INTRODUCTION TO ELECTRONICS

CONTENTS AT A GLANCE

In this appendix, I wanted to introduce you to basic electronic theory, different component types and how they are built into circuits. This appendix (and "Introduction to Programming") are not meant to be complete explanations of the topics, but should give you a background on the applications presented in "Programming and Customizing PICmicro® Microcontrollers.

Engineering Mathematics

In this book, I use a lot of the different aspects of mathematics to find answers in the operation of the circuits and application code that is presented here. For the most part, I have tried to keep the mathematics at a high-school level and have avoided referencing college or university level mathematics and concepts, unless I couldn't avoid it. To work through the information in this book, you should be reasonably well grounded in the disciplines of numbering systems, multiplication, division, exponents, and logarithms. I have utilized my knowledge of statistics and calculus to help find the answers needed to solve some of the problems that I came across in developing the information for this book, but I have avoided solving equations and values that I present should just be accepted as "givens."

You should be able to develop your own PICmicro® MCU applications with no more than junior-level high-school mathematics. To be successful, you should understand num-

bering systems and working with exponents very well and, in this section, I review the skills and basic knowledge that you should have. Some people might consider the basic skills to be able to develop PICmicro® MCU applications to be no more than the ability of counting from zero (and not) one and being able to multiply exponents together. But there really is a lot more to creating electronic circuits and computer application code with the calculations necessary to make them work properly.

All numbers can be represented as the sum of multiple exponents. For example, decimal 123 can be represented as 100 plus two 10 plus three ones.

The basic numbering system used by people is ten, so sum can be expressed as a series of exponents:

$$123 = 1 \times (10^2) + 2 \times (10^1) + 3 \times (10^0)$$

It is important to remember that one is actually 10 (or any other base value) raised to the exponent zero.

The usual base that we humans use is 10—probably because we have 10 fingers, which makes it easy to count and keep track of things. Other bases can be used and are often used in electronics and computer science.

For example, if we had eight fingers, the "base" we would use is eight. The actual numbers in each digit would be zero through seven and not eight. Eight would not exist in a base-eight numbering system (just as 10 doesn't exist in our base-10 numbering system), it would actually be written as 10. This can be somewhat confusing. To avoid problems, the base is usually identified when a number is presented in this (and most other) books.

The term *base* is also known as *radix*. This term is used by Microchip when working with the PICmicro® MCU.

In the base-eight case, the *ones* would be the digits zero through seven, the next digit (the *eights*) would be zero through seven multiplied by eight, the second digit (the *sixty-fours*) would be zero through seven multiplied by eight times eight (eight to the power of two), and so on.

To convert between numbering systems, the value would be divided into individual digits until the actual result was found. Going back to the base 10 (which is usually known as *decimal*) number, it can be converted to base 8 (often known as *octal*) by working through each of the digits.

To start off, the decimal number is divided by powers of eight until they are larger than the number. Then the power is backed off and the next power down is divided into the number. This continues until there is an equivalent order of magnitude.

For example, if decimal 123 was to be converted into octal, the first operation would be to find the most-significant digit of the octal number of the same order of magnitude. Starting with an exponent of 0 (the *1*s), the conversion number would be compared until the base and exponent are greater than the number to be converted. For decimal 123, this operation is:

$$\text{Exponent } 0 - 8^0 = 1 < 123$$
$$\text{Exponent } 1 - 8^1 = 8 < 123$$
$$\text{Exponent } 2 - 8^2 = 64 < 123$$
$$\text{Exponent } 3 - 8^3 = 512 > 123$$

In this case, the fourth digit (exponent 3) is greater than the conversion value. This means that the octal equivalent will be three digits in length and the order of magnitude would be 64.

Now, the value of each digit is divided into the number until the number is less than the value of the digit. This is shown in Table 1, starting with the third digit (exponent 2).

So, 173 octal is the equivalent to 123 decimal. Notice that the octal number is greater than the equivalent decimal number; this is true for any conversion from a higher base to a lower base.

Although octal is a possible base for working with computer systems, I don't recommend it. Octal was popular in the 1970s and early 1980s because the "C" program language and DEC VAX mini-computer systems used it, but I find it to be a very confusing numeric base.

The most popular bases for working with computer systems is base 2 (binary) and base 16 (hexadecimal) because of the binary operation of digital systems. Conversions between decimal, binary, and hexadecimal numbers can be carried out by many common hand-held calculators or, once you are familiar with working with them, done in your head.

Digital logic circuits can only operate in one of two states, on or off, which is usually represented as *0* and *1,* respectively. These state circuits (known as *bits*) are usually grouped together into sets of four (nybbles) or eight (bytes). Grouping four bits together is the basis for providing a base-16 (hexadecimal) number, which the normal way to express binary values.

Decimal-to-binary conversions are carried out exactly the same way as was done converting a decimal number to an octal value. First, the exponent greater than the value is

TABLE 1 Conversion Of Decimal 123 To Octal

EXPONENT	TEST	REMAINDER	CURRENT OCTAL
2	123 − 8 ** 2 = 123 − 64 = 59	59	100
2	59 − 8 ** 2 = 59 − 64 = −5	Can't be Done	100
1	59 − 8 ** 1 = 59 − 8 = 51	51	110
1	51 − 8 ** 1 = 51 − 8 = 43	43	120
1	43 − 8 ** 1 = 43 − 8 = 35	35	130
1	35 − 8 ** 1 = 35 − 8 = 27	27	140
1	27 − 8 ** 1 = 27 − 8 = 19	19	150
1	19 − 8 ** 1 = 19 − 8 = 11	11	160
1	11 − 8 ** 1 = 11 − 8 = 3	3	170
1	3 − 8 ** 1 = 3 − 8 = −5	Can't be Done	170
0	3 − 8 ** 0 = 3 − 1 = 2	2	171
0	2 − 8 ** 0 = 2 − 1 = 1	1	172
0	1 − 8 ** 0 = 1 − 1 = 0	0	173
0	0 − 8 ** 0 = 0 − 1 = −1	Can't be Done	173

TABLE 2 Conversion Of Decimal 123 To Binary

EXPONENT	TEST	REMAINDER	CURRENT BINARY
6	123 − 2 ** 6 = 123 − 64 = 59	59	1000000
6	59 − 2 ** 6 = 59 − 64 = −5	Can't be Done	1000000
5	59 − 2 ** 5 = 59 − 32 = 27	27	1100000
5	27 − 2 ** 5 = 27 − 32 = −5	Can't be Done	1100000
4	27 − 2 ** 4 = 27 − 16 = 11	11	1110000
4	11 − 2 ** 4 = 11 − 16 = −5	Can't be Done	1110000
3	11 − 2 ** 3 = 11 − 8 = 3	3	1111000
3	3 − 2 ** 3 = 3 − 8 = −5	Can't be Done	1111000
2	3 − 2 ** 2 = 3 − 4 = −1	Can't be Done	1111000
1	3 − 2 ** 1 = 3 − 2 = 1	1	1111010
1	1 − 2 ** 1 = 1 − 2 = −1	Can't be Done	1111010
0	1 − 2 ** 0 = 1 − 1 = 0	0	1111011
0	0 − 2 ** 0 = 0 − 1 = −1	Can't be Done	1111011

found and the conversion value is generated from the lower values. This can be shown with decimal 123, which will only require eight bits to be represented. The eighth bit (two to the power 7, 128) is greater than the value we are converting. The conversion can use a table the same way the decimal-to-octal conversion was carried out (Table 2).

Normally, binary numbers are written in groups of eight bits, so the binary equivalent of 123 decimal is 01111011.

These eight bits are usually converted into two hexadecimal (base 16) digits. The hexadecimal digits are *0* through *9*, and then followed by A, B, C, D, E, and F for the remaining six digits. So, the hexadecimal number 7B can refer to the binary value 01111011.

The hexadecimal digits, *A* through *F*, are usually referred to by their phonetic names, which are listed in Table 3.

TABLE 3 Hexadecimal Letter Phonetic Names

LETTER	DECIMAL VALUE	PHONETIC NAME
A	10	"Able"
B	11	"Baker"
C	12	"Charlie"
D	13	"Dog"
E	14	"Easy"
F	15	"Fox"

I can go through the same conversion operation on decimal 123 to hexadecimal to show that four bits grouped together work out to the same value. The second digit of a hexadecimal number is groups of 16 and the third digit is 256. Thus, the decimal value *123* is only going to be two digits in size. The conversion operation is shown in Table 4.

In this case, going from a small base to a larger one, notice that the number of digits in the number is decreased (which is expected).

To differentiate between numbering systems, a prefix is placed in front of each number type or specialized data format is used to help avoid any confusion over what the value is. Microchip has specified the four data types and prefix/format for the PICmicro® MCU listed in Table 5.

This book uses decimal numbers almost exclusively. When I use another system, I will identify it by using the prefix. If no prefix is assigned, then the value is decimal. For binary numbers, I use the prefix *0b0* to follow with the convention started by *0x0* for hexadecimal numbers.

TABLE 4 Converting Decimal 123 To Hexadecimal

EXPONENT	TEST	REMAINDER	CURRENT HEXADECIMAL
1	123 − 16 ** 1 = 123 − 16 = 107	107	10
1	107 − 16 ** 1 = 107 − 16 = 91	91	20
1	91 − 16 ** 1 = 91 − 16 = 75	75	30
1	75 − 16 ** 1 = 75 − 16 = 59	59	40
1	59 − 16 ** 1 = 59 − 16 = 43	43	50
1	43 − 16 ** 1 = 43 − 16 = 27	27	60
1	27 − 16 ** 1 = 27 − 16 = 11	11	70
1	11 − 16 ** 1 = 11 − 16 = −5	Can't be Done	70
0	11 − 16 ** 0 = 11 − 1 = 10	10	71
0	10 − 16 ** 0 = 10 − 1 = 9	9	72
0	9 − 16 ** 0 = 9 − 1 = 8	8	73
0	8 − 16 ** 0 = 8 − 1 = 7	7	74
0	7 − 16 ** 0 = 7 − 1 = 6	6	75
0	6 − 16 ** 0 = 6 − 1 = 5	5	76
0	5 − 16 ** 0 = 5 − 1 = 4	4	77
0	4 − 16 ** 0 = 3 − 1 = 3	3	78
0	3 − 16 ** 0 = 3 − 1 = 2	2	79
0	2 − 16 ** 0 = 2 − 1 = 1	1	7A
0	1 − 16 ** 0 = 1 − 1 = 0	0	7B
0	0 − 16 ** 0 = 0 − 1 = −1	Can't be Done	7B

TABLE 5 Numbering System Prefixes

NUMBERING SYSTEM	PREFIX	DATA FORMAT
Decimal	.##	D′##′
Hexadecimal	0x##	H′##′
Binary	None	B′##′
Octal	None	O′##′

"Introduction to Programming" goes into more detail about working with the different numbering systems and some of the tricks that are used to convert them and work with constants within them without having to perform manual conversions.

When working with binary (or hexadecimal) values, notice that the first number that is valid is zero. This is different than what we are usually taught in grade school, where you start counting at one. In electronics, numbers are equivalent to states and the first one is zero (or 0b000000000 in binary). This is important to remember when you start working with electronics.

Counting can be confused by the concept of gray codes. *Gray codes* are a way to encode data in such a way that increasing a number only changes one bit at a time. Table 6 shows

TABLE 6 Four Bit "Gray Code" Example

DECIMAL	BINARY	GRAY CODE
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0100
6	0110	0101
7	0111	0111
8	1000	1111
9	1001	1011
10	1010	1010
11	1011	1110
12	1100	1100
13	1101	1101
14	1110	1001
15	1111	1000

a 16 number (four bit) progression in traditional decimal and binary counting, along with the gray codes associated with them.

The set of gray codes presented in Table 6 are arbitrary. The important thing to notice is that only one bit changes at a time and each of the 16 different states are represented, just not at the same time. Notice that going from state 15 back to state 0 only changes one bit as well.

Gray codes are used in such equipment optical encoders, where both the position and direction can be decoded by comparing two sequential codes. For example, if 0b00111 was received followed by 0b00101, it can be determined that the optical encoder's last position is *6* and it is turning in a *downward* (from high to low number) direction.

So far, I have described integer values, which do not have values less than one (fractions). Fractions can be expressed in binary as well as decimal values.

This is accomplished by using negative exponents to the base numbers. For example, the decimal value 123.4 can be represented using binary numbers, recognizing that 2^{-1} is 0.5 (decimal), 2^{-2} is 0.25, etc. Table 7 uses this progression to convert 123.4 decimal to B′1111011.011′.

As the negative binary exponents are worked through, notice that the result is not an even value. As you work through the fractions in the two's base, you will find that frac-

TABLE 7 Converting 123.4 Decimal To Binary

EXPONENT	TEST	REMAINDER	CURRENT BINARY
6	123.4 − 2 ** 6 = 123.4 − 64 = 59.4	59.4	1000000.000
6	59.4 − 2 ** 6 = 59.4 − 64 = −4.6	Invalid	1000000.000
5	59.4 − 2 ** 5 = 59.4 − 32 = 27.4	27.4	1100000.000
5	27.4 − 2 ** 5 = 27.4 − 32 = −4.6	Invalid	1100000.000
4	27.4 − 2 ** 4 = 27.4 − 16 = 11.4	11.4	1110000.000
4	11.4 − 2 ** 4 = 11.4 − 16 = −4.6	Invalid	1110000.000
3	11.4 − 2 ** 3 = 11.4 − 8 = 3.4	3	1111000.000
3	3.4 − 2 ** 3 = 3.4 − 8 = −4.6	Invalid	1111000.000
2	3.4 − 2 ** 2 = 3.4 − 4 = −0.6	Invalid	1111000.000
1	3.4 − 2 ** 1 = 3.4 − 2 = 1.4	1	1111010.000
1	1.4 − 2 ** 1 = 1.4 − 2 = −0.6	Invalid	1111010.000
0	1.4 − 2 ** 0 = 1.4 − 1 = 0.4	0.4	1111011.000
0	0.4 − 2 ** 0 = 0.4 − 1 = −0.4	Invalid	1111011.000
−1	0.4 − 2 ** −1 = 0.4 − 0.5 = −0.1	Invalid	1111011.000
−2	0.4 − 2 ** −2 = 0.4 − 0.25 = 0.15	0.15	1111011.010
−2	0.15 − 2 ** −2 = 0.15 − 0.25 = −0.15	Invalid	1111011.010
−3	0.15 − 2 ** −3 = 0.15 − 0.125 = −0.025	0.025	1111011.011

tional values (such as 0.4), which are very easily expressed in base 10, are not easily represented in other numbering systems.

I will express very large and very small numbers in scientific format with a base-10 exponent. This format consists of a single whole number (with a two-digit fraction) multiplied by an exponent of 10. For example, 123 can be expressed in scientific format as:

$1.23 \times (10^2)$

The advantage of scientific format is that its order of magnitude can be very easily converted to a logarithm and then be multiplied or divided with another number. I use scientific format where appropriate throughout the book with two or three significant digits (the fraction displayed in the number).

Of more use are the standard-unit exponent multipliers. In the SI (metric) measurement system, each three orders of magnitude are indicated by a prefix to a unit. The prefixes are listed in Table 8.

Using this system, very large units are expressed in more convenient terms. For example, 10,000 ohms is usually expressed as 10 Ω or just 10K.

In another case, 0.0000001 farads is expressed as 0.1 microfarads or 0.1 μF.

The only other engineering mathematics that you should be aware of is how logarithms work. Logarithms are the complementary operation to exponents. Passing a value to a logarithmic function will return the exponent (to the specific base), along with a logarithm value. In the previous text, I refer to the exponent as the "order of magnitude." The operation of finding a logarithm also finds the order of magnitude of a number.

In this book and for most electronics, you should only be concerned with the returning of the exponent only from a logarithmic operation. For example, you should be aware that 10,000 logarithm base 10 is 4.

Along with knowing the base-10 exponent logarithms, you should also be aware of the base-2 exponent logarithms. These values for the first 17 exponents of 2 are shown in Table 9.

TABLE 8 Prefix Exponent Table

PREFIX	EXPONENT MULTIPLIER
p ("pico")	10 ** −12
n ("nano")	10 ** −9
u ("micro")	10 ** −6
m ("milli")	10 ** −3
k ("kilo")	10 ** 3
M ("Mega")	10 ** 6
G ("Giga")	10 ** 9
T ("Tera")	10 ** 12

TABLE 9 Decimal To Base 2 Exponent Conversion Table

DECIMAL VALUE	BASE 2 LOGARITHM / BASE 2 EXPONENT
1	0
2	1
4	2
8	3
16	4
32	5
64	6
128	7
256	8
512	9
1,024	10
2,048	11
4,096	12
8,192	13
16,384	14
32,768	15
65,536	16

DC Current Flow

Electricity is often referred to as "flowing" in a circuit, which probably doesn't seem to be that analogous to us in our physical world. When given the word *flowing,* we tend to think of a stream, in which water flows to a lower level. This analogy is only 50% accurate to how circuits work.

To really get an appropriate analogy, we have to go back to grade school and look at the water cycle (Fig. 1).

In the water cycle, oceans are warmed by the sun (the power source) and water evaporates from them to become clouds. These clouds then move over land, where they rain and form the flowing streams mentioned previously. The water itself flows from a higher point to the lowest possible point (the oceans) and starts the process all over again.

The path water is taking is actually a repeating "circuit" and, as surprising as it seems, this process is remarkably analogous to how electronic circuits work. A basic electronic circuit is shown in Fig. 2.

In the electronic circuit, electrons are given energy from the power supply to move through the circuit (the same as the sun evaporating the ocean's water so that it can form

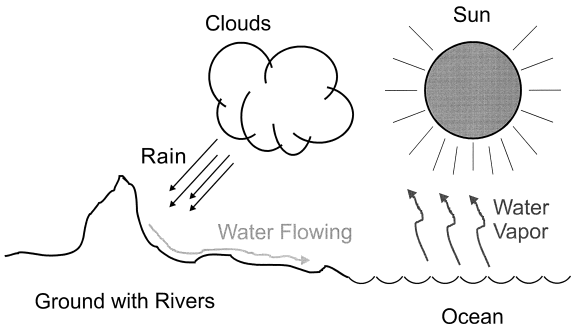


Figure 1 The water cycle

clouds, rain, and eventually flow back into the ocean). The power supply gives the electrons a *potential* for moving within the circuit. The electrons that are given this potential will then flow through the circuit to the point of lowest potential ("Ground" in Fig. 2). This process is repeated as long as power is applied to the circuit.

This process was worked out by Benjamin Franklin (remember the guy with the kite and the key in the thunderstorm?) and the notion that electrical current flows from positive (+) to negative (−) has become the basic convention used for all circuits. Unfortunately, it is wrong.

It is wrong because Franklin assumed that positively charged particles make up electrical current flow in the circuit. In actuality, these particles (called *electrons*) that flow in the circuit are negatively charged and actually flow in the opposite direction. This convention is kept simply because it has been used for over two centuries now and we are comfortable with it.

If you take (or have taken) Quantum Physics, you will be introduced with the concept of *holes,* which are spaces within atoms that can accept electrons. As electrons flow from negative to positive, the holes must flow from positive to negative to provide a place for the electrons to flow into. Although these holes actually exist (as empty electron positions in an atom's electron "cloud"), I find that they are really more of a crutch for people who can't accept that current actually flows in the opposite direction.

As you work through the circuits in this book (and other electronic circuits), just accept the convention that current flows from positive to negative, with the positive terminal of the power supply being at the highest electrical energy potential and the negative terminal the lowest. Going back to Fig. 2, I want to describe a few of the conventions I used when making the diagram because I continue to use these conventions for the rest of the book.

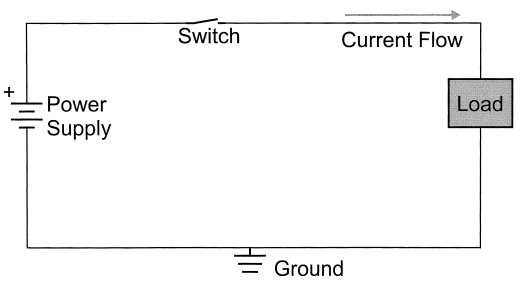


Figure 2 Basic electronic circuit

First is the idea of electrical potential. As described, electrical current flows from the point of highest potential to the lowest. With old Ben's convention, the positive terminal (marked with the + sign) is at the highest potential. It is therefore at the top of the diagram. After the current has gone through the load, it is now at ground potential. This is normally referred to as just *ground* because it has no where to flow down to (it is at the bottom) and because, in many circuits, this ground is also the actual electrical potential of the planet. In some countries, such as Britain, the term *ground* is replaced with *Earth,* but they really mean the same thing.

For safety's sake, many circuits' ground is connected to the earth to provide a safe path for current to flow if there is ever any problems. Going back to Ben Franklin, remember that one of his greatest inventions was the lightning rod which is a connection directly from a high point into the ground to provide a safe path for the current from a lightning bolt (instead of through a house or a barn). In most homes, electrical ground is provided by a connection to the cold water inlet pipe.

In many electrical circuit diagrams, you will see a circuit like Fig. 3, in which there is no continuous path between the negative terminal of the power supply and the load's negative terminal. Instead, it is assumed that there is one common path to ground that is not shown to keep the diagram from being cluttered with many lines. This convention is also followed for the power, with an upward arrow or a closed circle used to describe a connection to the positive power terminal.

Current is only able to flow if the circuit is closed. Thus, there is a continuous path for electrical current to flow from the power-supply positive terminal to its negative terminal. This is done by "closing" the switch that establishes the continuous current path.

In Fig. 2, I have added the *load,* which is the circuit that is being powered. This might seem like an ingenuous way of referring to the microcontroller circuits that we are going to be working with in the text of the book, but it is an accurate representation from the power supply's perspective.

As shown in the next section, the current through the load is calculated from its electrical resistance to the applied electrical force (known as *voltage*). An important aspect of designing any electrical circuit is knowing how much current is going to be passed through the circuit. The amount of current is dependent on the electrical resistance of the load (the greater the resistance the less current that can pass through it). Like water in a stream, electrical current also follows the "path of least resistance," which means if you have two loads wired side by side (known as in *parallel*), more current will flow through the load with the lesser resistance.

These are important concepts to understand because one condition that you will encounter when building your circuits could potentially damage your power supply and com-

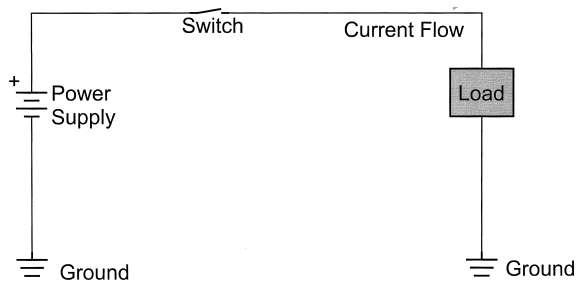


Figure 3 Basic electronic circuit with common ground

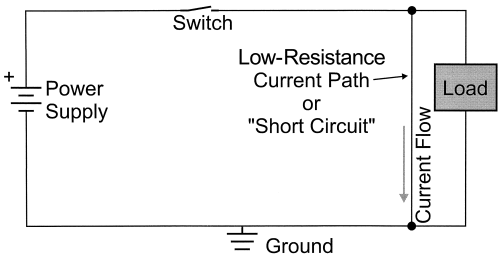


Figure 4 Short circuit

ponents as well as yourself. If an error is made in the circuit wiring and a path is allowed to exist directly between the positive terminal of the power supply and the negative terminal (Fig. 4), a great deal of current will flow through this path. This low-resistance path (known as a *short circuit*) could be caused by a wire fragment that has fallen on your circuit, a defective component, or an error in the assembly of the circuit. It can damage to the power supply or the circuit. You must always be careful around short circuits because large amounts of current can pass through them, resulting in temperatures that could burn you.

For this reason, power supplies should always be protected with a fuse or the capability to *crowbar* (turn itself off when the current draw is too great). For added protection, I recommend placing an indicator (such as an LED light) into a circuit that will allow you to see if any current is available for the circuit or if it is being lost to a short circuit.

OHM'S AND OTHER BASIC DC LAWS

In the previous section, I pointed out that the term *current* is used to describe the movement of electrons in a wire is analogous to the flow of water in a stream and in a pipe. Like electron current, water current is also subject to similar laws that determine how fast and how much water flows. This section introduces *Ohm's Law*, which is considered the most basic voltage/current law. It also introduces some of the other basic electricity laws and how they can be visualized as water flowing through pipes.

For water to flow through a pipe, some force must be exerted on it. As the water flows through the pipe, it will experience resistances that impede its flow and lessen the total volume that flows through the pipe over a given period of time. As Fig. 5 shows, if the "resistance" to the water is small, then large amounts of water can flow through it. If there is a greater restriction (Fig. 6), then a smaller volume of water will flow through the pipe over a given period of time.

This situation is analogous to the simple circuit shown in Fig. 7. Electrons are supplied by the power supply with a force known as *voltage* (volts or V). R is a resistor or load that restricts the flow of electrons in a circuit. All electrical devices have some resistance built into them, but for most circuits (including the PICmicro® MCU circuits presented in this

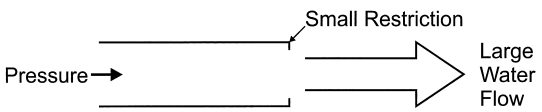


Figure 5 Water analogy with a small restriction

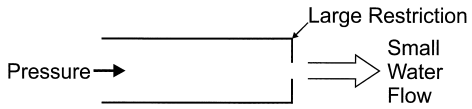


Figure 6 Water analogy with a large restriction

book), only resistors are assumed to have electrical resistance. All others are assumed to be ideal and not have any resistance at all.

