishes to cover the largest possible subcubes in a Karnaugh map.

Product-of-Sums Realizations

Thus far, we have exclusively worked with sum-of-products expressions, that is, logic functions of the form $A \cdot B + C \cdot D$. We know, however, that De Morgan's laws state that there is an equivalent form that appears as a product of sums, for example, $(W + Y) \cdot (Y + Z)$. The two forms are completely equivalent, logically, but one of the two forms may lead to a realization involving a smaller number of gates. When using Karnaugh maps, we may obtain the product-of-sums form very simply by following these rules:

FOCUS ON METHODOLOGY

Product-of-Sums Realizations

1. Solve for the 0s exactly as for the 1s in sum-of-products expressions.
2. Complement the resulting expression.

The same principles stated earlier apply in covering the map with subcubes and determining the minimal expression. The following examples illustrate how one form may result in a more efficient solution than the other.

EXAMPLE 13.14 Comparison of Sum-of-Products and Product-of-Sums Designs

Problem

x	y	z	f
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

Realize the function f described by the accompanying truth table using both 0 and 1 coverings in the Karnaugh map.

Solution

Known Quantities: Truth table for logic function.

Find: Realization in both sum-of-products and product-of-sums forms.

Analysis:

1. *Product-of-sums expression.* Product-of-sums expressions use zeros to determine the logical expression from a Karnaugh map. Figure 13.46 depicts the Karnaugh map covering with zeros, leading to the expression

$$f = (x + y + z) \cdot (\overline{x} + \overline{y})$$

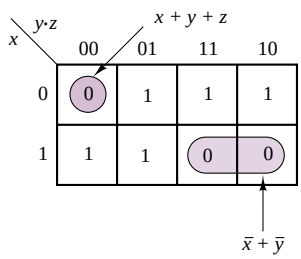


Figure 13.46

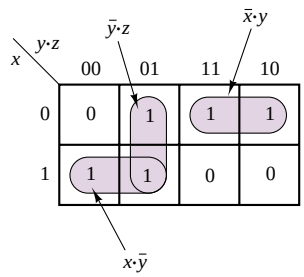


Figure 13.47

2. *Sum-of-products expression.* Sum-of-products expressions use ones to determine the logical expression from a Karnaugh map. Figure 13.47 depicts the Karnaugh map covering with ones, leading to the expression

$$f = (\overline{x} \cdot y) + (\overline{x} \cdot \overline{y}) + (\overline{y} \cdot z)$$

Comments: The product-of-sums solution requires the use of five gates (two OR, two NOT, and one AND), while the sum-of-products solution will use six gates (one OR, two NOT, and three AND). Thus, solution 1 leads to the simpler design.

EXAMPLE 13.15 Product-of-Sums Design

Problem

Realize the function f described by the accompanying truth table in minimal product of sums form.

Solution

Known Quantities: Truth table for logic function.

Find: Realization in minimal product-of-sums forms.

Analysis: We cover the Karnaugh map of Figure 13.48 using zeros, and obtain the following function:

$$f = \overline{z} \cdot (\overline{x} + \overline{y})$$

x	y	z	f
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

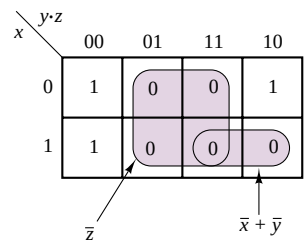


Figure 13.48

Comments: Is the sum-of-products solution simpler? Try it for yourself.

Safety Circuit for Operation of a Stamping Press

In this example, the techniques illustrated in the preceding examples will be applied to a practical situation. To operate a stamping press, an operator must press two buttons (b_1 and b_2) one meter apart from each other and away from the press (this ensures that the operator's hands cannot be caught in the press). When the buttons are pressed, the logical variables b_1 and b_2 are equal to 1. Thus, we can define a new variable $A = b_1 \cdot b_2$; when $A = 1$, the operator's hands are safely away from the press. In addition to the safety requirement, however, other conditions must be satisfied before the operator can activate the press. The press is designed to operate on one of two workpieces, part I and part II, but not both. Thus, acceptable logic states for the press to be operated are "part I is in the press, but not part II" and "part II is in the press, but not part I." If we denote the presence of part I in the press by the logical variable $B = 1$ and the presence of part II by the logical variable $C = 1$, we can then impose additional requirements on the operation of the press. For example, a robot used to place either part in the press could activate a pair of switches (corresponding to logical variables B and C) indicating which part, if any, is in the press. Finally, in order for the press to be operable, it must be "ready," meaning that it has to have completed any previous stamping operation. Let the logical variable $D = 1$

represent the ready condition. We have now represented the operation of the press in terms of four logical variables, summarized in the truth table of Table 13.12. Note that only two combinations of the logical variables will result in operation of the press: $ABCD = 1011$ and $ABCD = 1101$. You should verify that these two conditions correspond to the desired operation of the press. Using a Karnaugh map, realize the logic circuitry required to implement the truth table shown.

Table 13.12 Conditions for operation of stamping press

(*A*) $b_1 \cdot b_2$	**(*B*) Part I is in press**	**(*C*) Part II is in press**	**(*D*) Press is operable**	**Press operation 1 = pressing; 0 = not pressing**
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

↑ Both buttons (b_1, b_2) must be pressed for this to be a 1.

Solution:

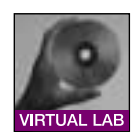
Table 13.12 can be converted to a Karnaugh map, as shown in Figure 13.49. Since there are many more 0s than 1s in the table, the use of 0s in covering the map will lead to greater simplification. This will result in a product-of-sums expression. The four subcubes shown in Figure 13.49 yield the equation

$$A \cdot D \cdot (C + B) \cdot (\overline{C} + \overline{B})$$

By De Morgan's law, this equation is equivalent to

$$A \cdot D \cdot (C + B) \cdot \overline{(C \cdot B)}$$

which can be realized by the circuit of Figure 13.50.



Multisim

For the purpose of comparison, the corresponding sum-of-products circuit is shown in Figure 13.51. Note that this circuit employs a greater number of gates and will therefore lead to a more expensive design.

Focus on Computer-Aided Solutions— An *Electronics Workbench*™ simulation of the logic circuit of Figure 13.50 may be found in the accompanying CD-ROM.

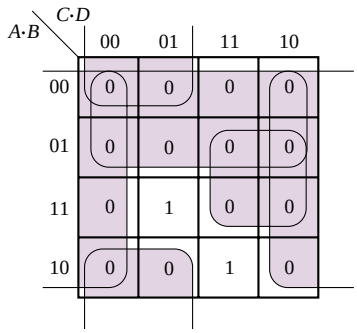


Figure 13.49

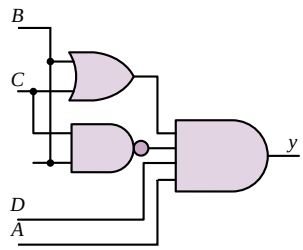


Figure 13.50

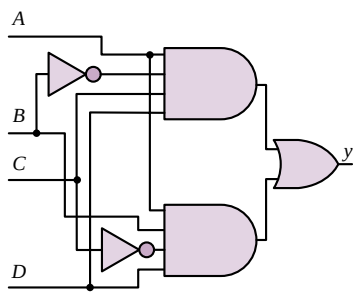


Figure 13.51

Don't Care Conditions

Another simplification technique may be employed whenever the value of the logic function to be implemented can be either a 1 or a 0. This condition may result from the specification of the problem and is not uncommon. Whenever it does not matter whether a position in the map is filled by a 1 or a 0, we use a so-called **don't care** entry, denoted by an **x**. Then the don't care can be used as either a 1 or a 0, depending on which results in a greater simplification (i.e., helps in forming the smallest number of maximal subcubes). The following examples illustrate the use of don't cares.

EXAMPLE 13.16 Using Don't Cares to Simplify Expressions—1

Problem

Use don't care entries to simplify the expression:

$$f(a,b,c,d) = \overline{a}\cdot\overline{b}\cdot\overline{c}\cdot d + \overline{a}\cdot\overline{b}\cdot c\cdot\overline{d} + \overline{a}\cdot\overline{b}\cdot c\cdot d + \overline{a}\cdot b\cdot\overline{c}\cdot d + a\cdot\overline{b}\cdot c\cdot d + a\cdot b\cdot\overline{c}\cdot\overline{d}$$

Note that the **x**'s never occur, and so they may be assigned a 1 or a 0, whichever will best simplify the expression.

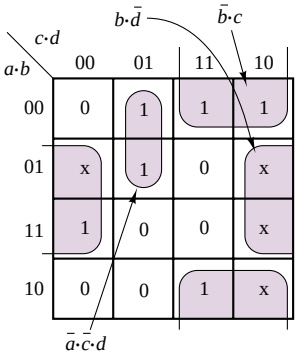


Figure 13.52

Solution

Known Quantities: Logical expression; don't care conditions.

Find: Minimal realization.

Schematics, Diagrams, Circuits, and Given Data: Don't care conditions: $f(a, b, c, d) = \{0100, 0110, 1010, 1110\}$.

Analysis: We cover the Karnaugh map of Figure 13.52 using ones, and also using x entries for each don't care condition. Treating all of the x entries as ones, we complete the covering with two four-cell subcubes and one two-cell subcube, to obtain the following simplified expression:

$$f(a, b, c, d) = b \cdot \overline{d} + \overline{b} \cdot c + \overline{a} \cdot \overline{c} \cdot d$$

Comments: Note that we could have also interpreted the don't care entries as zeros and tried to solve in product-of-sums form. Verify that the expression obtained above is indeed the minimal one.

EXAMPLE 13.17 Using Don't Cares to Simplify Expressions—2

Problem

Find a minimum product-of-sums realization for the expression $f(a, b, c)$.

Solution

Known Quantities: Logical expression, don't care conditions.

Find: Minimal realization.

Schematics, Diagrams, Circuits, and Given Data:

$f(a, b, c) = 1$ for $\{a, b, c\} = \{000, 010, 011\}$

$f(a, b, c) =$ don't care for $\{a, b, c\} = \{100, 101, 110\}$

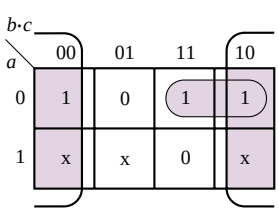


Figure 13.53

Analysis: We cover the Karnaugh map of Figure 13.53 using ones, and also using x entries for each don't care condition. By appropriately selecting two of the three don't-care entries to be equal to 1, we complete the covering with one four-cell subcube and one two-cell subcube, to obtain the following minimal expression:

$$f(a, b, c) = \overline{a} \cdot b + \overline{c}$$

Comments: Note that we have chosen to set one of the don't care entries equal to zero, since it would not lead to any further simplification.

EXAMPLE 13.18 Using Don't Cares to Simplify Expressions—3

Problem

Find a minimum sum-of-products realization for the expression $f(a, b, c, d)$.

Solution

Known Quantities: Logical expression; don't care conditions.

Find: Minimal realization.

Schematics, Diagrams, Circuits, and Given Data

$f(a, b, c, d) = 1$ for $\{a, b, c, d\} = \{0000, 0011, 0110, 0101\}$

$f(a, b, c, d) =$ don't care for $\{a, b, c, d\} = \{1010, 1011, 1101, 1110, 1111\}$

Analysis: We cover the Karnaugh map of Figure 13.54 using ones, and also using x entries for each don't care condition. By appropriately selecting three of the four don't care entries to be equal to 1, we complete the covering with one four-cell subcube, two two-cell subcubes, and one one-cell subcube, to obtain the following expression:

$$f(a, b, c) = \overline{a} \cdot \overline{b} \cdot \overline{c} \cdot \overline{d} + b \cdot c \cdot \overline{d} + a \cdot d + \overline{b} \cdot c \cdot d$$

Comments: Would the product-of-sums realization be simpler? Verify.

