electrical characteristics over recommended operating free-air temperature range (unless otherwise noted)

PARAMETER	TEST CONDITIONS		V_{CC}	T_A = 25°C			SN54HC04		SN74HC04		UNIT
				MIN	TYP	MAX	MIN	MAX	MIN	MAX	
V_{OH}	$V_I = V_{IH}$ or V_{IL}	I_{OH} = –20 µA	2 V	1.9	1.998		1.9		1.9		V
			4.5 V	4.4	4.499		4.4		4.4		
			6 V	5.9	5.999		5.9		5.9		
		I_{OH} = –4 mA	4.5 V	3.98	4.3		3.7		3.84		
		I_{OH} = –5.2 mA	6 V	5.48	5.8		5.2		5.34		
V_{OL}	$V_I = V_{IH}$ or V_{IL}	I_{OL} = 20 µA	2 V		0.002	0.1		0.1		0.1	V
			4.5 V		0.001	0.1		0.1		0.1	
			6 V		0.001	0.1		0.1		0.1	
		I_{OL} = 4 mA	4.5 V		0.17	0.26		0.4		0.33	
		I_{OL} = 5.2 mA	6 V		0.15	0.26		0.4		0.33	
I_I	$V_I = V_{CC}$ or 0		6 V		±0.1	±100		±1000		±1000	nA
I_{CC}	$V_I = V_{CC}$ or 0, I_O = 0		6 V			2		40		20	µA
C_i			2 V to 6 V		3	10		10		10	pF

switching characteristics over recommended operating free-air temperature range, CL = 50 pF (unless otherwise noted) (see Figure 1)

PARAMETER	FROM (INPUT)	TO (OUTPUT)	V_{CC}	T_A = 25°C			SN54HC04		SN74HC04		UNIT
				MIN	TYP	MAX	MIN	MAX	MIN	MAX	
t_{pd}	A	Y	2 V		45	95		145		120	ns
			4.5 V		9	19		29		24	
			6 V		8	16		25		20	
t_t		Y	2 V		38	75		110		95	ns
			4.5 V		8	15		22		19	
			6 V		6	13		19		16	

operating characteristics, T_A = 25°C

PARAMETER		TEST CONDITIONS	TYP	UNIT
C_{pd}	Power dissipation capacitance per inverter	No load	20	pF

Figure A.9 7404 datasheet page 3

A.6 LOGIC FAMILIES

The 74xx-series logic chips have been manufactured using many different technologies, called *logic families*, that offer different speed, power, and logic level trade-offs. Other chips are usually designed to be compatible with some of these logic families. The original chips, such as the 7404, were built using bipolar transistors in a technology called *Transistor-Transistor Logic* (*TTL*). Newer technologies add one or more letters after the 74 to indicate the logic family, such as 74LS04, 74HC04, or 74AHCT04. Table A.2 summarizes the most common 5-V logic families.

Advances in bipolar circuits and process technology led to the *Schottky* (*S*) and *Low-Power Schottky* (*LS*) families. Both are faster than TTL. Schottky draws more power, whereas Low-power Schottky draws less. *Advanced Schottky* (*AS*) and *Advanced Low-Power Schottky* (*ALS*) have improved speed and power compared to S and LS. *Fast* (*F*) logic is faster and draws less power than AS. All of these families provide more current for LOW outputs than for HIGH outputs and hence have asymmetric logic levels. They conform to the "TTL" logic levels: $V_{IH} = 2$ V, $V_{IL} = 0.8$ V, $V_{OH} > 2.4$ V, and $V_{OL} < 0.5$ V.

Table A.2 Typical specifications for 5-V logic families

	Bipolar / TTL						CMOS		CMOS / TTL Compatible	
Characteristic	TTL	S	LS	AS	ALS	F	HC	AHC	HCT	AHCT
t_{pd} (ns)	22	9	12	7.5	10	6	21	7.5	30	7.7
V_{IH} (V)	2	2	2	2	2	2	3.15	3.15	2	2
V_{IL} (V)	0.8	0.8	0.8	0.8	0.8	0.8	1.35	1.35	0.8	0.8
V_{OH} (V)	2.4	2.7	2.7	2.5	2.5	2.5	3.84	3.8	3.84	3.8
V_{OL} (V)	0.4	0.5	0.5	0.5	0.5	0.5	0.33	0.44	0.33	0.44
I_{OH} (mA)	0.4	1	0.4	2	0.4	1	4	8	4	8
I_{OL} (mA)	16	20	8	20	8	20	4	8	4	8
I_{IL} (mA)	1.6	2	0.4	0.5	0.1	0.6	0.001	0.001	0.001	0.001
I_{IH} (mA)	0.04	0.05	0.02	0.02	0.02	0.02	0.001	0.001	0.001	0.001
I_{DD} (mA)	33	54	6.6	26	4.2	15	0.02	0.02	0.02	0.02
C_{pd} (pF)			n/a				20	12	20	14
cost* (US $)	obsolete	0.57	0.29	0.53	0.33	0.20	0.15	0.15	0.15	0.15

* Per unit in quantities of 1000 for the 7408 from Texas Instruments in 2006

As CMOS circuits matured in the 1980s and 1990s, they became popular because they draw very little power supply or input current. The *High Speed CMOS* (*HC*) and *Advanced High Speed CMOS* (*AHC*) families draw almost no static power. They also deliver the same current for HIGH and LOW outputs. They conform to the "CMOS" logic levels: $V_{IH} = 3.15$ V, $V_{IL} = 1.35$ V, $V_{OH} > 3.8$ V, and $V_{OL} < 0.44$ V. Unfortunately, these levels are incompatible with TTL circuits, because a TTL HIGH output of 2.4 V may not be recognized as a legal CMOS HIGH input. This motivates the use of *High Speed TTL-compatible CMOS* (*HCT*) and *Advanced High Speed TTL-compatible CMOS* (*AHCT*), which accept TTL input logic levels and generate valid CMOS output logic levels. These families are slightly slower than their pure CMOS counterparts. All CMOS chips are sensitive to *electrostatic discharge* (*ESD*) caused by static electricity. Ground yourself by touching a large metal object before handling CMOS chips, lest you zap them.

The 74xx-series logic is inexpensive. The newer logic families are often cheaper than the obsolete ones. The LS family is widely available and robust and is a popular choice for laboratory or hobby projects that have no special performance requirements.

The 5-V standard collapsed in the mid-1990s, when transistors became too small to withstand the voltage. Moreover, lower voltage offers lower power consumption. Now 3.3, 2.5, 1.8, 1.2, and even lower voltages are commonly used. The plethora of voltages raises challenges in communicating between chips with different power supplies. Table A.3 lists some of the low-voltage logic families. Not all 74xx parts are available in all of these logic families.

All of the low-voltage logic families use CMOS transistors, the workhorse of modern integrated circuits. They operate over a wide range of V_{DD}, but the speed degrades at lower voltage. *Low-Voltage CMOS* (*LVC*) logic and *Advanced Low-Voltage CMOS* (*ALVC*) logic are commonly used at 3.3, 2.5, or 1.8 V. LVC withstands inputs up to 5.5 V, so it can receive inputs from 5-V CMOS or TTL circuits. *Advanced Ultra-Low-Voltage CMOS (AUC)* is commonly used at 2.5, 1.8, or 1.2 V and is exceptionally fast. Both ALVC and AUC withstand inputs up to 3.6 V, so they can receive inputs from 3.3-V circuits.

FPGAs often offer separate voltage supplies for the internal logic, called the *core*, and for the input/output (I/O) pins. As FPGAs have advanced, the core voltage has dropped from 5 to 3.3, 2.5, 1.8, and 1.2 V to save power and avoid damaging the very small transistors. FPGAs have configurable I/Os that can operate at many different voltages, so as to be compatible with the rest of the system.

Table A.3 Typical specifications for low-voltage logic families

	LVC			ALVC			AUC		
V_{dd} (V)	3.3	2.5	1.8	3.3	2.5	1.8	2.5	1.8	1.2
t_{pd} (ns)	4.1	6.9	9.8	2.8	3	?[1]	1.8	2.3	3.4
V_{IH} (V)	2	1.7	1.17	2	1.7	1.17	1.7	1.17	0.78
V_{IL} (V)	0.8	0.7	0.63	0.8	0.7	0.63	0.7	0.63	0.42
V_{OH} (V)	2.2	1.7	1.2	2	1.7	1.2	1.8	1.2	0.8
V_{OL} (V)	0.55	0.7	0.45	0.55	0.7	0.45	0.6	0.45	0.3
I_O (mA)	24	8	4	24	12	12	9	8	3
I_I (mA)		0.02			0.005			0.005	
I_{DD} (mA)		0.01			0.01			0.01	
C_{pd} (pF)	10	9.8	7	27.5	23	?*	17	14	14
cost (US $)		0.17			0.20			not available	

* Delay and capacitance not available at the time of writing

A.7 PACKAGING AND ASSEMBLY

Integrated circuits are typically placed in *packages* made of plastic or ceramic. The packages serve a number of functions, including connecting the tiny metal I/O pads of the chip to larger pins in the package for ease of connection, protecting the chip from physical damage, and spreading the heat generated by the chip over a larger area to help with cooling. The packages are placed on a breadboard or printed circuit board and wired together to assemble the system.

Packages

Figure A.10 shows a variety of integrated circuit packages. Packages can be generally categorized as *through-hole* or *surface mount* (*SMT*). Through-hole packages, as their name implies, have pins that can be inserted through holes in a printed circuit board or into a socket. *Dual inline packages* (*DIPs*) have two rows of pins with 0.1-inch spacing between pins. *Pin grid arrays* (*PGAs*) support more pins in a smaller package by placing the pins under the package. SMT packages are soldered directly to the surface of a printed circuit board without using holes. Pins on SMT parts are called *leads*. The *thin small outline package* (*TSOP*) has two rows of closely spaced leads (typically 0.02-inch spacing). *Plastic leaded chip carriers* (*PLCCs*) have J-shaped leads on all four

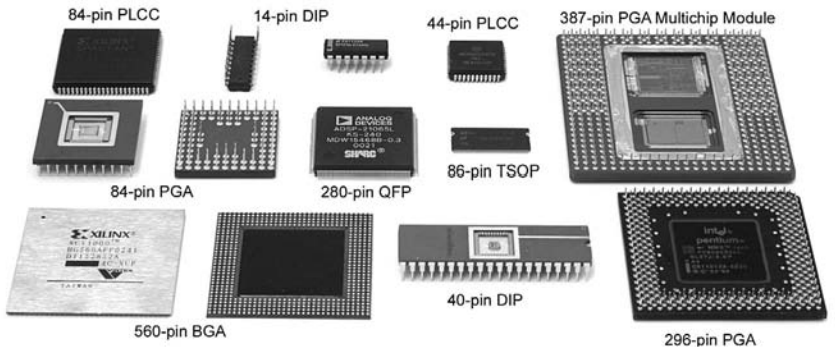


Figure A.10 Integrated circuit packages

sides, with 0.05-inch spacing. They can be soldered directly to a board or placed in special sockets. *Quad flat packs* (*QFPs*) accommodate a large number of pins using closely spaced legs on all four sides. *Ball grid arrays* (*BGAs*) eliminate the legs altogether. Instead, they have hundreds of tiny solder balls on the underside of the package. They are carefully placed over matching pads on a printed circuit board, then heated so that the solder melts and joins the package to the underlying board.

Breadboards

DIPs are easy to use for prototyping, because they can be placed in a *breadboard*. A breadboard is a plastic board containing rows of sockets, as shown in Figure A.11. All five holes in a row are connected together. Each pin of the package is placed in a hole in a separate row. Wires can be placed in adjacent holes in the same row to make connections to the pin. Breadboards often provide separate columns of connected holes running the height of the board to distribute power and ground.

Figure A.11 shows a breadboard containing a majority gate built with a 74LS08 AND chip and a 74LS32 OR chip. The schematic of the circuit is shown in Figure A.12. Each gate in the schematic is labeled with the chip (08 or 32) and the pin numbers of the inputs and outputs (see Figure A.1). Observe that the same connections are made on the breadboard. The inputs are connected to pins 1, 2, and 5 of the 08 chip, and the output is measured at pin 6 of the 32 chip. Power and ground are connected to pins 14 and 7, respectively, of each chip, from the vertical power and ground columns that are attached to the banana plug receptacles, Vb and Va. Labeling the schematic in this way and checking off connections as they are made is a good way to reduce the number of mistakes made during breadboarding.

Unfortunately, it is easy to accidentally plug a wire in the wrong hole or have a wire fall out, so breadboarding requires a great deal of care (and usually some debugging in the laboratory). Breadboards are suited only to prototyping, not production.

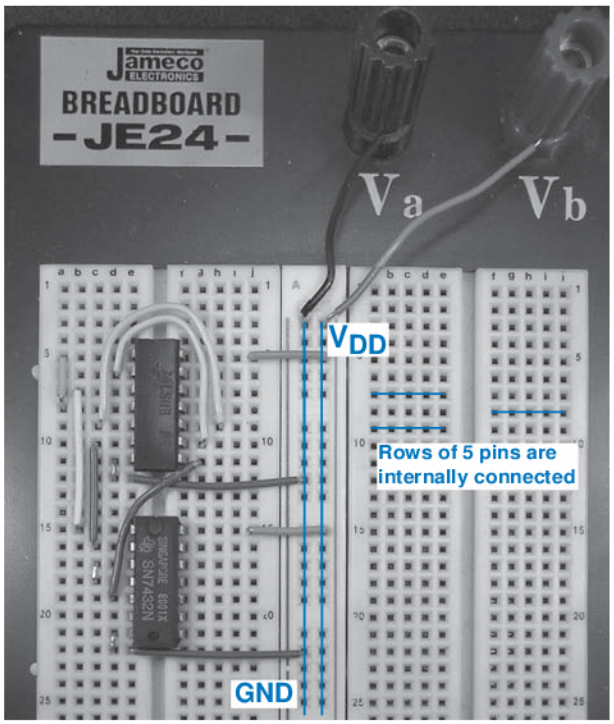


Figure A.11 **Majority circuit on breadboard**

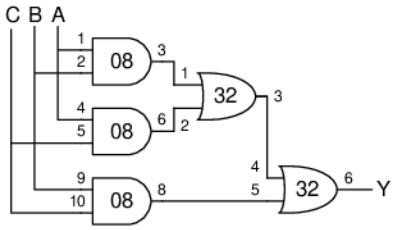


Figure A.12 **Majority gate schematic with chips and pins identified**

Printed Circuit Boards

Instead of breadboarding, chip packages may be soldered to a *printed circuit board* (*PCB*). The PCB is formed of alternating layers of conducting copper and insulating epoxy. The copper is etched to form wires called *traces*. Holes called *vias* are drilled through the board and plated with metal to connect between layers. PCBs are usually designed with *computer-aided design* (*CAD*) tools. You can etch and drill your own simple boards in the laboratory, or you can send the board design to a specialized factory for inexpensive mass production. Factories have turnaround times of days (or weeks, for cheap mass production runs) and typically charge a few hundred dollars in setup fees and a few dollars per board for moderately complex boards built in large quantities.