For the circuits presented in this book, this is a reasonable assumption. Although other devices have *parasitic resistance* (i.e., the wires that connect them and are used inside them have resistance current flow). For the purpose of this book (and most circuits), this is assumed to be zero. Active devices (such as the PICmicro® MCU) do require power to operate and their equivalent resistances or loads are described when the devices are presented.

In the water pipe examples (Figs. 5 and 6), if more force ("pressure") is applied to the water entering the pipe and restriction, common sense dictates that the volume of water going through the pipe in a given period of time increases as well. This is also true for the simple circuit shown in Fig. 7. As the force applied to the electrons (voltage) is increased, the electron current through the circuit will also increase. In Fig. 7, I wrote this relationship as:

```
I = V/R
```

where current (*I*) (which is measured in amperes and has the short form *amps* or uses the label *A*) is the volume of electrons flowing through the wire leading to the resistor (R) in a given period of time. Resistance is measured in ohms and often is referred to using the Greek letter omega, which is the character Ω. As indicated, *V* is the force applied to the electrons by the power supply that is known as *voltage* and is measured in *volts*.

The formula listed can be used to calculate the current expected to flow through the circuit. For example, if 3.5 volts is applied by the power supply and the resistance was 2 Ω, the formula can be used to find the current flowing through the circuit:

```
I = V/R
  = 3.5 volts/2 Ω
  = 1.75 amps
```

This is known as *Ohm's Law* and is normally expressed as:

```
V = I × R
```

You will have to commit this formula to memory because it is basic to the operation of all electronics. Once you understand it and can apply it along with the concepts explained

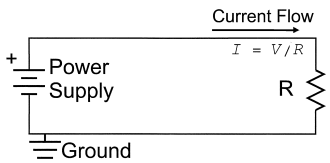


Figure 7 Basic resistance circuit operation

Ohm's Law: $V = I \times R$

Ohm's Law Triangle:

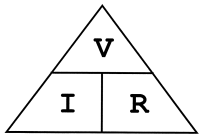


Figure 8 Ohm's Law triangle

in the rest of this section, you will understand almost all of the electronics presented in this book.

Ohm's Law can be re-arranged to find the voltage, current, or resistance of a circuit if any two of the parameters are known. The formula can be re-arranged using algebraic techniques or you can use the Ohm's Law triangle shown in Fig. 8.

I first learned about the "Ohm's Law triangle" when I was in grade 10, before I had learned any formal algebra. The triangle is a tool that will allow you to easily figure out what is the formula for finding a given parameter by placing your finger over the parameter you want to calculate. As is shown in Fig. 9, by placing your finger over the *I*, *V* over *R* is left, which is the formula used to calculate *I*.

The water analogy can also be used for other cases as well. In Fig. 10, a pipe can be drawn with multiple restrictions, which corresponds to multiple pressure drops (as shown in the graph below the diagram of the pipe). This is analogous to the situation where multiple series resistances are in a circuit (Fig. 11).

The resistances are said to be in series because the electron current has to flow through the series of them. To calculate the current in the circuit, the current through the equivalent resistance is calculated. The equivalent resistance is the sum of the resistances. So, for the circuit in Fig. 11, the equivalent resistance to the three resistances is:

$$R_t = R_1 + R_2 + R_3$$

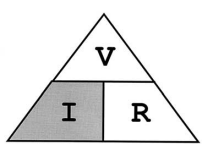
Each resistor has a voltage drop across it, just as the resistances in the pipe had a corresponding drop across them. This drop can be calculated by just finding the current through the equivalent resistance (which is V/R_t) and then multiplying it by the individual resistor values. So, for the voltage drop across R2, the formula:

$$V_2 = R_2 \times V/(R_1 + R_2 + R_3)$$

is used.

Find: I = ?

From Ohm's Law triangle:



Result: $I = \frac{V}{R}$

Figure 9 Ohm's Law triangle example

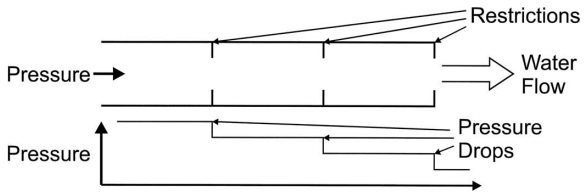


Figure 10 Water analogy with a multiple restrictions

The voltage across a resistor is measured at the current input and output points and the ground level is not used unless that is one of the two points of the resistor. In the example circuit in Fig. 11, if the voltage at the current input of R2 is measured along with ground, the voltage drop across R2 and R3 would be returned (which would be higher than just across R2). This is a common mistake made by people first working with electronics.

When you total the voltages across R1, R2, and R3, you will find that this value equals the voltage applied to the circuit. This can be confirmed very easily by summing the voltage drops across each resistor:

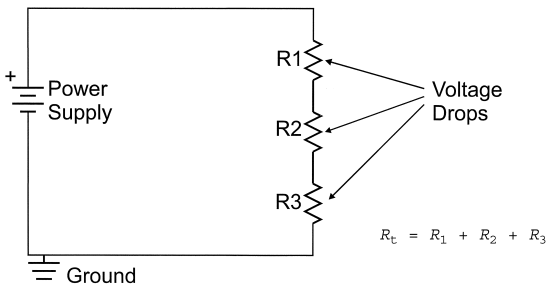
$$\begin{aligned} V_{resistors} &= V_1 + V_2 + V_3 \\ &= R_1 \times V/(R_1 + R_2 + R_3) \\ &\quad + R_2 \times V/(R_1 + R_2 + R_3) \\ &\quad + R_3 \times V/(R_1 + R_2 + R_3) \\ &= (R_1 + R_2 + R_3) \times V/(R_1 + R_2 + R_3) \\ &= V \end{aligned}$$

This is known as *Kirchoff's Law* and can be stated simply as "the voltage applied to a circuit is equal to the sum of the voltage drops within it."

Kirchoff's Law can be demonstrated with an LED that is lit by a 5-volt supply. As shown in Fig. 12, the LED has a 0.7-volt drop and a maximum current through it of 20 mA. The problem is to find the correct resistor value for the LED to have 0.7 volts applied to it and 20 mA of current flowing through it.

Using Kirchoff's Law, the voltage across the resistor and LED must equal the applied voltage. Because we know the applied voltage and the voltage across the LED, we can find the voltage across the resistor as:

$$\begin{aligned} V_{applied} &= V_r + V_1 \\ 5.0 \text{ volts} &= V_r + 0.7 \text{ volts} \\ V_r &= 5.0 \text{ volts} - 0.7 \text{ volts} \\ &= 4.3 \text{ volts} \end{aligned}$$



$R_t = R_1 + R_2 + R_3$

Figure 11 Series-resistance circuit

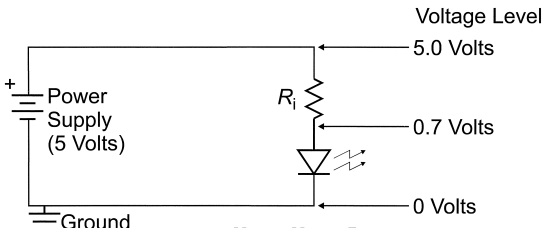


Figure 12 LED circuit operation

Now, to find the appropriate resistance that will limit the current in the circuit (and through the LED) to 20 mA, Ohm's Law is used:

$$R = V/I$$
$$= 4.3 \text{ volts}/20 \text{ mA}$$
$$= 215 \Omega$$

The resistor used to limit the current is called, not surprisingly, a current-limiting resistor and should be 215 Ω. In this type of application, I use a 220-Ω resistor, although other values can be used based on the actual voltage drop and current requirements of a specific LED. 220 Ω is a good convention to use for LEDs because it will allow most LEDs to light properly without any problems.

To lower the resistance in a pipe, either it can be made smoother (actually lessen the resistance) or the surface area could be increased. Figure 13 shows a pipe situation, where the resistance is lessened by splitting the pipe into two parallel paths for the water.

Like series resistance, parallel resistances work exactly the same as the water analog. In Fig. 14, two resistances are wired in parallel as the load with the equivalent resistance defined as:

$$R_p = (R_1 \times R_2)/(R_1 + R_2)$$

It is important to notice and remember that the equivalent parallel resistance will always be less than either one of the two resistances. This is a good check for when you are calculating the equivalent resistance and you aren't sure if you have the right answer.

For example, if you had 2-Ω and 5-Ω resistors in parallel, their equivalent resistance would be:

$$R_p = (R_1 \times R_2)/(R_1 + R_2)$$
$$= 2 \times 5/(2 + 5) \Omega$$
$$= 10/7 \Omega$$
$$= 1.43 \Omega$$

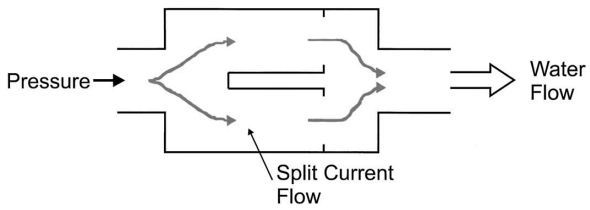


Figure 13 Water analogy with multiple restriction paths

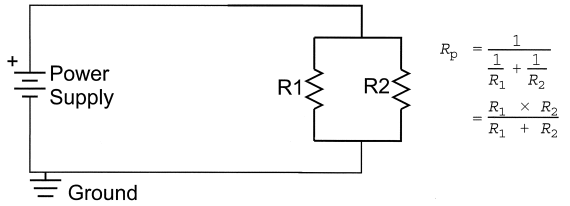


Figure 14 Parallel resistance circuit

The last concept and law that I would like to introduce to you is Thevinin's Law, which states that any series of loads that are ultimately connected to a power supply by two terminals can be represented as a single two-terminal load. Figure 15 shows how R_1, R_2 and R_3 can be combined into an equivalent resistance (R_e) using the series and parallel resistance formulas presented previously.

Using Thevinin's law, the equivalent resistance can be calculated and used to find the current supplied by the power supply. From this current, the voltages across R3 and the currents through R1 and R2 can be calculated.

Measuring voltage and current Measuring voltage and current probably strikes you as being something that is very easy to do, but you should be aware of a few things to help you to understand what your digital multimeter is showing you.

The basic meter consists of a coil and a piece of iron that is drawn into the coil by the magnetic force of current flowing through it. The basic operation is shown in Fig. 16. The higher the current through the coil, the more force that is applied to the piece of iron. As more force is applied to the iron, it pulls against a spring, which moves the needle proportionately to the current going through the coil. This is a basic ammeter.

In an ideal situation, the ammeter's coil has a resistance of 0 Ω. Obviously, this is not possible, so some voltage drop will occur across the coil. A good general rule is, the lower the voltage drop, the better quality (and the higher the cost) of the meter.

The ammeter is inserted in series with the circuit that the current is to be measured as shown in Fig. 17.

A voltmeter works similarly, but places a high-value resistor in series with the coil to allow a small amount of current to deflect the needle. The voltmeter is shown in Fig. 18. The in-line resistance (*R* in Fig. 18) is typically greater than 100 K. The voltmeter is connected in parallel to the device being checked. Ideally, the resistance would be infinite because the act of putting the voltmeter across a device will affect voltage current through the

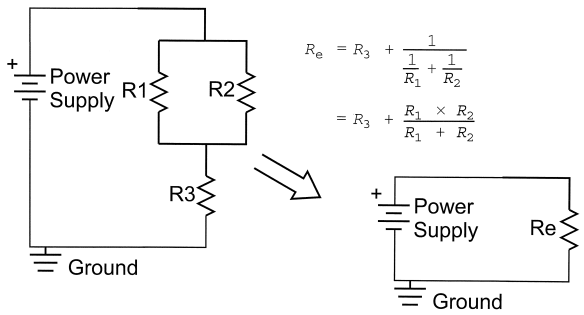


Figure 15 Thevinin equivalent resistance circuit

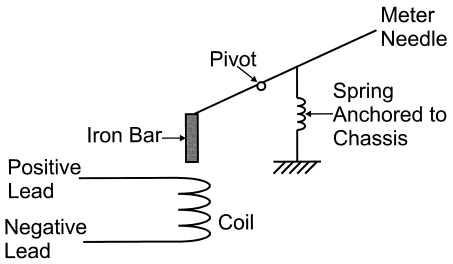


Figure 16 Basic meter circuit

device due to the parallel resistance. In the "real world" case, the current drawn by the voltmeter is very small. Measuring the voltage across a device in a circuit is shown in Fig. 19.

Both an ammeter and a voltmeter can be used to measure the resistance of a device by combining the two previous circuits into the one shown in Fig. 20. In this circuit, the resistance of the load can be calculated using Ohm's Law from the values determined by the ammeter and the voltmeter.

I'm showing how these meter types work because modern *Digital MultiMeters (DMMs)* have voltage, current, and resistance measurement capabilities built in and emulate these functions. Usually in a DMM, the current measurement circuitry has 1/10 Ω (or less) of internal resistance. For the voltmeter function, the internal resistance is 1 MΩ or greater (10 MΩ is not unheard of).

BASIC ELECTRONIC DEVICES AND THEIR SYMBOLS

Before working through all the different circuits, I want to go through some of the basic electronic devices, describe their characteristics, and things to watch out with them. I focus on the practical aspects of the devices, rather than the theory behind them and how they are manufactured. Along with this information, I expand on various devices later in the chapter.

The basic two devices in any circuit are the power supply and the load. Earlier in the chapter, these devices were presented as being necessary to calculate current power requirements. The power supply provides an electron source with the electronics given a specific voltage potential for a maximum rated current. Voltage can either be unchanging (direct current, DC) or changing according to a sine wave above and below 0 V (alternating current, AC). The projects presented in this book largely focus on using DC power supplies with 5-V output. The symbol used for power is the battery symbol (Fig. 21), with the positive terminal connected to V_{cc} (V_{dd}, for CMOS devices, such as the PICmicro® MCU). The negative terminal is connected to ground (GND) or V_{ss} for CMOS. The battery

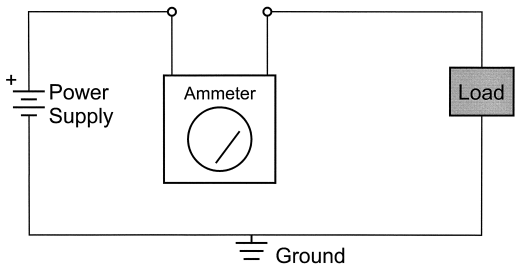


Figure 17 Basic electronic meter circuit

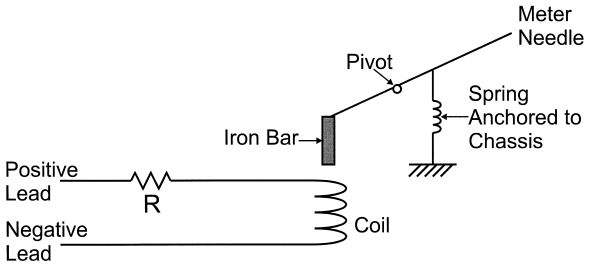


Figure 18 Basic voltage meter circuit

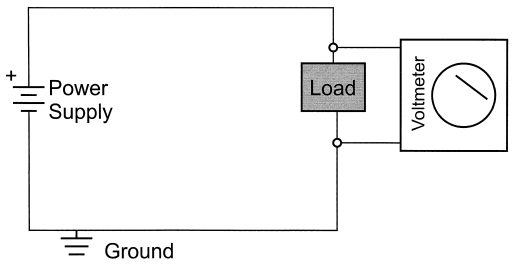


Figure 19 Basic electronic voltage meter circuit

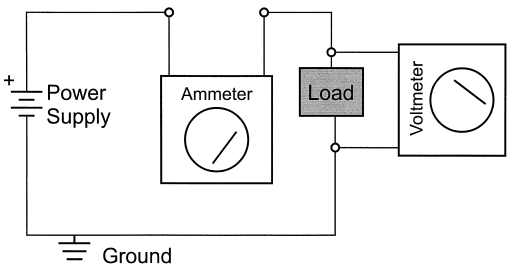


Figure 20 Voltage/current meter circuit

Figure 21 Battery/ power-supply symbol

symbol is meant to represent the multiple plates in a physical battery; I always remember positive because the symbol kind of looks like an actual cylindrical radio battery, with the small ground (which is positive in the physical device) connected backwards.

A resistor (Fig. 22) can characterize the load that is a device that impedes current flow through a circuit.

Resistors are not polarized and you should have a pretty good idea of how they work from the previous sections. The only point I would like to add about them is their power rating.

Occasionally, when you place a resistor in a circuit, you will find that it gets very hot. This is because more power being dissipated than is specified for the resistor. Resistors are normally rated at 1/8, 1/4, 1/2, one watt, and larger power-dissipation ratings with the rating being dependent on the power through it using the power formula:

$$P = I \times V$$

or substituting in Ohm's Law:

$$P = (V^2)/R$$
$$= (I^2) \times R$$

For most applications, a 1/4-watt resistor will be more than adequate, but occasionally, you will have situations that you should be aware of. For example, in the El Cheapo programmer circuit, I use a 220-Ω resistor as a current-limiting device when dropping a power supply from 13.4 to 5.1 V. The maximum current through the resistor is 40 mA and using the power formula:

$$P = (I^2) \times R$$
$$= 0.040 \times 0.040 \times 220$$
$$= 0.35 \text{ W}$$

the power dissipated by the resistor is greater than 1/4 watt. For this reason, in this circuit, I have specified a 1/2-watt resistor, rather than a traditional 1/4 watt.

Resistors normally have a series of four or five color bands printed on them. These bands are used to specify the resistance of the circuit using the formula:

$$Resistance = [(Band1 \times 100) + (Band2 \times 10) + (Band3 \times 1)] \times 10^{Band4}$$

Table 10 lists the values for each band and what they mean in the resistor-specification formula. So, for a 220-Ω resistor, the bands would have the colors: red, red, black, and black.

Remembering what the color codes is usually accomplished either by noting that the colors are in the order of the colors of the rainbow or by an obscene mnemonic. For all

Figure 22 Resistor symbol

TABLE 10 Resistor Color Code Chart

NUMBER	COLOR	BAND1	BAND2	BAND3	BAND4	OPTIONAL BAND5
0	Black	N/A	0	0	10 ** 0	N/A
1	Brown	1	1	1	10 ** 1	1% Tolerance
2	Red	2	2	2	10 ** 2	2% Tolerance
3	Orange	3	3	3	10 ** 3	N/A
4	Yellow	4	4	4	10 ** 4	N/A
5	Green	5	5	5	10 ** 5	0.5% Tolerance
6	Blue	6	6	6	10 ** 6	0.25% Tolerance
7	Violet	7	7	7	10 ** 7	0.1% Tolerance
8	Gray	8	8	8	10 ** 8	0.05% Tolerance
9	White	9	9	9	10 ** 9	N/A
N/A	Gold	N/A	N/A	N/A	10 ** −1	5% Tolerance
N/A	Silver	N/A	N/A	N/A	10 ** −2	10% Tolerance

practical purposes, nobody today reads the bands and tries to figure out what the resistance is. Instead, the ohmmeter function of a DMM is used to read the resistance value. This process is usually faster and definitely more accurate in terms of finding the actual resistance value.

Variable resistors (potentiometers) are used to change the resistance of a circuit or provide a user-selected voltage. Its symbol is shown in Fig. 23.

I typically use very high-value potentiometers (10 K or above) so that the issues of power in potentiometers in my circuits are not as significant as with simple resistors. Most PC board-mounted potentiometers can dissipate up to 1/4 watt (or so) of power. Potentiometers are available that can handle much larger amounts of current, but these devices tend to be expensive and difficult to work with. If I have a circuit where I want to control a DC device with varying power levels, I typically use a PWM control, rather than a potentiometer.

This book is concerned with two types of switches. Push buttons are normally referred to as *momentary-on* or *momentary-off* switches and their symbols (i.e., operation) are shown in Fig. 24. Momentary-on switches close the circuit when they are pressed and open

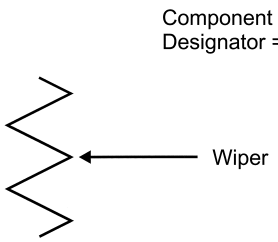


Figure 23 Potentiometer/variable resistor symbol

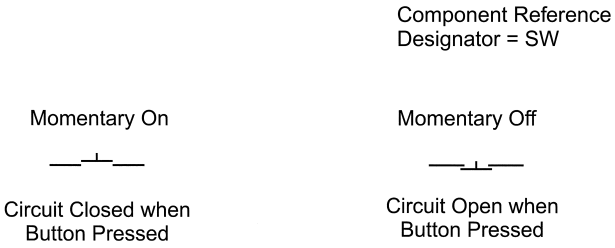


Figure 24 Momentary switch symbols

the circuit when they are released. Momentary-off switches work in the opposite manner, with the circuit opened when the button is pushed.

Momentary switches are used to satisfy a variety of different microcontroller application input requirements, which are described elsewhere in the book.

Slide switches are used to make a specific contact, based on the position in which the switch is left. The most basic switch has one or two throws, which result in a circuit closed. Figure 25 shows single- and double-throw switches.

More than one "throw" or position can be in the switch and multiple switches can be ganged together as *multipole switches*. Many different combinations can be built out these basic types of switches.

This book mostly covers *single-pole, single-throw (SPST) switches* for turning on and off power or being used to control basic signals. The symbol for the SPST is shown in Fig. 26.

You will find that most of slide switches are *Single Pole/Double Throw (SPDT)*, in which case, just the common connection and one switched one are used.

You will have to wire multiple boards together, which is accomplished by the use of connectors. It seems like an infinite variety of different connectors are available. In many cases, the application developer selects the connectors, in which they feel most comfortable. For this reason, except for some very specific instances, where high current or convenience is required, I have not specified the type of connector to be used. The connector symbol name is J.

Fuses can be used within your application to limit current draw and protect circuits, wiring and power supplies. Like home fuses, electronic ones are rated in amps and are available in a variety of different board-mountable or in-line packages and cartridges. The symbol is shown in Fig. 27.

Fuses are not rated in volts. For a joke you can always ask somebody to go out and buy you a "six-volt, half amp" fuse and tell them not to come back until they have it. If somebody asks you to go out for a "six-volt, half-amp" fuse, take their money and buy yourself a good dinner with it and return hours later, saying all the stores you went to were sold out.

In my own applications, I prefer to avoid specifying fuses, except when absolutely

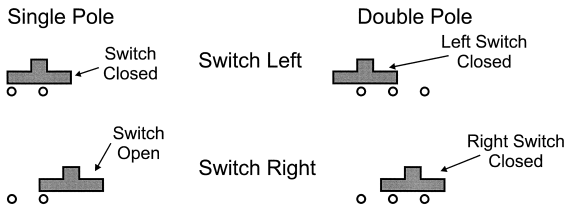


Figure 25 Throw switch types

Figure 26 Single-pole single-throw switch

necessary, instead I prefer to use voltage regulators that shut down, rather than burn out, when too much current is drawn through them. This does not eliminate the need for fuses in situations where they are indicted for safety or regulatory reasons, but it does mean that they are not required in experimental applications.

I could probably write an entire chapter on capacitors, their operation, dos and don'ts, and hints. It has actually been a lot of work for me to write just a simple explanation of the types of capacitors and their characteristics.

Capacitors are designed to store an electrical charge. The capacitor symbol is shown in Fig. 28.

The charge placed in a capacitor is measured in farads (symbol F) and are measured in the units of coulombs/volt. A *coulomb* is a measure of 1.6×10^{19} electrons, making one Farad a very large number. This very large number is why most capacitors are measured in millionths (μF) or trillionths (pF) of a Farad.

Capacitors consist of two metal plates (which explains the symbols shown in Fig. 28), separated by a dielectric. A *dielectric* is a material that doesn't conduct electricity. The type of dielectric used in a capacitor usually becomes their name. The most common types of capacitors are:

1 Ceramic disk
2 Polyester
3 Tantalum
4 Electrolytic

Ceramic disk and polyester are the most "benign" type of capacitors you can work with. They are typically used for applications that require small capacitance values at reasonably high frequencies. The reason these dielectrics do not enhance the capacitance of the two plates as significantly as the other two and large capacitances (such as μF) would be measured in inches across and would probably weigh up to a pound. These types also tend to be the least expensive of the capacitors you can buy.

Tantalum capacitors are a relatively new invention and offer excellent capacitive densities at high frequencies. Tantalum capacitors are well suited for use as decoupling capacitors. I typically keep an electronics drawer full of 0.1-μF, 16-V tantalum capacitors at all times.

Tantalum capacitors do not have a good reputation amongst many people because of the tendency of early devices to explode. Pin-hole shorts between the plates in the capacitor caused this. These shorts would heat up, burning away the dielectric, grow larger in sur-

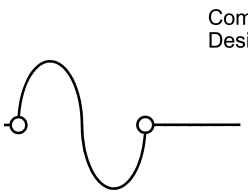


Figure 27 Fuse symbol

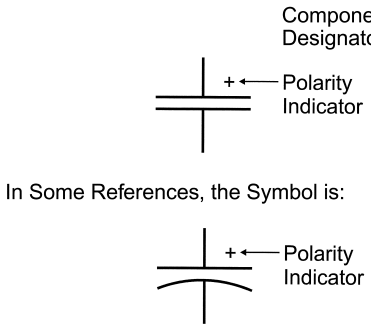


Figure 28 Capacitor symbol

face area until there was a catastrophic failure, which is very spectacular. This problem is accelerated by reverse biasing the capacitor; always be sure that the side marked + is connected to V_{cc}.