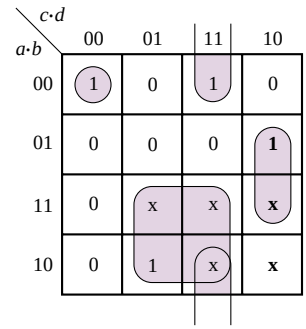


Figure 13.54

Check Your Understanding

13.15 Simplify the following expression to show that it corresponds to the function $\overline{Z}$:

$$\overline{W} \cdot \overline{X} \cdot \overline{Y} \cdot \overline{Z} + \overline{W} \cdot X \cdot \overline{Y} \cdot \overline{Z} + W \cdot X \cdot \overline{Y} \cdot \overline{Z} + W \cdot \overline{X} \cdot \overline{Y} \cdot \overline{Z} + \overline{W} \cdot \overline{X} \cdot Y \cdot \overline{Z} + \overline{W} \cdot X \cdot Y \cdot \overline{Z} + W \cdot X \cdot Y \cdot \overline{Z} + W \cdot \overline{X} \cdot Y \cdot \overline{Z}$$

13.16 Simplify the following expression, using a Karnaugh map:

$$\overline{W} \cdot \overline{X} \cdot \overline{Y} \cdot \overline{Z} + \overline{W} \cdot \overline{X} \cdot Y \cdot \overline{Z} + W \cdot X \cdot \overline{Y} \cdot \overline{Z} + W \cdot \overline{X} \cdot \overline{Y} \cdot \overline{Z} + W \cdot \overline{X} \cdot Y \cdot \overline{Z} + W \cdot X \cdot Y \cdot \overline{Z}$$

13.17 Simplify the following expression, using a Karnaugh map:

$$\overline{W} \cdot \overline{X} \cdot \overline{Y} \cdot \overline{Z} + \overline{W} \cdot \overline{X} \cdot Y \cdot \overline{Z} + W \cdot X \cdot \overline{Y} \cdot \overline{Z} + W \cdot \overline{X} \cdot \overline{Y} \cdot \overline{Z} + W \cdot \overline{X} \cdot Y \cdot \overline{Z} + \overline{W} \cdot X \cdot \overline{Y} \cdot \overline{Z}$$

13.18 The function y of Example 13.9 can be obtained with fewer gates if we use gates with three or four inputs. Find the minimum number of gates needed to obtain this function.

13.19 Verify that the product-of-sums expression for Example 13.14 can be realized with fewer gates.

13.20 Would a sum-of-products realization for Example 13.15 require fewer gates?

13.21 Prove that the circuit of Figure 13.51 can also be obtained from the sum of products.

13.22 In Example 13.16, assign a value of 0 to the don't care terms and derive the corresponding minimal expression. Is the new function simpler than the one obtained in Example 13.16?

13.23 In Example 13.17, assign a value of 0 to the don't care terms and derive the corresponding minimal expression. Is the new function simpler than the one obtained in Example 13.17?

13.24 In Example 13.17, assign a value of 1 to all don't care terms and derive the corresponding minimal expression. Is the new function simpler than the one obtained in Example 13.17?

13.25 In Example 13.18, assign a value of 0 to all don't care terms and derive the corresponding minimal expression. Is the new function simpler than the one obtained in Example 13.18?

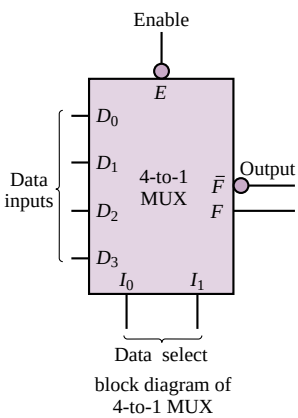
13.26 In Example 13.18, assign a value of 1 to all don't care terms and derive the corresponding minimal expression. Is the new function simpler than the one obtained in Example 13.18?

13.5 COMBINATIONAL LOGIC MODULES

The basic logic gates described in the previous section are used to implement more advanced functions and are often combined to form logic modules, which, thanks to modern technology, are available in compact integrated circuit (IC) packages. In this section and the next, we discuss a few of the more common **combinational logic modules**, illustrating how these can be used to implement advanced logic functions.

Multiplexers

Multiplexers, or **data selectors,** are combinational logic circuits that permit the selection of one of many inputs. A typical multiplexer (MUX) has 2^n **data lines,** n **address** (or **data select**) **lines,** and one output. In addition, other control inputs (e.g., enables) may exist. Standard, commercially available MUXs allow for n up to 4; however, two or more MUXs can be combined if a greater range is needed. The MUX allows for one of 2^n inputs to be selected as the data output; the selection of which input is to appear at the output is made by way of the address lines. Figure 13.55 depicts the block diagram of a four-input MUX. The input data lines are labeled D_0, D_1, D_2, and D_3; the **data select**, or address, **lines** are labeled I_0 and I_1; and the output is available in both complemented and uncomplemented form, and is thus labeled F, or $\overline{F}$. Finally, an **enable** input, labeled E, is also provided, as a means of enabling or disabling the MUX: if $E = 1$, the MUX is disabled; if $E = 0$, it is enabled. The negative logic (MUX off when $E = 1$ and on when $E = 0$) is represented by the small "bubble" at the enable input, which represents a complement operation (just as at the output of NAND and NOR gates). The enable input is useful whenever one is interested in a cascade of MUXs; this would be of interest if we needed to select a line from a large number, say $2^8 = 256$. Then two 4-input MUXs could be used to provide the data selection of 1 of 8.



block diagram of 4-to-1 MUX

I_1	I_0	F
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

Truth table of 4-to-1 MUX

Figure 13.55 4-to-1 MUX

The material described in the previous sections is quite adequate to describe the internal workings of a multiplexer. Figure 13.56 shows the internal construction of a 4-to-1 MUX using exclusively NAND gates (inverters are also used, but the reader will recall that a NAND gate can act as an inverter if properly connected).

In the design of digital systems (for example, microcomputers), a single line is often required to carry two or more different digital signals. However, only one signal at a time can be placed on the line. A MUX will allow us to select, at different instants, the signal we wish to place on this single line. This property is shown here for a 4-to-1 MUX. Figure 13.57 depicts the functional diagram of a 4-to-1 MUX, showing four data lines, D_0 through D_3, and two select lines, I_0 and I_1.

The data selector function of a MUX is best understood in terms of Table 13.13. In this truth table, the **x**'s represent don't care entries. As can be seen from the truth table, the output selects one of the data lines depending on the values of

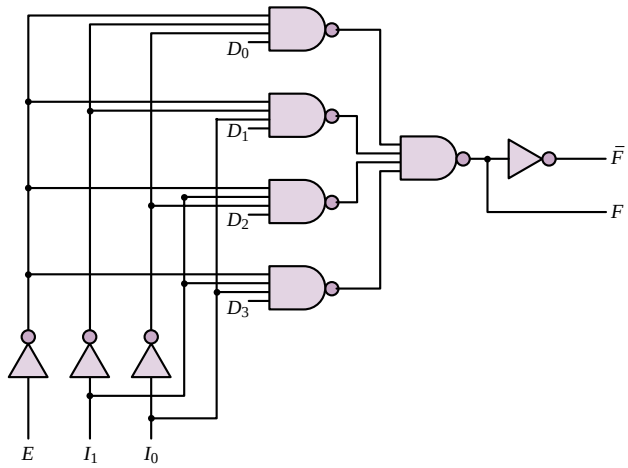


Figure 13.56 Internal structure of the 4-to-1 MUX

Table 13.13

I_1	I_0	D_3	D_2	D_1	D_0	F
0	0	x	x	x	0	0
0	0	x	x	x	1	1
0	1	x	x	0	x	0
0	1	x	x	1	x	1
1	0	x	0	x	x	0
1	0	x	1	x	x	1
1	1	0	x	x	x	0
1	1	1	x	x	x	1

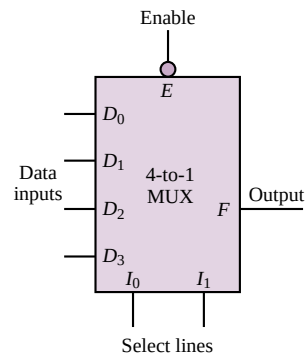


Figure 13.57 Functional diagram of four-input MUX

I_1 and I_0, assuming that I_0 is the least significant bit. As an example, $I_1 I_0 = 10$ selects D_2, which means that the output, F, will select the value of the data line D_2. Therefore $F = 1$ if $D_2 = 1$ and $F = 0$ if $D_2 = 0$.

Read-Only Memory (ROM)

Another common technique for implementing logic functions uses a **read-only memory,** or **ROM.** As the name implies, a ROM is a logic circuit that holds in storage ("memory") information—in the form of binary numbers—that cannot be altered but can be "read" by a logic circuit. A ROM is an array of memory cells, each of which can store either a 1 or a 0. The array consists of $2^m \times n$ cells, where n is the number of bits in each word stored in ROM. To access the information stored in ROM, m address lines are required. When an address is selected, in a fashion similar to the operation of the MUX, the binary word corresponding to the address selected appears at the output, which consists of n bits, that is, the same number of bits as the stored words. In some sense, a ROM can be thought of as a MUX that has an output consisting of a word instead of a single bit.

Figure 13.58 depicts the conceptual arrangement of a ROM with $n = 4$ and $m = 2$. The ROM table has been filled with arbitrary 4-bit words, just for the

ROM address		ROM content (4-bit words)				
I_1	I_0	b_3	b_2	b_1	b_0	
0	0	0	1	1	0	W_0
0	1	1	0	0	1	W_1
1	0	0	1	1	0	W_2
1	1	1	1	1	1	W_3

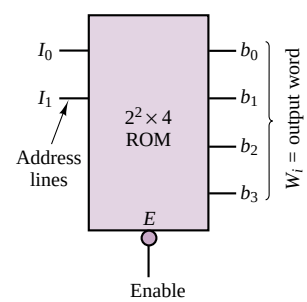


Figure 13.58 Read-only memory

purpose of illustration. In Figure 13.58, if one were to select an enable input of 0 (i.e., on) and values for the address lines of $I_0 = 0$ and $I_1 = 1$, the output word would be $W_2 = 0110$, so that $b_0 = 0$, $b_1 = 1$, $b_2 = 1$, $b_3 = 0$. Depending on the content of the ROM and the number of address and output lines, one could implement an arbitrary logic function.

Unfortunately, the data stored in read-only memories must be entered during fabrication and cannot be altered later. A much more convenient type of read-only memory is the **erasable programmable read-only memory (EPROM),** the content of which can be easily programmed and stored and may be changed if needed. EPROMs find use in many practical applications, because of their flexibility in content and ease of programming. The following example illustrates the use of an EPROM to perform the linearization of a nonlinear function.