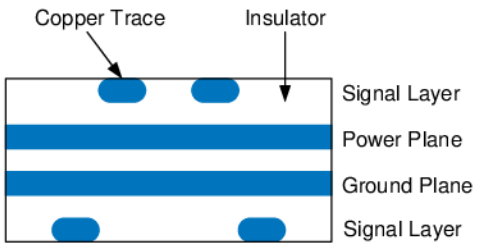


Figure A.13 Printed circuit board cross-section

PCB traces are normally made of copper because of its low resistance. The traces are embedded in an insulating material, usually a green, fire-resistant plastic called FR4. A PCB also typically has copper power and ground layers, called *planes*, between signal layers. Figure A.13 shows a cross-section of a PCB. The signal layers are on the top and bottom, and the power and ground planes are embedded in the center of the board. The power and ground planes have low resistance, so they distribute stable power to components on the board. They also make the capacitance and inductance of the traces uniform and predictable.

Figure A.14 shows a PCB for a 1970s vintage Apple II+ computer. At the top is a Motorola 6502 microprocessor. Beneath are six 16-Kb ROM chips forming 12 KB of ROM containing the operating system. Three rows of eight 16-Kb DRAM chips provide 48 KB of RAM. On the right are several rows of 74xx-series logic for memory address decoding and other functions. The lines between chips are traces that wire the chips together. The dots at the ends of some of the traces are vias filled with metal.

Putting It All Together

Most modern chips with large numbers of inputs and outputs use SMT packages, especially QFPs and BGAs. These packages require a printed circuit board rather than a breadboard. Working with BGAs is especially challenging because they require specialized assembly equipment. Moreover, the balls cannot be probed with a voltmeter or oscilloscope during debugging in the laboratory, because they are hidden under the package.

In summary, the designer needs to consider packaging early on to determine whether a breadboard can be used during prototyping and whether BGA parts will be required. Professional engineers rarely use breadboards when they are confident of connecting chips together correctly without experimentation.

A.8 TRANSMISSION LINES

We have assumed so far that wires are *equipotential* connections that have a single voltage along their entire length. Signals actually propagate along wires at the speed of light in the form of electromagnetic waves. If the wires are short enough or the signals change slowly, the equipotential assumption

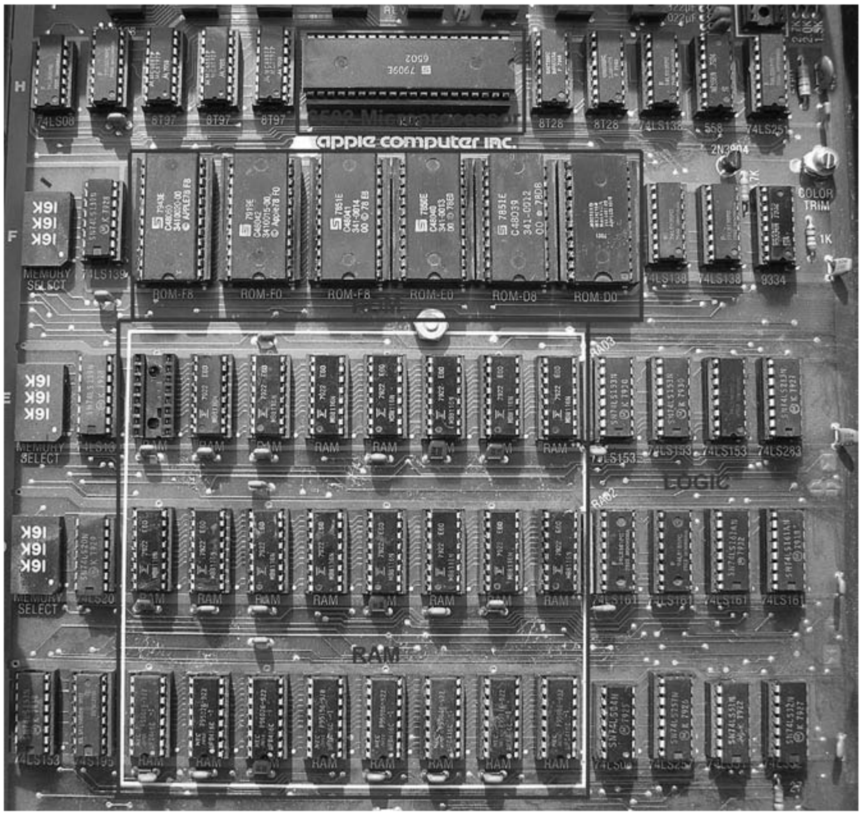


Figure A.14 **Apple II+ circuit board**

is good enough. When the wire is long or the signal is very fast, the *transmission time* along the wire becomes important to accurately determine the circuit delay. We must model such wires as *transmission lines*, in which a wave of voltage and current propagates at the speed of light. When the wave reaches the end of the line, it may reflect back along the line. The reflection may cause noise and odd behaviors unless steps are taken to limit it. Hence, the digital designer must consider transmission line behavior to accurately account for the delay and noise effects in long wires.

Electromagnetic waves travel at the speed of light in a given medium, which is fast but not instantaneous. The speed of light depends on the *permittivity*, ε, and *permeability*, μ, of the medium[2]: $v = \frac{1}{\sqrt{\mu\varepsilon}} = \frac{1}{\sqrt{LC}}$.

[2] The capacitance, C, and inductance, L, of a wire are related to the permittivity and permeability of the physical medium in which the wire is located.

The speed of light in free space is $v = c = 3 \times 10^8$ m/s. Signals in a PCB travel at about half this speed, because the FR4 insulator has four times the permittivity of air. Thus, PCB signals travel at about 1.5×10^8 m/s, or 15 cm/ns. The time delay for a signal to travel along a transmission line of length l is

$$t_d = l/v \tag{A.4}$$

The *characteristic impedance* of a transmission line, Z_0 (pronounced "Z-naught"), is the ratio of voltage to current in a wave traveling along the line: $Z_0 = V/I$. It is *not* the resistance of the wire (a good transmission line in a digital system typically has negligible resistance). Z_0 depends on the inductance and capacitance of the line (see the derivation in Section A.8.7) and typically has a value of 50 to 75 Ω.

$$Z_0 = \sqrt{\frac{L}{C}} \tag{A.5}$$

Figure A.15 shows the symbol for a transmission line. The symbol resembles a *coaxial cable* with an inner signal conductor and an outer grounded conductor like that used in television cable wiring.

The key to understanding the behavior of transmission lines is to visualize the wave of voltage propagating along the line at the speed of light. When the wave reaches the end of the line, it may be absorbed or reflected, depending on the termination or load at the end. Reflections travel back along the line, adding to the voltage already on the line. Terminations are classified as matched, open, short, or mismatched. The following subsections explore how a wave propagates along the line and what happens to the wave when it reaches the termination.

A.8.1 Matched Termination

Figure A.16 shows a transmission line of length l with a *matched termination*, which means that the load impedance, Z_L, is equal to the characteristic impedance, Z_0. The transmission line has a characteristic impedance of 50 Ω. One end of the line is connected to a voltage

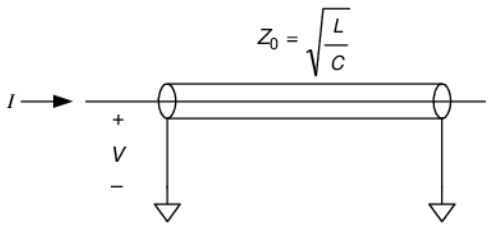


Figure A.15 Transmission line symbol

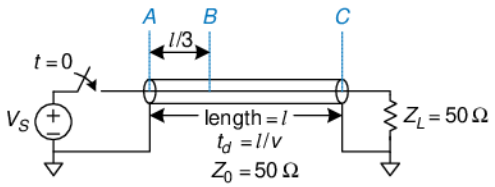


Figure A.16 Transmission line with matched termination

source through a switch that closes at time $t = 0$. The other end is connected to the 50 Ω matched load. This section analyzes the voltages and currents at points A, B, and C—at the beginning of the line, one-third of the length along the line, and at the end of the line, respectively.

Figure A.17 shows the voltages at points A, B, and C over time. Initially, there is no voltage or current flowing in the transmission line, because the switch is open. At time $t = 0$, the switch closes, and the voltage source launches a wave with voltage $V = V_S$ along the line. Because the characteristic impedance is Z_0, the wave has current $I = V_S/Z_0$. The voltage reaches the beginning of the line (point A) immediately, as shown in Figure A.17(a). The wave propagates along the line at the speed of light. At time $t_d/3$, the wave reaches point B. The voltage at this point abruptly rises from 0 to V_S, as shown in Figure A.17(b). At time t_d, the *incident wave* reaches point C at the end of the line, and the voltage rises there too. All of the current, I, flows into the resistor, Z_L, producing a voltage across the resistor of $Z_L I = Z_L(V_S/Z_0) = V_S$ because $Z_L = Z_0$. This voltage is consistent with the wave flowing along the transmission line. Thus, the wave is *absorbed* by the load impedance, and the transmission line reaches its *steady state*.

In steady state, the transmission line behaves like an ideal equipotential wire because it is, after all, just a wire. The voltage at all points along the line must be identical. Figure A.18 shows the steady-state equivalent model of the circuit in Figure A.16. The voltage is V_S everywhere along the wire.

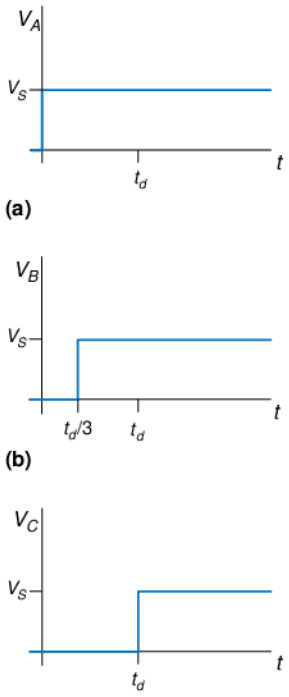


Figure A.17 Voltage waveforms for Figure A.16 at points *A*, *B*, and *C*

Example A.2 TRANSMISSION LINE WITH MATCHED SOURCE AND LOAD TERMINATIONS

Figure A.19 shows a transmission line with matched source and load impedances Z_S and Z_L. Plot the voltage at nodes A, B, and C versus time. When does the system reach steady-state, and what is the equivalent circuit at steady-state?

SOLUTION: When the voltage source has a source impedance, Z_S, in series with the transmission line, part of the voltage drops across Z_S, and the remainder

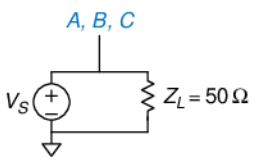


Figure A.18 Equivalent circuit of Figure A.16 at steady state

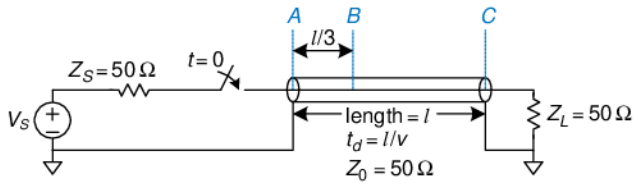


Figure A.19 Transmission line with matched source and load impedances

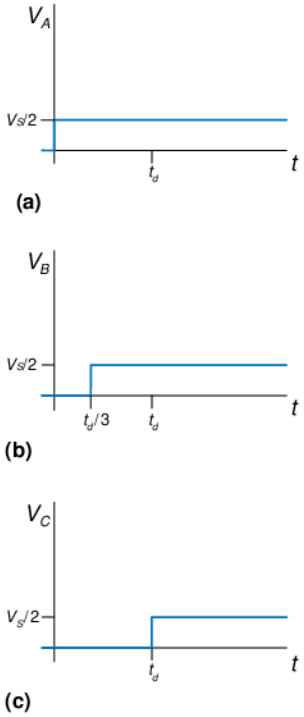


Figure A.20 Voltage waveforms for Figure A.19 at points *A*, *B*, and *C*

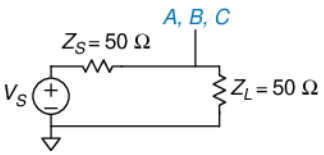


Figure A.21 Equivalent circuit of Figure A.19 at steady state

propagates down the transmission line. At first, the transmission line behaves as an impedance Z_0, because the load at the end of the line cannot possibly influence the behavior of the line until a speed of light delay has elapsed. Hence, by the *voltage divider equation*, the incident voltage flowing down the line is

$$V = V_S\left(\frac{Z_0}{Z_0 + Z_S}\right) = \frac{V_S}{2} \tag{A.6}$$

Thus, at $t = 0$, a wave of voltage, $V = \frac{V_S}{2}$, is sent down the line from point *A*. Again, the signal reaches point *B* at time $t_d/3$ and point *C* at t_d, as shown in Figure A.20. All of the current is absorbed by the load impedance Z_L, so the circuit enters steady-state at $t = t_d$. In steady-state, the entire line is at $V_S/2$, just as the steady-state equivalent circuit in Figure A.21 would predict.

A.8.2 Open Termination

When the load impedance is not equal to Z_0, the termination cannot absorb all of the current, and some of the wave must be reflected. Figure A.22 shows a transmission line with an open load termination. No current can flow through an open termination, so the current at point C must always be 0.

The voltage on the line is initially zero. At $t = 0$, the switch closes and a wave of voltage, $V = V_S\frac{Z_0}{Z_0 + Z_S} = \frac{V_S}{2}$, begins propagating down the line. Notice that this initial wave is the same as that of Example A.2 and is independent of the termination, because the load at the end of the line cannot influence the behavior at the beginning until at least t_d has elapsed. This wave reaches point *B* at $t_d/3$ and point C at t_d as shown in Figure A.23.

When the incident wave reaches point C, it cannot continue forward because the wire is open. It must instead reflect back toward the source. The reflected wave also has voltage $V = \frac{V_S}{2}$, because the open termination reflects the entire wave.

The voltage at any point is the sum of the incident and reflected waves. At time $t = t_d$, the voltage at point C is $V = \frac{V_S}{2} + \frac{V_S}{2} = V_S$. The reflected wave reaches point *B* at $5t_d/3$ and point *A* at $2t_d$. When it reaches point *A*, the wave is absorbed by the source termination

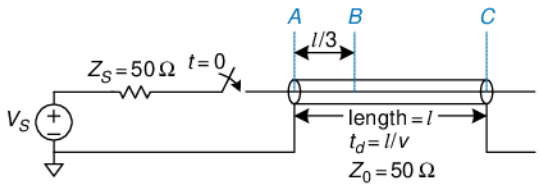


Figure A.22 Transmission line with open load termination

impedance that matches the characteristic impedance of the line. Thus, the system reaches steady state at time $t = 2t_d$, and the transmission line becomes equivalent to an equipotential wire with voltage V_S and current $I = 0$.