New tantalum capacitors do not have this problem if they are properly derated. All tantalum (and electrolytic) capacitors are given an operating voltage for which they are designed. By derating the input voltage specification, you are actually selecting devices with thicker dielectrics, which makes the likelihood of "burn through" much more remote.

A good general rule is to derate tantalum capacitors by 60%. This means that a tantalum capacitor used for decoupling in a 5-volt system should be rated at 12.5 volts or more (which is why I normally use 16-volt rated parts).

Electrolytic capacitors rely on a chemical liquid for its dielectric and are in sealed metal packages. The plates and the chemical are designed to be biased in one way to reach the desired capacitance. Reverse biasing on electrolytic capacitor will cause the liquid to break down, which is not as spectacular as a tantalum capacitor explosion, but you could still be hit by debris or sprayed by hot electrolytic. At the very least, you will have to clean the residue from your circuit board.

The electrolytic capacitor's big advantage is their dense capacitance in an inexpensive package. They do not have very fast responses and are not well suited to be used for decoupling and other high-speed filtering applications.

Some capacitors have been given a set of colored bands similar to the markings used on resistors. Making the issue confusing is that there are at least two sets of standards for marking capacitors. I recommend that you either buy capacitors with their values printed on them or segregate the parts in different marked bags or bins until you need them. Many modern DMMs can read capacitor values down to 30 pF or less, which can be useful when figuring out what is in that old bag of dust-gathering parts.

Where a capacitor stores energy in the form of a charge, inductors (coils) store energy in the form of a magnetic field. Inductors Fig. 29 are not often used in PICmicro® MCU circuits, except for switch-mode power supplies, which increase or decrease incoming power using the characteristics of coils to resist changes in current flow. Like the capaci-

Figure 29 Inductor symbol

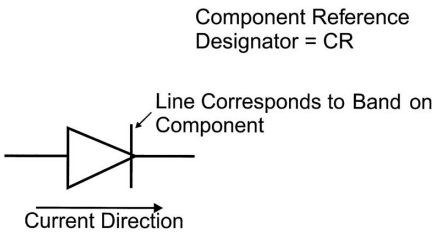


Figure 30 Diode symbol

tor's farad unit of measurement, the normal values of an inductor are in millionths of a Henry for the devices to be effective in a circuit.

Inductors can be difficult to work with if you are new to electronics. I would recommend that they not be used, except in the case where a chip manufacturer specifies them to be used in a specific application.

Modern application switching is accomplished by semiconductors, so named because their ability to conduct current can be controlled in different situations. The most basic of these circuits is the diode (Fig. 30), which only passes current in one direction.

Diodes are also known as *rectifiers* because they can be used to convert AC voltages into DC, (which is described later in this appendix). This is accomplished by manufacturing the device out of a piece of silicon that has been chemically impregnated (doped) with two different materials. These materials create an interface in which the energy of electrons is much higher in one than the other. This interface is known as the *PN junction* because of how the silicon is doped.

To cross the boundary, electrons can only flow from the high potential (N doped) side to the low potential (P doped) side. If electrons enter the device from the P side, they cannot "climb the wall" to the N-doped side, thus limiting current to only flowing in one direction (Fig. 31).

The electron potential drop results in some voltage drop in the applied current. For silicon diodes, this drop is 0.7 volts. I use this function of PN junctions (and diodes in general) to create simple voltage references for power supplies.

When electrons fall down from the high potential side to the low potential side, they release energy. This energy release, according to Professor Einstein, is a set amount and is released from the diode as a *quanta* of energy, which takes the form of a *photon.*

Figure 31 shows the release of photons as the electrons go from a higher energy state to a lower one. Standard silicon diodes release photons in the "far" infrared spectrum of light. By changing the diode materials, the photon released can be in a visible frequency of light. These diodes are known as *Light-Emitting Diodes (LEDs)* and are available in a variety of colors.

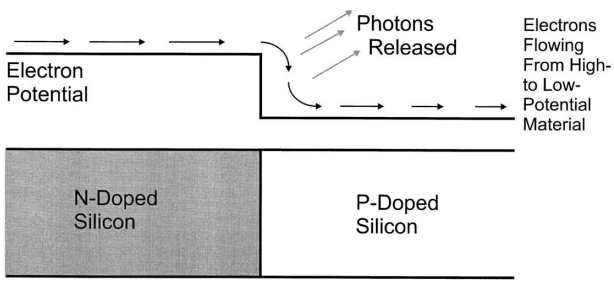


Figure 31 Diode operation

In the applications presented in this book, LEDs are used a lot. It is important that you remember these devices are diodes and attempts to drive current through them in the wrong direction will be resisted and no light will be output.

Transistors

When I was in university, the primary type of transistor taught was the bipolar device. It is quite difficult to understand how this type of transistor works and is manufactured. Despite this, it was the first transistor that was successfully built and became the primary device used in the 1960s and 1970s. In the 1980s, *Field-Effect Transistors (FETs)* became the most popular type of transistors. This trend continues today.

When I was in school, roughly 60% of all transistors were bipolar devices and 40% were FETs. Today, it is something like 99% are FET and 1% is bipolar. Bipolar transistors are used for very specific reasons that have to do with their electrical characteristics. FET transistors are easier to manufacture and can be placed in much smaller amounts of silicon chip "real estate," which allows for the amazing circuit densities of modern *VLSI (Very Large-Scale Integration) chips*. I remember my third-year microelectronics professor stating that FETs were just a passing fad. Boy, was he wrong.

This section introduces the two different classes of transistors and their characteristics. As well, I will describe how integrated circuits are made with transistors and what you would expect to see if you could magnify a PICmicro® MCU chip and look at it. I concentrate on how transistors work in computer systems, but I describe some of the concepts that are necessary to understand in analog applications.

The most basic transistor is the *bipolar NPN*, which consists of two pieces of N-doped silicon sandwiching a piece of P-doped silicon. *N-doped silicon* has a material added to it that causes electrons to be available in the crystal lattice. *P-doped* silicon has a material added that causes an affinity to accept electrons in the crystal lattice.

As I shown in the previous section, these materials can be combined to form a rectifier or diode, which only allows current to pass in one direction. In the NPN transistor, this doped silicon prevents current from passing through them unless current is "injected" at the P-doped region (which is known as the *base*). The NPN transistor is shown in Fig. 32.

As indicated above, transistors are switches. Bipolar transistors are actually variable

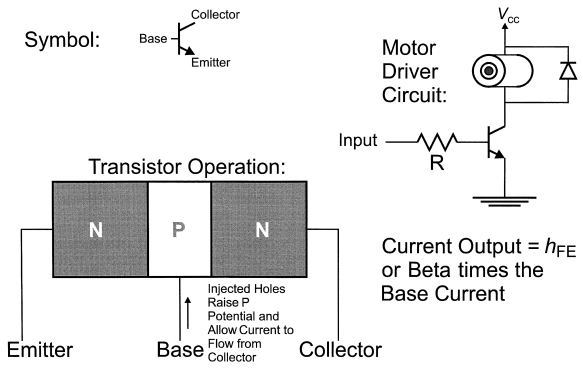


Figure 32 NPN bipolar transistor operation

switches in which the amount of current passing through the collector and emitter can be controlled by controlling the amount of current passed through the base. Figure 32 shows a simple circuit for a motor controller. When current is passed through the base to the emitter (which is connected to ground), the collector will be able to accept current from the V_{cc} source.

This current is dependent on the formula:

$$I_{ce} = I_{be} \times h_{FE}$$

Where I_{ce} is the current through the collector and emitter and I_{be} is the current through the base and the emitter. h_{FE}, also known as *Beta*, is the multiplication factor for the current flowing through the base and emitter to the collector and emitter.

For a common transistor like the 2N3904 (which I use a few of in this book), h_{FE} is in the range of 150 to 200. Thus, a significant amount of current reduction through the base and emitter is required to not burn out the transistor (or cause other PICmicro® MCU problems, which are described in the book). If a 2N3904 was used to control a motor, using the circuit in Fig. 32, then the value of *R*, the base-emitter current-limiting resistor would have to be calculated for the current passing through the motor.

For this example, assume that the 2N3904 has an h_{FE} of 150 and the motor requires 100 mA when it is running. To calculate the value of *R*, the current flowing through the base and emitter has to be calculated using the formula:

$$\begin{aligned} I_{ce} &= I_{be} \times h_{FE} \\ &= 100\ \text{mA}/150 \\ &= 667\ \mu\text{A} \end{aligned}$$

Now, assuming that the voltage applied to the base is V_{cc} (+5 volts for most applications), the resistance can be calculated using Ohm's Law (although it should first be recognized that there is 0.7-volt drop is across the base-emitter junction to ground). The resistance is:

$$\begin{aligned} R &= V/I \\ &= (5\ \text{volts} - 0.7\ \text{volts})/667\ \mu\text{A} \\ &= 4.3\ \text{volts}/667\ \mu\text{A} \\ &= 6447\ \Omega \end{aligned}$$

So, a 6.5-k resistor would be used in this application to allow a 2N3904 to drive an electric motor that requires 100 mA with five volts applied to it.

For most of the applications that use bipolar transistors in this book, I specify a 330-Ω base-emitter current-limiting resistor to be used. Using these calculations, this will work out to an I_{be} of 13 mA with a maximum I_{ce} of 1.96 Amps! For the applications presented in this book, the current flowing through the collector and emitter of the transistor is limited to just a few milliamps.

When the current is limited (but not shut off) by a transistor, then there will be a definite voltage drop across it with a specific current passing through it. These values can be translated into a specific amount of power dissipated by the transistor. This is why the relatively small base-emitter current-limiting resistor is used. With 13 mA driving into the transistor, any current that is available at the collector will be passed through to the emitter with a minimum of resistance. When this is done, the transistor is said to be in *saturation* and can pass any current applied to the collector very quickly.

Along with the NPN bipolar transistor that is turned on when current is applied to it, the PNP bipolar transistor that will pass current between its collector and emitter when current is drawn from it. This device (Fig. 33) behaves similarly to the NPN transistor. The amount of current passing through the collector and emitter is proportional to the current being drawn from it.

When I use a PNP transistor, I primarily use it to supply power to a circuit. By drawing current from it, it will "close" and provide power to other devices.

Bipolar devices have the advantage of being quite fast, but they require a lot of space on a silicon integrated circuit. This is because of the need for each transistor to be built in a "tub" in the silicon to ensure that it is isolated from all other transistors built on the chip. As well, for proper operation, the transistors also require the current-limiting resistors connected to their bases that are very hard to build on silicon chips.

The FETs do not have these limitations. The most popular of these devices are *Metal-Oxide Silicon Field-Effect Transistors (MOSFETs)*. These components are much smaller and use a fraction of the power of the bipolar transistors. Their only real drawback is the amount of time required for switching. Bipolar transistors can natively switch must faster. For this reason, bipolar transistors were the primary transistors used in "supercomputers" up until about 10 years ago. MOSFET devices have replaced bipolar ones as they have become much faster. Newer computer architectures with parallel circuits have allowed MOSFETs to become the building block of choice for these systems.

The basic type of MOSFET transistor is the N-channel device (Fig. 34). When a positive voltage is applied to the gate, an electrical field is set up in the P-doped silicon substrate. This field will cause the electrical characteristics in the substrate below the gate (the conducting region in Fig. 34) to mimic those of N-doped silicon.

N-doped silicon is a conductor and current is allowed to pass between the source and the drain. No current passes from the gate to either the source or the drain, so there is no bipolar I_{be} current analog in the MOSFET transistor.

The gate consists of a metal plate that is separated by a layer of silicon oxide (also known as *glass*) from the substrate. FETs were originally patented in the 1870s, but it was the difficulty in growing the silicon oxide onto the substrate, which resulted in working FETs not being demonstrated until the 1960s (almost 100 years later). The silicon-oxide layer must be as thin as possible to maximize the performance of the MOSFET transistor.

The size of the conducting region can be controlled by the amount of voltage applied to the gate. For digital applications, a set amount of voltage is constantly applied, making the N-channel behave like an on/off switch in digital applications.

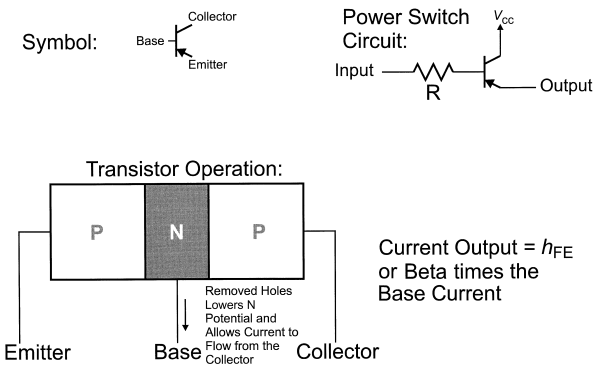


Figure 33 PNP bipolar transistor operation

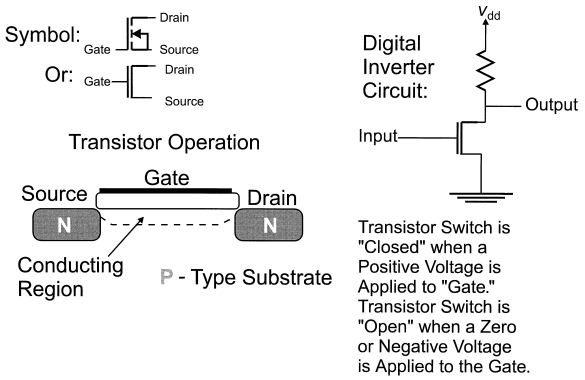


Figure 34 N-channel MOSFET transistor operation

MOSFETs are usually defined by the resistance between the drain and the source when the transistor is on or conducting. For an N-channel device, this resistance is normally measured in fractions of an ohm.

Figure 35 shows an N-channel transistor being used as a simple switch that inverts the logic state of the input signal to the output. This is an example of what was known as *NMOS logic*.

The N-channel MOSFET transistor is very easy to build on a P-doped silicon substrate and it became the first type of MOSFET logic used. If you look at older General Instruments PIC microcontroller datasheets, you will see that this type of logic was used for them. Although N-channel MOSFET devices are very easy to place on the substrate, there was still the need for placing resistors on the silicon. For NMOS logic, this was accomplished by doping regions of the substrate with N doping materials. The resulting resistor was only accurate to a few tens of percent, which made the devices unsuitable for analog applications.

N-channel MOSFET transistors also have a complementary device, the P-channel MOSFET. These transistors normally conduct when a zero voltage is applied to them because a conducting "tub" of N-doped silicon has been placed under the gate Fig. 35. When a positive voltage is applied to the gate, the N-doped silicon changes its electrical characteristics to P-doped silicon and stops conducting. When the depletion region grows to the point where the entire N-doped silicon behaves like P-doped silicon underneath the gate, the transistor is no longer conducting and is turned off.

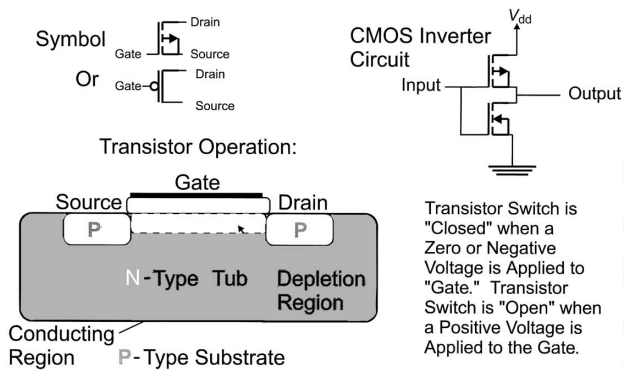


Figure 35 P-channel MOSFET transistor operation

Varying the amount of voltage applied to the gate can control this "pinching off" of the N-channel "tub." The P-channel MOSFET has an on resistance, which at several ohms (or more), is much higher than the N-channel MOSFET. It can be difficult to "match" the two devices for analog applications.

Where P-channel MOSFETs have found a niche is in working with N-channel MOSFETs in *CMOS (Complementary Metal-Oxide Silicon) logic*. As shown in Fig. 35, a P-channel MOSFET can be combined with an N-channel MOSFET to produce an inverter, but not require the current-limiting resistor of the NMOS inverter. This has been an important breakthrough. By providing P-channel and N-channel MOSFET transistors in the same circuit, the need for resistors is eliminated. When the N-channel transistor is conducting (and pulling the output to ground), current is being passed through a current-limiting resistor to ground.

When you see the symbols for MOSFET transistors, it is easy to forget which is which. Always remember that the N-channel device has the arrow going "iN." This is not true for NPN transistors, where the arrow indicates current output.

In CMOS circuits, the only real opportunity for current flow is when the gates are switching and stored charges are passed through the transistors. This accounts for the phenomenally low current (and power) requirements of CMOS circuits and why the current requirement goes up when the clock frequencies go up. As number of switch transitions per second increases, the amount of charge moved within the chips goes up proportionally. This charge movement averages out to a current flow.

For digital circuits, transistors are combined to form "logic gates," which are explained later in this appendix. Logic gates change the electrical state of one or more inputs into another one. Figure 36 shows how a bipolar NPN transistor can be used to implement a NOT or inverter gate, which changes the state of the input from low to high and visa versa.

The other circuit in Fig. 36 is a logical NOR, which only outputs a high voltage if both the input voltages are low. This is the basic gate used by *TTL (Transistor to Transistor Logic)*. TTL is a bipolar technology that has been around since the mid-1960s. In TTL, these two basic gates are combined (using the Boolean Logic rules presented later in this appendix and elsewhere in the book) to provide complex logic functions. CMOS is a NAND-based technology in which the logic circuits are built from NAND and NOT gates.

By building logic functions out of these basic transistors, you can see that creating complex logic functions requires a lot of transistors. Typically, each gate requires 12 to 30 transistors. When I look at logic functions, I tend to think in terms of gates. This comes

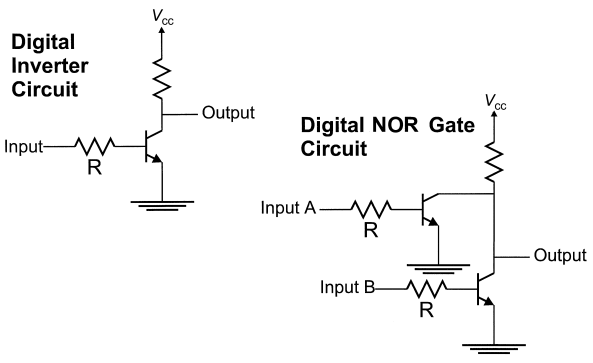


Figure 36 Example transistor logic gates

from designing *ASICs (Application-Specific Integrated Circuits)* and *FPGAs (Field Programmable Gate Arrays)*, which provide the functions in terms of gate macros and allocate transistors for the required functions as they are needed.

Power Supplies

One of the most overlooked areas of microcontroller-application development is the specification and design of power supplies. Although most PICmicro® MCU applications are very insensitive to the power supply design, a properly specified power supply can make an application much more inexpensive, reliable in hostile environments, or better able to handle changing power loads.

For DC circuits, the power required by a circuit is defined by the equation:

$$P = V \times I$$

where P is the power consumed (in watts), V is the voltage applied to the circuit, and I is the current (in amperes or amps) drawn from the power supply. For example, in a +5-volt digital circuit, if 0.15 amps is drawn from the power supply, then the circuit is dissipating 0.75 watts (750 mW). Because most digital electronic circuits require +5 volts, the primary focus of power-supply specification and design is how much current is required by the application and the parts used within it.

When I develop my PICmicro® MCU applications, I only want to use one voltage (normally +5 volts for the PICmicro® MCU's V_{dd}) for simplicity and to keep the application cost down. Most interfaces that require different voltages have parts available to generate the voltages and provide the level conversion for the application. The best example I can think of is the MAXIM Max232, which generates +12 and −12 volts on the chip and provides an interface for TTL/CMOS logic.

In many of the applications (especially the experiments) presented in this book, I will appear to be pretty casual in terms of the power supplies that I use. For all of the experiments, I specify a 1.0-amp, +5-volt power supply. For the experiments, this supply is adequate for the circuits that I present—it supplies more than enough current for any of the experiments and it is simple to set up. For the projects and tools, which are more-specific applications, I always recommend that an estimate should be made of the current required by all of the different components used in the application. Once the estimate for each part has been completed, they should be summed and 25% added the total to find the specified current. The extra 25% will give the supply some margin without burning out, crowbarring, or blowing a fuse.

When looking up the specified current required for a device from its manufacturer's datasheet, be sure to look up the current required at the expected operating frequency and temperature. Because of their design, CMOS circuits use more current the faster they are run. Operating temperatures can also affect the amount of current required by a device. If you are unsure what is the correct current requirements are for a device, use the maximum listed in the datasheet. It is always better to plan for too more current draw than is actually required than finding out that you planned for too little.

BATTERY POWER

In our society, battery-powered electronic devices are found just about everywhere. When I look around my office, I can see a "palm pilot," a digital clock, a DMM, a radio, and a couple of calculators, all of which operate on alkaline "radio" batteries. It is probably a big surprise then to find out that I don't recommend that you use battery power for your own electronic projects, unless you follow some specific rules.

These rules are based on an understanding of how batteries operate. You are probably familiar with three types of batteries: alkaline disposable "*dry cells*" or "*radio*" batteries (which I usually just refer to as "*alkaline batteries*") and rechargeable nickel-cadmium cells (*nicads*). The last type of battery is the lithium cell, which is normally used for clocks and retaining data in CMOS devices. These batteries might appear to be similar, but they each work differently and have different operating characteristics.

All types of batteries are given a "rated life" in the units of ampere-hours (AH). This rating is used to determine how long a battery will provide its rated operating characteristics for with a given drain. For example, a battery with a 2-AH rating will supply two amps of current for one hour—or 200 mA of current for 10 hours. A typical value for a AA alkaline radio battery is in the range of 100 to 300 mA-h. Of the three types of batteries described in this section, alkaline batteries tend to have the greatest amp-hour rating and lithium has the smallest. Lithium batteries are designed to provide a "trickle" (very small) current for a very long time.

When using different types of batteries in an application, I make one suggestion and warning. When prototyping a circuit, never use nicad batteries. Nicads can provide a very large amount of current when short-circuited (alkaline batteries provide a set current, usually in the range of hundreds of mA). If you are prototyping with nicads and a short circuit is encountered, the full charge within the batteries can be released. The batteries will become very hot and potentially explode. This will not happen with alkaline batteries.

One of the biggest differences between the battery types is the voltage output. For alkaline batteries, the voltage output on new batteries is 1.7 to 2.0 volts per cell. This voltage tends to drop linearly over use and 1.5 volts per cell is output when half the charge has been used up. Nicads tend to output around 1.2 volts per cell and lithium provide much higher voltage (about 2.7 or higher volts per cell. To provide higher voltages, cells are connected in series and their voltage outputs are summed together. For example, for a 9-volt "square" batteries have six 1.5-volt cells built into them to provide the nominal 9-volt output.

This leads to the question of how many cells are required to provide a stable V_{cc} (normally 5.0 volts) for the PICmicro® MCU and digital logic of your circuits. If alkaline batteries are used, then three are required, four for nicad, and two for Lithium. But, as I will explain in this section, this is not an optimal arrangement for powering applications.

The next major difference between battery types is the voltage output over time as current is drawn. This is different for the three different types of cells Fig. 37.

Nicads and lithium batteries tend to provide a constant voltage while they are being drained, but "die off" suddenly when their charge is depleted. Alkalines, as noted, tend to have their output voltage linearly drop as the charge within the cell is depleted. This characteristic is what separates the alkaline cell from the other two types.

Another characteristic that is different among the types of battery cells is how long a charge can be retained. Lithium and nicads will keep their charge for a very long time, if

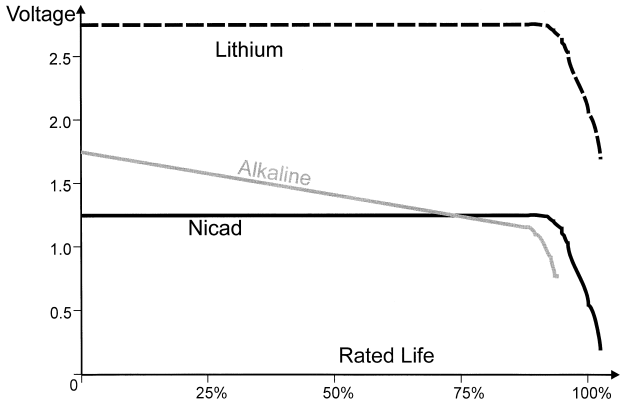


Figure 37 Different battery-type operation

there is no drain on them. As you have probably found, alkaline and other dry cells will leak and lose their charge over time. Nicad cells are particularly good at maintaining charge and will stay charged for 10 years or more. Lithium cells also have good charge-retention capabilities, although they generally need to be replaced after five or so years—even if their rated capacity hasn't been used.

Lithium batteries, with a greater than 2.5-volt output, is not enough for microcontroller operation, but can be used to maintain a device in Sleep or Standby modes. Ideally, the lithium battery should only power the circuit with V_{cc} (or regulate power) to the PICmicro® MCU. The circuit I normally use for this function is shown in Fig. 38.

In this circuit, as long as power is being applied to the circuit, no current will flow from the lithium battery because the voltage applied to the PICmicro® MCU is greater than what is being applied by the battery. When V_{cc} is lost, the Lithium cell supplies current to the PICmicro® MCU. Ideally, there should be some kind of sensor to detect the low-voltage condition (when the lithium cell is powering the circuit because the primary power has been lost). The low-voltage sensor should cause the PICmicro® MCU to enter Sleep mode to minimize the required power. This can be accomplished by using a comparator or the Brown Out Reset function of some PICmicro® MCU's reset circuits.