EPROM-Based Lookup Table for Automotive Fuel Injection System Control

One of the most common applications of EPROMs is the *arithmetic lookup table.* A lookup table is similar in concept to the familiar multiplication table and is used to store precomputed values of certain functions, eliminating the need for actually computing the function. A practical application of this concept is present in every automobile manufactured in the United States since the early 1980s, as part of the **exhaust emission control system.** In order for the catalytic converter to minimize the emissions of exhaust gases (especially hydrocarbons, oxides of nitrogen, and carbon monoxide), it is necessary to maintain the *air-to-fuel ratio* (A/F) as close as possible to the stoichiometric value, that is, 14.7 parts of air for each part of fuel. Most modern engines are equipped with fuel injection systems that are capable of delivering accurate amounts of fuel to each individual cylinder—thus, the task of maintaining an accurate A/F amounts to measuring the mass of air that is aspirated into each cylinder and computing the corresponding mass of fuel. Many automobiles are equipped with a *mass airflow sensor,* capable of measuring the mass of air drawn into each cylinder during each engine cycle. Let the output of the mass airflow sensor be denoted by the variable M_A, and let this variable represent the mass of air (in g) actually entering a cylinder during a particular stroke. It is then desired to compute the mass of fuel, M_F (also expressed in g), required to achieve and A/F of 14.7. This computation is simply:

$$M_F = \frac{M_A}{14.7}$$

Although the above computation is a simple division, its actual calculation in a low-cost digital computer (such as would be used on an automobile) is rather complicated. It would be much simpler to tabulate a number of values of M_A, to precompute the variable M_F, and then to store the result of this computation into an EPROM. If the EPROM address were made to correspond to the tabulated values of air mass, and the content at each address to the corresponding fuel mass (according to the precomputed values of the expression $M_F = M_A/14.7$), it would not be necessary to perform the division by 14.7. For each measurement of air mass into one

cylinder, an EPROM address is specified and the corresponding content is read. The content at the specific address is the mass of fuel required by that particular cylinder.

In practice, the fuel mass needs to be converted into a time interval corresponding to the duration of time during which the fuel injector is open. This final conversion factor can also be accounted for in the table. Suppose, for example, that the fuel injector is capable of injecting K_F g of fuel per second; then the time duration, T_F, during which the injector should be open in order to inject M_F g of fuel into the cylinder is given by:

$$T_F = \frac{M_F}{K_F} \text{ s}$$

Therefore, the complete expression to be precomputed and stored in the EPROM is:

$$T_F = \frac{M_A}{14.7 \times K_F} \text{ s}$$

Figure 13.59 illustrates this process graphically.

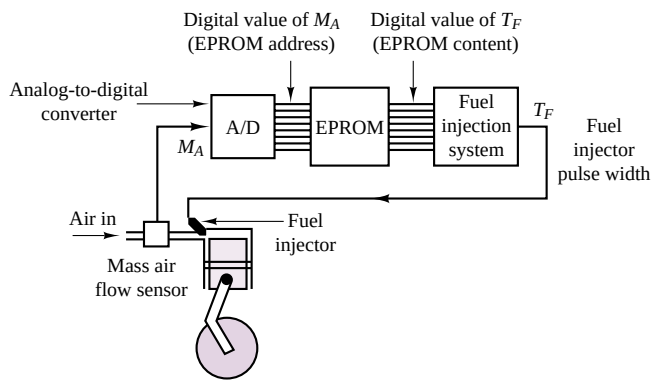


Figure 13.59 Use of EPROM lookup table in automotive fuel injection system

To provide a numerical illustration, consider a hypothetical engine capable of aspirating air in the range $0 < M_A < 0.51$ g and equipped with fuel injectors capable of injecting at the rate of 1.36 g/s. Thus, the relationship between T_F and M_A is:

$$T_F = 50 \times M_A \text{ ms} = 0.05 M_A \text{ s}$$

If the digital value of M_A is expressed in dg (decigrams, or tenths of g), the lookup table of Figure 13.60 can be implemented, illustrating the conversion capabilities provided by the EPROM. Note that in order to represent the quantities of interest in an appropriate binary format compatible with the 8-bit EPROM, the units of air mass and of time have been scaled.

M_A (g) $\times 10^{-2}$	Address (digital value of M_A)	Content (digital value of T_F)	T_F(ms) $\times 10^{-1}$
0	00000000	00000000	0
1	00000001	00000101	5
2	00000010	00001010	10
3	00000011	00001111	15
4	00000100	00010100	20
5	00000101	00011001	25
⋮	⋮	⋮	⋮
51	00110011	11111111	255

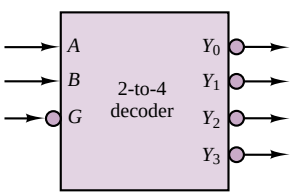
Figure 13.60 Lookup table for automotive fuel injection application

Decoders and Read and Write Memory

Decoders, which are commonly used for applications such as address decoding or memory expansion, are combinational logic circuits as well. Our reason for introducing decoders is to show some of the internal organization of semiconductor memory devices. An important application of decoders in the organization of a memory system is discussed in Chapter 14.

Figure 13.61 shows the truth table for a 2-to-4 decoder. The decoder has an enable input, $\overline{G}$, and select inputs, B and A. It also has four outputs, Y_0 through Y_3. When the enable input is logic 1, all decoder outputs are forced to logic 1 regardless of the select inputs.

This simple description of decoders permits a brief discussion of the internal organization of an **SRAM (static random-access** or **read and write memory).** SRAM is internally organized to provide memory with high speed (i.e., short access time), a large bit capacity, and low cost. The memory array in this memory device has a column length equal to the number of words, W, and a row length equal to the number of bits per word, N. To select a word, an n-to-W decoder is needed. Since the address inputs to the decoder select only one of the decoder's outputs, the decoder selects one word in the memory array. Figure 13.62 shows the internal organization of a typical SRAM.



Inputs			Outputs			
Enable	Select					
$\overline{G}$	A	B	Y_0	Y_1	Y_2	Y_3
1	**x**	**x**	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

Figure 13.61 2-to-4 decoder

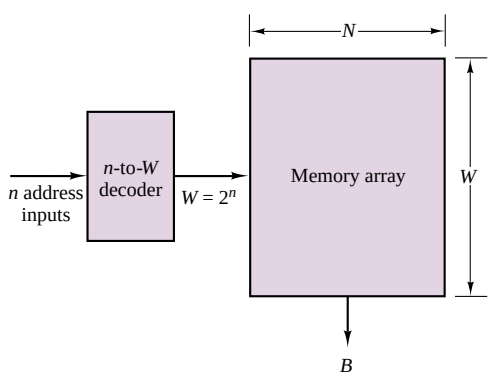


Figure 13.62 Internal organization of SRAM

Thus, to choose the desired word from the memory array, the proper address inputs are required. As an example, if the number of words in the memory array is 8, a 3-to-8 decoder is needed. Data sheets for 2-to-4 and 3-to-8 decoders from a CMOS family data book are provided in the accompanying CD-ROM.

Check Your Understanding

13.27 Which combination of the control lines will select the data line D_3 for a 4-to-1 MUX?

13.28 Show that an 8-to-1 MUX with eight data inputs (D_0 through D_7) and three control lines (I_0 through I_2) can be used as a data selector. Which combination of the control lines will select the data line D_5?

13.29 Which combination of the control lines will select the data line D_4 for an 8-to-1 MUX?

13.30 How many address inputs do you need if the number of words in a memory array is 16?

CONCLUSION

- Digital logic circuits are at the basis of digital computers. Such circuits operate strictly on binary signals according to the laws of Boolean algebra.
- Combinational logic circuits can implement arbitrary Boolean logic functions.
- Combinational logic circuits include all of the logic gates—AND, OR, NAND, NOT, and XOR—as well as logic modules such as multiplexers and read-only memory.

CHECK YOUR UNDERSTANDING ANSWERS

CYU 13.1 (a) 100111; (b) 111011; (c) 100000000; (d) 0.011100; (e) 0.11001; (f) 0.110011; (g) 100000000.11; (h) 10000001.1001; (i) 1000000000000.11101

CYU 13.2 (a) 13; (b) 27; (c) 23; (d) 0.6875; (e) 0.203125; (f) 0.2128906 0.2128906255; (g) 59.6875; (h) 91.203125; (i) 22.340820312

CYU 13.3 (a) 20.75_{10}; (b) 74_{10}; (c) 1.5_{10}; (d) 21_{10}; (e) 100000_2; (f) 1000000_2; (g) 110010.11_2; (h) 100011.11111_2

CYU 13.4 4,096

CYU 13.5 39 mV

CYU 13.6 (a) 111110000011; (b) 00111001001; (c) 10100110; (d) 1AE; (e) B9; (f) 6ED

CYU 13.7 (a) 00010111; (b) 01101001; (c) 0100010

CYU 13.8 (a) 0000 1011 1101; (b) 1101 0100 0111; (c) 0101 1010

CYU 13.16 $W \cdot \overline{Z} + \overline{X} \cdot \overline{Z}$

CYU 13.17 $\overline{Y} \cdot \overline{Z} + \overline{X} \cdot \overline{Z}$

CYU 13.18 Nine gates

CYU 13.20 No

CYU 13.22 $f = a \cdot b \cdot \overline{c} \cdot \overline{d} + \overline{a} \cdot \overline{c} \cdot d + \overline{a} \cdot \overline{b} \cdot c + \overline{b} \cdot c \cdot d$

CYU 13.23 $f = \overline{a} \cdot b + \overline{a} \cdot \overline{c}$

CYU 13.24 $f = \overline{a} \cdot b + a \cdot \overline{b} + \overline{c}$

CYU 13.25 $f = \overline{a} \cdot \overline{b} \cdot \overline{c} \cdot \overline{d} + \overline{a} \cdot \overline{b} \cdot c \cdot d + a \cdot \overline{b} \cdot \overline{c} \cdot d + \overline{a} \cdot b \cdot c \cdot \overline{d}$

CYU 13.26 $f = \overline{a} \cdot \overline{b} \cdot \overline{c} \cdot \overline{d} + b \cdot c \cdot \overline{d} + a \cdot d + \overline{b} \cdot c \cdot d + a \cdot c$

CYU 13.27 $I_1 I_0 = 11$

CYU 13.28 For the first part, use the same method as in Check Your Understanding Exercise 13.27, but for an 8-to-1 MUX. For the second part, $I_2 I_1 I_0 = 101$.

CYU 13.29 $I_2 I_1 I_0 = 100$

CYU 13.30 4

HOMEWORK PROBLEMS

Section 1: Number Systems

13.1 Convert the following base 10 numbers to hex and binary:

a. 401 b. 273 c. 15 d. 38 e. 56

13.2 Convert the following hex numbers to base 10 and binary:

a. A b. 66 c. 47 d. 21 e. 13

13.3 Convert the following base 10 numbers to binary:

a. 271.25 b. 53.375 c. 37.32 d. 54.27

13.4 Convert the following binary numbers to hex and base 10:

a. 1111 b. 1001101 c. 1100101 d. 1011100
e. 11101 f. 101000

13.5 Perform the following additions all in the binary system:

a. $11001011 + 101111$
b. $10011001 + 1111011$
c. $11101001 + 10011011$

13.6 Perform the following subtractions all in the binary system:

a. $10001011 - 1101111$
b. $10101001 - 111011$
c. $11000011 - 10111011$

13.7 Assuming that the most significant bit is the sign bit, find the decimal value of the following sign-magnitude form eight-bit binary numbers:

a. 11111000 b. 10011111 c. 01111001

13.8 Find the sign-magnitude form binary representation of the following decimal numbers:

a. 126 b. -126 c. 108 d. -98

13.9 Find the two's complement of the following binary numbers:

a. 1111 b. 1001101 c. 1011100 d. 11101

Section 2: Combinational Logic

13.10 Use a truth table to prove that $B = AB + \overline{A}B$.

13.11 Use truth tables to prove that $BC + B\overline{C} + \overline{B}A = A + B$.

13.12 Using the method of proof by perfect induction, show that

$$(X + Y) \cdot (\overline{X} + X \cdot Y) = Y$$

13.13 Using De Morgan's theorems and the rules of Boolean algebra, simplify the following logic function:

$$F(X, Y, Z) = \overline{X} \cdot \overline{Y} \cdot \overline{Z} + \overline{X} \cdot Y \cdot Z + X \cdot (\overline{Y + Z})$$

13.14 Simplify the expression
$f(A, B, C, D) = ABC + \overline{A}CD + \overline{B}CD$.

13.15 Simplify the logic function
$F(A, B, C) = \overline{A} \cdot B \cdot \overline{C} + \overline{A} \cdot B \cdot C + A \cdot B \cdot \overline{C} + A \cdot B \cdot C$
using Boolean algebra.