A.8.3 Short Termination

Figure A.24 shows a transmission line terminated with a short circuit to ground. Thus, the voltage at point C must always be 0.

As in the previous examples, the voltages on the line are initially 0. When the switch closes, a wave of voltage, $V = \frac{V_S}{2}$, begins propagating down the line (Figure A.25). When it reaches the end of the line, it must reflect with opposite polarity. The reflected wave, with voltage $V = \frac{-V_S}{2}$, adds to the incident wave, ensuring that the voltage at point C remains 0. The reflected wave reaches the source at time $t = 2t_d$ and is absorbed by the source impedance. At this point, the system reaches steady state, and the transmission line is equivalent to an equipotential wire with voltage $V = 0$.

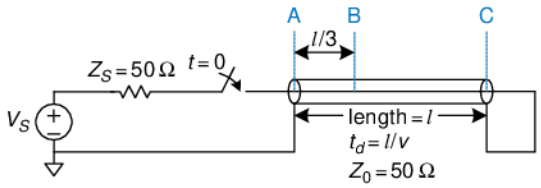
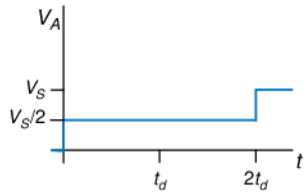
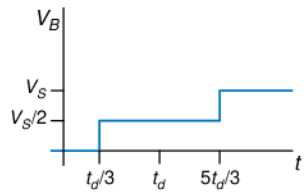


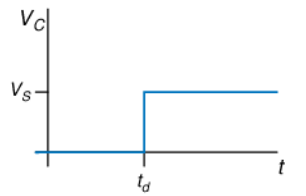
Figure A.24 Transmission line with short termination



(a)



(b)



(c)

Figure A.23 Voltage waveforms for Figure A.22 at points *A*, *B*, and *C*

A.8.4 Mismatched Termination

The termination impedance is said to be *mismatched* when it does not equal the characteristic impedance of the line. In general, when an incident wave reaches a mismatched termination, part of the wave is absorbed and part is reflected. The reflection coefficient, k_r, indicates the fraction of the incident wave (V_i) that is reflected: $V_r = k_r V_i$.

Section A.8.8 derives the reflection coefficient using conservation of current arguments. It shows that, when an incident wave flowing along a

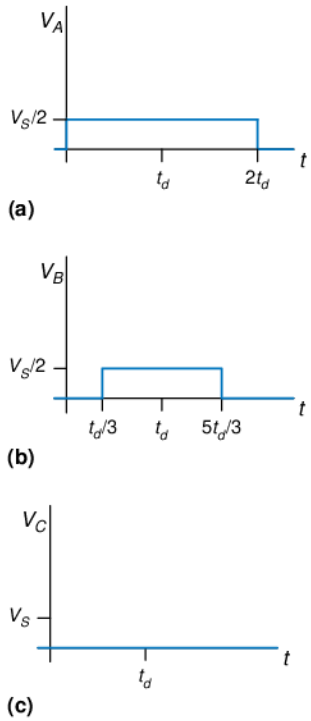


Figure A.25 Voltage waveforms for Figure A.24 at points *A*, *B*, and *C*

transmission line of characteristic impedance Z_0 reaches a termination impedance, Z_T, at the end of the line, the reflection coefficient is

$$k_r = \frac{Z_T - Z_0}{Z_T + Z_0} \tag{A.7}$$

Note a few special cases. If the termination is an open circuit ($Z_T = \infty$), $k_r = 1$, because the incident wave is entirely reflected (so the current out the end of the line remains zero). If the termination is a short circuit ($Z_T = 0$), $k_r = -1$, because the incident wave is reflected with negative polarity (so the voltage at the end of the line remains zero). If the termination is a matched load ($Z_T = Z_0$), $k_r = 0$, because the incident wave is absorbed.

Figure A.26 illustrates reflections in a transmission line with a *mismatched load termination* of 75 Ω. $Z_T = Z_L = 75\ \Omega$, and $Z_0 = 50\ \Omega$, so $k_r = 1/5$. As in previous examples, the voltage on the line is initially 0. When the switch closes, a wave of voltage, $V = \frac{V_S}{2}$, propagates down the line, reaching the end at $t = t_d$. When the incident wave reaches the termination at the end of the line, one fifth of the wave is reflected, and the remaining four fifths flows into the load impedance. Thus, the reflected wave has a voltage $V = \frac{V_S}{2} \times \frac{1}{5} = \frac{V_S}{10}$. The total voltage at point C is the sum of the incoming and reflected voltages, $V_C = \frac{V_S}{2} + \frac{V_S}{10} = \frac{3V_S}{5}$. At $t = 2t_d$, the reflected wave reaches point A, where it is absorbed by the matched 50 Ω termination, Z_S. Figure A.27 plots the voltages and currents along the line. Again, note that, in steady state (in this case at time $t > 2t_d$), the transmission line is equivalent to an equipotential wire, as shown in Figure A.28. At steady-state, the system acts like a voltage divider, so

$$V_A = V_B = V_C = V_S\left(\frac{Z_L}{Z_L + Z_S}\right) = V_S\left(\frac{75\ \Omega}{75\ \Omega + 50\ \Omega}\right) = \frac{3V_S}{5}$$

Reflections can occur at both ends of the transmission line. Figure A.29 shows a transmission line with a source impedance, Z_S, of 450 Ω and an open termination at the load. The reflection coefficients at the load and source, k_{rL} and k_{rS}, are 4/5 and 1, respectively. In this case, waves reflect off both ends of the transmission line until a steady state is reached.

The *bounce diagram*, shown in Figure A.30, helps visualize reflections off both ends of the transmission line. The horizontal axis represents

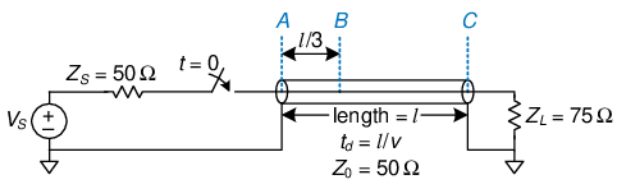


Figure A.26 Transmission line with mismatched termination

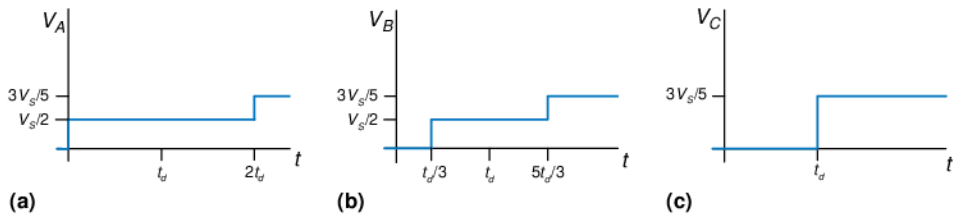


Figure A.27 Voltage waveforms for Figure A.26 at points *A*, *B*, and *C*

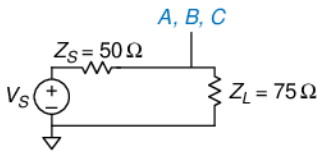


Figure A.28 Equivalent circuit of Figure A.26 at steady-state

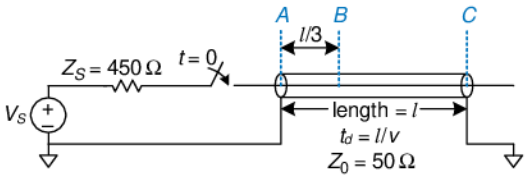


Figure A.29 Transmission line with mismatched source and load terminations

distance along the transmission line, and the vertical axis represents time, increasing downward. The two sides of the bounce diagram represent the source and load ends of the transmission line, points *A* and C. The incoming and reflected signal waves are drawn as diagonal lines between points *A* and C. At time $t = 0$, the source impedance and transmission line behave as a voltage divider, launching a voltage wave of $\frac{V_S}{10}$ from point *A* toward point C. At time $t = t_d$, the signal reaches point C and is completely reflected ($k_{rL} = 1$). At time $t = 2t_d$, the reflected wave of $\frac{V_S}{10}$ reaches point *A* and is reflected with a reflection coefficient, $k_{rS} = 4/5$, to produce a wave of $\frac{2V_S}{25}$ traveling toward point C, and so forth.

The voltage at a given time at any point on the transmission line is the sum of all the incident and reflected waves. Thus, at time $t = 1.1t_d$, the voltage at point C is $\frac{V_S}{10} + \frac{V_S}{10} = \frac{V_S}{5}$. At time $t = 3.1t_d$, the voltage at point C is $\frac{V_S}{10} + \frac{V_S}{10} + \frac{2V_S}{25} + \frac{2V_S}{25} = \frac{9V_S}{25}$, and so forth. Figure A.31 plots the voltages against time. As t approaches infinity, the voltages approach steady-state with $V_A = V_B = V_C = V_S$.

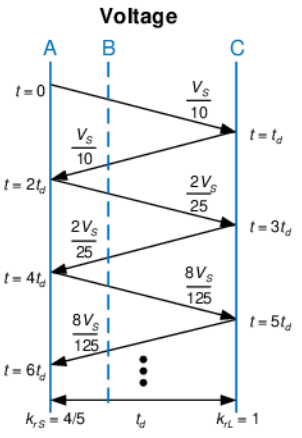


Figure A.30 Bounce diagram for Figure A.29

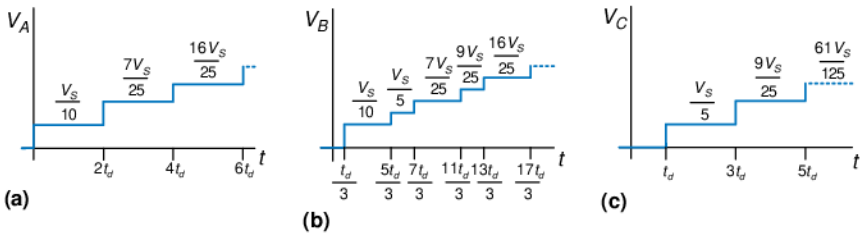


Figure A.31 Voltage and current waveforms for Figure A.29

A.8.5 When to Use Transmission Line Models

Transmission line models for wires are needed whenever the wire delay, t_d, is longer than a fraction (e.g., 20%) of the edge rates (rise or fall times) of a signal. If the wire delay is shorter, it has an insignificant effect on the propagation delay of the signal, and the reflections dissipate while the signal is transitioning. If the wire delay is longer, it must be considered in order to accurately predict the propagation delay and waveform of the signal. In particular, reflections may distort the digital characteristic of a waveform, resulting in incorrect logic operations.

Recall that signals travel on a PCB at about 15 cm/ns. For TTL logic, with edge rates of 10 ns, wires must be modeled as transmission lines only if they are longer than 30 cm (10 ns × 15 cm/ns × 20%). PCB traces are usually less than 30 cm, so most traces can be modeled as ideal equipotential wires. In contrast, many modern chips have edge rates of 2 ns or less, so traces longer than about 6 cm (about 2.5 inches) must be modeled as transmission lines. Clearly, use of edge rates that are crisper than necessary just causes difficulties for the designer.

Breadboards lack a ground plane, so the electromagnetic fields of each signal are nonuniform and difficult to model. Moreover, the fields interact with other signals. This can cause strange reflections and crosstalk between signals. Thus, breadboards are unreliable above a few megahertz.

In contrast, PCBs have good transmission lines with consistent characteristic impedance and velocity along the entire line. As long as they are terminated with a source or load impedance that is matched to the impedance of the line, PCB traces do not suffer from reflections.

A.8.6 Proper Transmission Line Terminations

There are two common ways to properly terminate a transmission line, shown in Figure A.32. In *parallel termination* (Figure A.32(a)), the driver has a low impedance ($Z_S \approx 0$). A load resistor (Z_L) with impedance Z_0 is placed in parallel with the load (between the input of the receiver gate and

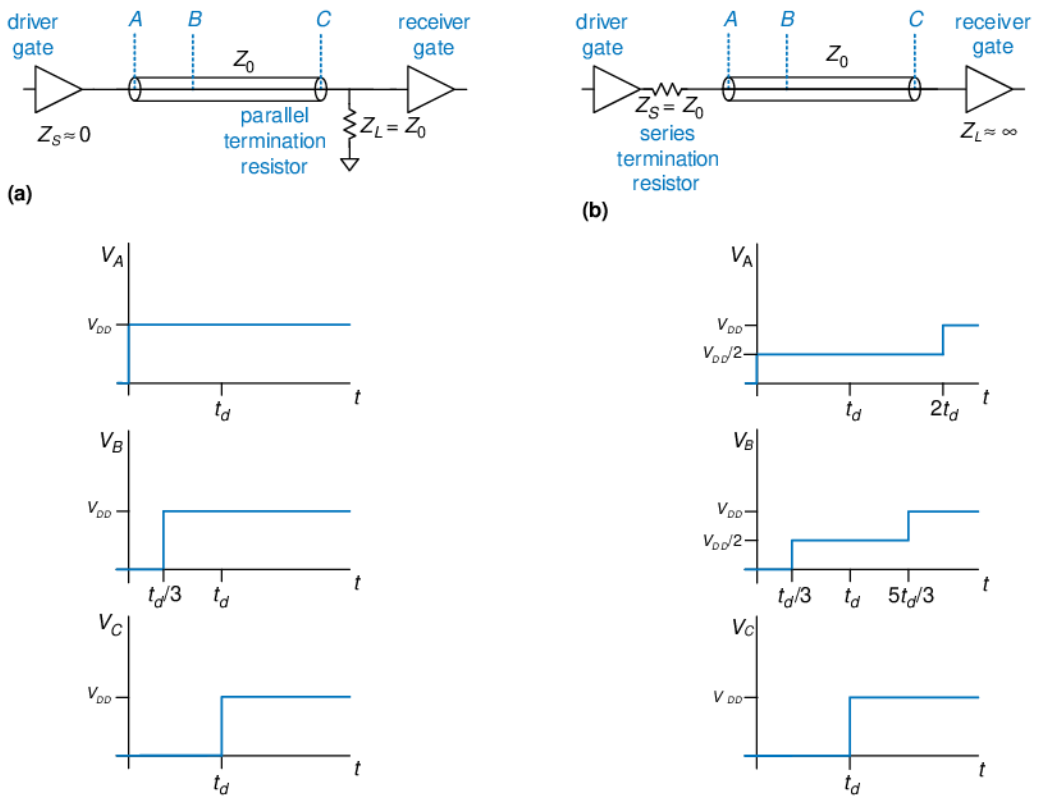


Figure A.32 Termination schemes: (a) parallel, (b) series

ground). When the driver switches from 0 to V_{DD}, it sends a wave with voltage V_{DD} down the line. The wave is absorbed by the matched load termination, and no reflections take place. In *series termination* (Figure A.32(b)), a source resistor (Z_S) is placed in series with the driver to raise the source impedance to Z_0. The load has a high impedance ($Z_L \approx \infty$). When the driver switches, it sends a wave with voltage $V_{DD}/2$ down the line. The wave reflects at the open circuit load and returns, bringing the voltage on the line up to V_{DD}. The wave is absorbed at the source termination. Both schemes are similar in that the voltage at the receiver transitions from 0 to V_{DD} at $t = t_d$, just as one would desire. They differ in power consumption and in the waveforms that appear elsewhere along the line. Parallel termination dissipates power continuously through the load resistor when the line is at a high voltage. Series termination dissipates no DC power, because the load is an open circuit. However, in series terminated lines, points near the middle of the transmission line initially see a voltage of $V_{DD}/2$, until the reflection returns. If other gates are attached to the

middle of the line, they will momentarily see an illegal logic level. Therefore, series termination works best for *point-to-point* communication with a single driver and a single receiver. Parallel termination is better for a *bus* with multiple receivers, because receivers at the middle of the line never see an illegal logic level.