The voltage over time characteristic of alkaline batteries means that battery power should be regulated in some manner to provide the nominal 5.0 volts required for the PICmicro® MCU and logic. This can be done either by providing a small voltage and increasing it by a Switch-mode supply (which is described later in this appendix) or providing more than 5.0 volts and regulating it downwards (described in the next section). Although some PICmicro® MCUs can run from lower than 5.0-volt sources and some

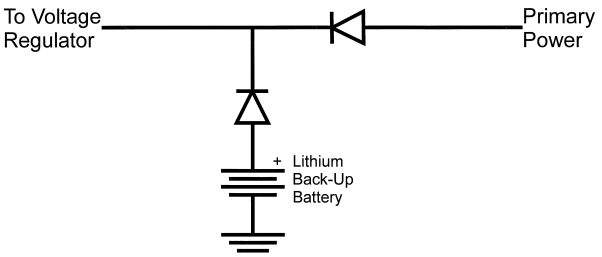


Figure 38 Isolating a battery in a supply

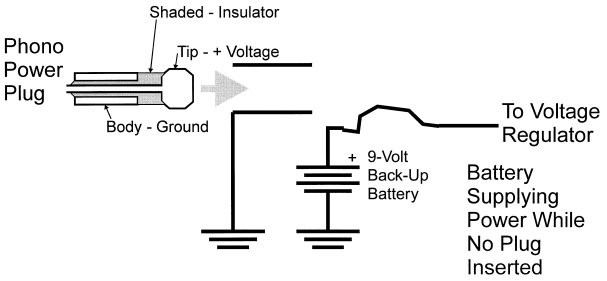


Figure 39 Battery in a multiple power-source circuit

have on-board regulators that can regulate voltage down to 5.0 volts for the PICmicro® MCU functions, you generally cannot run your applications directly from batteries. For "quick and dirty" circuits, I will often use a 9-volt alkaline battery connected to a voltage regulator (as shown in the next section).

To get the most out of the battery, I will add a wall power source socket to the circuit in such a way that the battery will provide power if the socket is not available. This can be done using the previous diode circuit or by using a socket that connects a battery when the wall source is no longer available. The power connection operation looks like Figs. 39 and 40.

In this circuit, when the "phono" plug is not plugged in, the switch built into the socket is closed and power is drawn from the battery. When the plug is installed, the battery is isolated from the circuit.

Although nicad cells can be recharged, they cannot be recharged by a straight DC power source (and some variants can be damaged by a straight DC power). Placing the battery "downstream" of the plug (Fig. 41) should never be made.

Warning! Lithium cells and alkaline cells cannot be recharged using nicad recharging equipment. Attempting to recharge the batteries can result in their exploding. Although some alkaline batteries can be recharged, only use the manufacturer's recharging equipment. When a lithium cell is depleted, throw it out!

Nicad batteries have one potential problem. If they are used for the same length of time before being recharged, they can develop what is known as *memory* and will stop producing current after this length of time. To avoid this problem, nicads should be used for varying lengths of time and be periodically "deep discharged" (run to almost zero volts output). This will break up the dendrites that build up over time and are the cause of the memory problem. For many people, memory is not a problem with their applications. Some of the newer nicad cells are designed in such a way that memory is not a problem.

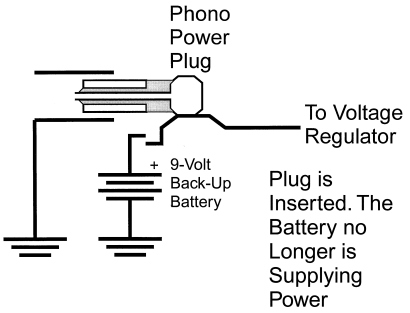


Figure 40 Battery disabled in a multiple power-source circuit

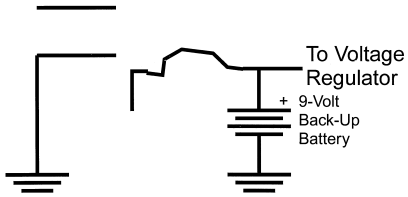


Figure 41 Invalid battery charging in a multiple power-source circuit

VOLTAGE REGULATORS

When I first started being interested in electronics. Power-supply design consisted of a number discrete parts that were designed so that the circuit would meet the requirements of the project. Designing the power supply and wiring the parts needed for it together was often a difficult chore with many people just using designs they had used for a long time and hoping that they would be appropriate for the application. I remember approaching a number of projects with trepidation because I had had some power supplies burn out on me. In one case, a specified electrolytic capacitor exploded and its replacement exploded as well. As a teenager, I developed a healthy respect for mains current because it melted the wiring in the power supply I had built.

Today, the situation has changed drastically, with the availability of cheap and simple integrated circuits that can carry out the regulation function. When these chips are used with a commercial *wall-wart* transformer, they are very safe as well. However, I still see a lot of circuits that do not use what I would consider to be an appropriate power supply.

This section introduces the issues of regulating power. I go through my favorite microcontroller power supply circuits with an explanation of why I feel they are superior.

Voltage regulators are circuits that lower an incoming voltage to a specific level that can be used by another circuit (often referred to as the *load*). Along with lowering this voltage, sufficient current must be produced to drive all the devices in the load without affecting the regulated voltage.

Earlier in this appendix, I showed how water could be used as an analogy to electrical voltage and current. This is also a useful way of describing how a voltage regulator works.

The water example, water from a higher-pressure source must be provided at a lower pressure. A common way to do this is using a bowl that contains a floating block connected to a valve that regulates the flow of water into the bowl. This is used in automobile carburetors (Fig. 42). As water is drawn out of the bowl, it is replaced when the float drops and allows more water in from the high-pressure source (Fig. 43).

Although this device is very simple and easy to understand, the actual implementation could be quite "fiddly." Issues that have to be accounted for is the ability of float of pro-

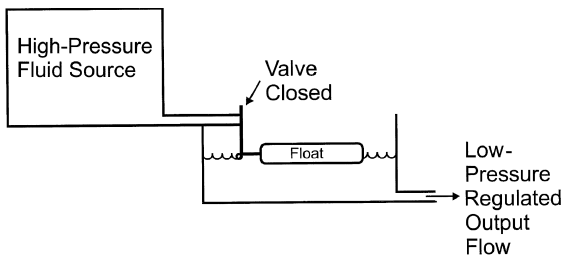


Figure 42 Car carburator as a flow regulator

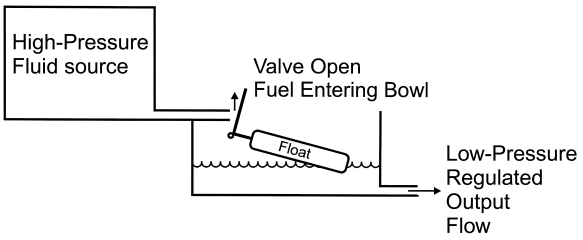


Figure 43 Car carburator allowing in fuel

vide enough force to close off the valve as, well as respond quickly when a volume of fuel is drawn from the bowl. A similar situation exists with electronic voltage regulators (which work in almost the same way as a carburetor. A linear voltage regulator has the block diagram shown in Fig. 44.

The "comparator" behaves as the float, which provides the input to the transistor (acting like the valve in the carburetor). As the current draw form the load increases, the comparator will sense the increased load, which causes an output voltage drop and allows the transistor to pass more power through.

Although this seems very simple in theory, in practice, this circuit is very hard to correctly implement (this is the genesis of my "fiddly" comment about carburetors).

One of the problems of this circuit is dissipating the heat that is dissipated by the transistor switch. When the circuit is operating, a voltage drop occurs across this transistor, along with the current being drawn across it. The power dissipation can be expressed using the formula:

$$P = V \times I$$

Where V is the voltage drop across the transistor and I is the current drawn by the load. For a voltage regulator, which supplies one amp of current at 5 volts from a 12-volt supply, the voltage drop will be 7 volts across it. The actual power dissipated will be 7 watts (7 volts times 1 amp) and will require some kind of heatsink to dissipate this power to prevent the transistor and other aspects of the circuit from being damaged.

This power loss (in the previous example, higher than what the load dissipates) gives the accurate impression that the linear voltage regulator is inefficient. For the relatively low-power applications presented in this book, they are more than adequate and the low cost of integrated devices makes them very attractive. The following sections present high-efficiency Switched-mode power supplies that are better suited for high-current applications or applications that require the voltage to be stepped up or converted into a negative voltage.

Linear voltage regulators have to be designed in such a way that they will track the load's demands and not begin to oscillate when the load changes. Oscillation is a very easy

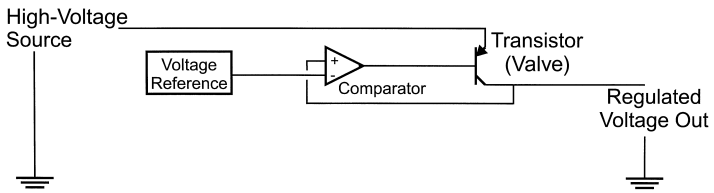


Figure 44 Simple regulator controlling voltage

trap to fall into in this type of circuit (which is why the power supply I talked about at the beginning of this section caused electrolytic capacitors to blow up). Take the case, where the load requires instantaneous increased power and the comparator and transistor switch cannot keep up. This potentially translates to the power supply providing more power when it is not required. The comparator will recognize that the power is too high and cut it down, but the change will not occur until the load power requirements has increased or stayed the same. This is shown graphically in Fig. 45.

The output "lags" the comparator output, which causes these oscillations to continue indefinitely. This is actually how an oscillator circuit works in a radio or even in the PICmicro® MCUs. There are methods to eliminate this problem, but they involve understanding the characteristics of the individual parts from the perspective of "control theory." It is really not necessary when small, easy-to-use integrated circuits provide this function for much less cost than the parts required for a circuit using discrete parts.

For my applications, I like to use the 78xx and 78Lxx series of integrated voltage regulators. These three-pin devices can provide 1 A or 100 mA of current, respectively, at a specific voltage. They are designed to avoid the oscillation problems caused by varying loads and have automatic shut-down capabilities if their internal temperatures get too high or if the load is too great for their driver circuits.

Before starting to build the circuits in this book, I suggest you invest in a few 7805s. This 5-volt regulator, which can be found for forty cents each or less, is packaged in a TO-220 transistor package (Fig. 46).

I always remember the pin out by the convention that inputs come into a schematic from the left and outputs are on right.

The 7805 is capable of sourcing up to an amp and should have a heatsink associated with it (although I often don't follow this rule). A good general rule for heatsinks is that one square centimeter of surface area is required to dissipate the heat from one watt of power. For the example above of 7 watts being dissipated by a transistor with a 7-volt drop across it and passing one amp of current is 7 square centimeters.

The 78L05 (and the 12-volt output 78L12) can supply up to 100 mA of current and is packaged in a transistor TO-92 package (Fig. 47). 78Lxx parts are more expensive than the straight 78xx versions: they can cost up to 60 cents each. Even though they are more expensive, you would be hard pressed to design a circuit that could be built as cheaply as the 78Lxx.

Notice that the pinout, relative to the label, is reversed to the 78xx parts. Because of the relatively small output current (and proportionately lower dissipated power) heatsinking is usually not required for the 78Lxx parts.

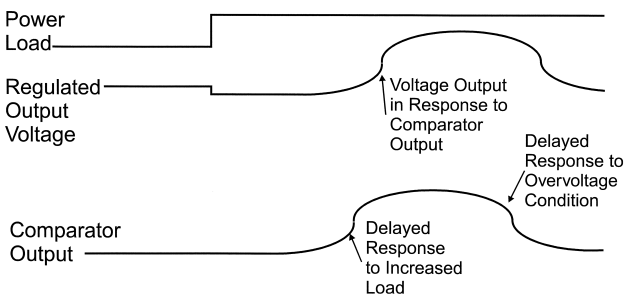


Figure 45 Power-supply oscillations

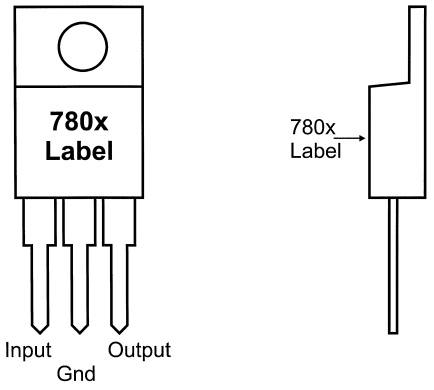


Figure 46 780x voltage regulator in To-220 package

Other voltage regulators are available with similar output capabilities and are cheaper, but lack the current- and temperature-overload capabilities of the 78(L)xx parts. The overcurrent/temperature features of the 78(L)xx parts have saved me having to replace the voltage regulators on numerous occasions. Not having this feature might save you a few cents, (literally just a few cents) but will expose you and your customers to the possibility of having to replace the voltage regulator.

A good example of this is the Parallax Basic Stamp. The number one complaint for the product is burned-out power regulators. The most recommended fix for the problem is to replace the original regulator with a 78L05.

The 78(L)xx voltage regulator's output can be "shifted" upward to provide different voltages by providing a different ground (which acts as a voltage reference) to the rest of the application. I'm mentioning this because the PICmicro® MCU requires a +13- to +14-volt supply for programming and I don't want to design a unique power supply for this application unless I can help it.

The easiest way to provide this capability is to use silicon diodes to shift the ground reference by the 0.7 volts of the forward junction. Adding two silicon diodes (Fig. 48) will create a 13.4-volt power supply, which can be used to program the PICmicro® MCU.

The inputs and outputs to voltage regulators should have filtering capacitors on their inputs and outputs. The typical filter capacitor I use is a 10-μF electrolytic across the input

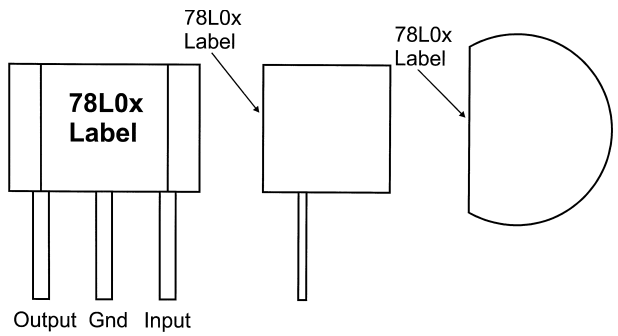


Figure 47 78L0x voltage regulator in TO-92 package

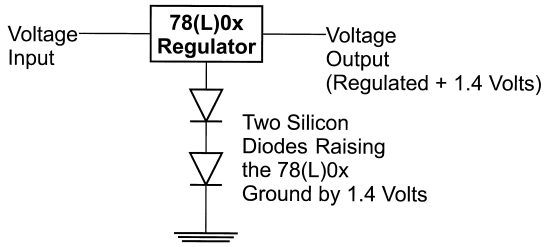


Figure 48 Raising a voltage-regulator output

and a 0.1-μF tantalum across the output. The basic circuit for using these parts is shown in Fig. 49. Both the 10-μF electrolytic and 0.1-μF tantalum capacitors should be "derated" by 50% or more to eliminate any chance they will burn out or blow up. This is to say the actual voltage rating should be twice (or more) the voltage levels that the parts are subjected to in the application.

These capacitors will eliminate any noise from the input voltage and help provide a very clean supply to your application circuit. They are also required for proper operation of the voltage regulator chips. You might find that they work properly without these capacitors for small loads, but for larger loads, the capacitors are definitely required.

A good idea for your applications should be to provide a power-on indicator to your circuit. For this function, I normally use an LED and current-limiting resistor. You could drive the LED from the PICmicro® MCU (and use it for other functions as well). But for prototyping, be sure that you have an LED powered by the power supply so that you don't do something dumb like add or remove parts while the power is on (or even solder a circuit with the power on). The latter situation can be potentially dangerous because you could have a short circuit on mains power lines.

If a switch is to be added to turn on and off the circuit, then it should be "upstream" of the input filter capacitor. This makes the recommended power-supply circuit look like the Bread-Board Power Adapter presented later in this appendix.

I'm mentioning this because sometimes it will be easier to wire in the switch between the capacitor and the voltage regulator. In this case, the switch could be opened and input voltage removed with a charge left on the capacitor. In this case, you might get shocks for no apparent reason.

Another type of voltage regulator circuit that is useful, quite inexpensive, and well suited to applications where parts might be inserted or removed while power is applied. Going back to the carburetor analogy to the voltage regulator, there is another, simpler way to provide water at a specific pressure: let higher-pressure fuel fill the bowl and allow anything extra to spill over. The fuel in the bowl will provide a constant-pressure output at the bottom of the bowl (Fig. 50).

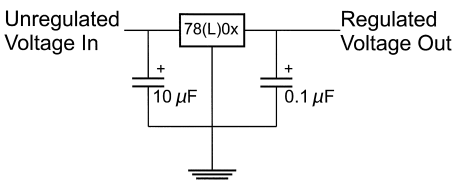


Figure 49 Using a 780x as a voltage regulator

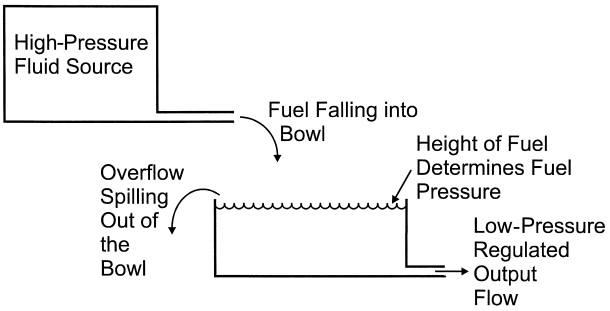


Figure 50 Simple carburator operation

This type of pressure regulator is obviously very inefficient because of the overflow spilling out of the bowl. This overflow can be minimized by limiting the amount of fuel falling into the bowl to the volume of fuel to the amount output from the bowl.

This method of voltage regulation can be provided using a zener diode and a current-limiting resistor. A zener diode is an interesting beast because it behaves like a regular diode unless the voltage across it is greater than its rated voltage, at which point it passes current until the voltage across it is the rated value.

A +5-volt regulator can be built using a 5-volt zener diode and a resistor (Fig. 51).

In this circuit, the voltage across the zener diode's anode will be always be at 5.1 volts, unless the load is so high that it exceeds the current available through the resistor.

The current-limiting resistor is used to regulate the amount of current that is available to the load. Without it, the zener would pass as much current as the V_{in} power supply could source (essentially being a short circuit). The current-limiting resistor provides a voltage drop and current limit to avoid this problem.

The value of the current limiting resistor is calculated from the expected voltage input the zener voltage and current drawn by the load. For example, if 10 volts was supplied by the voltage input supply and 5 volts was required by the load and it required 25 mA, the resistance value would be calculated by first finding the voltage across the resistor and the current through it.

In this example, the voltage across it is 5 volts (10 volts − 5 volts) and 25 mA through it. Using Ohm's Law, the resistance is:

```
R = V/I
  = (10 V − 5 V)/25 mA
  = 5 V/25 mA
  = 200 Ω
```

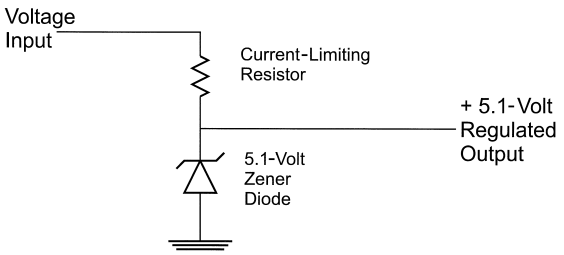


Figure 51 Zener diode voltage regulator

So, a 200-Ω current-limiting resistor is required for this power supply in this application. As well as the resistor value, you should be cognizant of the power being dissipated through the resistor and zener diode. For the resistor power is calculated as:

```
P = V × I
  = (10 V − 5 V) × 25 mA
  = 5 V × 25 mA
  = 125 mW
```

which means that a 1/4-watt resistor can be used safely.

For the zener diode's power rating, I always go with the worst case, which is if there is no load. In this case, because the voltage diode is acting as the resistor, the power dissipated is also the same (125 mW). A 1/2-watt zener diode can be used safely in this circuit.

This circuit is best suited for situations where low power is required. The circuit is essentially a voltage divider (which is described in more detail elsewhere in this appendix) and is not well suited for supplying current to high power loads. This characteristic makes it well suited to applications where chips with the load can be pulled out and replaced with power active. Like the linear voltage regulator, this circuit should have filtering caps provided with it to avoid noise from the voltage input power supply from being passed to the load circuit and visa versa.

SWITCH-MODE SUPPLIES

The voltage regulator (or "linear regulator") is used to provide a regulated voltage from a higher, often unregulated voltage. As shown, the voltage regulator is accomplished by limiting power to the load. This method is quite inefficient in terms of power and is not a good way of using batteries in an application.

For example, if a 9-volt battery is used to drive a 100-mA load at 5 volts, stepping down the 9 volts to 5 volts will cause a 4-volt drop at 100 mA and consume a total of 0.4 watts. Because the application consumes a total of 0.5 watts, it is only being 55% efficient with the power being supplied to it. This isn't a problem for a PICmicro® MCU application, which is powered from an AC mainline, because the total power is relatively small, but it is an issue for battery-powered applications.

The *Switch-Mode P/S (SWPS)* is a device that uses the characteristic of coil fly back after current passing through it is turned off. By rapidly switching on and off a small coil, a higher voltage can be produced by the circuit shown in Fig. 52. This one of the most popular of four or five SWPS designs available.

In the circuit diagram, the N-channel MOSFET is used as a switch and should be capable of handling the peak current loads. A PWM signal (switch control) turns the transistor on and off. The purpose of the diode is to prevent current from "sliding back down" into

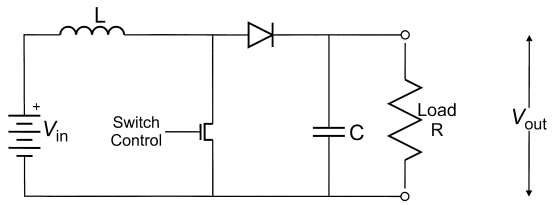


Figure 52 Switch-mode power supply circuit

the switch after being driven from the coil. Finally, the capacitor is used to filter out as much voltage ripple as possible.

This circuit will transform V_{in} in to V_{out} for a given load based on a simple set of equations.

$$I_{peak} = 2 \times I_{out} \times (V_{out}/V_{in})$$
$$T_{off} = (L \times I_{peak})/(V_{out} - V_{in})$$
$$T_{on} = [(V_{out}/V_{in}) - 1]$$

These formulas might seem like they are recursive. That is to say they are based on each other, but this reflects one of the basic aspect of SWPS design. The process is iterative, based on the parts available for the power supply and the requirements of the load.

The T_{on} and T_{off} values are used to determine the PWM's characteristics. T_{on} is the time that the switch is on and current is passing through it. T_{off} (also known as T_{don} in some references) is the time the switch is off and current is flowing through the diode. These two values are combined to produce the switch PWM signal shown in Fig. 53.

When I was working on the YAP-II and EMU-II, one of the concepts I tried was to use the on board PICmicro® MCUs to drive a SWPS that would produce the 13 volts necessary for V_{pp}. For this case, I wanted to raise 5 to 13 volts for a 50-mA load using a 50-kHz PWM frequency generated by the PICmicro® MCU. Using these formulas, I wanted to calculate T_{off} and T_{on} using a 22-μH coil (which I happened to have on hand). To generate the switch-control PWM signal, I wanted to use a PICmicro® MCU running at 4 MHz (1 ms/instruction cycle). These formulas could then be applied:

$$\begin{aligned} I_{peak} &= 2 \times I_{out} \times (V_{out}/V_{in}) \\ &= 2 \times 50\ \text{mA} \times (13/5) \\ &= 260\ \text{mA} \end{aligned}$$

$$\begin{aligned} T_{off} &= L \times I_{peak}/(V_{out} - V_{in}) \\ &= 22\ \mu\text{H} \times 260\ \text{mA}/(13 - 5) \\ &= 22\ (10^{-6}) \times 260\ (10^{-3})/8 \\ &= 0.715\ \mu\text{s} \end{aligned}$$

$$\begin{aligned} T_{on} &= T_{off} \times [(V_{out}/V_{in}) - 1] \\ &= 0.715\ \text{ms} \times [(13/5) - 1] \\ &= 1.144\ \mu\text{s} \end{aligned}$$

$$\begin{aligned} T_{period} &= T_{on} + T_{off} \\ &= 1.859\ \text{ms} \end{aligned}$$

For this case and this is where I mean determining the values is an iterative process, T_{on} and T_{off} specified are too fast to be produced by the PICmicro® MCU. So, to implement this PWM, I would have had to change the parameters and parts until I got components that met the needs of the application.

For example, if the coil was changed to 5 μH, the values for T_{on} and T_{off} become

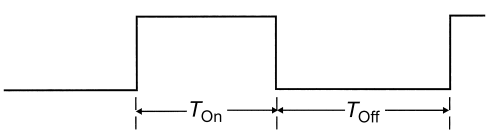


Figure 53 Switch-mode power supply PWM signal

0.1625 ms and 0.26 ms, respectively. This is much faster then what I had previously (and even more inappropriately used for the PICmicro® MCU to drive the PWM in the SWPS). Instead, the coil size has to be increased: going up to a 220-mH coil, the values for T_{on} and T_{off} are 7 μs and 11.4 μs, which are easier to output from a PICmicro® MCU running at 4 MHz.