13.16 Find the logic function defined by the truth table given in Figure P13.16.

A	*B*	*C*	*F*
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Figure P13.16

13.17 Determine the Boolean function describing the operation of the circuit shown in Figure P13.17.

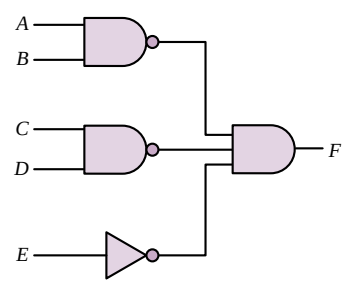


Figure P13.17

13.18 Use a truth table to show when the output of the circuit of Figure P13.18 is 1.

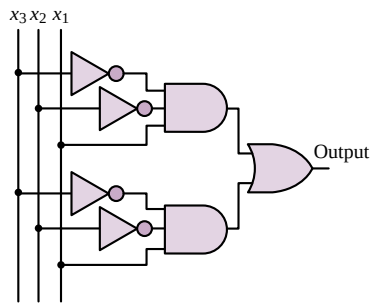


Figure P13.18

13.19 Baseball is a complicated game and often the manager has a difficult time keeping track of all the rules of thumb that guide decisions. To assist your favorite baseball team you have been asked to design a logic circuit that will flash a light when the manager should give the steal sign. The rules have been laid out for you by a baseball fan with limited knowledge of the game as follows: Give the steal sign if there is a runner on first base and

a. There are no other runners, the pitcher is right-handed, and the runner is fast, or

b. There is one other runner on third-base, and one of the runners is fast, or

c. There is one other runner on second-base, the pitcher is left-handed, and both runners are fast.

Under no circumstances should the steal sign be given if all three bases have runners. Design a logic circuit that implements these rules to indicate when the steal sign should be given.

13.20 A small county board is composed of three commissioners. Each commissioner votes on measures presented to the board by pressing a button indicating whether the commissioner votes for or against a measure. If two or more commissioners vote for a measure it passes. Design a logic circuit that takes the three votes as inputs and lights either a green or red light to indicate whether or not a measure passed.

13.21 A water purification plant uses one tank for chemical sterilization and a second larger tank for settling and aeration. Each tank is equipped with two sensors that measure the height of water in each tank and the flow rate of water into each tank. When the height of water or flow rate is too high the sensors produce a logic high output. Design a logic circuit that sounds an alarm whenever the height of water in both tanks is too high and either of the flow rates is too high, or whenever both flow rates are too high and the height of water in either tank is also too high.

13.22 Many automobiles incorporate logic circuits to alert the driver of problems or potential problems. In one particular car, a buzzer is sounded whenever the ignition key is turned and either a door is open or a seat belt is not fastened. The buzzer also sounds when the key is not turned but the lights are on. In addition, the car will not start unless the key is in the ignition, the car is in park, and all doors are closed and seat belts fastened. Design a logic circuit that takes all of the inputs listed and sounds the buzzer and starts the car when appropriate.

13.23 An on/off start-up signal governs the compressor motor of a large commercial air conditioning unit. In general, the start-up signal should be on whenever the output of a temperature sensor (S) exceeds a reference temperature. However, you are asked to limit the compressor start-ups to certain hours of the day and also enable service technicians to start up or shut down the compressor through a manual override. A time-of-day indicator (D) is available with on/off outputs as is a manual override switch (M). A separate timer (T) prohibits a compressor start-up within 10 minutes of a previous shutdown. Design a logic diagram that incorporates the state of all four devices (S, D, M, and T) and produces the correct on/off condition for the motor start-up.

Section 3: Logic Design

13.24 Find the logic function corresponding to the truth table of Figure P13.24 in the simplest sum-of-products form.

A	*B*	*C*	*F*
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Figure P13.24

13.25 Find the minimum expression for the output of the logic circuit shown in Figure P13.25.

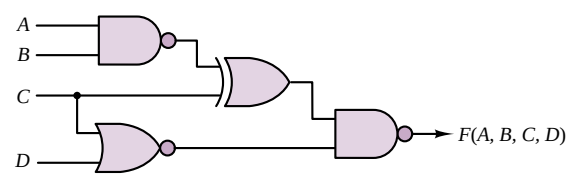


Figure P13.25

13.26 Use a Karnaugh map to minimize the function $f(A, B, C) = ABC + AB\overline{C} + A\overline{B}\overline{C}$.

13.27

a. Build the Karnaugh map for the logic function defined by the truth table of Figure P13.27.

b. What is the minimum expression for this function?

c. Realize F using AND, OR, and NOT gates.

A	*B*	*C*	*D*	*F*
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

Figure P13.27

13.28 Fill in the Karnaugh map for the function defined by the truth table of Figure P13.28, and find the minimum expression for the function.

A	*B*	*C*	*f*(*A*,*B*,*C*)
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Figure P13.28

13.29 A function, F, is defined such that it equals 1 when a 4-bit input code is equivalent to any of the decimal numbers 3, 6, 9, 12 or 15. F is 0 for input codes 0, 2, 8 and 10. Other input values cannot occur. Use a Karnaugh map to determine a minimal expression for this function. Design and sketch a circuit to implement this function using only AND and NOT gates.

13.30 The function described in Figure P13.30 can be constructed using only two gates. Design the circuit.

Input			Output
A	*B*	*C*	*F*
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	x

Figure P13.30

13.31 Design a logic circuit which will produce the one's complement of an 8-bit signed binary number.

13.32 Construct the Karnaugh map for the logic function defined by the truth table of Figure P13.32, and find the minimum expression for the function.

A	*B*	*C*	*D*	*F*
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

Figure P13.32

13.33 Modify the circuit for Problem 13.31 so that it produces the two's complement of the 8-bit signed binary input.

13.34 Find the minimum output expression for the circuit of Figure P13.34.

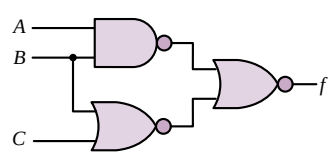


Figure P13.34

13.35 Design a combinational logic circuit which will add two 4-bit binary numbers.

13.36 Minimize the expression described in the truth table of Figure P13.36 and draw the circuit.

A	B	C	F
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Figure P13.36

13.37 Find the minimum expression for the output of the logic circuit of Figure P13.37.

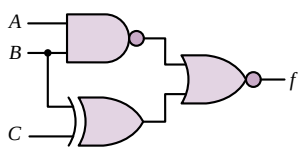


Figure P13.37

13.38 The objective of this problem is to design a combinational logic circuit which will aid in determination of the acceptability of emergency blood transfusions. It is known that human blood can be categorized into four types—A, B, AB, and O. Persons with type A blood can donate to both A and AB types, and can receive blood from both A and O types. Persons with type B blood can donate to both B and AB, and can receive from both B and O types. Persons with type AB blood can donate only to type AB, but can receive from any type. Persons with type O blood can donate to any type, but can receive only from type O. Make appropriate variable assignments and design a circuit which will approve or disapprove any particular transfusion based on these conditions.

13.39 Find the minimum expression for the logic function at the output of the logic circuit of Figure P13.39.

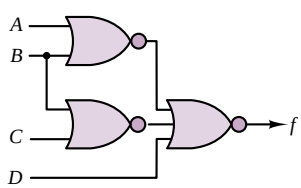


Figure P13.39

13.40 Design a combinational logic circuit which will accept a 4-bit binary number and:

If the number is even, divide it by 2_{10} and produce the binary result.
If the number is odd, multiply it by 2_{10} and produce the binary result.

13.41

a. Fill in the Karnaugh map for the function defined in the truth table of Figure P13.41.
b. What is the minimum expression for the function?
c. Draw the circuit, using AND, OR, and NOT gates.

A	B	C	f(A,B,C)
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Figure P13.41

13.42

a. Fill in the Karnaugh map for the logic function defined by the truth table of Figure P13.42.
b. What is the minimum expression for the function?

A	B	C	D	F
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

Figure P13.42

13.43

a. Fill in the Karnaugh map for the logic function defined by the truth table of Figure P13.43.
b. What is the minimum expression for the function?
c. Realize the function, using only NAND gates.

A	B	C	D	F
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

Figure P13.43

13.44 Design a circuit with a four-bit input representing the binary number $A_3A_2A_1A_0$. The output should be 1 if the input value is divisible by 3. Assume that the circuit is to be used only for the digits 0 through 9 (thus, values for 10 to 15 can be don't cares).

a. Draw the Karnaugh map and truth table for the function.

b. Determine the minimum expression for the function.

c. Draw the circuit, using only AND, OR, and NOT gates.

13.45 Find the simplified sum-of-products representation of the function from the Karnaugh map shown in Figure P13.45. Note that x is the don't care term.

C·D \ A·B	00	01	11	10
00	0	1	0	0
01	1	1	0	0
11	0	x	1	0
10	0	0	1	0

Figure P13.45

13.46 Can the circuit for Problem 13.40 be simplified if it is known that the input represents a BCD (binary-coded decimal) number, i.e., it can never be greater than 10_{10}? If not, explain why not. Otherwise, design the simplified circuit.

13.47 Find the simplified sum-of-products representation of the function from the Karnaugh map shown in Figure P13.47.

C·D \ A·B	00	01	11	10
00	0	1	x	0
01	0	1	x	0
11	0	1	0	1
10	x	x	1	0

Figure P13.47

13.48 One method of ensuring reliability in data transmission systems is to transmit a parity bit along with every nibble, byte, or word of binary data transmitted. The parity bit confirms whether an even or odd number of 1's were transmitted in the data. In even-parity systems, the parity bit is set to 1 when the number of 1's in the transmitted data is odd. Odd-parity systems set the parity bit to 1 when the number of 1's in the transmitted data is even. Assume that a parity-bit is transmitted for every nibble of data. Design a logic circuit that checks the nibble of data and transmits the proper parity bit for both even- and odd-parity systems.

13.49 Assume that a parity bit is transmitted for every nibble of data. Design two logic circuits that check a nibble of data and its parity bit to determine if there may have been an data transmission error. First assume an even-parity system, then an odd-parity system.

13.50 Design a logic circuit that takes a 4-bit Gray code input from an optical encoder and translates it into two 4-bit nibbles of BCD code.

13.51 Design a logic circuit that takes a 4-bit Gray code input from an optical encoder and determines if the input value is a multiple of 3.

13.52 The 4221 code is a base 10–oriented code that assigns the weights 4221 to each of 4 bits in a nibble of data. Design a logic circuit that takes a BCD nibble as input and converts it to its 4221 equivalent. The logic circuit should also report an error in the BCD input if its value exceeds 1001.

13.53 The 4-bit digital output of each of two sensors along an assembly line conveyor belt is proportional to the number of parts which pass by on the conveyor belt in a 30-second period. Design a logic circuit that reports an error if the outputs of the two sensors differ by more than one part per 30-second period.

Section 4: Logic Modules

13.54

a. Fill in the Karnaugh map for the logic function defined by the truth table of Figure P13.54.

b. What is the minimum expression for the function?

c. Realize the function using a 1-of-8 multiplexer.

A	B	C	D	f(A,B,C,D)
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

Figure P13.54

13.55

a. Fill in the truth table for the multiplexer circuit shown in Figure P13.55.

b. What binary function is performed by these multiplexers?

x	y	C	S
0	0		
0	1		
1	0		
1	1		

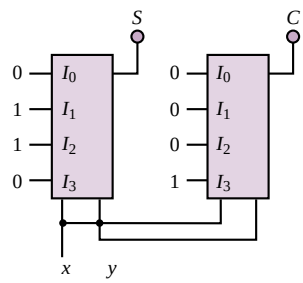


Figure P13.55

13.56 The circuit of Figure P13.56 can operate as a 4-to-16 decoder. Terminal EN denotes the enable input. Describe the operation of the 4-to-16 decoder. What is the role of logic variable A?