A.8.7 Derivation of Z_0*

Z_0 is the ratio of voltage to current in a wave propagating along a transmission line. This section derives Z_0; it assumes some previous knowledge of resistor-inductor-capacitor (RLC) circuit analysis.

Imagine applying a step voltage to the input of a semi-infinite transmission line (so that there are no reflections). Figure A.33 shows the semi-infinite line and a model of a segment of the line of length *dx*. *R*, *L*, and *C*, are the values of resistance, inductance, and capacitance per unit length. Figure A.33(b) shows the transmission line model with a resistive component, *R*. This is called a *lossy* transmission line model, because energy is dissipated, or lost, in the resistance of the wire. However, this loss is often negligible, and we can simplify analysis by ignoring the resistive component and treating the transmission line as an *ideal* transmission line, as shown in Figure A.33(c).

Voltage and current are functions of time and space throughout the transmission line, as given by Equations A.8 and A.9.

$$\frac{\partial}{\partial x}V(x,t) = L\frac{\partial}{\partial t}I(x,t) \tag{A.8}$$

$$\frac{\partial}{\partial x}I(x,t) = C\frac{\partial}{\partial t}V(x,t) \tag{A.9}$$

Taking the space derivative of Equation A.8 and the time derivative of Equation A.9 and substituting gives Equation A.10, the *wave equation*.

$$\frac{\partial^2}{\partial x^2}V(x,t) = LC\frac{\partial^2}{\partial t^2}I(x,t) \tag{A.10}$$

Z_0 is the ratio of voltage to current in the transmission line, as illustrated in Figure A.34(a). Z_0 must be independent of the length of the line, because the behavior of the wave cannot depend on things at a distance. Because it is independent of length, the impedance must still equal Z_0 after the addition of a small amount of transmission line, *dx*, as shown in Figure A.34(b).

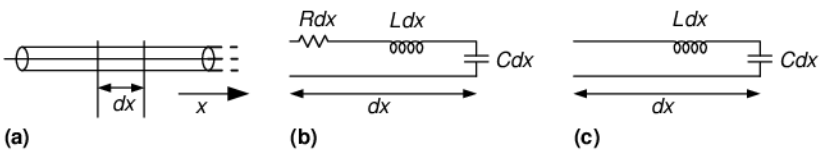


Figure A.33 Transmission line models: (a) semi-infinite cable, (b) lossy, (c) ideal

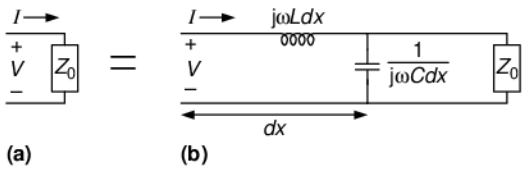


Figure A.34 Transmission line model: (a) for entire line and (b) with additional length, *dx*

Using the impedances of an inductor and a capacitor, we rewrite the relationship of Figure A.34 in equation form:

$$Z_0 = j\omega L dx + [Z_0 \| (1/(j\omega C dx))] \tag{A.11}$$

Rearranging, we get

$$Z_0^2(j\omega C) - j\omega L + \omega^2 Z_0 L C dx = 0 \tag{A.12}$$

Taking the limit as dx approaches 0, the last term vanishes and we find that

$$Z_0 = \sqrt{\frac{L}{C}} \tag{A.13}$$

A.8.8 Derivation of the Reflection Coefficient*

The reflection coefficient, k_r, is derived using conservation of current. Figure A.35 shows a transmission line with characteristic impedance, Z_0, and load impedance, Z_L. Imagine an incident wave of voltage V_i and current I_i. When the wave reaches the termination, some current, I_L, flows through the load impedance, causing a voltage drop, V_L. The remainder of the current reflects back down the line in a wave of voltage, V_r, and current, I_r. Z_0 is the ratio of voltage to current in waves propagating along the line, so $\frac{V_i}{I_i} = \frac{V_r}{I_r} = Z_0$.

The voltage on the line is the sum of the voltages of the incident and reflected waves. The current flowing in the positive direction on the line is the difference between the currents of the incident and reflected waves.

$$V_L = V_i + V_r \tag{A.14}$$

$$I_L = I_i - I_r \tag{A.15}$$

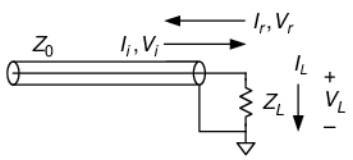


Figure A.35 Transmission line showing incoming, reflected, and load voltages and currents

Using Ohm's law and substituting for I_L, I_i, and I_r in Equation A.15, we get

$$\frac{V_i + V_r}{Z_L} = \frac{V_i}{Z_0} - \frac{V_r}{Z_0} \tag{A.16}$$

Rearranging, we solve for the reflection coefficient, k_r:

$$\frac{V_r}{V_i} = \frac{Z_L - Z_0}{Z_L + Z_0} = k_r \tag{A.17}$$

A.8.9 Putting It All Together

Transmission lines model the fact that signals take time to propagate down long wires because the speed of light is finite. An ideal transmission line has uniform inductance, L, and capacitance, C, per unit length and zero resistance. The transmission line is characterized by its characteristic impedance, Z_0, and delay, t_d, which can be derived from the inductance, capacitance, and wire length. The transmission line has significant delay and noise effects on signals whose rise/fall times are less than about $5t_d$. This means that, for systems with 2 ns rise/fall times, PCB traces longer than about 6 cm must be analyzed as transmission lines to accurately understand their behavior.

A digital system consisting of a gate driving a long wire attached to the input of a second gate can be modeled with a transmission line as shown in Figure A.36. The voltage source, source impedance (Z_S), and switch model the first gate switching from 0 to 1 at time 0. The driver gate cannot supply infinite current; this is modeled by Z_S. Z_S is usually small for a logic gate, but a designer may choose to add a resistor in series with the gate to raise Z_S and match the impedance of the line. The input to the second gate is modeled as Z_L. CMOS circuits usually have little input current, so Z_L may be close to infinity. The designer may also choose to add a resistor in parallel with the second gate, between the gate input and ground, so that Z_L matches the impedance of the line.

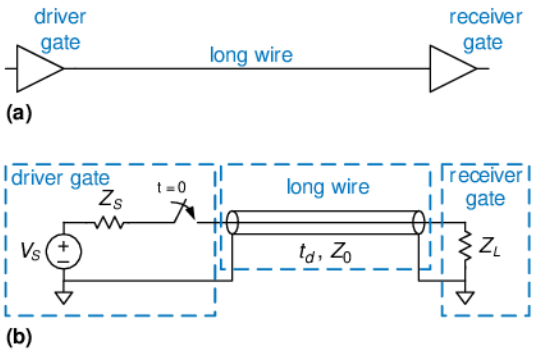


Figure A.36 Digital system modeled with transmission line

When the first gate switches, a wave of voltage is driven onto the transmission line. The source impedance and transmission line form a voltage divider, so the voltage of the incident wave is

$$V_i = V_S \frac{Z_0}{Z_0 + Z_S} \tag{A.18}$$

At time t_d, the wave reaches the end of the line. Part is absorbed by the load impedance, and part is reflected. The reflection coefficient, k_r, indicates the portion that is reflected: $k_r = V_r/V_i$, where V_r is the voltage of the reflected wave and V_i is the voltage of the incident wave.

$$k_r = \frac{Z_L - Z_0}{Z_L + Z_0} \tag{A.19}$$

The reflected wave adds to the voltage already on the line. It reaches the source at time $2t_d$, where part is absorbed and part is again reflected. The reflections continue back and forth, and the voltage on the line eventually approaches the value that would be expected if the line were a simple equipotential wire.

A.9 ECONOMICS

Although digital design is so much fun that some of us would do it for free, most designers and companies intend to make money. Therefore, economic considerations are a major factor in design decisions.

The cost of a digital system can be divided into *nonrecurring engineering costs* (*NRE*), and *recurring costs*. NRE accounts for the cost of designing the system. It includes the salaries of the design team, computer and software costs, and the costs of producing the first working unit. The fully loaded cost of a designer in the United States in 2006 (including salary, health insurance, retirement plan, and a computer with design tools) is roughly $200,000 per year, so design costs can be significant. Recurring costs are the cost of each additional unit; this includes components, manufacturing, marketing, technical support, and shipping.

The sales price must cover not only the cost of the system but also other costs such as office rental, taxes, and salaries of staff who do not directly contribute to the design (such as the janitor and the CEO). After all of these expenses, the company should still make a profit.

Example A.3 BEN TRIES TO MAKE SOME MONEY

Ben Bitdiddle has designed a crafty circuit for counting raindrops. He decides to sell the device and try to make some money, but he needs help deciding what implementation to use. He decides to use either an FPGA or an ASIC. The

development kit to design and test the FPGA costs $1500. Each FPGA costs $17. The ASIC costs $600,000 for a mask set and $4 per chip.

Regardless of what chip implementation he chooses, Ben needs to mount the packaged chip on a printed circuit board (PCB), which will cost him $1.50 per board. He thinks he can sell 1000 devices per month. Ben has coerced a team of bright undergraduates into designing the chip for their senior project, so it doesn't cost him anything to design.

If the sales price has to be twice the cost (100% profit margin), and the product life is 2 years, which implementation is the better choice?

SOLUTION: Ben figures out the total cost for each implementation over 2 years, as shown in Table A.4. Over 2 years, Ben plans on selling 24,000 devices, and the total cost is given in Table A.4 for each option. If the product life is only two years, the FPGA option is clearly superior. The per-unit cost is $445,500/24,000 = $18.56, and the sales price is $37.13 per unit to give a 100% profit margin. The ASIC option would have cost $732,000/24,000 = $30.50 and would have sold for $61 per unit.

Table A.4 ASIC vs FPGA costs

Cost	ASIC	FPGA
NRE	$600,000	$1500
chip	$4	$17
PCB	$1.50	$1.50
TOTAL	$600,000 + (24,000 × $5.50) = **$732,000**	$1500 + (24,000 × $18.50) = **$445,500**
per unit	$30.50	$18.56

Example A.4 BEN GETS GREEDY

After seeing the marketing ads for his product, Ben thinks he can sell even more chips per month than originally expected. If he were to choose the ASIC option, how many devices per month would he have to sell to make the ASIC option more profitable than the FPGA option?

SOLUTION: Ben solves for the minimum number of units, N, that he would need to sell in 2 years:

$$\$600{,}000 + (N \times \$5.50) = \$1500 + (N \times \$18.50)$$

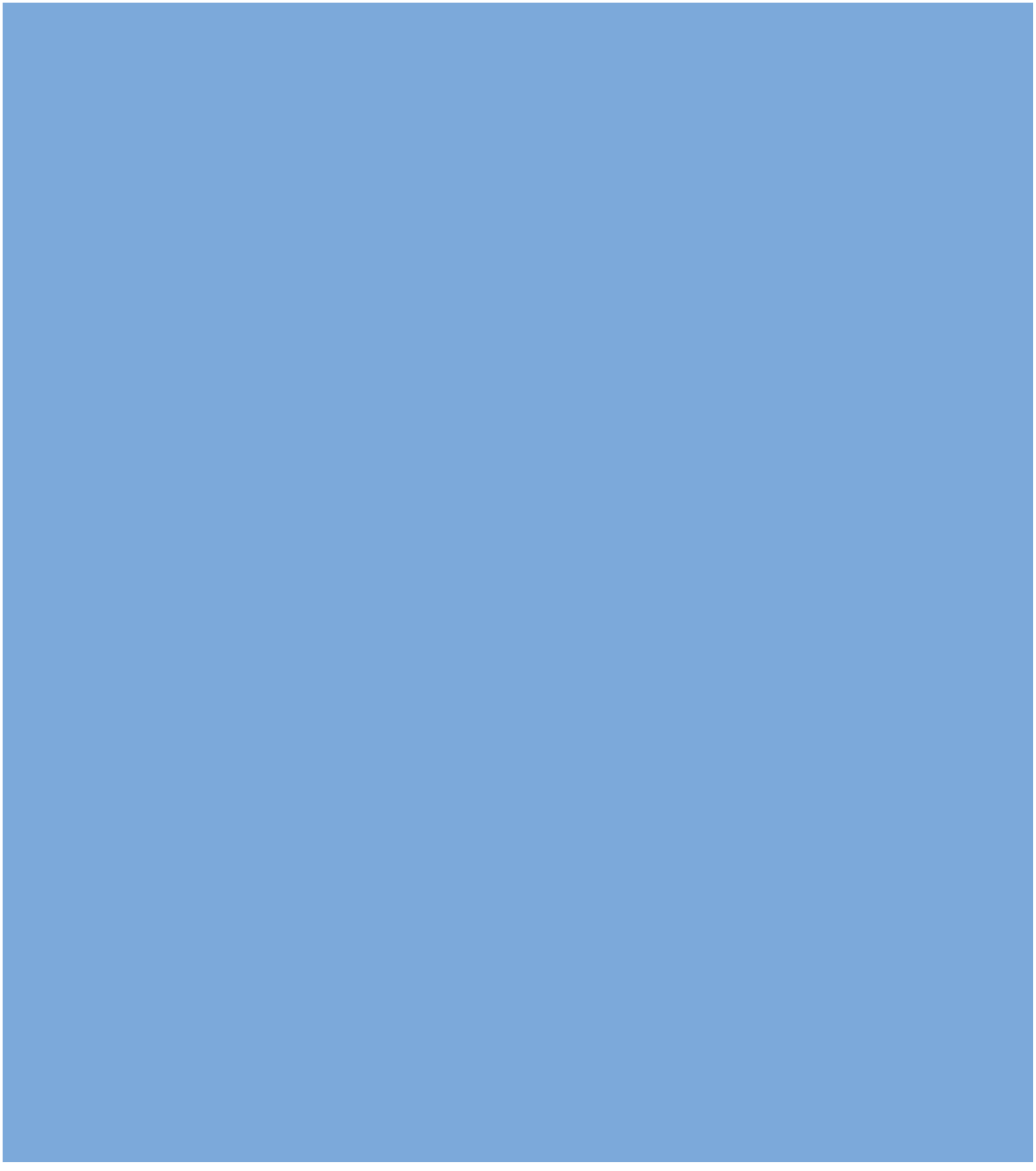
Solving the equation gives $N = 46{,}039$ units, or 1919 units per month. He would need to almost double his monthly sales to benefit from the ASIC solution.