When this PWM is being output by a PICmicro® MCU which is driving the output using bit-banging code, it will be very difficult (if not impossible) for the microcontroller to execute "mainline" code and do other work. These calculations show that the PICmicro® MCU is not well suited as a SWPS controller, unless the built-in PWM output available in some PICmicro® MCUs is used.

Instead of using a PICmicro® MCU, many chips are available on the market that will control an SWPS cheaply and reliably. Many of these chips also monitor and regulate the output voltage level (something that can't be done easily with a PICmicro® MCU), as well as provide a method to produce negative voltages and "step-down" Switch-mode power supplies.

If you are just looking for a clocking circuit to drive the switch control, then I recommend looking at the venerable 555 timer. This chip is cheap and a lot of information is written about it (including using it for this type of application).

MAINS VOLTAGE CONVERSION

By far, the most popular way to provide power to an application is by simply plugging the application into the wall. For the most part, this method is cheap and reliable, but working with mains power does have some risks associated with it. This section introduces the issues of working with 110 (and 220) volts *Alternating Current (AC)* and using it to power your applications. The next section provides a circuit and PC board design that can be used to power most of the applications presented here, as well as your own projects.

I must caution that the power coming out of your wall socket can conceivably destroy your PICmicro® MCU application, burn you, cause a fire, or even electrocute you. Despite the fact that it is commonly used for appliances, lighting, and electronic devices in the home, it is not to be trifled with. At the end of this section, theory and a circuit are provided, which can be built safely and inexpensively. I highly recommend that this circuit or a commercial bench power supply be used with your PICmicro® MCU projects.

Power coming from your wall sockets ("the mains"), comes in as either 110 or 220 volts peak to peak as a sine wave with a frequency of 50 or 60 cycles per second (Hertz, Hz) as shown by Fig. 54. In North America, power is provided at 110 volts (or higher) peak-to-peak voltage (typically 115 volts) at 60 Hz. Different countries around the world use dif-

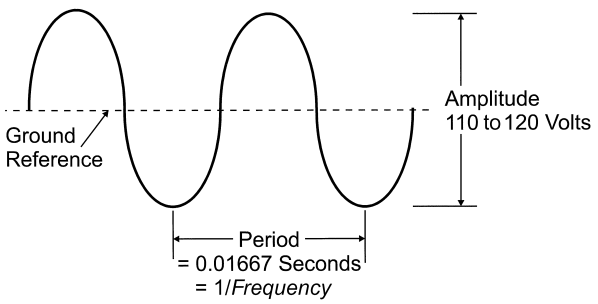


Figure 54 North American 110-V, 60-Hz power

ferent peak-to-peak voltage levels and operating frequencies. If you are going to design a power supply for a specific country's use, be sure that you understand the characteristics of the power supply. The circuits provided in this book might not be appropriate for your country.

As well, be sure that you understand what are the legal requirements for circuits that plug into mains power for specific countries. The information provided here is strictly "rule of thumb" to ensure that the circuit is safe for use in North America. The information contained here might be incorrect or illegal for different jurisdictions within North America. These safety and legal issues are the primary reasons why I recommend that you avoid building your own mains interfacing power supplies unless there are compelling reasons to do so.

This power coming in is normally provided by a "socket," which is built into your walls. Figure 55 shows the layout of the socket and labels the individual connections. *Power* is the alternating voltage sine wave shown in Fig. 54. *Neutral* is the return rate for this voltage. *Ground* is a shunt to earth ground if the circuit is damaged and the live voltage is not returned to the neutral connection.

This power input is *Alternating Current (AC)* and works somewhat differently from the basic *Direct Current (DC)* circuits described earlier in this appendix. Notice that the ground is not used as a return path like, as in the DC circuit. The neutral line should be considered as the only return path for current. Ground (Gnd), as mentioned, is an emergency return path if the circuit or power cord running from the socket to the device is damaged. Normally, ground is connected to the device's case to prevent the user from getting a shock from the circuit (Fig. 56).

Because the voltage coming from the mains is so high, it has to be converted into a lower DC voltage for the PICmicro® MCU application circuit. This is done in three stages. The first reduces the voltage from more than 100 volts to 10 volts (or less) using a transformer. A *transformer* is a device consisting of two coils that share their magnetic field. As is shown in Fig. 57, the transformer can simply consist of a magnetic toroidal ("doughnut shaped") core with power supplied at the primary-side coil and the transformed AC voltage out provided at the secondary-side coil. Figure 57 also gives the relationship between the voltage and current on the secondary side coil based on the number of turns for each coil.

Notice that the current is inversely proportional to the turns ratio. In North America (which has 110 VAC), an 8:1 transformer is often used. Thus, with 110 volts in, there will be 14 volts out. For 220 volts, a 16:1 transformer should be used for the same voltage output.

Although the voltage has been lowered, it is still AC that is going positive and negative. This voltage has to be "rectified" into a straight DC voltage. This is done using diodes in either a half-wave or full-wave rectifier (Fig. 58).

Full-wave rectifiers transform the positive and negative lobes of the AC circuit into a positive voltage. Half-wave rectifiers clip the negative wave (resulting in half the total power available to the circuit). For this reason (and because full-wave rectifiers can be cheaply purchased in electronic stores), I prefer to use the full-wave rectifier.

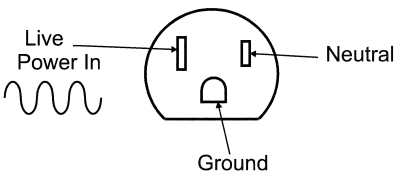


Figure 55 North American 110-V, 60-Hz wall plug.

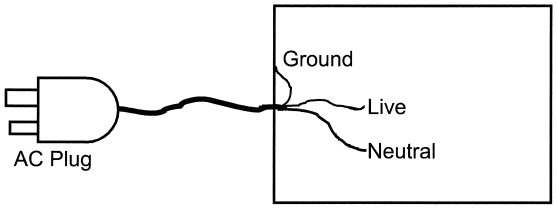
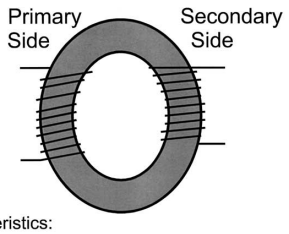


Figure 56 AC plug connection



Electrical Characteristics:

$$\frac{\#\ Turns\ primary}{\#\ Turns\ secondary} = \frac{Voltage\ primary}{Voltage\ secondary} = \frac{Secondary\ current}{Primary\ current}$$

Figure 57 Transformer construction and operation

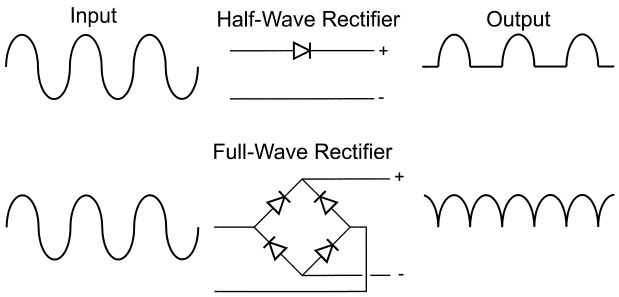


Figure 58 AC current rectifiers

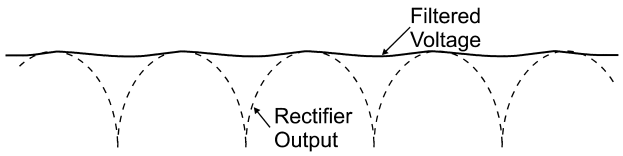


Figure 59 Filtered and rectified AC current

Inputting the rectified signal directly from the diodes into a voltage regulator should not be attempted. Instead, use a filtering electrolytic capacitor of a few tens of μF. The filtered signal output from the full-wave rectifier looks like that shown in Fig. 59.

As long as the rectified signal does not drop below the minimum voltage of the voltage regulator, the regulated DC voltage output will be constant. The filtering cap should be a minimum of 10 μF. A good general rule is that for digital circuits, 20 μF is required for each amp of current drawn. For DC electric motors, this value increases to 100 μF per amp drawn to help prevent inductive "kick-back" spikes from being driven back through the transformer to the mains circuit. For the +5-volt power supplies presented in this book, a 10-μF electrolytic capacitor is used to provide 500 mA to the PICmicro® MCU application.

Using the transformer, a full-wave rectifier, an electrolytic filter capacitor, and a 7805 voltage regulator, a +5-volt 0.5-amp power supply for PICmicro® MCU applications could be created (Fig. 60).

Notice a few things in this circuit. First, the mains ground is connected to the case and not to the digital ground. In any DC-powered circuit, the negative terminal of the full-wave rectifier can be called *digital ground*, but should be left floating relative to earth ground, which is provided by the AC plug. In this case, *digital ground* is simply a common negative terminal for the circuit.

I have put a fuse in the power line, which will cut out in high-current draw situations (like short circuits). It is rated at 0.1 amps, which might seem low, but remember that current output is inversely proportional to the turns ratio of the transformer. 0.1 amps at 110 volts translates into 1.4 amps at 8 volts at the output. Without this fuse, very large (and very dangerous) currents could build up inside the circuit. For example, 2 amps at 110 volts translates to 28 amps at 8 volts (224 watts of power).

If you can imagine the heat from a 60-watt bulb and how easily it can burn you, think of what can happen with almost four times the power!

Along with the fuse, the switch in the circuit should be certified for switching AC voltages. AC switches usually have a mechanical assembly inside them that snaps the switch contacts on and off. This minimizes arcing within the switch. This might seem hard to believe, but if you look inside an AC switch while it is opening or closing, you will see a blue spark and sometimes hear a "pop." This is caused by high inductive voltages produced by the transformer coils that "kick back" when the AC power to the transformer is shut off.

Notice that the switch is connected to the live wire after the fuse. A switch should be on the live terminal of the circuit to prevent active voltages when the switch is opened. I rec-

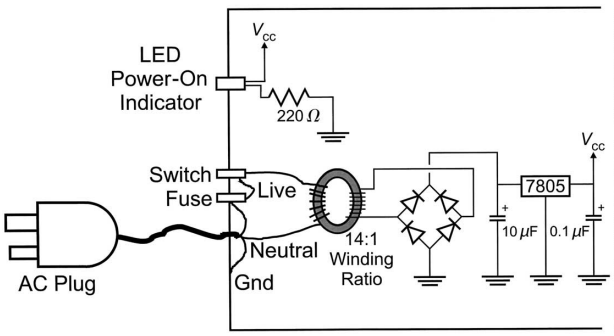


Figure 60 Complete voltage regulator from mains

ommend only working on the circuit when the power cord is unplugged from the wall socket, but putting the switch on the "hot" wire helps reduce this risk.

If you do build mains power-supply circuits like this one, I recommend that you use 14-gauge stranded wire for all connections with heat-shrink tubing over all solder joints and bare wire. As well, only UL/CSA-approved (or the local country testing organization) plugs, wires, switches, fuse holders, and transformers should be used in a properly grounded metal case.

If any of these terms are unfamiliar to you or if you doubt your ability to build the circuit safely, then don't build it!

After going through several paces explaining the theory and practice of a mains power supply, I want to present to you what I consider to be a superior method to provide mains power to your PICmicro® MCU (or other digital logic) application. Instead of the plug, wire transformer, case, fuse, switch, and rectifier combination, I suggest that you use a wall-socket-mounted AC-to-DC power supply (often known as a *wall-wart*) similar to the one in Fig. 61.

This device encapsulates many of the elements listed, is UL/CSA approved, is much easier to work with, and will be cheaper than if you bought the parts separately. Chances are, you already have a few wall-warts in your home that are used for toys, personal tape and CD sound systems, or other electronic devices. As long as you have a few volts greater than the regulator's output (I use three volts above the regulated output as a general rule), then you don't have to buy anything at all. The regulator circuit shown in Fig. 60 can be simplified to that shown in Fig. 62, which is much cheaper and safer than the do-it-yourself transformer solution. This is the basis of the power supply circuit presented in the next section.

When using a wall-wart power supply, be sure to use a power plug instead of a phono plug for connecting to your application. A power plug is different from a phono plug in that the positive terminal is a barrel inside of the outside negative terminal (Fig. 63).

Figure 61

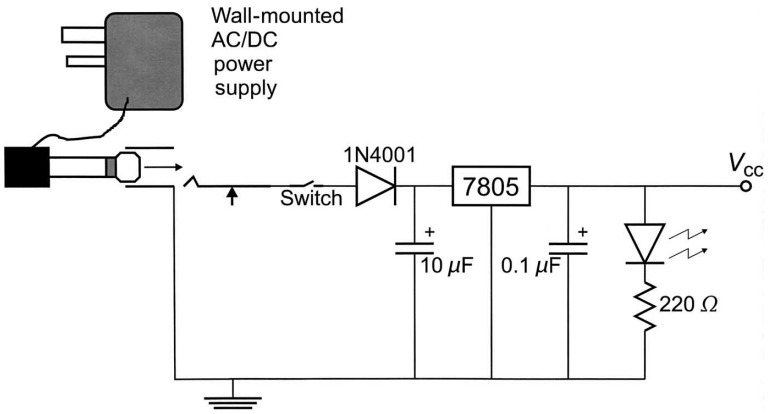


Figure 62 Digital power supply using a wall-mounted AC/DC supply

The advantage of the power plug is that the positive and negative terminals can never be shorted together (as in the case of the phono plug). The socket for the power plug has a pole that is inserted into the power plug. Like the phono plug, power-plug sockets are available in designs that will disable alternative power sources (e.g., batteries) when the plug is inserted. I realize that I used a phono plug to describe the action of a socket in the previous section, but this was done because the operation of the phono plug is easier to visualize (and draw) than the power plug.

Example application power source One of the most useful circuits you can build for yourself is a simple 5-volt power supply. The last few sections have introduced the issues regarding power supplies. To round off this description, I want to give you the design for a simple power supply (Fig. 64) that you can use to build the applications presented in this book.

The power supply can be powered either from a wall-wart power supply or a 9-volt alkaline radio battery, which provides up to 1 amp at +5 volts. The circuit itself is very simple (Fig. 65).

The bill of materials for this circuit is shown in Table 11. In this circuit, I used a center-pole power connector, which has the battery ground interrupted when a power plug is inserted into the connector. In the previous sections, I showed a phono plug, which interrupts the power. For most applications, the power plug is used because it avoids a chance for shorting and arcing when the plug is inserted or withdrawn. Disconnecting the ground (negative) connection of the battery might seem unnatural (especially after I have described it as the common connection to the circuit), but it serves the same purpose as disconnecting the battery's positive connection. The unregulated power sources can be a home electronics 8- to 15-volt wall-mounted transformer that can be bought new for just a

Figure 63 Power-plug types

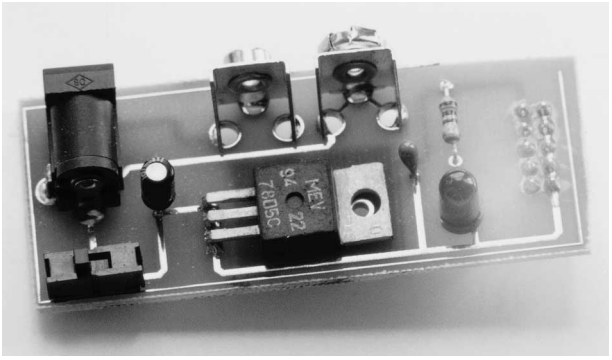


Figure 64

few dollars. Chances are that you have more than a few lying around your home. If you build any of the programmer or emulator circuits, this wall wart can be shared between them if the supply outputs 14 volts (or higher). The higher voltage is required to provide a stable programming voltage.

If just this power supply circuit is going to use the wall transformer adapter, as little as eight volts are required. Notice that a DC output is required with up to 1 amp of current if the full capabilities of the 7805 voltage regulator are to be realized.

The higher the input voltage, the greater the voltage drop across the 7805 regulator and the more power (heat) that will have to be dissipated. You might want to put a heatsink on the 7805 to reduce its temperature and the opportunity for "thermal shutdown."

You might be considering building your own transformer/rectifier circuit instead of buying a wall wart and I wanted to discourage you from doing that. I am personally amazed at how cheaply you can buy a commercial wall transformer. At a local (Toronto) surplus store, I can buy a new 0.5-A, 9-volt power supply for $3.00 Canadian ($2.00 USD). The "Cadillac" wall wart with multiple voltages and output options is $10.00. There is no way I can buy the parts this cheaply. Using the commercial unit will save you time, money, and avoid any shock hazards that you would have for a circuit that you build yourself.

As the photograph for this power supply shows, I have laid this circuit out onto a PC board with the opportunity for the power outputs being plugged directly into a "bread-board." The layout's "Gerber files" are located on the book's CD-ROM or the following layouts can be used to design your own PC board boards (Fig. 66).

This circuit could also be built on the various prototyping systems described in this appendix.

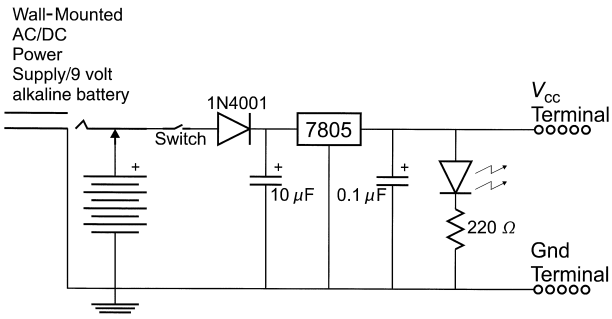


Figure 65 Bread-board power adapter

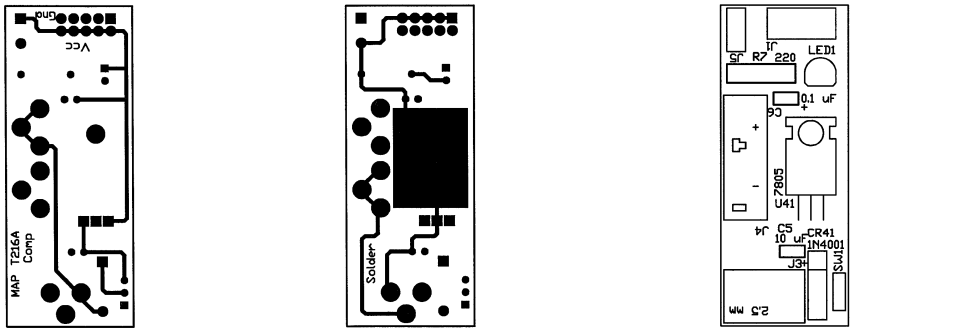
TABLE 11 Basic Digital Gates

PART	DESCRIPTION
7805	7805 5 Volt Regulator
1N4001	1N4001 Silicon Diode
LED	0.1″, 5 mm Red LED
10 μF	10 μF, 35 Volt Electrolytic Capacitor
0.1 μF	0.1 μF, 16 Volt Tantalum Capacitor
220	220 Ω, 1/4 Watt Resistor
Power Input	2.5 mm Power Plug
Battery	9 Volt Battery PCB Mount Connectors. Digi-Key Part Numbers BSPCF-ND and BSPCM-ND
Terminals	2 wire screw terminals and 2x5 IDC male connectors
Misc	PCB

Digital Electronics

Handling digital signals is one of the easiest things that you will have to do in electronics—and one of the most fun. The PICmicro® MCU will interface directly to most forms of digital circuits, just as most digital circuits will interface directly to each other.

Digital electronics consists of passing a signal, which is either a *1* or a *0*, depending on its voltage level between circuits and processing them. This processing is a branch of mathematics known as *Boolean mathematics,* named after George Boole, a nineteenth-century mathematician who was looking for a way to express symbolic logic mathematically. I have always felt that it is somewhat ironic that a tool developed to interpret philosophical statements has been used to design computer electronics.



PWRSPPLY Top Layer PWRSPPLY Bottom Layer PWRSPPLY Overlay

Figure 66 Power-supply PC board information

In Boolean mathematics (also known as *Boolean logic* or just *logic*), four primary operations can be carried out. Each operation can be used to test and/or modify a digital value based on its and other values. Digital values can be represented as *1* or *True* and *0* or *False* (Table 12).

These functions are available as discrete functions in a chip or are combined to provide complex functions in a chip. As I work through the next sections of this appendix, you will see how they can be combined to form more complex functions. In this section, I wanted to introduce you to the different logic functions, as well as describe some of the interfacing issues of digital circuits. The logic type that is most commonly used with PICmicro® MCUs (and most modern electronics) is known as *TTL (Transistor-to-Transistor Logic)*. As the name would imply, it is designed as transistor "drivers," which pass the Boolean logic levels to transistor "receivers." A number of different types of TTL logic are available—each with slightly different input threshold and output voltages.

When you look at a catalog of TTL chips, you'll find a bewildering array of different chip families available. Each chip starting with "74," followed by a letter code and a number. The letter code denotes the family the chip belongs to. The letter code (which might

TABLE 12 Basic Digital Gates

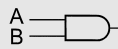
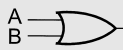
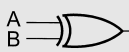
OPERATION	ELECTRONIC SYMBOL	DESCRIPTION AND "STATE TABLE"
NOT ("!")		Invert the input Signal. Input | Output 0 | 1 1 | 0
AND ("*")		Return "True" if both inputs are "True". A B | Output 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1
OR ("+")		Return "True" if either input is "True". A B | Output 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1
XOR ("^")		Return "True" if only one input is "True". A B | Output 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 0

TABLE 13 Logic Thresholds And Outputs For Different Logic Families

TYPE	THRESHOLD	"0" OUTPUT	"1" OUTPUT
TTL	1.4 Volts	0.3 Volts	3.3 Volts
HC	2.4 Volts	0.1 Volts	4.9 Volts
HCT	1.4 Volts	0.1 Volts	4.9 Volts
CMOS	2.5 Volts	0.1 Volts	4.9 Volts

not be present, or is S, LS, HC, HCT, AL, or ALS) is used to denote the type of circuit and driver. Each of these families work slightly differently and at different speeds (as well as require different amounts of current to operate).

The term *threshold* is used to specify the voltage in which an electrical signal is determined to be a *1* or *0*" For positive logic (which TTL is), a *1* is a high voltage, and a *0* is low voltage. Table 13 lists some different types of logic and their threshold and output voltages.

The circuits presented in this book are designed to work with *LS (Low-power Schottkey)* logic, which is a reasonably low-power logic family that works fast enough and with enough current drive for most application functions. If another type of a logic family is required, I make note of it and why. Table 14 lists the gate transition times (in terms of nanoseconds), which is the speed measurement of the different families, as well as their current sink capability, which is the amount of current each gate can pull to ground. I consider the current sink capability to be the most important consideration of each logic family because circuits driving an LED must have sufficient capability to light it.

TABLE 14 Gate Transition Times And Current Sink Capability For Different Logic Families

FAMILY	TRANSITION TIME	MAXIMUM CURRENT SINK
Straight TTL (No Letter)	8 nsec	12 mA
"L" TTL	15 nsec	5 mA
"LS" TTL	10 nsec	8 mA
"S" TTL	5 nsec	40 mA
"AS" TTL	2 nsec	20 mA
"ALS" TTL	4 nsec	8 mA
"F" TTL	3.5 nsec	20 mA
"C" CMOS	50 nsec	1.3 mA
"HC/"HCT CMOS	9 nsec	8 mA
"4000" CMOS	30 nsec	0.5 mA

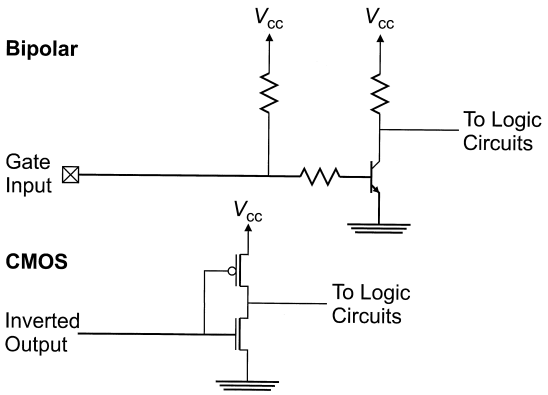


Figure 67 Digital input circuits

Table 14 lists general values for each logic family. Notice that the transition time is really per gate; multiple stage chips will have much slower transition times.

Two types of logic circuits are in common use today. *Bipolar (straight TTL)* uses NPN and PNP bipolar transistors for operation. *CMOS* uses *N-channel (NMOS)* and *P-channel (PMOS) MOSFETs* for the logic functions. These different transistor types reflect on the speed and drive characteristics of the different families.

This difference can be seen in the input circuits shown in Fig. 67. The bipolar input, drives a NPN transistor with a pull-up that provides an inverted signal to other circuits in the chip. The CMOS input drives a PMOS/NMOS inverter.

Notice that the bipolar input is pulled up, but the CMOS input is not. Elsewhere in this appendix (and the book), I use pull ups to the inputs of circuits. I don't depend on any built-in pull-up circuits when adding buttons and other circuits. This is a good approach to take when you design digital circuits because the actual circuit might not have an internal pull up, depending on the part that is chosen. By always placing a pull up on an input that isn't actively driven, you will avoid any issues where a floating input (which has an undefined value) is encountered. Pull ups are described in more detail later in this appendix.

The two types of digital-output circuits are shown in Fig. 68. The totem-pole output provides a push-pull driver for the circuit. It can both sink (pull down to ground) and source (provide from the power supply) power to the load (receiver) that it is going to drive. This circuit can be provided in either bipolar or CMOS logic.