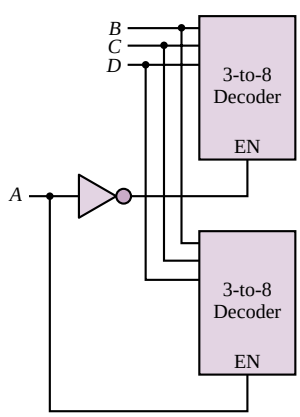


Figure P13.56

13.57 Show that the circuit given in Figure P13.57 converts 4-bit binary numbers to 4-bit Gray code.

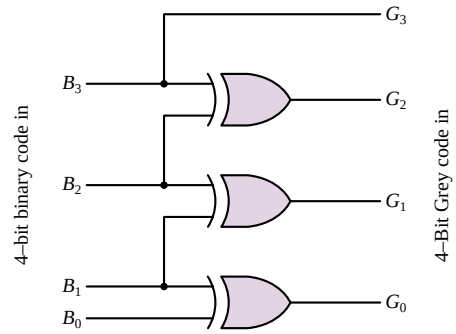


Figure P13.57

13.58 Suppose one of your classmates claims that the following Boolean expressions represent the conversion from 4-bit Gray code to 4-bit binary numbers:

$$B_3 = G_3$$
$$B_2 = G_3 \oplus G_2$$
$$B_1 = G_3 \oplus G_2 \oplus G_1$$
$$B_0 = G_3 \oplus G_2 \oplus G_1 \oplus G_0$$

a. Show that your classmate's claim is correct.

b. Draw the circuit which implements the conversion.

13.59 Select the proper inputs for a 4-input multiplexer to implement the function $f(A, B, C) = \overline{A}B\overline{C} + A\overline{B}\overline{C} + AC$. Assume the inputs I_0, I_1, I_2, and I_3 correspond to $\overline{A}\overline{B}$, $\overline{A}B$, $A\overline{B}$, and AB, respectively, and that each input may be 0, 1, $\overline{C}$, or C.

13.60 Select the proper inputs for an 8-bit multiplexer to implement the function $f(A, B, C, D) = \sum(2, 5, 6, 8, 9, 10, 11, 13, 14)_{10}$. Assume the inputs I_0 through I_7 correspond to $\overline{A}\overline{B}\overline{C}$, $\overline{A}\overline{B}C$, $\overline{A}B\overline{C}$, $\overline{A}BC$, $A\overline{B}\overline{C}$, $A\overline{B}C$, $AB\overline{C}$, and ABC, respectively, and that each input may be 0, 1, $\overline{D}$, or D.

CHAPTER

14

Digital Systems

The first half of Chapter 14 continues the analysis of digital circuits that was begun in Chapter 13 by focusing on sequential logic circuits, such as flip-flops, counters, and shift registers. The second half of the chapter is devoted to an overview of the basic functions of microcontrollers and microcomputers. During the last decade, microcomputers have become a standard tool in the analysis of engineering data, in the design of experiments, and in the control of plants and processes. No longer a specialized electronic device to be used only by appropriately trained computer engineers, today's microcomputer—perhaps more commonly represented by the ubiquitous *personal computer*—is a basic tool in the engineering profession. The common thread in its application in various engineering fields is its use in digital data acquisition instruments and digital controllers.

Modern microcomputers are relatively easy to program, have significant computing power and excellent memory storage capabilities, and can be readily interfaced with other instruments and electronic devices, such as transducers, printers, and other computers. The basic functions performed by the microcomputer in a typical digital data acquisition or control application are easily described: input signals (often analog, sometimes already in digital form) are acquired by the computer and processed by means of suitable software to produce the desired result (i.e., they undergo some kind of mathematical manipulation), which is then outputted to either a display or a storage device, or is used in controlling a process,

a plant, or an experiment. The objective of this chapter is to describe these various processes, with the aim of giving the reader enough background information to understand the notation used in data books and instruction manuals.

Upon completing this chapter you should be able to:

- Analyze sequential circuits including *RS*, *D*, and *JK* flip-flops.
- Understand the operation of binary, decade, and ring counters.
- Design simple sequential circuits using state transition diagrams.
- Understand the basic architecture of microprocessors and microcomputers.

14.1 SEQUENTIAL LOGIC MODULES

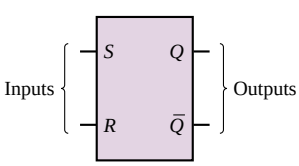
The discussion of logic devices in Chapter 13 focuses on the general family of combinational logic devices. The feature that distinguishes combinational logic devices from the other major family—**sequential logic devices**—is that combinational logic circuits provide outputs that are based on a combination of present inputs only. On the other hand, sequential logic circuits depend on present and past input values. Because of this "memory" property, sequential circuits can store information; this capability opens a whole new area of application for digital logic circuits.

Latches and Flip-Flops

The basic information-storage device in a digital circuit is called a **flip-flop.** There are many different varieties of flip-flops; however, all flip-flops share the following characteristics:

1. A flip-flop is a **bistable device;** that is, it can remain in one of two stable states (0 and 1) until appropriate conditions cause it to change state. Thus, a flip-flop can serve as a memory element.
2. A flip-flop has two outputs, one of which is the complement of the other.

RS Flip-Flop



S	R	Q
0	0	Present state
0	1	Reset
1	0	Set
1	0	Disallowed

Figure 14.1 *RS* flip-flop symbol and truth table

It is customary to depict flip-flops by their block diagram and a name—such as Q or X—representing the output variable. Figure 14.1 represents the so-called ***RS* flip-flop,** which has two inputs, denoted by S and R, and two outputs, Q and $\overline{Q}$. The value at Q is called the *state* of the flip-flop. If $Q = 1$, we refer to the device as *being in the 1 state.* Thus, we need define only one of the two outputs of the flip-flop. The two inputs, R and S, are used to change the state of the flip-flop, according to the following rules:

1. When $R = S = 0$, the flip-flop remains in its present state (whether 1 or 0).
2. When $S = 1$ and $R = 0$, the flip-flop is *set* to the 1 state (thus, the letter S, for **set**).
3. When $S = 0$ and $R = 1$, the flip-flop is *reset* to the 0 state (thus, the letter R, for **reset**).
4. It is not permitted for both S and R to be equal to 1. (This would correspond to requiring the flip-flop to set and reset at the same time.)

The rules just described are easily remembered by noting that 1s on the S and R inputs correspond to the set and reset commands, respectively.

A convenient means of describing the series of transitions that occur as the signals sent to the flip-flop inputs change is the **timing diagram.** A timing diagram is a graph of the inputs and outputs of the *RS* flip-flop (or any other logic device) depicting the transitions that occur over time. In effect, one could also represent these transitions in tabular form; however, the timing diagram provides a convenient visual representation of the evolution of the state of the flip-flop. Figure 14.2 depicts a table of transitions for an *RS* flip-flop Q, as well as the corresponding timing diagram.

S	R	Q
1	0	1
0	0	1
0	0	1
0	1	0
0	0	0
0	0	0
0	1	0
0	0	0
1	0	1
0	0	1

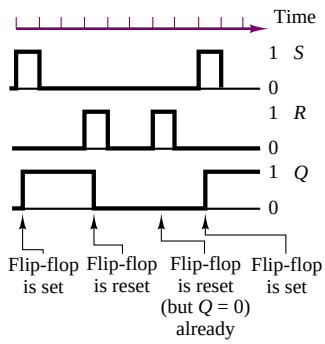


Figure 14.2 Timing diagram for the *RS* flip-flop

It is important to note that the *RS* flip-flop is **level-sensitive.** This means that the set and reset operations are completed only after the R and S inputs have reached the appropriate levels. Thus, in Figure 14.2 we show the transitions in the Q output as occurring with a small delay relative to the transitions in the R and S inputs.

It is instructive to illustrate how an *RS* flip-flop can be constructed using simple logic gates. For example, Figure 14.3 depicts a realization of such a circuit consisting of four gates: two inverters and two NAND gates (actually, the same result could be achieved with four NAND gates). Consider the case in which the circuit is in the initial state $Q = 0$ (and therefore $\overline{Q} = 1$). If the input $S = 1$ is applied, the top NOT gate will see inputs $\overline{Q} = 1$ and $\overline{S} = 0$, so that $Q = (\overline{\overline{S} \cdot \overline{Q}}) = (\overline{0 \cdot 1}) = 1$—that is, the flip-flop is set. Note that when Q is set to 1, $\overline{Q}$ becomes 0. This, however, does not affect the state of the Q output, since replacing $\overline{Q}$ with 0 in the expression

$$Q = (\overline{\overline{S} \cdot \overline{Q}})$$

does not change the result:

$$Q = (\overline{0 \cdot 0}) = 1$$

Thus, the cross-coupled feedback from outputs Q and $\overline{Q}$ to the input of the NAND gates is such that the set condition sustains itself. It is straightforward to show (by symmetry) that a 1 input on the R line causes the device to reset (i.e., causes $Q = 0$) and that this condition is also self-sustaining.

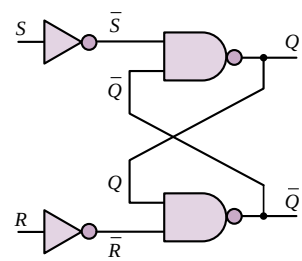


Figure 14.3 Logic gate implementation of the *RS* flip-flop

EXAMPLE 14.1 *RS* Flip-Flop Timing Diagram

Problem

Determine the output of an *RS* flip-flop for the series of inputs given in the table below.

R	0	0	0	1	0	0	0
S	1	0	1	0	0	1	0

Solution

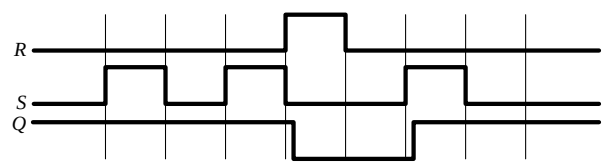
Known Quantities: *RS* flip-flop truth table (Figure 14.1).

Find: Output of *RS* flip-flop, Q.

Analysis: We complete the timing diagram for the *RS* flip-flop following the rules stated earlier to determine the output of the device; the result is summarized below.

R	0	0	0	1	0	0	0
S	1	0	1	0	0	1	0
Q	1	1	1	0	0	1	1

A sketch of the waveforms, shown below, can also be generated to visualize the transitions.