Example A.5 BEN GETS LESS GREEDY

Ben realizes that his eyes have gotten too big for his stomach, and he doesn't think he can sell more than 1000 devices per month. But he does think the product life can be longer than 2 years. At a sales volume of 1000 devices per month, how long would the product life have to be to make the ASIC option worthwhile?

SOLUTION: If Ben sells more than 46,039 units in total, the ASIC option is the best choice. So, Ben would need to sell at a volume of 1000 per month for at least 47 months (rounding up), which is almost 4 years. By then, his product is likely to be obsolete.

Chips are usually purchased from a distributor rather than directly from the manufacturer (unless you are ordering tens of thousands of units). Digikey (`www.digikey.com`) is a leading distributor that sells a wide variety of electronics. Jameco (`www.jameco.com`) and All Electronics (`www.allelectronics.com`) have eclectic catalogs that are competitively priced and well suited to hobbyists.



MIPS Instructions

This appendix summarizes MIPS instructions used in this book. Tables B.1–B.3 define the `opcode` and `funct` fields for each instruction, along with a short description of what the instruction does. The following notations are used:

- `[reg]`: contents of the register
- `imm`: 16-bit immediate field of the I-type instruction
- `addr`: 26-bit address field of the J-type instruction
- `SignImm`: sign-extended immediate
  `= {{16{imm[15]}}, imm}`
- `ZeroImm`: zero-extended immediate
  `= {16'b0, imm}`
- `Address`: `[rs] + SignImm`
- `[Address]`: contents of memory location `Address`
- `BTA`: branch target address[1]
  `= PC + 4 + (SignImm << 2)`
- `JTA`: jump target address
  `= {(PC + 4)[31:28], addr, 2'b0}`

[1] The SPIM simulator has no branch delay slot, so BTA is PC + (SignImm << 2). Thus, if you use the SPIM assembler to create machine code for a real MIPS processor, you must decrement the immediate field by 1 to compensate.

Table B.1 Instructions, sorted by opcode

Opcode	Name	Description	Operation
000000 (0)	R-type	all R-type instructions	see Table B.2
000001 (1) (rt = 0/1)	bltz/bgez	branch less than zero/ branch greater than or equal to zero	if ([rs] < 0) PC = BTA/ if ([rs] ≥ 0) PC = BTA
000010 (2)	j	jump	PC = JTA
000011 (3)	jal	jump and link	$ra = PC+4, PC = JTA
000100 (4)	beq	branch if equal	if ([rs]==[rt]) PC = BTA
000101 (5)	bne	branch if not equal	if ([rs]!=[rt]) PC = BTA
000110 (6)	blez	branch if less than or equal to zero	if ([rs] ≤ 0) PC = BTA
000111 (7)	bgtz	branch if greater than zero	if ([rs] > 0) PC = BTA
001000 (8)	addi	add immediate	[rt] = [rs] + SignImm
001001 (9)	addiu	add immediate unsigned	[rt] = [rs] + SignImm
001010 (10)	slti	set less than immediate	[rs] < SignImm ? [rt]=1 : [rt]=0
001011 (11)	sltiu	set less than immediate unsigned	[rs] < SignImm ? [rt]=1 : [rt]=0
001100 (12)	andi	and immediate	[rt] = [rs] & ZeroImm
001101 (13)	ori	or immediate	[rt] = [rs] | ZeroImm
001110 (14)	xori	xor immediate	[rt] = [rs] ^ ZeroImm
001111 (15)	lui	load upper immediate	[rt] = {Imm, 16'b0}
010000 (16) (rs = 0/4)	mfc0, mtc0	move from/to coprocessor 0	[rt] = [rd]/[rd] = [rt] (rd is in coprocessor 0)
010001 (17)	F-type	fop = 16/17: F-type instructions	see Table B.3
010001 (17) (rt = 0/1)	bc1f/bc1t	fop = 8: branch if fpcond is FALSE/TRUE	if (fpcond == 0) PC = BTA/ if (fpcond == 1) PC = BTA
100000 (32)	lb	load byte	[rt] = SignExt ([Address]$_{7:0}$)
100001 (33)	lh	load halfword	[rt] = SignExt ([Address]$_{15:0}$)
100011 (35)	lw	load word	[rt] = [Address]
100100 (36)	lbu	load byte unsigned	[rt] = ZeroExt ([Address]$_{7:0}$)
100101 (37)	lhu	load halfword unsigned	[rt] = ZeroExt ([Address]$_{15:0}$)
101000 (40)	sb	store byte	[Address]$_{7:0}$ = [rt]$_{7:0}$

(*continued*)

Table B.1 Instructions, sorted by opcode—Cont'd

Opcode	Name	Description	Operation
101001 (41)	sh	store halfword	$[Address]_{15:0} = [rt]_{15:0}$
101011 (43)	sw	store word	[Address] = [rt]
110001 (49)	lwc1	load word to FP coprocessor 1	[ft] = [Address]
111001 (56)	swc1	store word to FP coprocessor 1	[Address] = [ft]

Table B.2 R-type instructions, sorted by funct field

Funct	Name	Description	Operation
000000 (0)	sll	shift left logical	[rd] = [rt] << shamt
000010 (2)	srl	shift right logical	[rd] = [rt] >> shamt
000011 (3)	sra	shift right arithmetic	[rd] = [rt] >>> shamt
000100 (4)	sllv	shift left logical variable	[rd] = [rt] << $[rs]_{4:0}$ assembly: sllv rd, rt, rs
000110 (6)	srlv	shift right logical variable	[rd] = [rt] >> $[rs]_{4:0}$ assembly: srlv rd, rt, rs
000111 (7)	srav	shift right arithmetic variable	[rd] = [rt] >>> $[rs]_{4:0}$ assembly: srav rd, rt, rs
001000 (8)	jr	jump register	PC = [rs]
001001 (9)	jalr	jump and link register	$ra = PC + 4, PC = [rs]
001100 (12)	syscall	system call	system call exception
001101 (13)	break	break	break exception
010000 (16)	mfhi	move from hi	[rd] = [hi]
010001 (17)	mthi	move to hi	[hi] = [rs]
010010 (18)	mflo	move from lo	[rd] = [lo]
010011 (19)	mtlo	move to lo	[lo] = [rs]
011000 (24)	mult	multiply	{[hi], [lo]} = [rs] × [rt]
011001 (25)	multu	multiply unsigned	{[hi], [lo]} = [rs] × [rt]
011010 (26)	div	divide	[lo] = [rs]/[rt], [hi] = [rs]%[rt]

(continued)

Table B.2 R-type instructions, sorted by funct field—Cont'd

Funct	Name	Description	Operation
011011 (27)	divu	divide unsigned	[lo] = [rs]/[rt], [hi] = [rs]%[rt]
100000 (32)	add	add	[rd] = [rs] + [rt]
100001 (33)	addu	add unsigned	[rd] = [rs] + [rt]
100010 (34)	sub	subtract	[rd] = [rs] − [rt]
100011 (35)	subu	subtract unsigned	[rd] = [rs] − [rt]
100100 (36)	and	and	[rd] = [rs] & [rt]
100101 (37)	or	or	[rd] = [rs] | [rt]
100110 (38)	xor	xor	[rd] = [rs] ^ [rt]
100111 (39)	nor	nor	[rd] = ~([rs] | [rt])
101010 (42)	slt	set less than	[rs] < [rt] ? [rd] = 1 : [rd] = 0
101011 (43)	sltu	set less than unsigned	[rs] < [rt] ? [rd] = 1 : [rd] = 0

Table B.3 F-type instructions (fop = 16/17)

Funct	Name	Description	Operation
000000 (0)	add.s/add.d	FP add	[fd] = [fs] + [ft]
000001 (1)	sub.s/sub.d	FP subtract	[fd] = [fs] − [ft]
000010 (2)	mul.s/mul.d	FP multiply	[fd] = [fs] * [ft]
000011 (3)	div.s/div.d	FP divide	[fd] = [fs]/[ft]
000101 (5)	abs.s/abs.d	FP absolute value	[fd] = ([fs] < 0) ? [−fs] : [fs]
000111 (7)	neg.s/neg.d	FP negation	[fd] = [−fs]
111010 (58)	c.seq.s/c.seq.d	FP equality comparison	fpcond = ([fs] == [ft])
111100 (60)	c.lt.s/c.lt.d	FP less than comparison	fpcond = ([fs] < [ft])
111110 (62)	c.le.s/c.le.d	FP less than or equal comparison	fpcond = ([fs] ≤ [ft])

Further Reading

Berlin L., *The Man Behind the Microchip: Robert Noyce and the Invention of Silicon Valley*, Oxford University Press, 2005.

The fascinating biography of Robert Noyce, an inventor of the microchip and founder of Fairchild and Intel. For anyone thinking of working in Silicon Valley, this book gives insights into the culture of the region, a culture influenced more heavily by Noyce than any other individual.

Colwell R., *The Pentium Chronicles: The People, Passion, and Politics Behind Intel's Landmark Chips*, Wiley, 2005.

An insider's tale of the development of several generations of Intel's Pentium chips, told by one of the leaders of the project. For those considering a career in the field, this book offers views into the managment of huge design projects and a behind-the-scenes look at one of the most significant commercial microprocessor lines.

Ercegovac M., and Lang T., *Digital Arithmetic*, Morgan Kaufmann, 2003.

The most complete text on computer arithmetic systems. An excellent resource for building high-quality arithmetic units for computers.

Hennessy J., and Patterson D., *Computer Architecture: A Quantitative Approach, 4th ed.*, Morgan Kaufmann, 2006.

The authoritative text on advanced computer architecture. If you are intrigued about the inner workings of cutting-edge microprocessors, this is the book for you.

Kidder T., *The Soul of a New Machine*, Back Bay Books, 1981.

A classic story of the design of a computer system. Three decades later, the story is still a page-turner and the insights on project managment and technology still ring true.

Pedroni V., *Circuit Design with VHDL*, MIT Press, 2004.

A reference showing how to design circuits with VHDL.

Thomas D., and Moorby P., *The Verilog Hardware Description Language, 5th ed.*, Kluwer Academic Publishers, 2002.

Ciletti M., *Advanced Digital Design with the Verilog HDL*, Prentice Hall, 2003.

Both excellent references covering Verilog in more detail.

Verilog IEEE Standard (IEEE STD 1364).
The IEEE standard for the Verilog Hardware Description Language; last updated in 2001. Available at *ieeexplore.ieee.org*.

VHDL IEEE Standard (IEEE STD 1076).
The IEEE standard for VHDL; last updated in 2004. Available from IEEE. Available at *ieeexplore.ieee.org*.

Wakerly J., *Digital Design: Principles and Practices, 4th ed.*, Prentice Hall, 2006.
A comprehensive and readable text on digital design, and an excellent reference book.

Weste N., and Harris D., *CMOS VLSI Design, 3rd ed.*, Addison-Wesley, 2005.
Very Large Scale Integration (VLSI) Design is the art and science of building chips containing oodles of transistors. This book, coauthored by one of our favorite writers, spans the field from the beginning through the most advanced techniques used in commercial products.

Index

0, 23. *See also* LOW, OFF
1, 23. *See also* HIGH, ON
74xx series logic, 515–516
  parts,
    2:1 Mux (74157), 518
    3:8 Decoder (74138), 518
    4:1 Mux (74153), 518
    AND (7408), 517
    AND3 (7411), 517
    AND4 (7421), 517
    Counter (74161, 74163), 518
    FLOP (7474), 515–517
    NAND (7400), 517
    NOR (7402), 517
    NOT (7404), 515
    OR (7432), 517
    XOR (7486), 517
    Register (74377), 518
    Tristate buffer (74244), 518
  schematics of, 517–518

A

`add`, 291
Adders, 233–240
  carry-lookahead. *See* Carry-lookahead adder
  carry-propagate (CPA). *See* Carry-propagate adder
  prefix. *See* Prefix adder
  ripple-carry. *See* Ripple-carry adder
add immediate (`addi`), 327, 378
add immediate unsigned (`addiu`), 552
add unsigned (`addu`), 554
Addition, 14–15, 233–240, 291
  floating-point, 252–255
  overflow. *See* Overflow
  two's complement, 15, 240
  underflow. *See* Underflow
  unsigned binary, 15
Address, 485–490
  physical, 485
  translation, 486–489
  virtual, 485–490
  word alignment, 298
Addressing modes, 327–329
  base, 327
  immediate, 327
  MIPS, 327–329
  PC-relative, 327–328
  pseudo-direct, 328–329
  register-only, 327
Advanced microarchitecture, 435–447
  branch prediction. *See* Branch prediction
  deep pipelines. *See* Deep pipelines
  multiprocessors. *See* Multiprocessors
  multithreading. *See* Multithreading
  out-of-order processor. *See* Out-of-order processor
  register renaming. *See* Register renaming
  single instruction multiple data. *See* Single instruction multiple data (SIMD) units
  superscalar processor. *See* Superscalar processor
  vector processor. *See* Single instruction multiple data (SIMD) units
Alignment. *See* Word alignment
ALU. *See* Arithmetic/logical unit
ALU decoder, 374–376
ALU decoder truth table, 376
ALUControl, 370
ALUOp, 374–375
ALUOut, 383–385
ALUResult, 370–371
AMAT. *See* Average memory access time
Amdahl, Gene, 468
Amdahl's Law, 468
Anodes, 27–28
and immediate (`andi`), 306, 346–347
AND gate, 20–22, 32–33
Application-specific integrated circuits (ASICs), 523
Architectural state, 363–364
Architecture. *See* Instruction Set Architecture
Arithmetic, 233–249, 305–308, 515. *See also* Adders, Addition, Comparator, Divider, Multiplier
  adders. *See* Adders
  addition. *See* Addition
  ALU. *See* Arithmetic/logical unit
  circuits, 233–248
  comparators. *See* Comparators
  divider. *See* Divider
  division. *See* Division
  fixed-point, 249
  floating-point, 252–255
  logical instructions, 305–308
  multiplier. *See* Multiplier
  packed, 445
  rotators. *See* Rotators
  shifters. *See* Shifters
  signed and unsigned, 338–339
  subtraction. *See* Subtraction

Arithmetic (*Continued*)
subtractor. *See* Subtractor
underflow. *See* Addition, Underflow
Arithmetic/logical unit (ALU), 242–244, 515
32-bit, 244
adder. *See* Adders
ALUOp, 374–376
ALUResult, 370–371
ALUOut, 383–385
comparator. *See* Comparators
control, 242
subtractor. *See* Subtractor
Arrays, 314–318
accessing, 315–317
bytes and characters, 316–318
FPGA, *See* Field programmable gate array
logic. *See* Logic arrays
memory. *See* Memory
RAM, 265
ROM, 266
ASCII (American Standard Code for Information Interchange) codes, 317
table of, 317
Assembler directives, 333
ASICs. *See* Application-specific integrated circuits
Assembly language, MIPS. *See* MIPS assembly language
Associativity, 58–59
Average memory access time (AMAT), 467–468
Asynchronous circuits, 116–117
Asynchronous inputs, 144
Asynchronous resettable registers
HDL for, 192
Axioms. *See* Boolean axioms