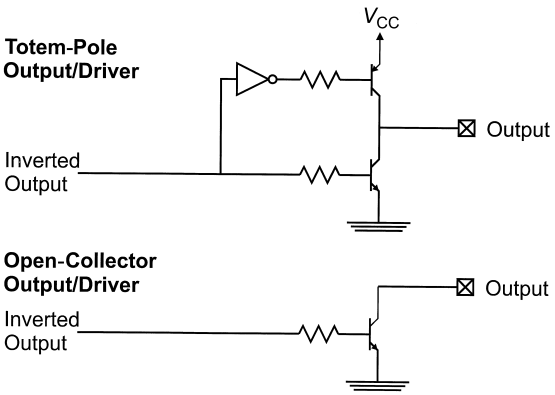


Figure 68 Digital output circuits

The second type of output in Fig. 68 is known as *open collector* because the collector of the output transistor that pulls to ground is "floating" or "open" to being externally pulled up. This CMOS equivalent is known as an *open drain driver*. This output must have a pull up when driving loads that don't have their own internal pull ups. They are best suited in special situations, where the ability to pull down a line is required.

One of the most common of these situations is the *Dotted And bus* (Fig. 69), in which multiple open-collector drivers can be tied together to provide a single logic function. This logic function is low unless every input to it is high. Because this function simulates the AND logic function and in many design systems, a dot would be placed on each output to show that they are connected together. Thus, this circuit is known as a *Dotted And*. Elsewhere, this book, shows how the open-drain output and Dotted And are used with the PICmicro® MCU in different applications and situations.

COMBINING AND OPTIMIZING DIGITAL CIRCUITS

When you first start working with digital circuits, you're going to have a lot of fun with combining simple logic gates. Once you understand the basic logic functions, you will see how they can be combined to create complex functions that are needed for applications. All digital logic chips (such as the PICmicro® MCU microcontrollers) consist of multiple gates, arranged in such a way as to implement a function. This section introduces the issues of combining circuits: some of the available tools that can help you and some things to watch out for.

The output of logic gates can be passed to the inputs of other logic gates to form these complex functions. Figure 70 combines four inputs together using an inverter, three AND gates, and an OR gate to implement the function:

```
Output = (A * B * C) + (B * _C * D)
```

which could be used in such things as selecting a value or decoding input data. Different logic circuits can be implemented for an infinite number of requested functions.

The equation above is known as an *output equation* because it relates the output value to the inputs.

Notice a number of things in Fig. 70 and in the equation listed previously. In the equation, I replaced the AND functions with an asterisk (*) and the OR function with an addition character (+). In standard mathematics, the * and + are normally used for multipli-

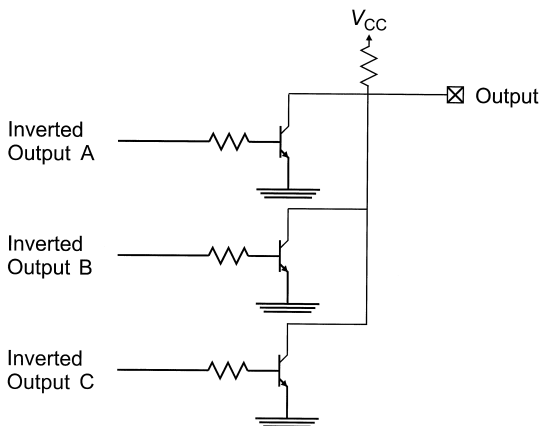
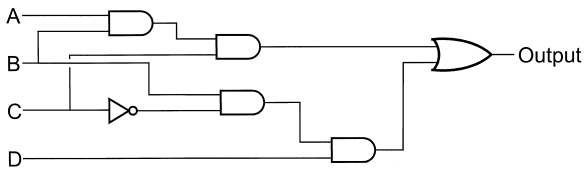


Figure 69 Dotted And digital circuits



Output = *(A * B * C) + (B * _C * D)*

Output has three or four Gate Delays

Figure 70 ANDs, NOT, and OR combined as logic

cation and addition, respectively. In digital logic or Boolean arithmetic, multiplication is assumed to be an AND operation and OR is addition.

The underscore before the *C* indicates that it is inverted. In some references, a bar will be over the character (or an expression). I use the leading underscore because it is available in the keyboard and does not require any graphics. Another way in which an inverted expression is indicated is by a leading exclamation mark (*!*).

Combining these functions (known as *gates* when the circuit is wired) to find a specific output is known as *Boolean arithmetic*. Notice that the result of a Boolean arithmetic statement is always either *1* or *0*. This single set of values is where the term *digital logic* comes from.

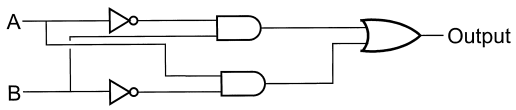
Each gate has its own characteristic delay. For LS TTL logic, I assume a 20-ns delay for each gate. Thus, for the circuit shown in Fig. 70, the delay from the input to the output will be 60 or 80 ns. To ensure there are never any differences between the logic functions, you will find that many circuit designers will add logic delays using AND and OR gates that don't change the output. These delays are used to match the delays in the circuit and avoid changing output timing for different inputs.

In the PICmicro® MCU projects presented in this book, the time through the gate is really not an issue, except if TTL logic chips with delays of 50 ns (or longer) were used in the applications. I just wanted to introduce you to the concept of gate delays and the effects they can have on your circuits.

Using the multiplication and addition terms described, the previous equation is said to be in the "sum of products" format. I use this convention when writing out digital functions. "Product of sums" format is also possible, but not as often because it can be somewhat hard to understand what the operations are actually doing. Sum of products operates by passing the inputs through the filtering AND gates first and then combines the filtered inputs with an OR gate as the final output.

Notice that most people (and I) very rarely use an XOR gate in these expressions. The reason for this really comes down to the idea that an XOR gate is actually a combination of gates (Fig. 71). Often, the XOR function is produced as an intermediate value by the logic circuits, so individual XOR gates are not required in the digital circuit.

Notice that in the XOR gate function shown in Fig. 71, three gate delays are available for the XOR output. This is quite typical of actual logic. Bipolar and CMOS logic circuits cannot



A XOR *B* = (*A* × *_B*) + (*_A* × *B*)

Figure 71 XOR gate from ANDs, NOTs, and OR logic

easily implement XOR functions and the input signals take a long time to "propagate" through the gates to the output. For this reason, XOR gates are generally avoided by many designers of high speed circuits unless there is no other way to implement a required function.

To understand how the digital logic function presented works, the first tool that you should use is the *truth table*. This table takes all the possible inputs and lists the intermediate values, as well as the final output. For the output function, I can create the truth table shown in Table 15.

In the truth table, I typically put in the values for the different AND filters of the sum-of-products logic function. This allows me to better see what the circuits are doing before being passed to the output.

Notice that for each product, $(A * B * C)$ and $(B * \_C * D)$, two cases are true and return a one. These two inputs are ORed together for an output. When creating truth tables from a logic diagram (Fig. 70), the number of *1* outputs should match the number of inputs to the OR gate.

Most logic circuits are designed from a "black box," like the one shown in Fig. 72. The black box takes a series of inputs (*A*, *B*, *C*, and *D* in Fig. 72) and performs the logic comparisons described previously to get a specific output. The best way to understand these outputs is to place them into a Karnaugh map (Fig. 72).

A Karnaugh map is a tool used to find relationships in inputs and outputs. For the example circuit shown in Fig. 72, the truth table is shown in Table 16.

TABLE 15 "TRUTH TABLE" For Logic Function

D C B A	(A * B * C)	(B * _C * D)	OUTPUT
0 0 0 0	0	0	0
0 0 0 1	0	0	0
0 0 1 0	0	0	0
0 0 1 1	0	0	0
0 1 0 0	0	0	0
0 1 0 1	0	0	0
0 1 0 1	0	0	0
0 1 1 1	1	0	1
1 0 0 0	0	0	0
1 0 0 1	0	0	0
1 0 1 0	0	1	1
1 0 1 1	0	1	1
1 1 0 0	0	0	0
1 1 0 1	0	0	0
1 1 1 0	0	0	0
1 1 1 1	1	0	0

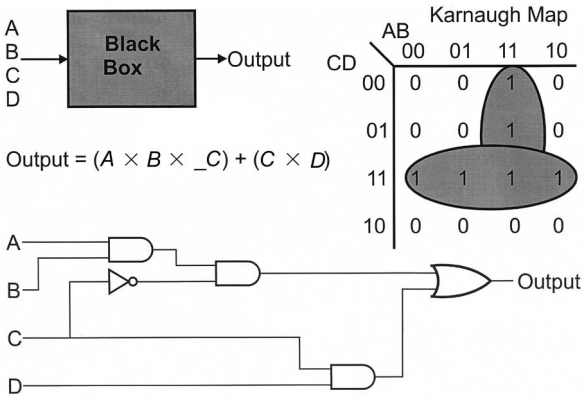


Figure 72 Deriving logic functions from requirements

This truth table isn't very helpful in showing the relationships between the circuits. Looking at the truth table, you might come up with an output equation that looked like:

```
Output = (A * B * _C * _D) + (A * B * _C * D) + (C * D)
```

which will require seven two-input AND gates, two two-input OR gates, and two inverters. The propagation delay through the function will range from two to four gate delays. By making up the Karnaugh map shown in Fig. 72 from the truth table, I look for rela-

TABLE 16 "TRUTH TABLE" For Karnaugh Map Example

D C B A	OUTPUT
0 0 0 0	0
0 0 0 1	0
0 0 1 0	0
0 0 1 1	1
0 1 0 0	0
0 1 0 1	0
0 1 1 0	0
0 1 1 1	0
1 0 0 0	0
1 0 0 1	0
1 0 1 0	0
1 0 1 1	1
1 1 0 0	1
1 1 0 1	1
1 1 1 0	1
1 1 1 1	1

tionships in the output settings. The Karnaugh map is drawn with the bits of each axis only changing by one value at any time (basically using gray codes). By doing this, relationships with bits staying the same can be more easily seen.

This is done by circling all the cases where the output is *1* with the cases that are side by side or up and down. Diagonal relationships are not circled. The result of circling the cases is to make relationships become more readily apparent.

For example, I can see that when *A* and *B* are both *1*, the output will be *1*, except when the *C* input is *1*. Going further with this, I can ignore the case when *A* and *B* are set and *D* and *C* are set because it is already encompassed in the AND function $C \times D$.

The Karnaugh map allows me to reduce the output equation to:

```
Output = (A * B * _C) + (C * D)
```

which, as can be seen in Fig. 72, only requires three AND gates, one inverter and one OR gate. The propagation delay is improved to two or three gate delays as well.

Karnaugh maps, although they probably seem pretty easy to work with in this way, do take some practice. When you first start working with them, you will discover that you might have to redraw them multiple times to best "see" the most efficient results. Over time, you will probably find that Karnaugh maps are less and less useful because you will learn how to "think" in ways of looking for relationships and seeing how to visualize the desired functions in your head.

One way to optimize circuits is look through their output equations and try to find relationships that you can take advantage of. To do this, you should be aware of the Boolean identities. These *identities* are simple formulas that you should remember or keep the list of them (in the appendices) handy when you are working with digital logic.

Going back to the example output equation that I came up with before optimizing with the Karnaugh map:

```
Output = (A * B * _C * _D) + (A * B * _C * D) + (C * D)
```

Notice that the first two terms are almost identical. The only difference is that one uses the positive value of *D* and the other uses the inverted value of *D*. Going to the appendix, I can look up the Associative law, which states that:

```
A * B * C = (A * B) * C
```

to make the first two terms even more similar:

```
Output = [(A * B * _C) * _D] + [(A * B * _C) * D] + (C * D)
```

Next, I can use the Distributive law:

```
[(A * B) * C] + [(A * B) * D] = (A * B) + (C * D)
```

To merge the first three values of the first two terms together:

```
Output = (A * B * _C) + (_D * D) + (C * D)
```

Finally, using the Complementary law, I know that:

```
A AND _A = 0
```

I can reduce the output equation to:

```
Output = (A * B * _C) + (C * D)
```

which is identical to what I came up with using the Karnaugh map.

Performing this type of analysis can also be quite a bit of fun, but it also requires some experience to know what to look for. This is a good tool to use with Karnaugh maps to confirm that you have produced the correct optimization for the output function that you require.

So far in this section, I have only shown combined logic gates as providing a single output. They can also be used to provide multiple outputs, such as the address decoder shown in Fig. 73.

This circuit combines two inputs in four different ways to produce four address select bits. This circuit could have been divided into four discrete circuits, each one providing an output to a different address select, but that would have required multiple inverters. By combining circuits and sharing the inverter outputs, I reduced the overall circuit requirements for the function.

Combining multiple circuits into a single circuit like this example is usually not as trivial to implement. When combining circuits, you should look for intermediate outputs that are common and merge them together. In the example circuit, the inverted A0 and A1 inputs that were common between the four circuits, which allowed them to be combined into one.

The digital logic circuits presented here are known as *combinatorial circuits* and are a lot of fun to play around with. In the early days of digital electronics, having skills in developing combinatorial circuits was crucial to being able to successfully develop an application. Today, for most application developers, combinatorial circuits have been reduced in status to being "glue" between other complex chips.

As I work through the experiments and projects in this book, you will see how little combinatorial circuits are required in actual applications. Of more use are circuits that provide some kind of memory function within them. This function is provided by feeding back outputs from the combinatorial circuit into its input. By doing this, previous values are used to affect the current output values. These circuits are known as *sequential circuits* because they follow a predefined set of operations based on a specific sequence, the previous value being "remembered" within the circuit.

Sequential circuits are quite a bit more difficult than combinatorial circuits, but they are much more useful in microcontroller applications. Later, this appendix introduces the concept of flip flops, which are simple sequential circuits that can "remember" information.

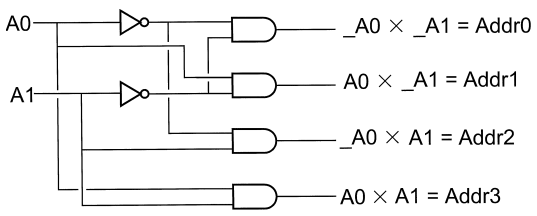


Figure 73 Memory address decoder logic

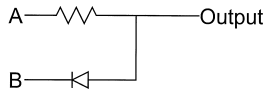


Figure 74 AND gate analog circuit

LOGIC ANALOGS

In many digital electronics applications, you will require individual logic gates that require very little real estate and are fast enough for the PICmicro® MCU. *Fast enough* being circuits that can switch in less than a microsecond. Going back to *RTL (Resistor-Transistor Logic),* a precursor to TTL, some simple analogies can be used when a full logic chip is not required.

For example, the AND function can be recreated using the resistor diode circuit (Fig. 74).

In the resistor/diode equivalent AND gate equivalent, if input A is low, the output will be low. If input A is high, then current can flow through the resistor to the output or through input B, if it is driven low. When input B is high and input A is low, no voltage will be driven to the output. Only if input A and input B are high, will the output then be high. A typical value for *R* in this (and the other circuits) is 10 kΩ. Diodes for these circuits can be virtually any small-signal silicon diodes (I like to use 1N914s because they are very cheap).

In the circuit shown in Fig. 75, if either input is driving high, the output voltage will be high. The resistor will pull the output low if neither input A nor input B is driving high. A 10-k resistor can be used in this circuit as the pull-down.

The last analog, which one you're probably very familiar with, is the inverter, that can be cobbled together using a pulled-up transistor (as shown in Fig. 76).

In this circuit, if the input is high, the N-channel MOSFET transistor will be turned on, pulling the line low. When input is low, the transistor will be off and the resistor will provide a high-voltage output.

A common term for these types of circuits is *M**L* or *MML,* which stands for *"Mickey Mouse" Logic*. This expression implies that solution is not very high tech or complex, but, if designed right, it will eliminate the cost of adding a chip for just one gate.

FLIP-FLOPS

Multiple gates can be arranged into circuits that provide feedback from the outputs into the inputs. The resulting circuits are known as *flip-flops,* which can store information. This information storage can range from saving state information and turning a combinatorial circuit into a sequential circuit for providing counting or arithmetic functions to implementing memory circuits that are used in processors.

When you are first introduced to a flip-flop, its operation will seem unusual at best and it probably seems like some law is violated in any case. This isn't true, but (as you can see in Fig. 77) the circuit appears to be somewhat strange.

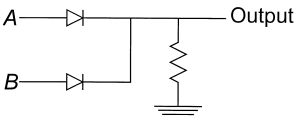


Figure 75 OR gate analog circuit

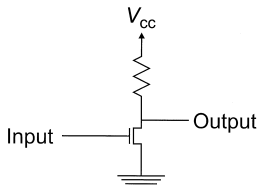


Figure 76 Inverter gate analog circuit

The output of each of the two NAND (AND gates with an inverter on the output) gates is used as an input of the other. When I first saw this, I imagined a signal racing around the two NAND gates. But this is the wrong way to visualize the circuit. Signals do not "race" between the gates, but state information is stored when the two input lines go high.

When input A and input B are high, the previous state of the NAND gates is stored in the flip-flop. For example, if input A was set and input B was reset and then both inputs were set, flip-flop A would output a high voltage on output A and flip-flop B would output a low voltage on output B.

Only one of the two input's can be pulled low at any one time to load in a bit to save in the flip-flop. Once the state is set, then both lines are pulled high and the state is stored. The truth table for the flip-flop is shown in Table 17.

At, both inputs *0* is invalid because it will result in both the outputs being high. When the two input lines are brought high, the outputs will initially go low and will enter an unstable state in which the flip-flop will jump to a data state based on any imbalance. This imbalance can be caused by one line rising faster than the other line or even an errant electron charge that is not balanced. The term for the flip-flop at this unstable state before it goes to one state is *metastable* because it can go equally easily to either state. Metastable flip-flops can be useful in some specific applications, but, for the most part, they should be avoided wherever possible.

The flip-flop circuit shown here, although usable, isn't that practical in most circuits. I often use the D flip-flop in applications because it can be more easily interfaced to a PICmicro® MCU. The Data In state is stored in the input when the clock line is brought from low to high. This is usually described as catching in data on the "rising edge" of the clock (Fig. 78).

The *toggling flip-flop* or *T flip-flop* (Fig. 78) will change the output state each time the clock inputs a rising edge. T flip-flops are often used to divide a clock by two. This is because if a clock is input, the outputs will change states at one half the rate of the clock. As can be seen in Fig. 78, this results in outputs that have two times the period of the clock.

The *set/reset flip-flop* is often referred to as an *RS flip-flop*. The output (*Q,* Fig. 80) is set on reset based on whether or not an input is pulled low.

The truth table for the RS_FF is shown in Table 18.

The last flip-flop is the JK (Fig. 81), which provides a similar function to the RS flip-flop, except when both lines are high, the output's are toggled.

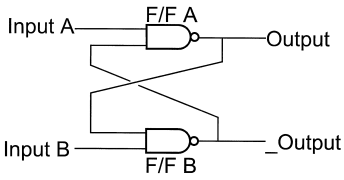


Figure 77 Simple flip-flop

TABLE 17 NAND Gate Based Flip-Flop Input and Output Truth Table

INPUTS		OUTPUTS		COMMENTS
A	B	A	B	
1	1	Ao	Bo	saved
0	1	1	0	set “A” high
1	0	0	1	set “B” high
0	0	N/A	N/A	invalid

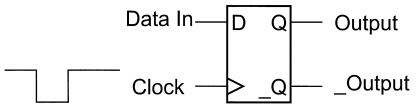


Figure 78 D flip-flop

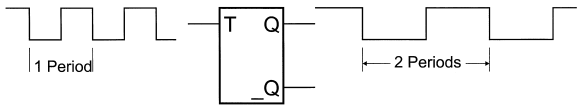


Figure 79 T flip-flop

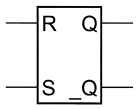


Figure 80 RS flip-flop

TABLE 18 RS Flip-Flop Input and Output Truth Table

R	S	0	_Q
1	1	INVALID	
0	0	Qo	_Qo / SAVED
0	1	0	1
1	0	1	0

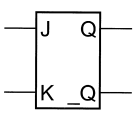


Figure 81 JK flip-flop

TABLE 19 Jk Flip-Flop Input And Output Truth Table

J	K	Q	_Q
0	0	Q0	_Q0
0	1	0	1
1	0	1	0
1	1	_Q0	Q0 / TOGGLE

The JK truth table is shown in Table 19. The different flip-flops presented here are based on the original circuit shown in Fig. 77. To provide the enhanced functions, circuits are put on the front end of the flip-flop. With the PICmicro® MCU providing many intelligent in/out functions on its own, there isn't a large call for many of these flip-flop circuits. I tend to primarily use the D flip-flop functions with the PICmicro® MCU because it can be easily interfaced to the chip and does not require a lot of thinking to develop the interfacing software.

COMMON DIGITAL CHIPS

The next 19 diagrams show the pin-outs for the TTL "glue" logic that I use most often in my designs. All the chips can very easily found from a variety of sources and vendors. Although in the diagrams, I reference them as "straight" parts (no technology references), I typically use LS-logic parts. So, instead of a 7400, I will typically use a 74LS00.

Because LS logic has a less-than-20-ns propagation delay, you should be able to use it almost exclusively for your PICmicro® MCU applications. The only instances where you would want to use another logic family are when the LS electrical output drive characteristics are not sufficient for the application. In these cases, you should go with S (i.e., 74S138 instead of 74LS138).

The chips outlined in Figs. 82 through 99 are those that I am most likely to reach for when developing a CMOS or TTL digital application.

Each one of these chips should have a decoupling capacitor as close to the V_{cc} pin as possible. These chips can be used with sockets that have built-in decoupling caps.

The chip operation is very straightforward, although, in a number of cases, I have added instructions and comments on specific aspects and features of the chips. The operation of these chips with the PICmicro® MCU can be seen in the projects. More complete chip data sheets can be found at National Semiconductor's web site: **http://www.national.com** or Philips' web site: **http://www.semiconductors.philips.com/.**

PULL UPS/PULL DOWNS

When you start understanding digital logic, and began to look at circuit schematics, you probably noticed that many digital input pins have resistors connecting them to V_{cc} or ground. As was described in the previous section, digital inputs can be characterized as a high-impedance circuit, so the resistors seem unnecessary. These resistors are known as

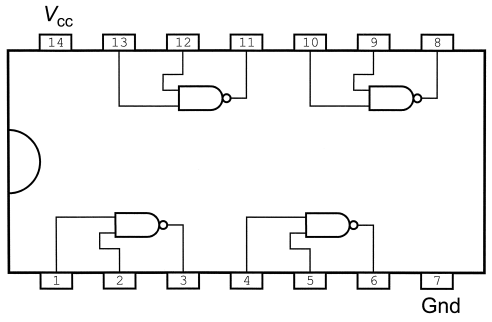


Figure 82 Quad two-input NAND gate TTL chip

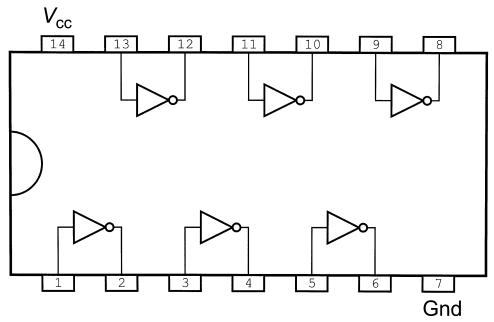


Figure 83 Hex inverters with totem-pole outputs

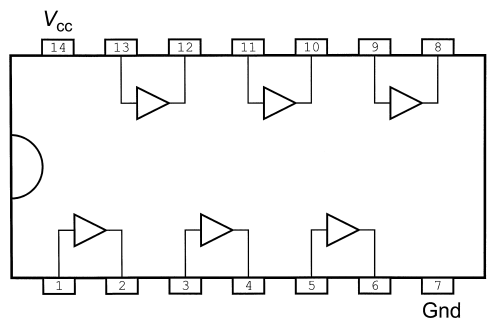


Figure 84 Hex buffers with open-collector outputs

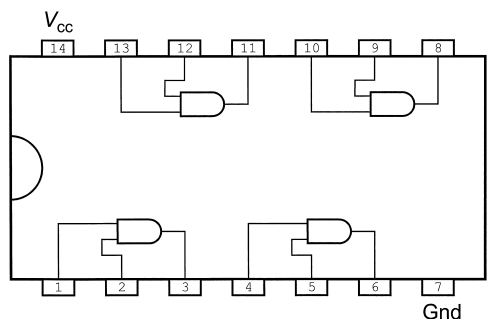


Figure 85 Quad two-input AND gate TTL chip

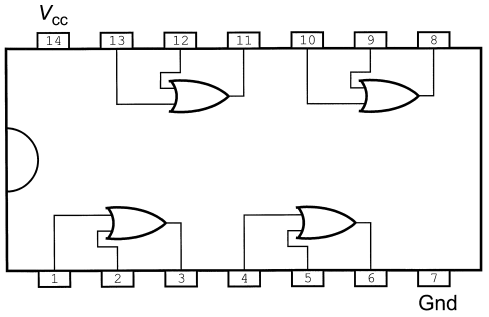


Figure 86 Quad two-input OR gate TTL chip

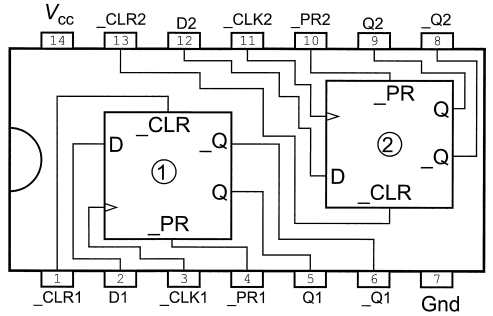


Figure 87 Dual D-type flip-flops

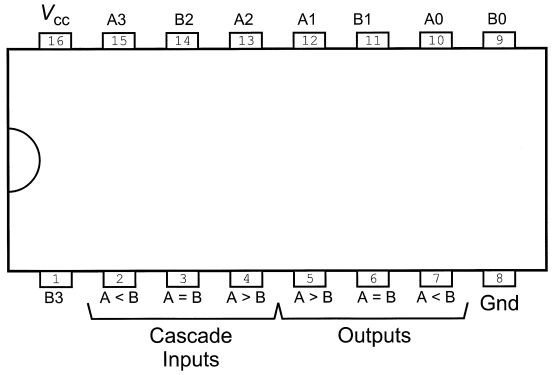


Figure 88 Four-bit magnitude comparator

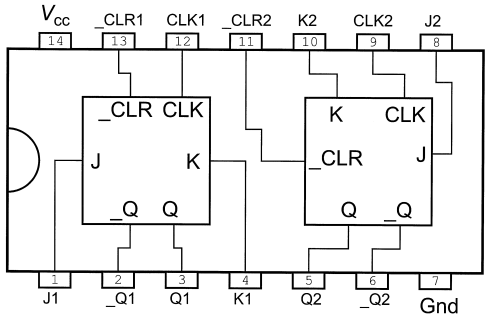


Figure 89 Dual JK-type flip-flops

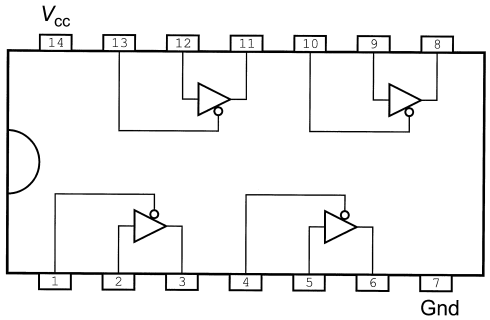
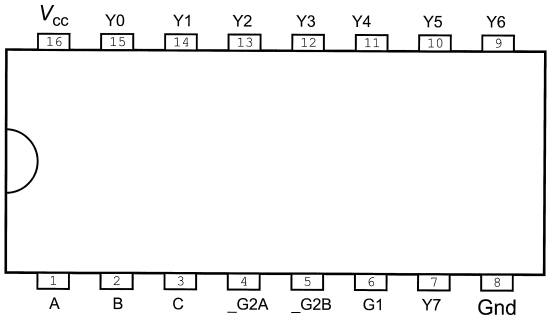
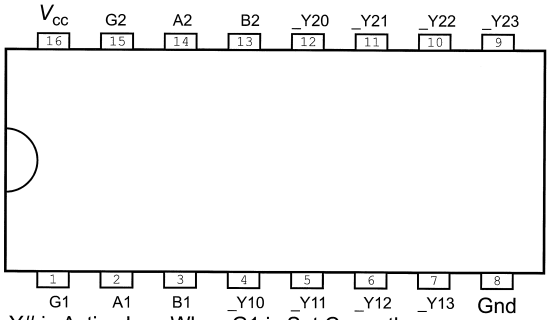