An extension of the *RS* flip-flop includes an additional enable input that is *gated* into each of the other two inputs. Figure 14.4 depicts an *RS* flip-flop consisting of two NOR gates. In addition, an enable input is connected through two AND gates to the *RS* flip-flop, so that an input to the R or S line will be effective only when the enable input is 1. Thus, any transitions will be controlled by the enable input, which acts as a synchronizing signal. The enable signal may consist of a **clock,** in which case the flip-flop is said to be **clocked** and its operation is said to be **synchronous.**

The same circuit of Figure 14.4 can be used to illustrate two additional features of flip-flops: the **preset** and **clear** functions, denoted by the inputs P and C, respectively. When P and C are 0, they do not affect the operation of the flip-flop. Setting $P = 1$ corresponds to setting $S = 1$, and therefore causes the flip-flop to go into the 1 state. Thus, the term *preset:* this function allows the user to preset the flip-flop to 1 at any time. When C is 1, the flip-flop is reset, or *cleared* (i.e., Q is made equal to 0). Note that these direct inputs are, in general,

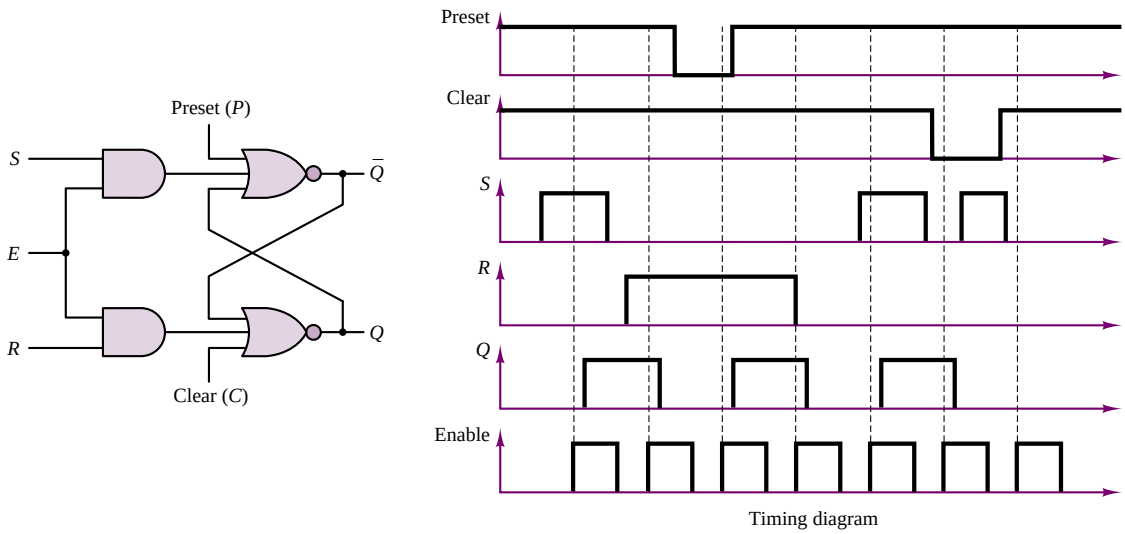


Figure 14.4 *RS* flip-flop with enable, preset, and clear lines

asynchronous; therefore, they allow the user to preset or clear the flip-flop at any time. A set of timing waveforms illustrating the function of the enable, preset, and clear inputs is also shown in Figure 14.4. Note how transitions occur only when the enable input goes high (unless the preset or clear inputs are used to override the *RS* inputs).

Another extension of the *RS* flip-flop, called the **data latch,** or **delay element,** is shown in Figure 14.5. In this circuit, the *R* input is always equal to the inverted *S* input, so that whenever the enable input is high, the flip-flop is set. This device has the dual advantage of avoiding the potential conflict that might arise if both *R* and *S* were high and reducing the number of input connections by eliminating the reset input. This circuit is called a data latch or delay because once the enable input goes low, the flip-flop is latched to the previous value of the input. Thus, this device can serve as a basic memory element, delaying the output by one clock count with respect to the input.

D Flip-Flop

The ***D* flip-flop** is an extension of the data latch that utilizes two *RS* flip-flops, as shown in Figure 14.6. In this circuit, a clock is connected to the enable input of each flip-flop. Since Q_1 sees an inverted clock signal, the latch is enabled when the clock waveform goes low. However, since Q_2 is disabled when the clock is low, the output of the *D* flip-flop will not switch to the 1 state until the clock goes high, enabling the second latch and transferring the state of Q_1 to Q_2. It is important to note that the *D* flip-flop changes state only on the positive edge of the clock waveform: Q_1 is set on the negative edge of the clock, and Q_2 (and therefore Q) is set on the positive edge of the clock, as shown in the timing diagram of Figure 14.6. This type of device is said to be **edge-triggered.** This feature is indicated by the "knife edge" drawn next to the CLK input in the

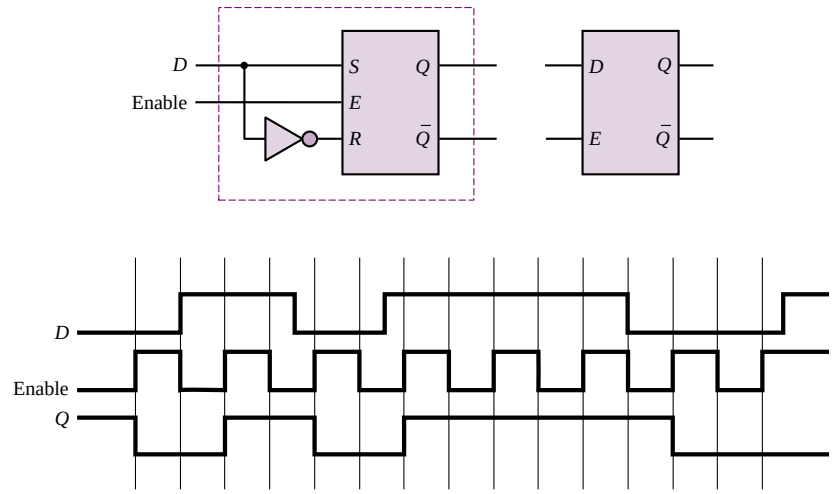


Figure 14.5 Data latch and associated timing diagram

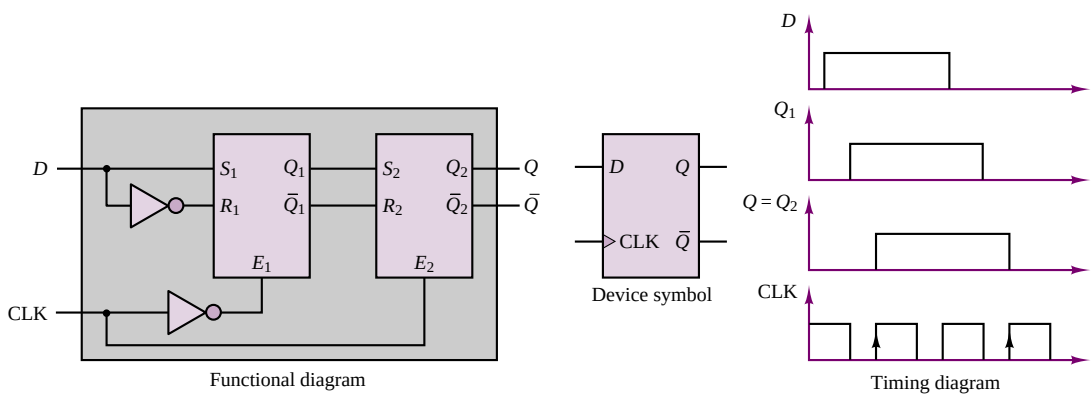


Figure 14.6 *D* flip-flop functional diagram, symbol, and timing waveforms

device symbol. The particular device described here is said to be positive edge–triggered, or **leading edge–triggered,** since the final output of the flip-flop is set on a positive-going clock transition.

On the basis of the rules stated in this section, the state of the *D* flip-flop can be described by means of the following truth table:

D	**CLK**	***Q***
0	↑	0
1	↑	1

where the symbol ↑ indicates the occurrence of a positive transition.

JK Flip-Flop

Another very common type of flip-flop is the ***JK* flip-flop,** shown in Figure 14.7. The *JK* flip-flop operates according to the following rules:

- When J and K are both low, no change occurs in the state of the flip-flop.
- When $J = 0$ and $K = 1$, the flip-flop is reset to 0.
- When $J = 1$ and $K = 0$, the flip-flop is set to 1.
- When both J and K are high, the flip-flop will toggle between states at every negative transition of the clock input, denoted from here on by the symbol $\downarrow$.

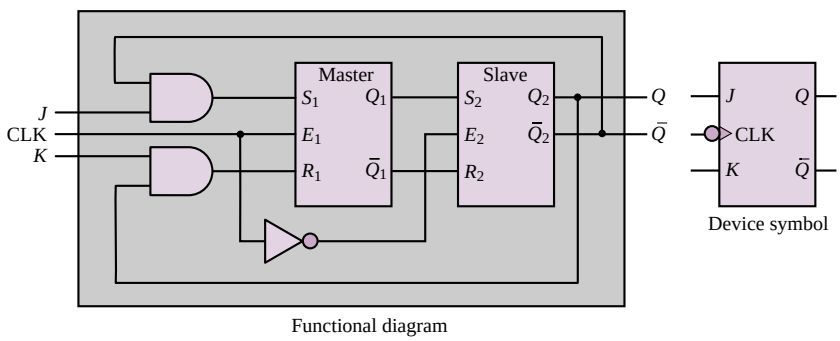
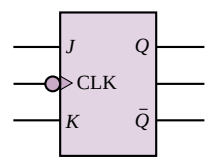


Figure 14.7 *JK* flip-flop functional diagram and device symbol

Note that, functionally, the operation of the *JK* flip-flop can also be explained in terms of two *RS* flip-flops. When the clock waveform goes high, the "master" flip-flop is enabled; the "slave" receives the state of the master upon a negative clock transition. The "bubble" at the clock input signifies that the device is negative or **trailing edge–triggered.** This behavior is similar to that of an *RS* flip-flop, except for the $J = 1$, $K = 1$ condition, which corresponds to a toggle mode rather than to a disallowed combination of inputs.

Figure 14.8 depicts the truth table for the *JK* flip-flop. It is important to note that when both inputs are 0 the flip-flop remains in its previous state at the occurrence of a clock transition; when either input is high and the other is low, the *JK* flip-flop behaves like the *RS* flip-flop, whereas if both inputs are high, the output "toggles" between states every time the clock waveform undergoes a negative transition.

Data sheets for various types of flip-flops may be found in the accompanying CD-ROM.



JK flip-flop

J_n	K_n	Q_{n+1}
0	0	Q_n
0	1	0 (reset)
1	0	1 (set)
1	1	$\bar{Q}_n$ (toggle)

Figure 14.8 Truth table for the *JK* flip-flop

EXAMPLE 14.2 The *T* Flip-Flop

Problem

Determine the truth table and timing diagram of the ***T* flip-flop** of Figure 14.9. Note that the *T* flip-flop is a *JK* flip-flop with its inputs tied together.

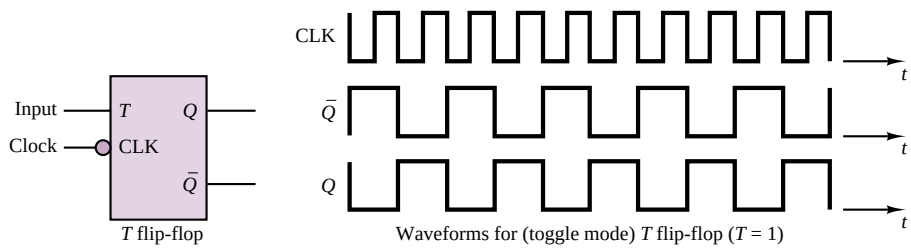


Figure 14.9 *T* flip-flop symbol and timing waveforms

Solution

Known Quantities: *JK* flip-flop rules of operation (Figure 14.8).

Find: Truth table and timing diagram for *T* flip-flop.

Analysis: We recognize that the *T* flip-flop is a *JK* flip-flop with its inputs tied together. Thus, the flip-flop will need only a two-element truth table to describe its operation, corresponding to the top and bottom entries in the *JK* flip-flop truth table of Figure 14.8. The truth table is shown below. A timing diagram is also included in Figure 14.9.

T	CLK	Q_{k+1}
0	↓	Q_k
1	↓	$\overline{Q_k}$

Comments: The *T* flip-flop takes its name from the fact that it *toggles* between the high and low state. Note that the toggling frequency is one half that of the clock. Thus the *T* flip-flop also acts as a *divide-by-2* counter. Counters are explored in more detail in the next subsection.

EXAMPLE 14.3 *JK* Flip-Flop Timing Diagram

Problem

Determine the output of a *JK* flip-flop for the series of inputs given in the table below. The initial state of the flip-flop is $Q_0 = 1$.