B

Babbage, Charles, 7–8, 26
Base address, 295–296
Base register, 302, 343, 347
Base 2 number representations. *See* Binary numbers
Base 8 number representations. *See* Octal numbers
Base 16 number representations. *See* Hexadecimal numbers
Block,
digital building. *See* Digital building blocks
in code,
else block, 311
if block, 310–311
Base addressing, 327
Baudot, Jean-Maurice-Emile, 317
Behavioral modeling, 171–185
Benchmarks, 367
SPEC2000, 398–399
Biased numbers, 41. *See also* Floating point
Big-Endian, 172, 296–297
Binary numbers, 9–11. *See also* Arithmetic
ASCII, 317
binary coded decimal, 252
conversion. *See* Number conversion
fixed point, 249–250
floating-point. *See* Floating-point numbers
signed, 15–19
sign/magnitude. *See* Sign/magnitude numbers
two's complement. *See* Two's complement numbers
unsigned, 9–15
Binary to decimal conversion, 10–11
Binary to hexadecimal conversion, 12
Bit, 9–11
dirty, 482–483
least significant, 13–14
most significant, 13–14
sign, 16
use, 478–479
valid, 472
Bit cells, 258
Bit swizzling, 182
Bitwise operators, 171–174
Boole, George, 8
Boolean algebra, 56–62
axioms, 57
equation simplification, 61–62
theorems, 57–60
Boolean axioms, 57
Boolean equations, 54–56
product-of-sums (POS) canonical form, 56
sum-of-products (SOP) canonical form, 54–55
terminology, 54
Boolean theorems, 57–60
`DeMorgan's`, 59
`complement`, 58, 59
`consensus`, 60
`covering`, 58
`combining`, 58
Branching, 308–310
calculating address, 309
conditional, 308
prediction, 437–438
target address (BTA), 327–328
unconditional (jump), 309
Branch equal (`beq`), 308–309
Branch not equal (`bne`), 308–309
Branch/control hazards, 407, 413–416. *See also* Hazards
Branch prediction, 437–438
Breadboards, 532–533
Bubble, 59
pushing, 67–69
Buffer, 20
tristate, 70–71
"Bugs," 169
Bus, 52
tristate, 71
Bypassing, 408
Bytes, 13–14
Byte-addressable memory, 295–297
Byte order, 296–297
Big-Endian, 296–297
Little-Endian, 296–297

C

C programming language, 290
overflows, 339
strings, 318
Caches, 468–484. *See also* Memory
accessing, 472–473, 476–477, 479
advanced design, 479–483
associativity, 469, 474–475, 478–479
block size, 476–477
blocks, 469–470, 476
capacity, 468–470
data placement, 469–470
data replacement, 478–479

definition, 468
direct mapped, 470–474
dirty bit, 482–483
entry fields, 472–473
evolution of MIPS, 483
fully associative, 475–476
hit, 466
hit rate, 467
IA-32 systems, 499–500
level 2 (L2), 480
mapping, 470–478
miss, 466
capacity, 481
compulsory, 481
conflict, 481
miss rate, 467
miss rate versus cache parameters, 481–482
miss penalty, 476
multiway set associative, 474–475
nonblocking, 500
organizations, 478
performance, 467
set associative, 470, 474–475
tag, 471–472
use bit, 478–479
valid bit, 472–473
write policy, 482–483
CAD. *See* Computer-aided design
Canonical form, 53
Capacitors, 28
Capacity miss, 481
Carry-lookahead adder, 235–237
Carry propagate adder (CPA), 274
Cathodes, 27–28
Cause register, 337–338
Chips, 28, 449
74xx series logic. *See* 74xx series logic
Circuits, 82–83, 87–88
74xx series logic. *See* 74xx series logic
application-specific integrated (ASIC), 523
arithmetic. *See* Arithmetic
asynchronous, 116–117
bistable device, 147
combinational, 53
definition, 51
delays, 73–75, 84–87
calculating, 86–87
dynamic, 273
multiple-output, 64
pipelining, 152
priority, 65, 202, 203
synchronous sequential, 114–116
synthesized, 186–190, 193–195, 199, 200
timing, 84–91
timing analysis, 138
types, 51–54
without glitch, 91
CISC. *See* Complex instruction set computers
CLBs. *See* Configurable logic blocks
Clock cycle. *See* Clock period
Clock period, 135–139
Clock rate. *See* Clock period
Clock cycles per instruction (CPI), 367–368
Clustered computers, 447
Clock skew, 140–143
CMOS. *See* Complementary Metal-Oxide-Semiconductor Logic
Code, 303
Code size, 334
Combinational composition, 52–53
Combinational logic design, 51–100
Boolean algebra, 56–62
Boolean equations, 54–56
building blocks, 79–84
delays, 85–87
don't cares, 65
HDLs and. *See* Hardware description languages
Karnaugh maps, 71–79
logic, 62–65
multilevel, 65–69
overview, 51–54
precedence, 54
timing, 84–91
two-level, 65–66
X's (contention). *See* Contention
X's (don't cares). *See* Don't cares
Z's (floating). *See* Floating
Comparators, 240–241
equality, 241
magnitude, 241, 242
Compiler, 331–333
Complementary Metal-Oxide-Semiconductor Logic (CMOS), 25
bubble pushing, 69
logic gates, 31–33
NAND gate, 32
NOR gate, 32–33
NOT gate, 31
transistors, 26–34
Complex instruction set computers (CISC), 292, 341
Compulsory miss, 481
Computer-aided design (CAD), 167
Computer Organization and Design (Patterson and Hennessy), 290, 363
Complexity management, 4–6
abstraction, 4–5
discipline, 5–6
hierarchy, 6
modularity, 6
regularity, 6
Conditional assignment, 175–176
Conditional branches, 308
Condition codes, 344
Conflict misses, 481
Conditional statements, 310–311
Constants, 298. *See also* Immediates
Contamination delay, 84–88
Contention (X), 69–70
Context switch, 446
Control signals, 79, 242–243
Control unit, 364, 366, 374–406
multicycle MIPS processor FSM, 381–395
pipelined MIPS processor, 405–406
single-cycle MIPS processor, 374–377
Configurable logic blocks (CLBs), 268–272
Control hazards. *See* Branch/control hazards, Pipelining
Coprocessor 0, 338
Counters, 254
Covalent bond, 27
CPA. *See* Carry propagate adder
CPI. *See* Clock cycles per instruction
Critical path, 85–89
Cyclic paths, 114
Cycle time. *See* Clock Period

D

Data hazards. *See* Hazards
Data memory, 365
Data sheets, 523–528
Datapath, 364. *See also* MIPS microprocessors

Datapath (*Continued*)
elements, 364
multicycle MIPS processor, 382–388
pipelined MIPS processor, 404
single-cycle MIPS processor, 368–374
Data segments. *See* Memory map
Data types. *See* Hardware description languages
DC. *See* Direct current
Decimal numbers, 9
conversion to binary and hexadecimal. *See* Number conversion
scientific notation, 249–250
Decimal to Binary conversion, 11
Decimal to hexadecimal conversion, 13
Decoders
implementation, 83
logic, 83
parameterized, 212
Deep pipelines, 435–436
Delay, 182
DeMorgan, Augustus, 56
DeMorgan's theorem, 59, 60
Dennard, Robert, 260
Destination register, 301, 305–306, 370–371, 377–378
Device under test (DUT), 214–218. *See also* Unit under test
Device driver, 496, 498
Dice, 28
Digital abstraction, 4–5, 8–9, 22–26
DC transfer characteristics, 23–24
logic levels, 22–23
noise margins, 23
supply voltage, 22
Digital design
abstraction, 7–9
discipline, 5–6
hierarchy, 6
modularity, 6
regularity, 6
Digital system implementation, 515–548
74xx series logic. *See* 74xx series logic
application-specific integrated circuits (ASICs), 523
data sheets, 523–528
economics, 546–548
logic families, 529–531
overview, 515
packaging and assembly, 531–534
breadboards, 532
packages, 531–532
printed circuit boards, 533–534
programmable logic, 516–523
transmission lines. *See* Transmission lines
Diodes, 27–28
DIP. *See* Dual-inline package
Direct current (DC), 23, 24
transfer characteristics, 23–24, 25
Direct mapped cache, 470–474
Dirty bit, 482–483
Discipline, 5–6
Disk. *See* Hard disk
divide (`div`), 308
divide unsigned (`divu`), 339
Divider, 247–248
Division, 247–248
floating-point, 253
instructions, 308
Divisor, 247
Don't care (X), 65
Dopant atoms, 27
Double. *See* Double-precision floating-point numbers
Double-precision floating point numbers, 251–252
DRAM. *See* Dynamic random access memory
Driver. *See* Device driver
Dual-inline package (DIP), 28, 531
DUT. *See* Device under test
Dynamic discipline, 134
Dynamic data segment, 331
Dynamic random access memory (DRAM), 257, 260, 463

E

Edison, Thomas, 169
Edge-triggered digital systems, 108, 112
Equations
simplification, 61–62
Electrically erasable programmable read only memory (EEPROM), 263
Enabled registers, 193
HDL for, 193
EPC. *See* Exception Program Counter register
Erasable programmable read only memory (EPROM), 263
Exceptions, 337–339
Exception handler, 337–338
Exception program counter (EPC), 337–338
Exclusive or. *See* XOR
Executable file, 334
Execution time, 367
Exponent, 250–253

F

Failures, 144–146
FDIV bug, 253
FET. *See* Field effect transistors
Field programmable gate array (FPGA), 268–272, 521–523
Field effect transistors, 26
FIFO. *See* First-in-first-out queue. *See also* Queue
Finite state machines (FSMs), 117–133
design example, 117–123
divide-by-3, 207–208
factoring, 129–132
HDLs and, 206–213
Mealy machines. *See* Mealy machines
Moore machines. *See* Moore machines
Moore versus Mealy machines, 126–129
state encodings, 123–126
state transition diagram, 118–119
First-in-first-out (FIFO) queue, 508
Fixed-point numbers, 249–250
Flash memory, 263–264
Flip-flops, 103–112, 257. *See also* Registers
comparison with latches, 106, 112
D, 108
enabled, 109–110
register, 108–109
resettable, 110, 427
asynchronous, 192
synchronous, 192
transistor count, 108
transistor-level, 110–111
Floating (Z), 69–71
Floating-point numbers, 250–253

addition, 252–255. *See also* Addition
converting binary or decimal to. *See* Number conversions
division, 253. *See also* Division
double-precision, 251–252
FDIV bug. *See* FDIV bug
floating-point unit (FPU), 253
instructions, 340–341
rounding, 252
single-precision, 251–252
special cases
infinity, 251
not a number (NaN), 251
Forwarding, 408–409
Fractional numbers, 274
FPGA. *See* Field programmable gate array
Frequency, 135. *See also* Clock period
FSM. *See* Finite state machines
Full adder, 52, 178. *See also* Adder, Addition
HDL using always/process, 197
using nonblocking assignments, 204
Fully associative cache, 475–476
Funct field, 299–300
Functional specification, 51
Functions. *See* Procedure calls
Fuse, 263

G

Gates, 19–22
AND, 20–22, 32–33
NAND, 21, 32
NOR, 21, 32–33
OR, 21
transistor-level implementation, 262
XOR, 21
XNOR, 21
Gedanken-Experiments on Sequential Machines (Moore), 111
Generate signal, 235, 237
Glitches, 75, 88–91
Global pointer (`$gp`), 294, 331. *See also* Static data segment
Gray, Frank, 65
Gray codes, 65, 72
Gulliver's Travels (Swift), 297

H

Half word, 339
Half adder, 233–234
Hard disk, 484
Hard drive. *See* Hard disk
Hardware reduction, 66–67
Hardware description languages (HDLs), 167–230
assignment statements, 177
behavioral modeling, 171–184
combinational logic, 171–185, 195–206
bit swizzling, 182
bitwise operators, 171–174
blocking and nonblocking assignments, 201–206
case statements, 198–199
conditional assignment, 175–176
delays, 182–183
if statements, 199–201
internal variables, 176–178
numbers, 179
precedence, 178–179
reduction operators, 174
synthesis tools. *See* Synthesis Tools
Verilog, 201
VHDL libraries and types, 183–185
Z's and X's, 179–182
finite state machines, 206–213
generate statement, 213
generic building blocks, 426
invalid logic level, 181
language origins, 168–169
modules, 167–168
origins of, 168–169
overview, 167
parameterized modules, 211–213
representation, 421–431
sequential logic, 190–195, 205
enabled registers, 193
latches, 195
multiple registers, 194–195
registers, 190–191. *See also* Registers
resettable registers, 191–193
simulation and synthesis, 169–171
single-cycle processor, 422
structural modeling, 185–189
testbenches, 214–218, 428–431
Hazards, 75, 406–418. *See also* Glitches
control hazards, 413–416
data hazards 408–413
solving, 408–416
forwarding, 408–410
stalls, 410–413
WAR. *See* Write after read
WAW. *See* Write after write
Hazard unit, 408, 411, 419
Heap, 331
HDLs. *See* Hardware description languages
Hennessy, John, 290, 364
Hexadecimal numbers, 11–13
to binary conversion table, 12
Hierarchy, 6, 189
HIGH, 23. *See also* 1, ON
High-level programming languages, 290–294
translating into assembly, 290–291
compiling, linking, and launching, 330–331
Hit, 466
High impedance. *See* Floating, Z
High Z. *See* Floating, High impedance, Z
Hold time, 133–154

I

I-type instruction, 301–302
IA-32 microprocessor, 290, 341–349, 447–453
branch conditions, 346
cache systems, 499–500
encoding, 346–348
evolution, 448, 500
instructions, 344, 345
memory and input/output (I/O) systems, 499–502
operands, 342–344
programmed I/O, 502
registers, 342
status flags, 344
virtual memory, 501
IEEE 754 floating-point standard, 251
Idempotency, 58
Idioms, 171
IEEE, 169