Figure 90 Quad tri-state buffers



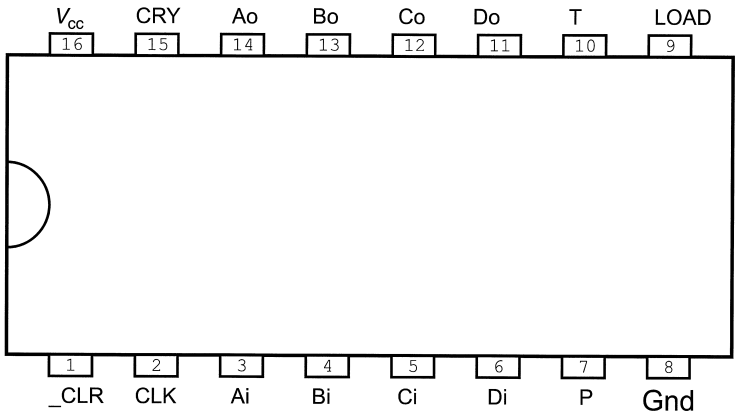
Y# is Active Low When G1/G2 are Set Correctly.
= (*C* x 4) + (*B* x 2) + (*A* x 1)

Figure 91 Three to eight decoder



Y# is Active Low When G1 is Set Correctly.
= (*B* x 2) + (*A* x 1)

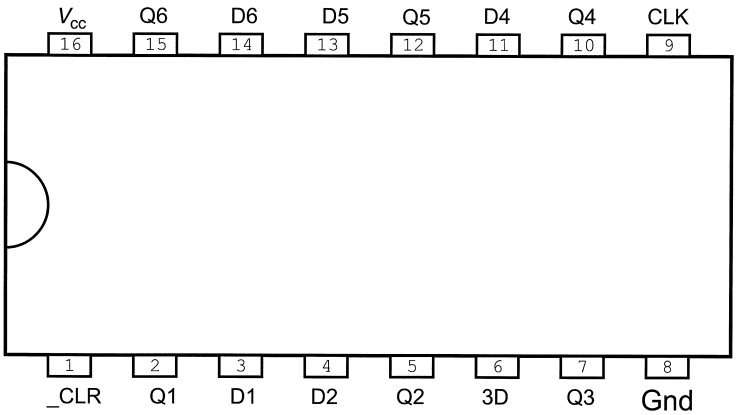
Figure 92 Dual two to four decoder



160 Decade Counter
161 Binary Counter

P & T are Enables, Must be High
Counter Update on Rising Edge of Clock

Figure 93 74160/74161 binary/decade counters



Data Latched on Rising Edge of CLK.

Figure 94 Hex D-type flip-flops

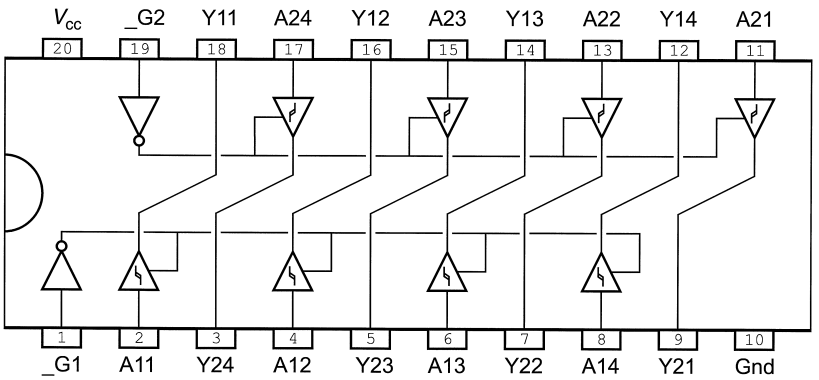
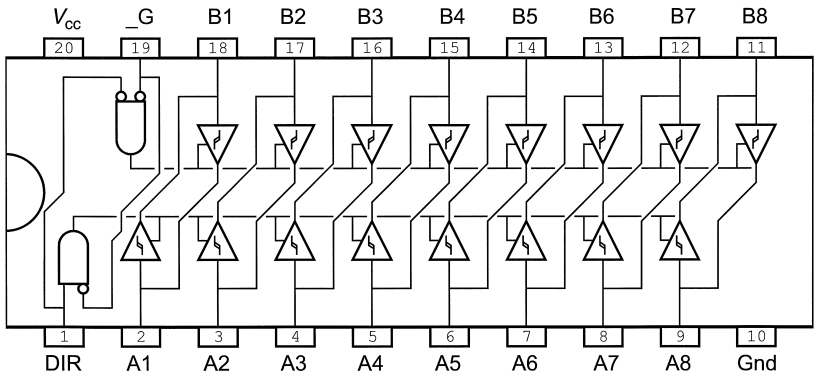
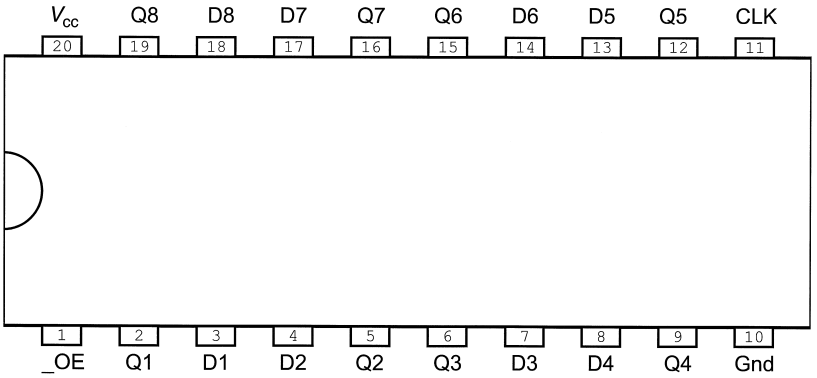


Figure 95 Eight-bit tri-state driver



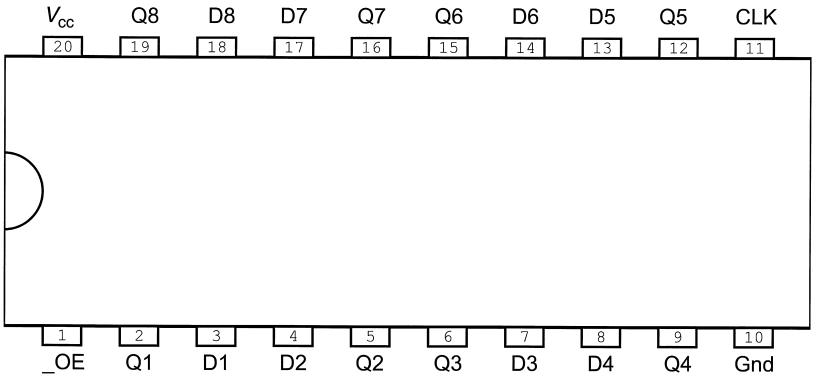
DIR = 1, Data From A to B
DIR = 0, Data From B to A

Figure 96 **Bidirectional eight-bit tri-state driver 74LS245**



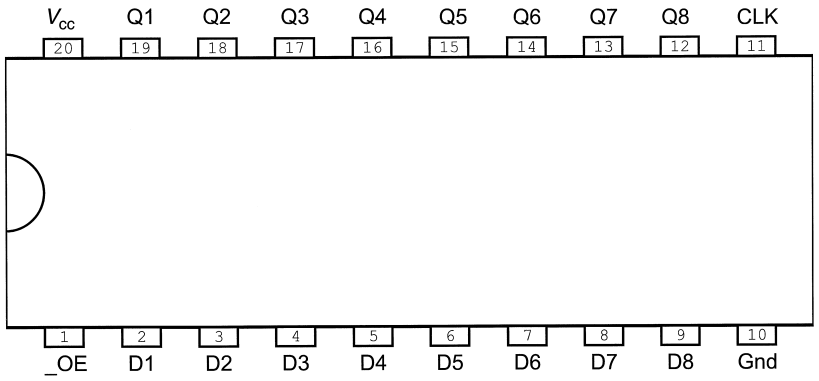
Latches are Transparent when CLK is High.
Data Latched in on Negative Edge of CLK.

Figure 97 **Eight-bit latch with tri-state output driver 74LS373**



Output Changes After Latch Operation.
Data Latched in on Positive Edge of CLK.

Figure 98 **Eight-bit latch with tri-state output driver 74LS374**



Latches are Transparent when CLK is High.
Data Latched in on negative Edge of CLK.

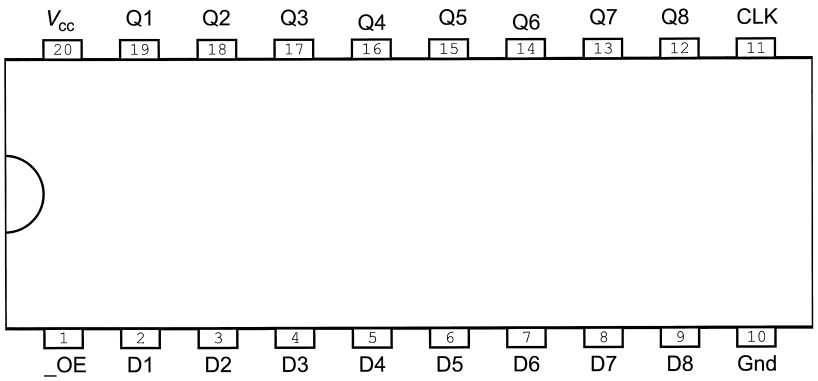
Figure 99 **Eight-bit latch with tri-state output driver 74LS573**

pull ups. Pull downs are used to allow other circuits to "overpower" them and change the input going into the pin.

A *pull up* is a resistor that connects an input pin to the V_{cc} power connection of a circuit (Fig. 101).

The normal value of a pull up is anywhere between 1 k and 10 k. These values are chosen for the current that will pass through them when the line is "pulled" to a low value. A 1-k resistor will have 5 mA flowing through it when it is pulled to ground. The higher the current, the faster and with less "rounded" edges, the input pin will transition from a high to low state and back again.

A common use for pull ups is to allow switches and buttons to be connected to a digital input. A common circuit for this is shown in Fig. 102.



Output Changes After Latch Operation.
Data Latched in on Positive Edge of CLK.

Figure 100 **Eight-bit latch with tri-state output driver 74LS574**

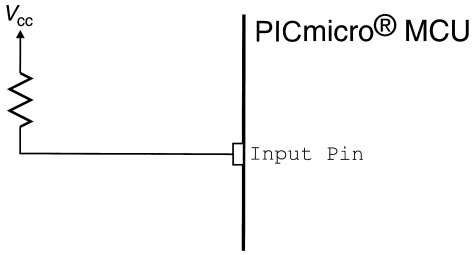


Figure 101 Pulled-up PICMicro® MCU input

In this circuit, a momentary-on push-button switch connects the pin to ground, passing a low voltage into the pin. When the switch is released and allowed to open, the input returns to a high value. This circuit is often used for the PICmicro® MCU's _MCLR reset pin. The other most popular reason for using a pull up on an input pin is when multiple drivers are connected like the circuit shown in Fig. 103.

When any of the output transistors in Fig. 103 pull the line low, the input signal will be pulled low. This is useful for putting multiple switches on a single input pin. This circuit is known as a *Dotted And* bus.

Pull downs work similarly to pull ups, except that they connect the input pin to ground through a resistor. *A* switch connected to V_{cc} could be used to connect a pulled-down line to V_{cc} (Fig. 104).

The values for pull downs are not as cut and dried as for pull ups. Internal pull ups to the input pin will require you to check the devices data sheet before selecting a pull-down resistor. The pull down must be selected to place the input voltage in the region below the switching voltage threshold for the input pin. For most devices, 0.5 volts is a good voltage to work toward when calculating the pull-down resistor value.

When developing PICmicro® MCU applications, I do not recommend using a pulled-down switch over a pulled-up one. Coming up with the correct resistor value is work that you simply do not have to do. If you want a function where the input is low until the button is pressed, I would suggest that you use the pulled-up switch with a momentary-off pushbutton switch instead of the momentary-on switch shown in Fig. 102.

LINEAR-FEEDBACK SHIFT REGISTERS

One of the most interesting logic devices you can work with is the *linear-feedback shift register (LFSR)*. This circuit can be used to pseudorandomize data, encrypt, and decrypt serial data and provide very good serial data integrity checking. You might have heard the term *Cyclical Redundancy Check (CRC)* when applied to data transmission; this is a type

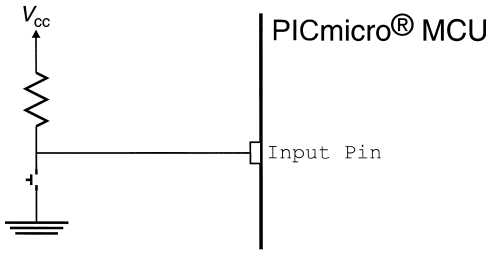


Figure 102 Pulled-up switch.

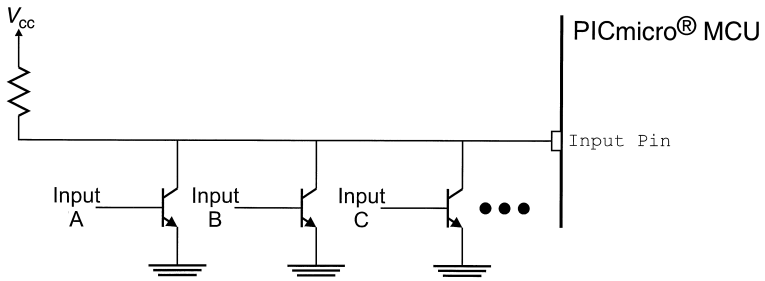


Figure 103 Pulled-up Dotted AND bus

of linear-feedback shift register. linear-feedback shift registers can also be implemented fairly easily in software, although it is in hardware where the device is the most efficient.

The form of a linear-feedback shift register is shown in Fig. 105. The XOR gates in between the shift register's inputs are known as *taps*, which provide a pseudorandom output based on the serial inputs.

For example, if you had an eight-bit shift register with a single tap coming from bit five, the circuit would look like that circuit shown in Fig. 106.

The data going into the shift register would be characterized as:

```
Shift register in = Data in ^ Bit 5
```

The actual shift register is a function of what was put in before. If the 32 bits 0x055AA00FF were input into the register, the data states possible data states are shown in Table 20.

This might seem like a cumbersome method of getting 0x080, but the important thing is that this value is unique to the original string of bits. If a bit input into the linear-feedback shift register were in error, then the result would not be 0x080; it would probably be something significantly different. In this mode, the LFSR is producing a control code for the data input to ensure that the data is correct. The advantages of producing a control on a data string are that the hardware to create it is very simple and the control result will detect multiple bit errors (something that a checksum or parity checks might not do).

Linear-feedback shift registers can also be used to generate pseudo-random displays. By using the output as the data-in circuit, I can produce a pseudorandom output that will change each time the data is shifted. This is actually the function of the circuit in Fig. 106. This circuit is useful for producing random numbers in an application or even displaying a seemingly random pattern of lights. I use this characteristic of linear-feedback shift registers to make "Christmas lights" or random numbers for games.

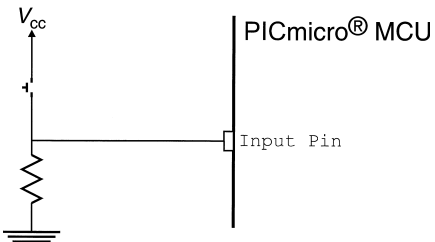


Figure 104 Pulled-down switch

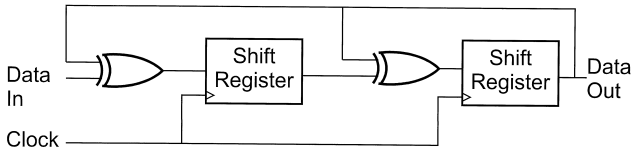


Figure 105 Linear feedback shift register

DIGITAL INPUTS AND OUTPUTS

I almost feel like the title of this section is a misnomer; as I go through the input and output circuitry of digital devices, you'll see quite a bit of analog circuitry as will as some peculiarities that you should watch out for. The information contained in this section is quite important to understand, although it is often ignored or passed over in school courses. I suggest that you read through this material and try to make as much sense of it, in relation to TTL and PICmicro® MCU device data sheets, as well as the experiments and projects presented in this book.

A digital input is normally described as the idealized circuit shown in Fig. 107, which consists of a CMOS inverter. The two transistor gates do not allow any current drain and the size of the gates have zero capacitance. The actual circuit usually has clamping diodes added to them, along with parallel inputs and potentially tristatable output drivers. These extras make the circuit somewhat less than ideal.

Making things worse, input pins might be TTL or TTL compatible (consisting of a pull up and base current-limiting resistor, Fig. 108). In this circuit, the input impedance changes according to the level of the input voltage. This can make the actual operation of the input pin unpredictable, depending on the other components connected to it and the driver characteristics.

These unexpected and parasitic impedances lower the maximum circuit logic switching speed and limit the total number of inputs that can be wired to an output driver. The number of gates that can be connected to a single output driver is known as *fan out* and is dependent on the input and output device technologies that are being connected. In the idealized case of the two-MOSFET transistor logic input, the number of devices that can be driven is literally in the hundreds. The only limiting factor being the resistance, capacitance, and inductance (collectively known as *impedance*) of the connecting circuits and wiring.

In the real world, because of the built-in TTL resistors, clamping diodes, and parallel peripheral functions, the maximum number of inputs that you can connect to is considerably less. The maximum fan out can range anywhere from one device to 10 or so. A good general rule is never exceed five; a fan out of two devices is the optimum.

In description of digital logic earlier in this appendix, I have shown typical CMOS and TTL inputs with the switching threshold voltage being constant. For TTL, this threshold voltage is 1.4 to 1.7 volts. For CMOS, it is 1.4 volts to 0.5 power in (V_{dd}). Another type of gate, the *Schmidt trigger* input, provides hysteresis to limit the opportunity for noisy signals to provide spurious logic changes.

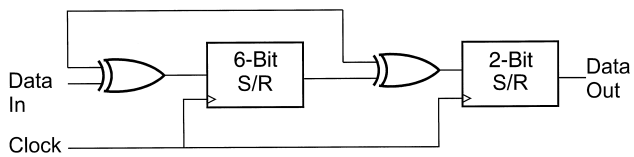


Figure 106 Linear feedback shift register application

TABLE 20 Linear Feedback Shift Register Operation Example

CYCLE	DATA IN	SHIFT REGISTER VALUES
0	1	0x000
1	1	0x001
2	1	0x003
3	1	0x007
4	1	0x00F
5	1	0x01F
6	1	0x03F
7	1	0x07E
8	0	0x0FD
9	0	0x0FB
10	0	0x0F7
11	0	0x0EF
12	0	0x0DF
13	0	0x0BE
14	0	0x07D
15	0	0x0FB
16	0	0x0F7
17	1	0x0EE
18	0	0x0DD
19	1	0x0BA
20	0	0x075
21	1	0x0EA
22	0	0x0D5
23	1	0x0AA
24	1	0x055
25	0	0x0AA
26	1	0x054
27	0	0x0A8
28	1	0x050
29	0	0x0A0
30	1	0x040
31	0	0x080

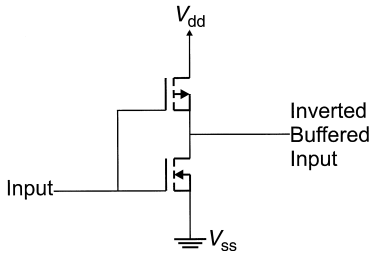


Figure 107 Basic CMOS input circuit

In the 74LS14 inverter chip, the input threshold voltage (at 25°C) for a low to high signal is 1.7 volts. The threshold voltage for a high to low signal is 0.7 volts. This can result in some unpredictable switching events (Fig. 109).

When the state input to a Schmidt Trigger is changing, the switching threshold voltage "moves" away from the source. This helps to filter noise on the input when the transition is being made. When discussing PICmicro® MCU interfacing, I will describe using this capability of Schmidt trigger inputs to "debounce" noisy signals.

The two basic types of output circuits and are *totem pole* and *open collector/open drain*. The totem-pole output consists of two transistors set up in a "push/pull" configuration (Fig. 110). When this circuit is driving out a *1* or high voltage, the upper transistor (which is connected to V_{cc}) is on and the lower one is off. For a *0* (low voltage) output, the lower transistor is on, pulling the output to ground while the upper transistor is off. Both transistors can never be turned on at the same time. This circuit responds very quickly to output state changes and can be implemented in a variety of different logic families.

The name *totem pole* comes from the resemblance of the two transistors to a Native American wooden totem pole.

The other type of digital output is known as the *open collector* (for TTL logic technology) or *open drain* (for CMOS technology). This type of output is unable to source current and can only sink it. TTL and CMOS circuits are shown in Fig. 111.

This type of output is used in situations where multiple outputs must be connected together. A common circuit that uses open collector outputs is putting multiple hardware interrupt requests on a single line (Fig. 69), shown earlier in this appendix.

The pull up provides a high voltage for when all the transistors are turned off. When any of the transistors are turned on, the interrupt request line will be pulled low and the interrupt request will be passed to the processor. This Dotted AND bus is high only as long as none of the outputs are pulling the line low.