J	0	1	0	1	0	0	1
K	0	1	1	0	0	1	1

Solution

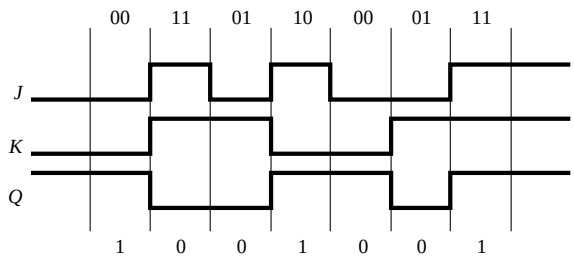
Known Quantities: *JK* flip-flop truth table (Figure 14.8).

Find: Output of *RS* flip-flop, Q, as a function of the input transitions.

Analysis: We complete the timing diagram for the *JK* flip-flop following the rules of Figure 14.8; the result is summarized below.

J	0	0	0	1	0	0	0
K	1	0	1	0	0	1	0
Q	1	0	0	1	1	0	1

A sketch of the waveforms, shown below, can also be generated to visualize the transitions. Each vertical line corresponds to a clock transition.



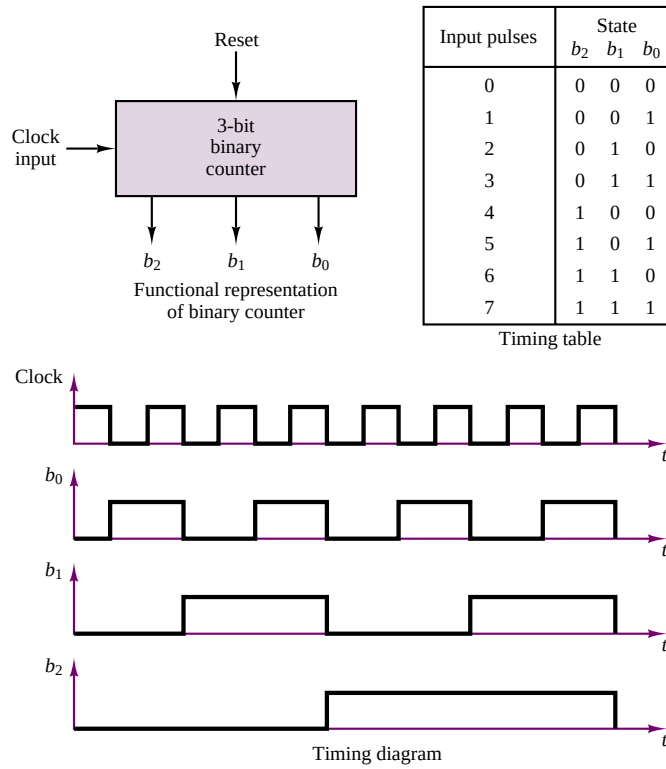
Comments: How would the timing diagram change if the initial state of the flip-flop were $Q_0 = 1$?

Digital Counters

One of the more immediate applications of flip-flops is in the design of **counters.** A counter is a sequential logic device that can take one of N possible states, stepping through these states in a sequential fashion. When the counter has reached its last state, it resets to zero and is ready to start counting again. For example, a three-bit **binary up counter** would have $2^3 = 8$ possible states, and might appear as shown in the functional block of Figure 14.10. The input clock waveform causes the counter to step through the eight states, making one transition for each clock pulse. We shall shortly see that a string of *JK* flip-flops can accomplish this task exactly. The device shown in Figure 14.10 also displays a reset input, which forces the counter output to equal 0: $b_2b_1b_0 = 000$.

Although binary counters are very useful in many applications, one is often interested in a **decade counter,** that is, a counter that counts from 0 to 9 and then resets. A four-bit binary counter can easily be configured in principle to provide this function by means of simple logic that resets the counter when it has reached the count $1001_2 = 9_{10}$. As shown in Figure 14.11, if we connect bits b_3 and b_1 to a four-input AND gate, along with $\overline{b}_2$ and $\overline{b}_0$, the output of the AND gate can be used to reset the counter after a count of 10. Additional logic can provide a "carry" bit whenever a reset condition is reached, which could be passed along to another decade counter, enabling counts up to 99. Decade counters can be cascaded so as to represent decimal digits in succession.

Although the decade counter of Figure 14.11 is attractive because of its simplicity, this configuration would never be used in practice, because of the



Input pulses	State b_2	b_1	b_0
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

Timing table

Figure 14.10 Binary up counter functional representation, state table, and timing waveforms

Input pulses	b_3	b_2	b_1	b_0	
0	0	0	0	0	
1	0	0	0	1	
2	0	0	1	0	
3	0	0	1	1	
4	0	1	0	0	
5	0	1	0	1	
6	0	1	1	0	
7	0	1	1	1	
8	1	0	0	0	
9	1	0	0	1	
10	1	0	1	0	Reset

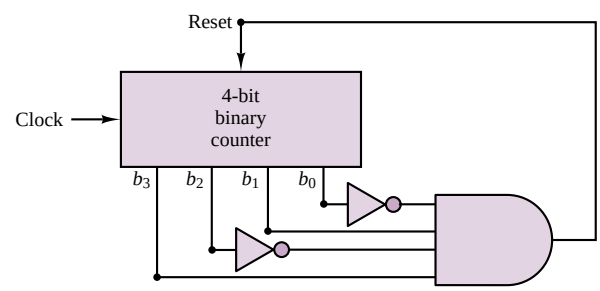


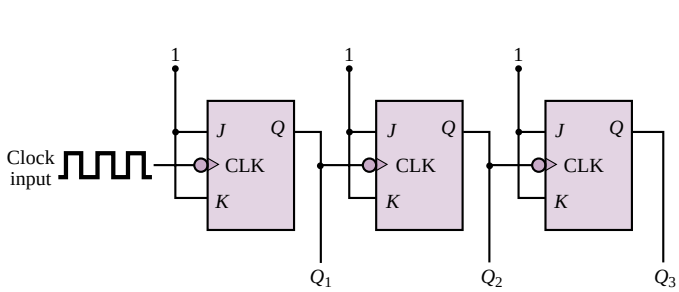
Figure 14.11 Decade counter

presence of **propagation delays.** These delays are caused by the finite response time of the individual transistors in each logic device and cannot be guaranteed to be identical for each gate and flip-flop. Thus, if the reset signal—which is presumed to be applied at exactly the same time to each of the four *JK* flip-flops in the four-bit

binary counter—does not cause the *JK* flip-flops to reset at exactly the same time on account of different propagation delays, then the binary word appearing at the output of the counter will change from 1001 to some other number, and the output of the four-input NAND gate will no longer be high. In such a condition, the flip-flops that have not already reset will then not be able to reset, and the counting sequence will be irreparably compromised.

What can be done to obviate this problem? The answer is to use a systematic approach to the design of sequential circuits making use of **state transition diagrams.** This topic will be discussed in the next section.

A simple implementation of the binary counter we have described in terms of its functional behavior is shown in Figure 14.12. The figure depicts a three-bit binary **ripple counter,** which is obtained from a cascade of three *JK* flip-flops. The transition table shown in the figure illustrates how the Q output of each stage becomes the clock input to the next stage, while each flip-flop is held in the toggle mode. The output transitions assume that the clock, CLK, is a simple square wave (all *JK*s are negative edge–triggered).



Input	Q_3	Q_2	Q_1
⎍	0	0	0
⎍	0	0	1
⎍	0	1	0
⎍	0	1	1
⎍	1	0	0
⎍	1	0	1
⎍	1	1	0
⎍	1	1	1

Figure 14.12 Ripple counter

This 3-bit ripple counter can easily be configured as a divide-by-8 mechanism, simply by adding an AND gate. To divide the input clock rate by 8, one output pulse should be generated for every eight clock pulses. If one were to output a pulse every time a binary 111 combination occurs, a simple AND gate would suffice to generate the required condition. This solution is shown in Figure 14.13. Note that the square wave is also included as an input to the AND gate; this ensures that the output is only as wide as the input signal. This application of ripple counters is further illustrated in the following example.

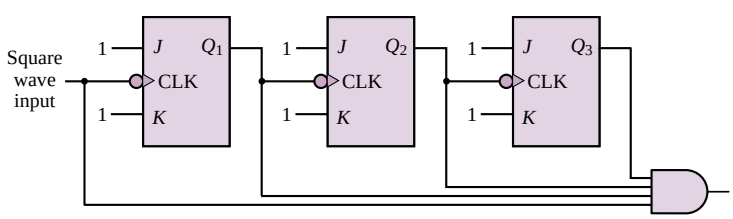


Figure 14.13 Divide-by-8 circuit

EXAMPLE 14.4 Divider Circuit

Problem

Draw the timing diagram for the clock input, Q_0 and Q_1, for the binary ripple counter of Figure 14.14.

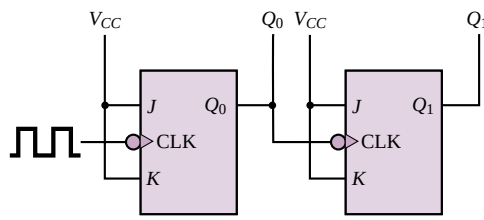


Figure 14.14

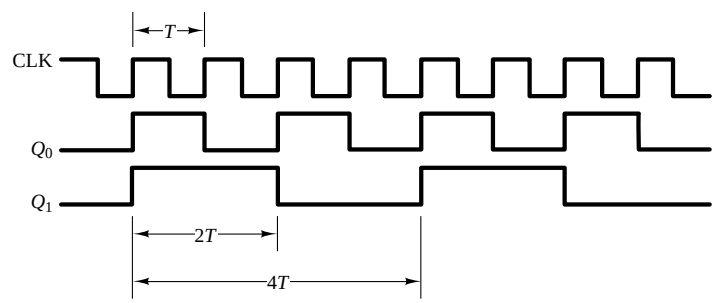
Solution

Known Quantities: *JK* flip-flop truth table (Figure 14.8).

Find: Output of each flip-flop, Q, as a function of the input clock transitions.

Assumptions: Assume negative-edge–triggered devices.

Analysis: Following the timing diagram of Figure 14.12, we see that Q_0 switches at half the frequency of the clock input, and that Q_1 switches at half the frequency of Q_0. Hence the timing diagram shown below.



A slightly more complex version of the binary counter is the so-called **synchronous counter,** in which the input clock drives all of the flip-flops simultaneously. Figure 14.15 depicts a three-bit synchronous counter. In this figure, we have chosen to represent each flip-flop as a T flip-flop. The clocks to all the flip-flops are incremented simultaneously. The reader should verify that Q_0 toggles to 1 first and then Q_1 toggles to 1, and that the AND gate ensures that Q_2 will toggle only after Q_0 and Q_1 have both reached the 1 state ($Q_0 \cdot Q_1 = 1$).

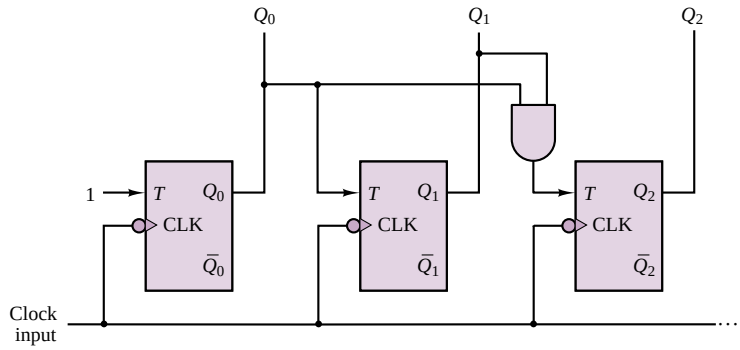


Figure 14.15 Three-bit synchronous counter

Other common counters are the **ring counter,** illustrated in Example 14.5, and the **up-down counter,** which has an additional select input that determines whether the counter counts up or down. Data sheets for various counters may be found in the accompanying CD-ROM.