If statements, 199–201, 310
If/else statements, 311
ILP. *See* Instruction-level parallelism
Immediates, 298
Immediate addressing, 327
Information, amount of, 8
IorD, 385
I/O (input/output), 337, 494–502
  communicating with, 494–495
  device driver, 496, 498
  devices, 494–496. *See also* Peripheral devices
  memory interface, 494, 502
  memory-mapped I/O, 494–499
Inputs, asynchronous, 144–145
Instruction encoding. *See* Machine Language
Instruction register (IR), 383, 390
Instruction set architecture (ISA), 289–361. *See also* MIPS instruction set architecture
Input/output blocks (IOBs), 268
Input terminals, 51
Institute of Electrical and Electronics Engineers, 250
Instruction decode, 401–402
Instruction encoding. *See* Instruction format
Instruction format
  F-type, 340
  I-type, 301–302
  J-type, 302
  R-type, 299–300
Instruction-level parallelism (ILP), 443, 446
Instruction memory, 365
Instruction set. *See* Instruction set architecture
Instructions. *See also* Language
  arithmetic/logical, 304–308
  floating-point, 340–341
  IA-32, 344–346
  I-type, 301–302
  J-type, 302
  loads. *See* Loads
  multiplication and division, 308
  pseudoinstructions, 336–337
  R-type, 299–300
  set less than, 339
  shift, 306–307
  signed and unsigned, 338–339
Intel, 30, 111, 290, 348, 367
  Inverter. *See* NOT gate
Integrated circuits (ICs), 26, 137, 515, 532
  costs, 137, 169
  manufacturing process, 515
Intel. *See* IA-32 microprocessors
Interrupts. *See* Exceptions
An Investigation of the Laws of Thought (Boole), 8
Involution, 58
IOBs. *See* Input/output blocks
I-type instructions, 301–302

J

Java, 316. *See also* Language
JTA. *See* Jump target address
J-type instructions, 302
Jump, 309–310. *See also* Branch, unconditional, Programming
Jump target address (JTA), 329

K

K-maps. *See* Karnaugh maps
Karnaugh, Maurice, 64, 71
Karnaugh maps (K-maps), 71–79
  logic minimization using, 73–76
  prime implicants, 61, 74
  seven-segment display decoder, 75–77
  with "don't cares," 78
  without glitches, 91
Kilby, Jack, 26
Kilobyte, 14
K-maps. *See* Karnaugh maps

L

Labels, 308–309
Language. *See also* Instructions
  assembly, 290–299
  high-level, 290–294
  machine, 299–304
  mnemonic, 291
  translating assembly to machine, 300
Last-in-first-out (LIFO) queue, 321. *See also* Stack, Queue
Latches, 103–112
  comparison with flip-flops, 106, 112
  D, 107
  SR, 105–107
  transistor-level, 110–111
Latency, 149
Lattice, 27
Leaf procedures, 324–325
Least recently used (LRU) replacement, 478–479
Least significant bit (lsb), 13
Least significant byte (LSB), 296
LIFO. *See* Last-in-first-out queue
Literal, 54, 61, 167
Little-Endian, 296–297
Load, 255
  byte (`lb`), 317
  byte unsigned (`lbu`), 317
  half (`lh`), 339
  immediate (`li`), 336
  upper immediate (`lui`), 308
  word (`lw`), 295–296
Loading, 335
Locality, 464
Local variables, 326–327
Logic, 62–65. *See also* Multilevel combinational logic; Sequential logic design
  bubble pushing. *See* Bubble pushing
  combinational. *See* Combinational logic
  families, 529–531
  gates. *See* Logic gates
  hardware reduction. *See* Hardware reduction, Equation simplification
  multilevel. *See* Multilevel combinational logic
  programmable, 516–523
  sequential. *See* Sequential logic
  synthesis, 170–171
  two-level, 65–66
  using memory arrays, 264. *See also* Logic arrays
Logic arrays, 266–274
  field programmable gate array, 268–272
  programmable logic array, 266–268
  transistor-level implementations, 273–274
Logic families, 25, 529–531

compatibility, 26
specifications, 529, 531
Logic gates, 19–22, 173
buffer, 20
delays, 183
multiple-input gates, 21–22
two-input gates, 21
types
AND. *See* AND gate
AOI (and-or-invert). *See* And-or-invert gate
NAND. *See* NAND gate
NOR. *See* NOR gate
NOT. *See* NOT gate
OAI (or-and-invert). *See* Or-and-invert gate
OR. *See* OR gate
XOR. *See* XOR gate
XNOR. *See* XNOR gate
Logic levels, 22–23
Logical operations, 304–308
Lookup tables (LUTs), 268
Loops, 311–314
for, 313
while, 312–313
LOW, 23. *See also* 0, OFF
Low Voltage CMOS Logic (LVCMOS), 25
Low Voltage TTL Logic (LVTTL), 25
LRU. *See* Least recently used replacement
LSB. *See* Least significant byte
LUTs. *See* Lookup tables
LVCMOS. *See* Low Voltage CMOS Logic
LVTTL. *See* Low Voltage TTL Logic

M

Machine language, 299–304
function fields, 299
interpreting code, 302–303
I-type instructions, 301–302
J-type instructions, 302
opcodes, 299–303
R-type instructions, 299–300
stored programs, 303–304
translating from assembly language, 300–302
translating into assembly language, 303
Main decoder, 374–379
Main memory, 466–469
Mapping, 470
Mantissa, 250, 252–253. *See also* Floating-point numbers
Masuoka, Fujio, 263
MCM. *See* Multichip module
Mealy, George H., 111
Mealy machines, 126–129, 130, 210
combined state transition and output table, 127
state transition diagram, 118–119
timing diagram, 131
Mean time between failures (MTBF), 146
Memory, 51, 295–298
access, 298
average memory access time (AMAT), 467
cache. *See* Caches
DRAM. *See* Dynamic random access memory
hierarchy, 466
interface, 464
main, 466–469
map, 330–331
dynamic data segment, 331
global data segment, 330–331
reserved segment, 331
text segment, 330
nonvolatile, 259–260
performance, 465
physical, 466, 485–486
protection, 491. *See also* Virtual memory
RAM, 259
ROM, 259
separate data and instruction, 430–431
shared, 71
stack. *See* Stack
types
flip-flops, 105–112
latches, 105–112
DRAM, 257
registers, 108–109
register file, 261–262
SRAM, 257
virtual, 466. *See also* Virtual memory
volatile, 259–261
word-addressable, 29. *See* Word-addressable memory
Memory arrays, 257–266
area, 261
bit cells, 258
delay, 261
DRAM. *See* Dynamic random access memory
HDL code for, 264–266
logic implementation using, 264. *See also* Logic arrays
organization, 258
overview, 257–260
ports, 259
register files built using, 261–262
types, 259–260
DRAM. *See* Dynamic random access memory
ROM. *See* Read only memory
SRAM. *See* Static random access memory
Memory-mapped I/O (input/output), 494–498
address decoder, 495–496
communicating with I/O devices, 495–496
hardware, 495
speech synthesizer device driver, 498
speech synthesizer hardware, 496–497
SP0256, 496
Memory protection, 491
Memory systems, 463–512
caches. *See* Caches
IA-32, 499–502
MIPS, 470–478
overview, 463–467
performance analysis, 467–468
virtual memory. *See* Virtual memory
Mercedes Benz, 268
Metal-oxide-semiconductor field effect transistors (MOSFETs), 26–31. *See also* CMOS, nMOS, pMOS, transistors
Metastability, 143–144
MTBF. *See* Mean time between failures
metastable state, 143
probability of failure, 145
resolution time, 144
synchronizers, 144–146
A Method of Synthesizing Sequential Circuits (Mealy), 111

Microarchitecture, 290, 363–461
advanced. *See* Advanced microarchitecture
architectural state. *See* Architectural State. *See also* Architecture
design process, 364–366
exception handling, 431–434
HDL representation, 421–431.
IA-32. *See* IA-32 microprocessor
instruction set. *See* Instruction set
MIPS. *See* MIPS microprocessor
overview, 363–366
performance analysis, 366–368
types,
advanced. *See* Advanced microarchitecture
multicycle. *See* Multicycle MIPS processor
pipelined. *See* Pipelined MIPS processor
single-cycle. *See* Single-cycle MIPS processor
Microprocessors, 3, 13
advanced. *See* Advanced microarchitecture
chips. *See* Chips
clock frequencies, 124
IA-32. *See* IA-32 microprocessor
instructions. *See* MIPS instructions, IA-32 instructions
MIPS. *See* MIPS microprocessor
Microsoft Windows, 501
Minterms, 54
Maxterms, 54
MIPS (Millions of instructions per second). *See* Millions of instructions per second
MIPS architecture. *See* MIPS instruction set architecture (ISA)
MIPS assembly language. *See also* MIPS instruction set architecture
addressing modes, 327–329
assembler directives, 333
instructions, 290–292
logical instructions, 304–308
mnemonic, 291
operands, 292–298
procedure calls, 319–327
table of instructions, 336
translating machine language to, 303
translating to machine language, 300
MIPS instruction set architecture (ISA)
addressing modes, 327–329
assembly language, 290–299
compiling, assembling, and loading, 330–335
exceptions, 337–338
floating-point instructions, 340–341
IA-32 instructions, 344–346
machine language, 299–304
MIPS instructions, 290–292
overview, 289–290
programming, 304–327
pseudoinstructions, 336–337
signed and unsigned instructions, 338–339
SPARC, 364
translating and starting a program. *See* Translating and starting a program
MIPS instructions, 551–554
formats
F-type, 340
I-type, 301–302
J-type, 302
R-type, 299–300
tables of, 552–554
opcodes, 552
R-type funct fields, 553–554
types,
arithmetic, 304–308
branching. *See* Branching
division, 308
floating-point, 340–341
logical, 304–308
multiplication, 308
pseudoinstructions, 336–337
MIPS microprocessor, 364
ALU, 242–244
multicycle. *See* Multicycle MIPS processor
pipelined. *See* Pipelined MIPS processor
single-cycle. *See* Single-cycle MIPS processor
MIPS processor. *See* MIPS microprocessor
MIPS registers, 293–294, 308
nonpreserved, 322–324
preserved, 322–324
table of, 294
MIPS single-cycle HDL implementation, 421–431
building blocks, 426–428
controller, 423
datapath, 425
testbench, 429
top-level module, 430
Misses, 466, 481
AMAT. *See* Average memory access time
cache, 466
capacity, 481
compulsory, 481
conflict, 481
page fault, 485
Miss penalty, 476
Miss rate, 467
Mnemonic, 291
Modeling, structural. *See* Structural modeling
Modeling, behavioral. *See* Behavioral modeling
Modularity, 6, 168
Modules, in HDL 167–168. *See also* Hardware description languages
behavioral, 168, 171
parameterized, 211–213
structural, 168
Moore, Edward F., 111
Moore, Gordon, 30
Moore machine, 126–129, 130, 208, 209
output table, 128
state transition diagram, 127
state transition table, 128
timing diagram, 131
Moore's law, 30
MOSFETs. *See* Metal-oxide-semiconductor field effect transistors
Most significant bit (msb), 13
Most significant byte (MSB), 296
Move from hi (`mfhi`), 308
Move from lo (`mflo`), 308
Move from coprocessor 0 (`mfc0`), 338
msb. *See* Most significant bit
MSB. *See* Most significant byte
MTBF. *See* Mean time before failure
Multichip module (MCM), 499
Multicycle MIPS processor, 366, 381–400
control, 388–394
control FSM, 394–395
datapath, 382–388
performance analysis, 397–400
Multilevel combinational logic, 65–69. *See also* Logic
Multilevel page table, 493
Multiplexers, 79–82, 175, 176, 428

instance, 188
logic, 80–82
parameterized, 211. *See also* Hardware description languages
symbol and truth table, 79
timing, 87–88
with type conversion, 185
wide, 80
Multiplicand, 246
Multiplication, 246–247. *See also* Arithmetic, Multiplier
architecture, 339
instructions, 308
Multiplier, 246–247
multiply (`mult`), 308, 339
multiply unsigned (`multu`), 339
Multiprocessors, 447
chip, 448
Multithreading, 446
Mux. *See* Multiplexers

N

NaN. *See* Not a number
Negation, 340. *See also* Taking the two's complement
Not a number (NaN), 251
NAND gate, 21, 32
`nor`, 179
NOR gate, 21, 32–33
NOT gate, 24, 172. *See also* Inverter
in HDL using always/process, 196
Nested procedure calls, 324–326
Netlist, 170
Next state, 115
Nibbles, 13–14
nMOS, 28–31
nMOS transistors, 28–31
No operation. *See* `nop`
Noise margins, 23, 24
`nop`, 336
Noyce, Robert, 26
Number conversion, 9–19 *See also* Number systems, Binary numbers
binary to decimal, 10–11
binary to hexadecimal, 12
decimal to binary, 11
decimal to hexadecimal, 13
taking the two's complement, 15, 240
Number systems, 9–19, 249–253
addition. *See* Addition
binary numbers. *See* Binary numbers
comparison of, 18–19
conversion of. *See* Number conversions
decimal numbers. *See* Decimal numbers
estimating powers of two, 14
fixed-point, 249–250. *See* Fixed-point numbers
floating-point, 250–253. *See* Floating-point numbers
in hardware description languages, 179
hexadecimal numbers. *See* Hexadecimal numbers
negative and positive, 15–19
rounding, 252. *See also* Floating-point numbers
sign bit, 16
signed, 15–19. *See also* Signed binary numbers
sign/magnitude numbers. *See* Sign/magnitude numbers
two's complement numbers. *See* Two's complement numbers
unsigned,

O

OAI gate. *See* Or-and-invert gate
Object files, 331–334
Octal numbers, 180
OFF, 23. *See also* 0, LOW
Offset, 295–296
ON, 23. *See also* 1, HIGH, Asserted
One-cold, 124
One-hot, 82
Opcode, 299
Operands, 292–298
Operators
bitwise, 171–174
or immediate (`ori`), 308
OR gate, 21
Or-and-invert (OAI) gate, 43
precedence table of, 179
reduction, 174
Out-of-order execution, 443
Out-of-order processor, 441–443
Output devices, 466
Output terminals, 51
Overflow, 15
detecting, 15
with addition, 15

P

Page, 485
size, 485
Page fault, 485
Page offset, 486–488
Page table, 486
Parity, 22
Parallelism, 149–153
pipelining. *See* Pipelining, Pipelined MIPS processor
SIMD. *See* Single instruction multiple data unit
spatial and temporal, 151
vector processor. *See* Vector processor
Patterson, David, 290, 364
PCBs. *See* Printed circuit boards
PC-relative addressing, 327–328. *See also* Addressing modes
PCSrc, 281, 387–390
PCWrite, 385
Pentium processors, 449–452. *See also* Intel, IA-32
Pentium 4, 452
Pentium II, 450, 499
Pentium III, 450, 451, 499
Pentium M, 453
Pentium Pro, 450, 499, 501
Perfect induction, 60
Performance, 366–368
Peripheral devices. *See* I/O devices
Perl programming language, 20
Physical memory, 466, 485–486
Physical pages, 486–494
Physical page number, 486
Pipelined MIPS processor, 151, 366, 401–421. *See also* MIPS, Architecture, Microarchitecture
control, 405–406
datapath, 404
forwarding, 408–410
hazards. *See* Hazards
performance analysis, 418–421
processor performance comparison, 420
stalls, 410–413. *See also* Hazards
timing, 402