EXAMPLE 14.5 Ring Counter

Problem

Draw the timing diagram for the ring counter of Figure 14.16.

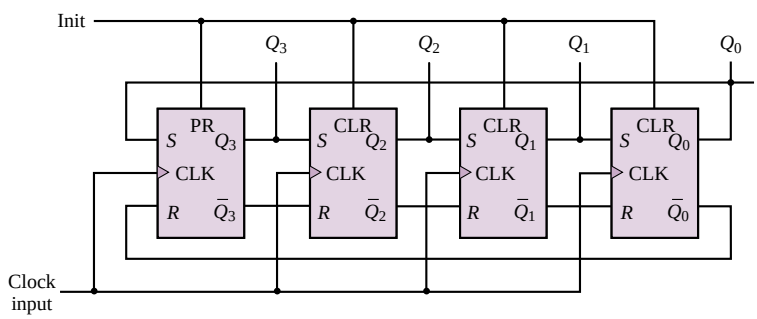


Figure 14.16 Ring counter

Solution

Known Quantities: *JK* flip-flop truth table (Figure 14.8).

Find: Output of each flip-flop, Q, as a function of the input clock transitions.

Assumptions: Assume that prior to applying the clock input the Init line sees a positive transition (this initializes the counter by setting the state of the first flip-flop to 1 through a PR (preset) input, and all other states to zero through a CLR (clear) input).

Analysis: With the initial state of $Q_3 = 0$, a clock transition will *set* $Q_3 = 1$. The clock also causes the other three flip-flops to see a *reset* input of 1, since

$Q_3 = Q_2 = Q_1 = Q_0 = 0$ at the time of the first clock pulse. Thus, Q_2, Q_1 and Q_0 remain in the zero state. At the second clock pulse, since Q_3 is now 1, the second flip-flop will see a *set* input of one, and its output will become $Q_2 = 1$. Q_1 and Q_0 remain in the zero state, and Q_3 is reset to 0. The pattern continues, causing the 1-state to ripple from left to right and back again. This rightward rotation gives the counter its name. The transition table is shown below.

CLK	Q_3	Q_2	Q_1	Q_0
↑	1	0	0	0
↑	0	0	1	0
↑	0	1	0	0
↑	0	0	0	1
↑	1	0	0	0
↑	0	1	0	0
↑	0	0	1	0

Comments: The shifting function implemented by the ring counter is used in the shift registers discussed in the following subsection.

Focus on Computer-Aided Solutions: A ring counter simulation generated by Electronics Workbench™ may be found in the accompanying CD-ROM.

Digital Measurement of Angular Position and Velocity

Another type of angular position encoder, besides the angular encoder discussed in Chapter 13 in "Focus on Measurements: Position Encoders," is the slotted encoder shown in Figure 14.17. This encoder can be used in conjunction with a pair of counters and a high-frequency clock to determine the speed of rotation of the slotted wheel. As shown in Figure 14.18, a clock of known frequency is connected to a counter while another counter records the number of slot pulses detected by an optical slot detector as the wheel rotates. Dividing the counter values, one could obtain the speed of the rotating wheel in radians per second. For example, assume a clocking frequency of 1.2 kHz. If both counters are started at zero and at some instant the timer counter reads 2,850 and the encoder counter reads 3,050, then the speed of the rotating encoder is found to be:

$$1{,}200\frac{\text{cycles}}{\text{second}} \cdot \frac{2{,}850 \text{ slots}}{3{,}050 \text{ cycles}} = 1{,}121.3\frac{\text{slots}}{\text{second}}$$

and

$$1{,}121.3 \text{ slots per second} \times 1^\circ \text{ per slot} \times 2\pi/360 \text{ rad/degree}$$
$$= 19.6 \text{ rad/s}$$

If this encoder is connected to a rotating shaft, it is possible to measure the angular position and velocity of the shaft. Such shaft encoders are used in **measuring the speed of rotation of electric motors, machine tools, engines,** and other rotating machinery.

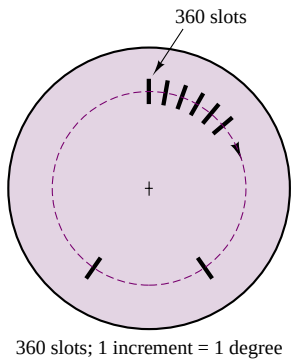


Figure 14.17

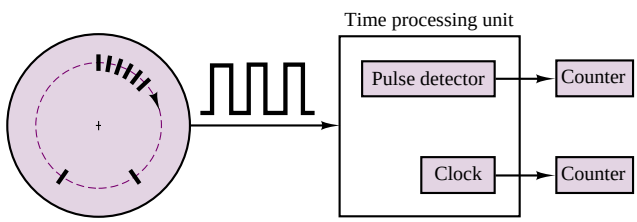


Figure 14.18 Calculating the speed of rotation of the slotted wheel

A typical application of the slotted encoder is to compute the ignition and injection timing in an automotive engine. In an automotive engine, information related to speed is obtained from the camshaft and the flywheel, which have known reference points. The reference points determine the timing for the ignition firing points and fuel injection pulses, and are identified by special slot patterns on the camshaft and crankshaft. Two methods are used to detect the special slots (reference points): *period measurement with additional transition detection (PMA),* and *period measurement with missing transition detection (PMM)*. In the PMA method, an additional slot (reference point) determines a known reference position on the crankshaft or camshaft. In the PMM method, the reference position is determined by the absence of a slot. Figure 14.19 illustrates a typical PMA pulse sequence, showing the presence of an additional pulse. The additional slot may be used to determine the timing for the ignition pulses relative to a known position of the crankshaft. Figure 14.20 depicts a typical PMM pulse sequence. Because the period of the pulses is known, the additional slot or the missing slot can be easily detected and used as a reference position. How would you implement these pulse sequences using ring counters?

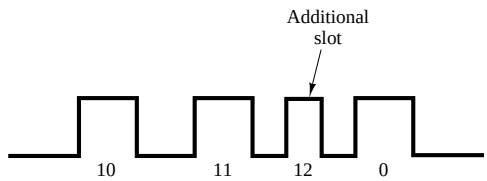


Figure 14.19 PMA pulse sequence

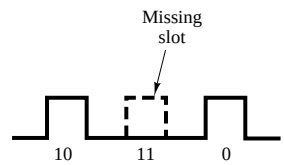


Figure 14.20 PMM pulse sequence

Registers

A register consists of a cascade of flip-flops that can store binary data, one bit in each flip-flop. The simplest type of register is the parallel input–parallel output register shown in Figure 14.21. In this register, the "load" input pulse, which acts on all clocks simultaneously, causes the parallel inputs $b_0b_1b_2b_3$ to be transferred to the respective flip-flops. The D flip-flop employed in this register allows the transfer from b_n to Q_n to occur very directly. Thus, D flip-flops are very commonly used in this type of application. The binary word $b_3b_2b_1b_0$ is now "stored," each bit being represented by the state of a flip-flop. Until the "load" input is applied again and a new word appears at the parallel inputs, the register will preserve the stored word.

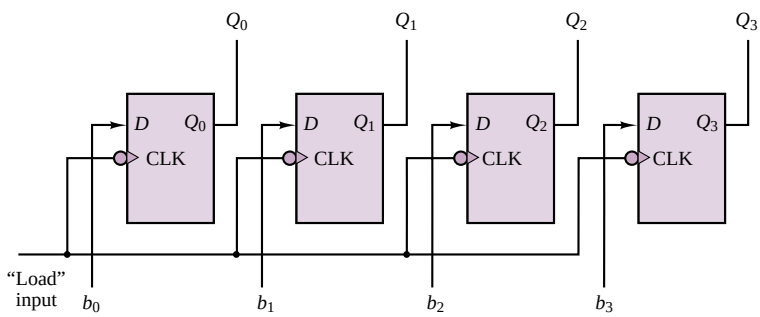


Figure 14.21 Four-bit parallel register

The construction of the parallel register presumes that the N-bit word to be stored is available in parallel form. However, it is often true that a binary word will arrive in serial form, that is, one bit at a time. A register that can accommodate this type of logic signal is called a **shift register.** Figure 14.22 illustrates how the same basic structure of the parallel register applies to the shift register, except that the input is now applied to the first flip-flop and shifted along at each clock pulse. Note that this type of register provides both a serial and a parallel output.

Data sheets for some common registers are included in the accompanying CD-ROM.

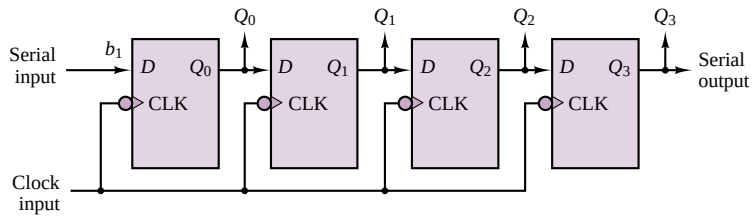


Figure 14.22 Four-bit shift register

Seven-Segment Display

A **seven-segment display** is a very convenient device for displaying digital data. The display is shown in Figure 14.23. Operation of a seven-segment display requires a decoder circuit to light the proper combinations of segments corresponding to the desired decimal digit.

This display, with the appropriate decoder driver, is capable of displaying values ranging from 0 to 9.

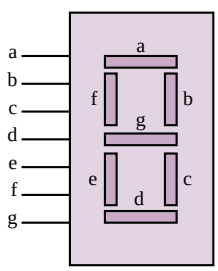


Figure 14.23 Seven-segment display

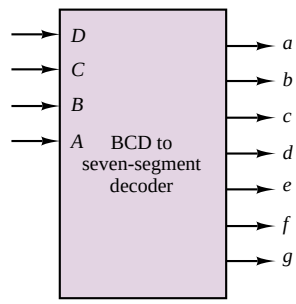
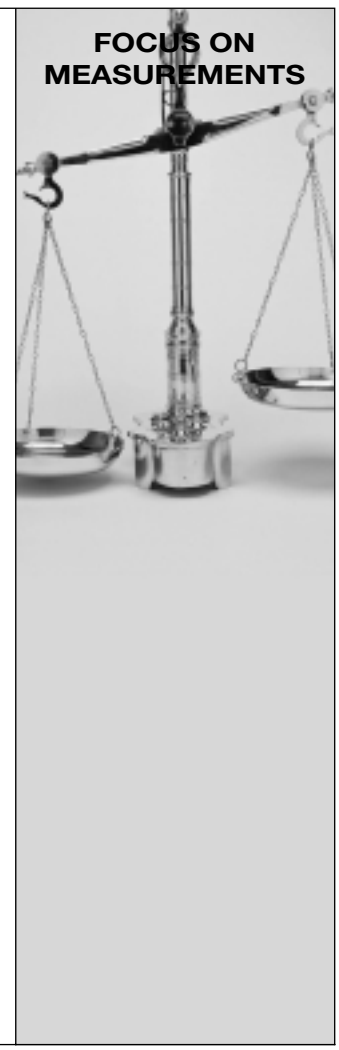


Figure 14.24

A typical BCD to seven-segment decoder function block is shown in Figure 14.24, where the lowercase letters correspond to the segments shown in Figure 14.23. The decoder features four data inputs (A, B, C, D), which are used to light the appropriate segment(s). The outputs of the decoder are connected to the seven-segment display. The decoder will light up the appropriate segments corresponding to the incoming value. A BCD to seven-segment decoder function is similar to the 2-to-4 decoder function described in Chapter 13 and shown in Figure 13.61. Data sheets for seven-segment display drivers may be found in the accompanying CD-ROM.



Check Your Understanding

14.1 The circuit shown in Figure 14.25 also serves as an *RS* flip-flop and requires only two NOR gates. Analyze the circuit to prove that it operates as an *RS* flip-flop. [*Hint:* Use a truth table with two variables, S and R.]