Pipelining hazards. *See* Hazards
PLAs. *See* Programmable logic arrays
Plastic leaded chip carriers (PLCCs), 531–532
PLCCs. *See* Plastic leaded chip carriers
PLDs. *See* Programmable logic devices
pMOS, 29–30
pMOS transistors, 28–31
Pointer, 321
  global, 331
  stack, 321
Pop, 345–346. *See also* Stack
Ports, 259
POS. *See* Product of sums form
Power consumption, 34–35
Prediction. *See* Branch prediction
Prefix adder, 237–239
Preserved registers, 322–324
Prime implicants, 61, 74
Printed circuit boards (PCBs), 533–534
Procedure calls, 319–327
  arguments and variables, 326–327
  nested, 324–326
  preserved versus nonpreserved registers, 323–324
  returns, 319–320
  return values, 320
  stack frame, 322
  stack usage, 321–322
Processors. *See* Microprocessors
Product-of-sums (POS) canonical form, 56
Program counter (`PC`), 365
Programmable logic arrays (PLAs), 63, 266–268, 520–521
Programmable logic devices (PLDs), 268
Programmable read only memories (PROMs), 516–520. *See also* Read only memories
Programming, 304–327
  arithmetic/logical instructions. *See* Arithmetic, 304–308
  arrays. *See* Arrays
  branching. *See* Branching
  conditional statements. *See* Conditional statements
  constants, 307–308
  immediates, 298
  loops. *See* Loops
  procedure calls. *See* Procedure calls
  shift instructions, 306–307
  translating and starting a program, 331–335
Programming languages, 290–294
Propagation delay, 84
Protection, memory. *See* Memory protection
Proving Boolean theorems. *See* Perfect induction
PROMs. *See* Programmable read only memories
Pseudo-direct addressing, 328–329. *See also* Addressing modes
Pseudo-nMOS logic, 33–34. *See also* Transistors
Pseudoinstructions, 336–337
Push, 67–69. *See also* Stack

Q

Queue, 321
  FIFO. *See* First-in-first-out queue
  LIFO. *See* Last-in-first-out queue
Q output. *See* Sequential logic design

R

R-type instructions, 299–300
RAM. *See* Random access memory
Random access memory (RAM), 257, 262–264. *See also* Memory arrays
  synthesized, 265
Read only memory, 199, 257, 262–264. *See also* Memory arrays
  EPROM. *See* Erasable programmable read only memory
  EEPROM. *See* Electrically erasable programmable read only memory
  flash memory. *See* Flash memory
  PROM. *See* Programmable read only memory
  transistor-level implementation, 273
Read/write head, 484
Recursive procedure calls, 324–325. *See also* Procedure calls
Reduced instruction set computer (RISC), 292, 364
Reduction operators, 174. *See also* Hardware description languages, Verilog
RegDst, 373–374
Register-only addressing, 327. *See also* Addressing modes
Register renaming, 443–445. *See also* Advanced microarchitecture
Register(s), 190–191, 261–262, 292–294. *See also* Flip-flops, Register file
  arguments (`$a0-$a3`)
  assembler temporary (`$at`)
  enabled. *See* Enabled registers
  file, 261
  global pointer (`$gp`), 331
  multiple, 194–195
  program counter (`PC`), 365
  preserved and nonpreserved, 322–324
  renaming, 443–445
  resettable. *See* Resettable registers
    asynchronous, 192
    synchronous, 192
  return address (`$ra`), 319–320
Register set, 294. *See also* Register file, MIPS registers, IA-32 registers
Regularity, 6, 188
RegWrite, 371
Replacement policies, 492. *See also* Caches, Virtual memory
Resettable registers,191–193
  asynchronous. *See* Asynchronous resettable registers
  synchronous. *See* Synchronous resettable registers
Return address (`$ra`), 319–320
Ripple-carry adder, 234
RISC. *See* Reduced instruction set computer
ROM, *See* Read Only Memory
Rotators, 244–246
Rounding, 252

S

Scalar processor, 438
Scan chains, 255. *See also* Shift registers
Schematic, 62–65
Scientific notation, 249–250

*See*k time, 484
Segments, memory, 330–331
Semiconductor industry, sales, 3
Semiconductors, 27. *See also* Transistors
  CMOS. *See* Complementary metal oxide silicon
  diodes. *See* Diodes
  transistors. *See* Transistors
  MOSFET. *See* Metal oxide silicon field effect transistors
  nMOS. *See* nMOS transistors
  pMOS. *See* pMOS transistors
  pseudo nMOS. *See* Pseudo nMOS
Sensitivity list, 190, 191, 196
Sequential building blocks. *See* Sequential logic
Sequential logic, 103–165, 190–195, 254–257
  enabled registers, 193
  latches, 103–112, 195
  multiple registers, 194–195
  overview, 103
  registers, 190–191
  shift registers, 255–257
  synchronous, 113–117
  timing of, 133–149. *See also* Timing
set if less than (`slt`), 313
set if less than immediate (`slti`), 339
set if less than immediate unsigned (`sltiu`), 339
set if less than unsigned (`sltu`), 339
Setup time, 133, 135–136
Seven-segment display decoder, 75–77
  with "don't cares," 78
Shared memory, 71
Shift amount (`shamt`), 245
shift left logical (`sll`), 306
shift left logical variable (`sllv`), 306
shift right arithmetic (`sra`), 306
shift right arithmetic variable (`srav`), 306
shift right logical (`srl`), 306
shift right logical variable (`srlv`), 306
Shifters, 244–246
  arithmetic, 244
  logical, 244
Shift instructions, 306–307
Shift registers, 255–257
Short path, 86
Sign/magnitude numbers, 15–16
Sign bit, 16
Sign extension, 18
Significand. *See* Mantissa
Silicon (Si), 27
Silicon dioxide (SO_2), 28
Silicon Valley, 26
Simplicity, 291–292
SIMD. *See* Single instruction multiple data units
Single-cycle MIPS processor, 366, 368–381. *See also* MIPS microprocessor, MIPS architecture, MIPS microarchitecture
  control, 374–377
    ALU decoder truth table. *See* ALU decoder
    Main decoder truth table. *See* Main decoder
  datapath, 368–374
  HDL representation, 422
  operation, 376–377 performance analysis, 380–381
  timing, 402
Single instruction multiple data (SIMD) units, 438, 445
Slash notation, 53
SPARC architecture, 364
SRAM. *See* Static random access memory
SP0256, 496
Spatial locality, 464
Speech synthesis, 496–498
  device driver, 498
  SP0256, 496
Stack, 321–322. *See also* Memory map, Procedure calls, Queue
  dynamic data segment, 331
  frame, 322
  LIFO. *See* Last-in-first-out queue
  pointer, 321
Stalls, 410–413. *See also* Hazards
Static discipline, 24–26
Static random access memory (SRAM), 257, 260
Status flags, 344
Stored program concept, 303–304
Stores
  store byte (`sb`), 318
  store half (`sh`), 553
  store word (`sw`), 296
Strings, 318
Structural modeling, 185–189
subtract (`sub`), 291
subtract unsigned (subu), 339
Subtraction, 17–18, 240, 291
Subtractor, 240
Sum-of-products (SOP) canonical form, 54–55
Sun Microsystems, 364
Superscalar processor, 438–440
Supply voltage, 22
Swap space, 492
Swift, Jonathan, 297
Switch/case statements, 311
Symbol table, 333
Synchronizers, 144–146
  asynchronous inputs, 146
  MTBF. *See* Mean time before failure
  probability of failure. *See* Probability of failure
Synchronous resettable registers, 192
  HDL for, 192
Synchronous sequential logic, 113–117
  problematic circuits, 113–114
Synthesis, 78–79, 186, 187, 188, 189, 190, 193, 194, 195, 199, 200
Synthesis Tools, 195

T

Taking the two's complement, 16
Temporal locality, 464
Testbenches, 171, 214–218, 428–431
  self-checking, 215
  with test vector file, 216
Text segment, 330
Theorems. *See* Boolean Theorems
Thin small outline package (TSOP), 531
Threshold voltage, 29
Throughput, 149
Timing
  analysis, 137–138
  delay, 86–87
  glitches, 88–91
  of sequential logic, 133–149
    clock skew, 140–143
    dynamic discipline, 134
    hold time. *See* Hold time
    hold time constraint, 136–137. *See* Hold time constraint, Hold time violation
    hold time violation. *See* Hold time violation, Hold time constraint, 139

Timing (*Continued*)
metastability, 143–144
setup time. *See* Setup time
setup time constraint, 135–136
setup time violations
resolution time, 146–149. *See also* Metastability
synchronizers, 144–146
system timing, 135–140
specification, 51
TLB. *See* Translation lookaside buffer
Token, 149
Transistors, 23, 28–31, 34
CMOS. *See* Complement metal oxide silicon
nMOS. *See* nMOS
pMOS. *See* pMOS
Transistor-Transistor Logic (TTL), 25
Translation lookaside buffer (TLB), 490
Translating and starting a program, 331–336
Transmission gates, 33. *See also* Transistors
Transmission lines, 534–546
reflection coefficient, 544–545
Z0, 543–544
matched termination, 536–538
mismatched termination, 539–541
open termination, 538–539
proper terminations, 542
short termination, 539
when to use, 41–542
Tristate buffer, 70–71
Truth tables, 55, 56, 60, 61, 177
ALU decoder, 376
"don't care," 77
main decoder, 376, 379
multiplexer, 79
seven-segment display decoder, 76
SR latch, 106
with undefined and floating inputs, 181
TSOP. *See* Thin small outline package
TTL. *See* Transistor-Transistor Logic
Two-level logic, 65–66
Two's complement numbers, 16–18. *See also* Binary numbers

U

Unconditional branches, 308. *See also* Jumps
Unicode, 316. *See also* ASCII
Unit under test (UUT), 201
Unity gain points, 24
Unsigned numbers, 18
Use bit, 478–479
UUT. *See* Unit under test

V

Valid bit, 472. *See also* Caches, Virtual memory
Vanity Fair (Carroll), 65
V_{CC}, 23
V_{DD}, 23
Vector processors, 438. *See also* Advanced microarchitecture
Verilog, 167, 169, 172, 173, 174, 175, 176, 178, 180, 181, 201, 203, 205
3:8 decoder, 199
accessing parts of busses, 189
adder, 426
ALU decoder, 424
AND, 168
architecture body, 168
assign statement, 168, 197, 178
asynchronous reset, 192
bad synchronizer with blocking assignments, 206
bit swizzling, 182
blocking assignment, 202
case sensitivity, 174
casez, 201
combinational logic, 168, 202
comments, 174
comparators, 242
continuous assignment statement, 173
controller, 423
counter, 254
datapath, 425
default, 198
divide-by-3 finite state machine, 207, 208
D latch, 195
eight-input AND, 174
entity declaration, 168
full adder, 178
using always/process, 197
using nonblocking assignments, 204
IEEE_STD_LOGIC_1164, 168, 183
IEEE_STD_LOGIC_SIGNED, 183
IEEE_STD_LOGIC_UNSIGNED, 183
inverters, 172
using always/process, 196
left shift, 427
library use clause, 168
logic gates, 173
with delays, 183
main decoder, 424
MIPS testbench, 429
MIPS top-level module, 430
multiplexers, 175, 176, 428
multiplier, 247
nonblocking assignment, 191, 202
NOT, 168
numbers, 180
operator precedence, 179
OR, 168
parameterized
N:2^N decoder, 212
N-bit multiplexer, 211
N-input AND gate, 213
pattern recognizer
Mealy FSM, 210
Moore FSM, 209
priority circuit, 201
RAM, 265
reg, 191, 194
register, 191
register file, 42
resettable flip-flop, 427
resettable enabled register, 193
resettable register, 192
ROM, 266
self-checking testbench, 215
seven-segment display decoder, 198
shift register, 256
sign extension, 427
single-cycle MIPS processor, 422
STD_LOGIC, 168, 183
statement, 173
structural models

2:1 multiplexer, 188
4:1 multiplexer, 187
subtractor, 241
synchronizer, 194
synchronous reset, 192
testbench, 214
with test vector file, 216
tristate buffer, 180
truth tables with undefined and floating inputs, 181
type declaration, 168
wires, 178
VHDL libraries and types, 167, 169, 172, 173, 174, 175, 176, 178, 180, 181, 183–185, 203, 205
3:8 decoder, 199
accessing parts of busses, 189
adder, 426
architecture, 187
asynchronous reset, 192
bad synchronizer with blocking assignments, 206
bit swizzling, 182
boolean, 183
case sensitivity, 174
clk'event, 191
combinational logic, 168
comments, 174
comparators, 242
concurrent signal assignment, 173, 178
controller, 423
CONV_STD_LOGIC_VECTOR, 212
counter, 254
datapath, 425
decoder, 424
divide-by-3 finite state machine, 207, 208
D latch, 195
eight-input AND, 174
expression, 173
full adder, 178
using always/process, 197
using nonblocking assignments, 204
generic statement, 211
inverters, 172
using always/process, 196
left shift, 427
logic gates, 173
with delays, 183
main decoder, 424
MIPS testbench, 429
MIPS top-level module, 430
multiplexers, 175, 176, 428
multiplier, 247
numbers, 180
operand, 173
operator precedence, 179
others, 198
parameterized
N-bit multiplexer, 211
N-input AND gate, 213
N:2^N decoder, 212
pattern recognizer
Mealy FSM, 210
Moore FSM, 209
priority circuit, 201
process, 191
RAM, 265
register, 191
register file, 426
resettable enabled register, 193
resettable flip-flop, 427
resettable register, 192
RISING_EDGE, 191
ROM, 266
selected signal assignment statements, 176
self-checking testbench, 215
seven-segment display decoder, 198
shift register, 256
sign extension, 427
signals, 178, 194
simulation waveforms with delays, 170, 183
single-cycle MIPS processor, 422
STD_LOGIC_ARITH, 212
structural models
2:1 multiplexer, 188
4:1 multiplexer, 187
subtractor, 241
synchronizer, 194
synchronous reset, 192
testbench, 214
with test vector file, 216
tristate buffer, 180
truth tables with undefined and floating inputs, 181
VHSIC, 169
Virtual address, 485–490
Virtual memory, 484–494
address translation, 486–488
IA-32, 501
memory protection, 491
pages, 485
page faults, 485
page offset, 486–488
page table, 488–489
multilevel page tables, 492–494
replacement policies, 492
translation lookaside buffer (TLB), 490. *See* Translation lookaside buffer
write policies, 482–483
Virtual pages, 485
Virtual page number, 487
Volatile memory, 259. *See also* DRAM, SRAM, Flip-flops
Voltage, threshold, 29
V_{SS}, 23

W

Wafers, 28
Wall, Larry, 20
WAR. *See* Write after read
WAW. *See* Write after write
While loop, 312–313
White space, 174
Whitmore, Georgiana, 7
Wire, 63
Word-addressable memory, 295
Write policies, 482–483
Write after read (WAR) hazard, 442. *See also* Hazards
Write after write (WAW) hazard, 442–443. *See also* Hazards

X

XOR gate, 21
XNOR gate, 21
Xilinx FPGA, 268
X. *See* Contention, Don't care.,

Z

Z,. *See* Floating
Zero extension, 302