Totem-pole outputs cannot be wired together (Fig. 112) because this circuit will result in bus contention when any of the outputs are driving at different levels. *Bus contention* lit-

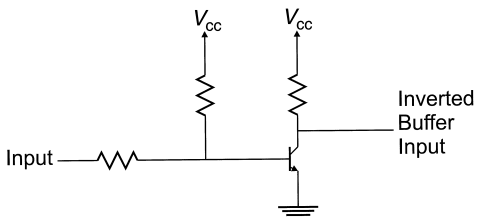


Figure 108 Basic TTL input circuit

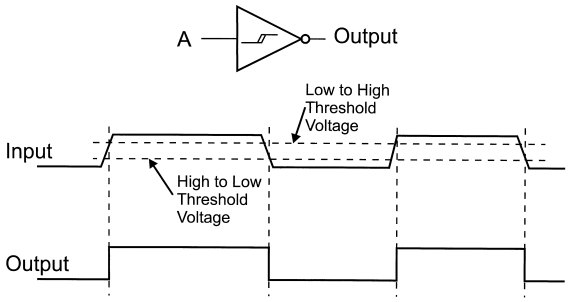


Figure 109 Schmidt trigger NOT gate response

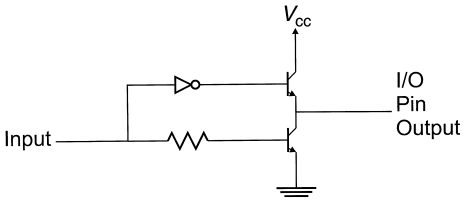


Figure 110 Basic TTL totem-pole output circuit

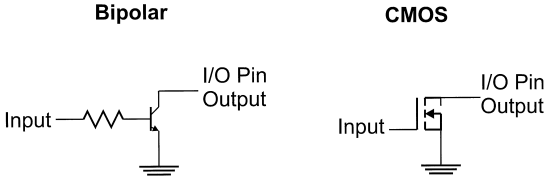


Figure 111 Basic open-collector/drain-output circuits

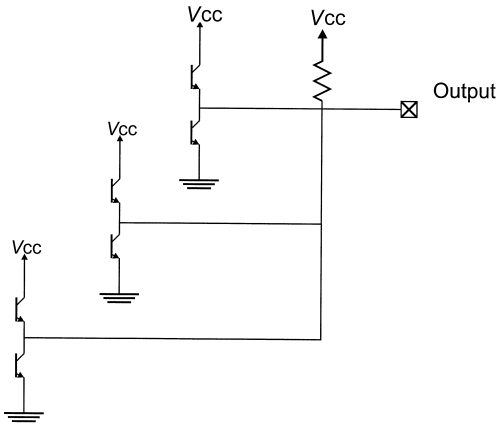


Figure 112 Bus contention situation

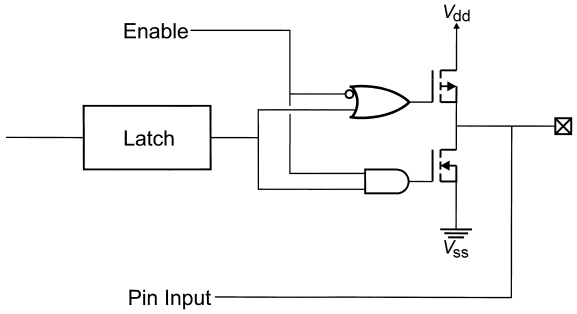


Figure 113
Microcontroller output

erally means that multiple drivers are trying to drive the line to their logic levels. Because of the voltage drops and current limiting aspects of the different gates, the actual voltage output is almost impossible to calculate (and is known as *indeterminate*).

Bus contention can actually taken advantage of in some applications. The adapter "Plug and Play" devices built into your PC and I2C addressing are excellent examples of this. For the most part, bus contention should be avoided at all costs. Two active digital drivers should never be on the same circuit.

The 8051 is an interesting device in how it provides bidirectional I/O capabilities to its pins. Most devices (the PICmicro® MCU included) provide a "tristate" output driver to the pin that can be turned on and off (Fig. 113). The enable line masks the operation of the totem-pole output transistors. If the output is not enabled, then neither transistor can source or sink current to or from the pin.

The 8051 uses a high-impedance pull up (11 k+) with an open-drain output (Fig. 114). When the output is high (the transistor is off), the pin is either in Output High mode or in Input mode and another driver can set its state. This method works reasonably well and is very simple, but you have to remember that it cannot source any kind of current and might be slow to rise to a high voltage because of capacitances attached to the I/O pin.

When digital outputs are shorted (connected) to ground or V_{cc} and these power terminals can provide more current than the gate, the output will be at the power level. The chip containing the output will grow warm (and could burn out) because it will supply the maximum amount of current to the voltage connection. The applications and description of the PICmicro® MCU point out any issues with the microcontroller's I/O pins and how to avoid any potential problems.

MEMORY TYPES

A variety of different types of memory are available for use with electronic devices. In this section, I would like to introduce you to a number of different types and how they are used.

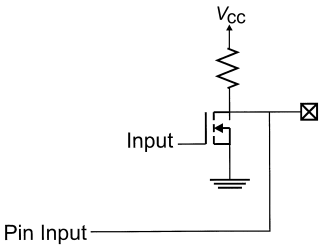


Figure 114 **Microcontroller open collector output**

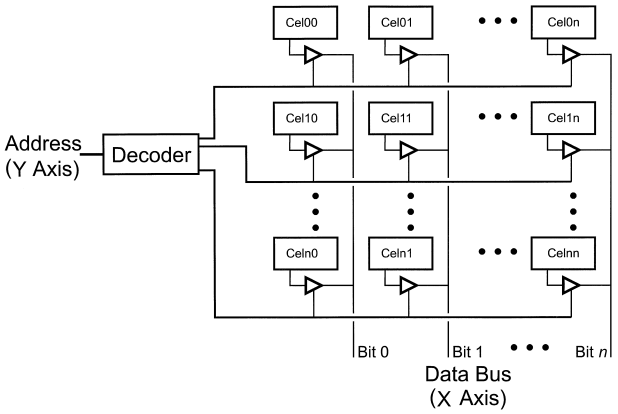


Figure 115 Array of memory cells for reading

The PICmicro® MCU is somewhat limited in the memory available, both in terms of types and amount. This will lead to some different methods of application, design, and programming.

Memory is normally arranged as two-dimensional arrays with one axis (the X or width) being the number of bits in a word. For data, this is normally eight bits for each data word being a byte. For program memory, this is often also eight, but, in some cases (like the PICmicro® MCU), it is the width of an instruction. The other axis (Y or depth) is the number of words that is available in the memory. Typically, the depth of the memory is some power of two (so that all the address bits are used to specify memory locations in memory).

These two axes are used to select and access a specific word based on its Y-axis address. A read array would look like the diagram shown in Fig. 115.

The bit address (Y axis) is used to select a specific row in the array. This selection enables the drivers in the line and the contents of the line are driven onto the databus.

When memory chips are specified, it is done one of three ways: it can be specified as the total number of bits, number of bytes, or number of words. For program memory, I am always most interested in the number of instruction words. For variable memory, knowing the number of bytes is most important.

Memory that can be read from and written to is known as *Random-Access Memory (RAM)* because the access can be reads or writes, depending on the application. RAM, in microcontrollers, is normally used for variable memory, in which the processor can read and write values to it. *Read-Only Memory (ROM),* on the other hand, can only be read. As described in the following section, ROM is best suited for storing application code data in the microcontroller.

There are two primary categories of memory. *Volatile* memory means that data stored in the memory will be lost when power is taken away from it. Nonvolatile memory retains its contents after power has been removed. It is best for code that is used when the application first powers up. In your PICmicro® MCU, nonvolatile memory is used for the application code ROM. Volatile memory is used for storing variables and applications that have been loaded from some nonvolatile source (such as ROM or external devices).

There are a number of types of nonvolatile memory. The most basic type is *mask ROM,* which has the memory's contents built into a chip. The *mask* adjective means that the contents of ROM are defined when the chip is built. This is usually accomplished by placing a pull down on a normally pulled-up circuit (Fig. 116).

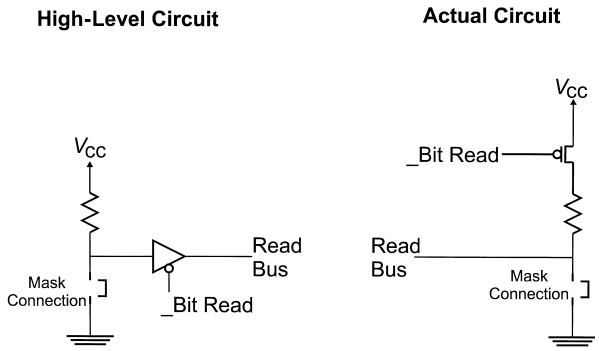


Figure 116 Mask programmable read-only memory

As Fig. 116 shows, if the mask connection is placed over the connection, when the memory cell is read, a *0* is returned. If the connection is not present, a *1* is returned because the cell is pulled up.

On a per-unit cost, mask ROM is the least expensive method of implementing nonvolatile memory. To implement it, a custom mask and parts have to be created. The mask usually costs several thousand dollars and can take up to 10 weeks to be made (along with parts). Mask ROM parts can also be used in only one type of application. This lessens its usefulness in situations where multiple part numbers are required. As indicated elsewhere in this book, mask ROM memory is best suited for applications where large numbers of parts are required on a schedule and the lowest possible cost is required (such as in the automotive market).

In the 1970s, *Programmable Read-Only Memory (PROM)* became available. In this type of nonvolatile memory, the pull ups were designed as fuses that could be blown during a programming step. A sample cell is shown in Fig. 117.

To blow a cell, a high current is driven through the fuse to ground, causing it to melt and open the path to ground. When the circuit is initially produced, all of cells are given the value *1*. After a cell is programmed, its value becomes *0*.

PROMs of this type (known as *fuseable link*) quickly fell out of favor as erasable PROMs came on the market. Although the parts themselves were quite inexpensive, they were unreliable; over time, some fuses would "grow back" as the metal migrated back into a pull-down state. This caused the contents of the memory to change over a relatively short time (a few months to a year). For this reason, fuseable-link PROM's have fallen out of favor and are very rarely used.

In the late 1960s, three university students started their own company, based on their design for a programmable read-only memory that could be erased. The company was Intel, which is better known for its microprocessor designs than its memory innovations. The *Erasable PROM (EPROM)* memory consisted of a MOSFET transistor, which had a float-

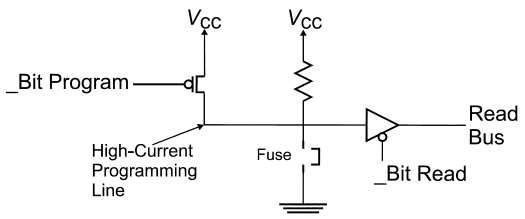


Figure 117 Fuse programmable read-only memory

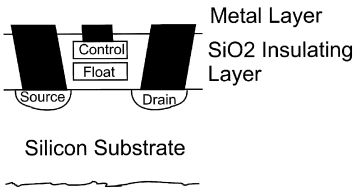


Figure 118 EPROM memory cell

ing gate that could be loaded with a charge, to cause it to conduct (and drive a pulled-up line to ground). To unload or erase this charge, *UltraViolet (UV)* light is shined on the transistor to free the trapped charge.

A cross-section of this transistor is shown in Fig. 118.

The EPROM memory cell is similar to the straight ROM cells described previously, except that the mask or fuse links are replaced with the EPROM cell. The EPROM cell is normally not conducting, which means that it returns a *1* in the unprogrammed state. When a cell is programmed (burned), it will return a zero because the EPROM cell will be conducting and pulling the cell's pull up to ground.

Most chips are encased in plastic packages (Fig. 119> which are opaque (which means that the erasing UV light cannot reach the chip). For EPROM parts that are to be erased, they are encased in a ceramic package, which has a quartz window built in to allow UV light to reach the chip (Fig. 120). The ceramic-windowed package is much more expensive than the plastic package and chips that use the straight plastic packages.

To help offset this cost in production, plastic-packaged EPROM parts are sold and are very cost effective for low and medium volumes. The chips can be programmed once, which leads to the label that is used for them: *One-Time Programmable (OTP)*. OTP parts have a number of advantages in different situations. The first is in manufacturing situations, where the parts are used in multiple products; it makes a lot of sense to stock unprogrammed OTP parts instead of preprogrammed mask ROM parts. Stocking unprogrammed parts and programming them, as required, reduces the part stock for the manufacturer and allows faster response to unplanned orders instead of waiting for mask-ROM parts to be built.

The second advantage is somewhat more obvious and refers to what happens if a mistake in the code is burned into the EPROM. In a mask-ROM part, the current parts must be scrapped and the mask redesigned, resulting in a *NRE (NonRecurring Expense)* bill and a wait for new parts to be built. In the meantime, all of the parts built with the incorrect code must be scrapped. If EPROM OTP parts are used, the parts are burned with the correct code for the next product build.

The last type of nonvolatile memory is *Electrically Erasable EPROM (EEPROM)*. These parts do not require UV light to prepare them to be reprogrammed (erased). EEPROM uses a similar cell, but the floating gate can be drained by the application of a high-voltage field, which causes the trapped charge to leave the floating gate.

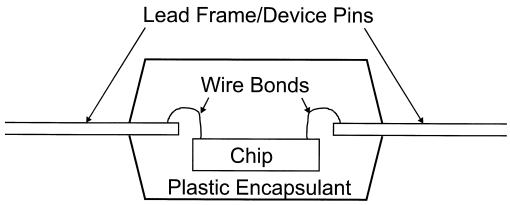


Figure 119 OTP plastic package

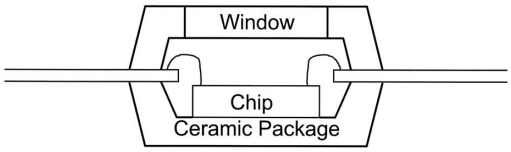


Figure 120 Windowed ceramic package

EEPROM is the term for memory that is erased and reprogrammed one bit or word at a time. Flash uses a common erase circuit for many cells to speed up the erase operation of the chip. When the cell is erased, it is a **1** state, ready to be programmed like EPROM.

EEPROM or Flash is somewhat more expensive than EPROM. Chances are your that PC's motherboard uses Flash for its *POST/BIOS (Power On Self-Test/Basic Input-Output System)* memory that can be updated if errors are discovered by the manufacturer. This is clearly an appropriate use for this technology.

I am not as convinced as to its suitability for microcontroller devices. The PIC16F84 and other Flash PICmicro® MCUs are wonderful devices for learning about the PICmicro® MCU and developing applications, but I do not believe they are very suitable for production. Along with the flash PICmicro® MCUs, EPROM PICmicro® MCUs have the same capabilities (and pinouts), they are cheaper to buy, and have the advantage that the program memory cannot be inadvertently erased and rewritten.

As for the argument that EEPROM program memory allows field upgrades, my response is to think about this for a few moments. Unlike the PC's flash, PICmicro® MCU EEPROM program memory cannot be reprogrammed without connecting the part to a programmer. To upgrade the PICmicro® MCU's program memory, either the product or the chip would have to be shipped to a field-support center, reprogrammed, tested, and shipped back.

A cheaper solution would be to ship out new EPROM parts when the code became available. The best solution is to write the code correctly the first time to prevent field upgrades.

All this does not make the EEPROM-based parts any less useful when you are developing applications. As I wrote this book, the PIC16F87x family has become available and PIC18Fxx parts will be available in the foreseeable future. These parts are pin and function compatible with other PICmicro® MCU parts which makes designing with EEPROM parts and shipping with EPROM parts a viable option that I think a lot of companies will take advantage of.

Two types of volatile memory are available. Dynamic RAM has been known as *single-transistor memory*. This is because a single transistor controls the flow of electrons to and from a capacitor, which makes up a memory cell. The DRAM circuit is shown in Fig. 121.

In this circuit, the amount of charge in the capacitor is used to determine the state of the cell. If a positive charge is in the cell, it is assumed to hold a *1*. No charge is a *0*. To write

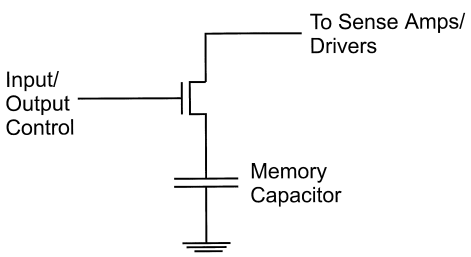


Figure 121 Single DRAM memory cell

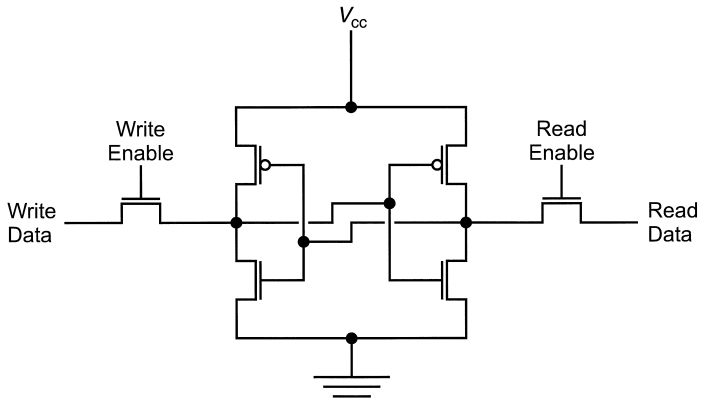


Figure 122 Static RAM (SRAM) memory cell

to the cell, the transistor is turned on. A high or low voltage on the bit line will cause the capacitor to be charged or discharged.

Reading the value in the capacitor is a bit tricky and requires a sense amp. A *sense amp* is simply an SRAM memory cell (Fig. 122) put into a metastable configuration. Thus, it is set at 1/2 V_{dd} instead of at V_{dd} or ground as a normal expected storage value would be. When a cell is read, the transistor is turned on and the charge in the capacitor will cause the sense amp to change state to a *0* for no charge or a *1* when there is a charge.

The capacitor used in the DRAM memory array is a silicon chip in which the charge will "leak" away after a short period of time (tens of milliseconds). To maintain the contents of the DRAM, the charge has to be refreshed periodically. This is accomplished by simply reading the cell. This is the *dynamic* aspect of DRAM; the cells must be continually refreshed to maintain its contents. *Static RAM* will retain its contents as long as power is applied to it.

DRAM is arranged differently than ROM or SRAM. It can even be thought of as three-dimensional memory because each bit has its own array (Fig. 123). The row is used to select a line of memory (each line has its own sense amp) and the bit to be accessed in the row is selected by the column.

In most DRAM memory chips, the row (RAS) and column (CAS) addresses are multi-

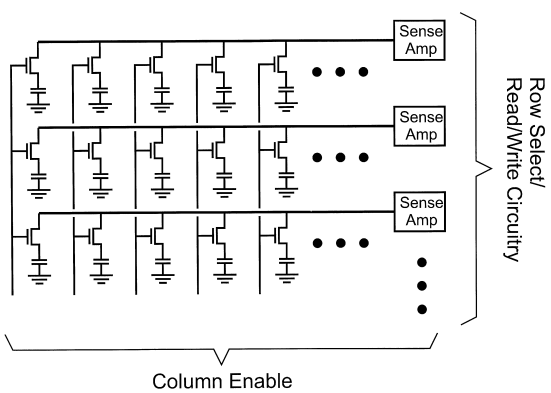


Figure 123 DRAM memory array

plexed on the same line. This makes the timing for a line to be somewhat difficult and somewhat masochistic to do with a PICmicro® MCU. DRAM is designed for large systems where large amounts of memory are required and dedicated hardware is used to interface to the DRAM. This makes DRAM memory very well suited for PCs and workstations.

In this circuit, when one cell is selected in the array. All the others in the row are read. This provides a "free" refresh during the access. Most DRAMS go one further with an *RAS-only refresh*. With only the row specified, all cells in that row are refreshed. A RAS-only refresh is a fast and simple way to refresh the DRAM array. A simple incrementing counter is used to specify the rows in order.

The other type of volatile memory is known as *SRAM (Static Random-Access Memory)*. As mentioned, SRAM will retain its contents for as long as power is applied to the cell. The most common SRAM cell is the "six transistor cell" (Fig. 122).

In this circuit, a write is accomplished by enabling the WE transistor. It "overpowers" the state of the flip/flop to the desired state. Reading is accomplished by simply turning on the Read-Enable transistor and letting it drive the read bus. SRAM, although requiring more space per cell than DRAM, is faster, uses less power and does not require refreshing.

BUSSES

The word *bus* is probably the most overused word in the field of engineering and the most confusing. Along with the uses in electronics that are present in this section, the word *bus* can be used to describe a number of different devices and layouts, including the mechanical framework for satellites. This seems to be a use of the word that is difficult to reconcile when I use it in this book to describe a shared electronic circuit that uses a common set of wires to pass electrical signals to multiple devices.

When working with the PICmicro® MCU, I describe two types of busses: serial and parallel. A *serial bus* passes digital data along a wire, one bit at a time. Your PC's network card passes serial data to a server on a serial data bus. Sending a byte of data serially looks like Fig. 124, with each bit sent for a specific period of time before the next bit is shifted out.

Figure 124 shows each bit as being either high (*1*) or low (*0*) with the lines going up and down. Data is presented as it travels along a point on the wire, which makes it actually backwards from how the data is transmitted on the bus. Only one device can put a signal on a bus, the driver. Multiple receivers can monitor the signal. If two drivers were to put their own signals on the bus, they could be said to be in *contention*. The resulting signal level would be a function of the two drivers. The bus itself and is often unknown or indeterminate.

There are types of serial signals. Asynchronous serial signals provide a start bit so that the receiver "knows" data is coming and it can synch to it. Elsewhere in the book, I'll describe asynchronous signals as they relate to the PICmicro® MCU and other electronic devices. The only issue to be aware of with asynchronous signals is that each bit must have the same period or length of time that the data is active.

The other type of serial data busses are the synchronous bus, in which a clocking signal is transmitted along with the data that is shown in Fig. 125.

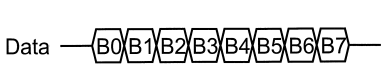


Figure 124 Serial data

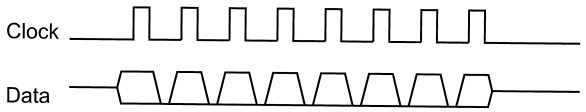


Figure 125 Synchronous data waveform

In Fig. 126, each time the clock line transitions from high to low, the data bit is latched in. This is known as a *negative* or *downward clock edge*. This clock does away with the need for the receiver to synch on a bit, based on its period as an asynchronous serial bus.

The synchronous bus, like the asynchronous bus, can only have one driver, which transmits the clock and data. In some busses, the receiver might produce the clock so that it will receive data at an appropriate rate. Actual synchronous serial buss implementations are also described in detail elsewhere in the book. This means the length of the synchronous serial bit is changeable, according to the operation of the clock.

Parallel busses are another way to pass data, but instead of doing it a bit at a time, multiple wires are used to send multiple bits simultaneously. The parallel bus is usually much faster than the serial bus and included its clocking information to avoid having to synch on bit edges. The most popular parallel busses actually have three distinct types of signals to address particular devices, pass data, and control the operation. This type of bus (Fig. 126) is often used by microprocessors and not embedded microcontrollers.

The read and write lines are used to select the operation of the device, with respect to the data bus (accept data from it, read, or drive data onto it, write).

The parallel bus is not very popular with the PICmicro® MCU because most of the memory and peripherals requiring a parallel bus are inside of it eliminating the need for many external devices. The internal PICmicro® MCU parallel busses allow data to be passed within the PICmicro® MCU faster than via a serial bus. The drawback of the parallel bus is that it requires a lot more wiring than the signal busses.

In this book, I will demonstrate different aspects of each bus and how they interface with devices external to the PICmicro® MCU. Different serial interfaces are presented along with using the PIC17Cxx as a parallel memory interface.

STATE MACHINES

One of the more interesting devices that you can work with is known as the *state machine*. This class of circuit allows you to create a complex application using a simple ROM, instead of a PICmicro® MCU or other microcontroller solution. It might seem like I'm lim-

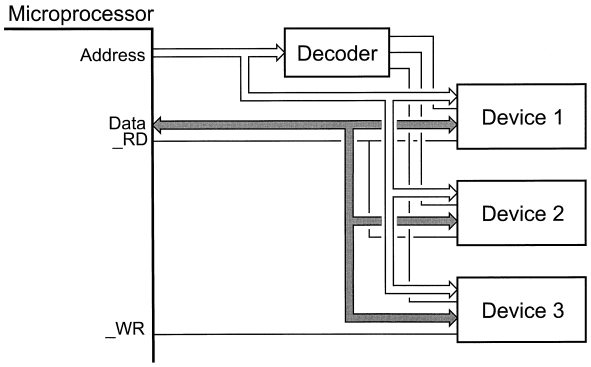


Figure 126 Parallel memory bus

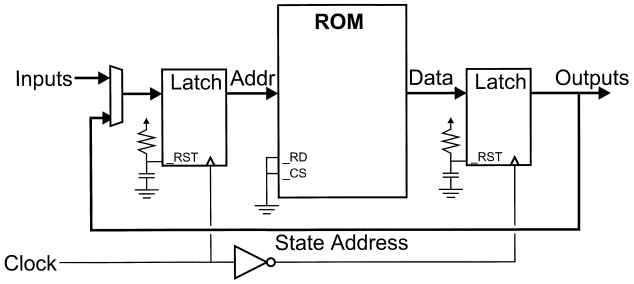


Figure 127 Basic state machine circuit

iting the appeal of this book by describing this circuit, but I think you would be hard pressed to develop a state-machine application as easily and as cheaply as a PICmicro® MCU application. Despite this, the *state machine* concept is good to know about because it provides a different perspective on solving problems.

The typical state machine is shown in Fig. 127. This circuit consists of a ROM (usually EPROM), which has part of its output data fed back as a state address. Other address lines are used as circuit inputs and the state machine changes its state address based on these inputs.

The clock is used to pass the new address to the ROM and then pass the output from the ROM to the outputs and input state circuits. The two latches are operated 180° out of phase to prevent glitches from the ROM changing state and invalidly affecting any output circuits.

As few output bits are used as the "state address" as possible. The reason for this is to maximize the number of outputs and minimize the number of states, which have to be programmed. Each state requires two to the number of inputs to function. Each state responds differently according to the inputs that receives.

A typical application for state machines is a traffic light. If a press-button crossing light (Fig. 128) was implemented, a state-machine circuit (Fig. 129) could be used.

In normal operation (known as *state 0*), the green light is on and the button is not pressed. If the button is pressed, then execution jumps to state one which turns on the yellow light for five seconds (states 2, 3, 4, and 5), after which the red light is put on for 26 seconds (states 6 to 31). If the button is pressed during states 7 to 31, execution jumps to state six to reset the timer.

To keep the circuit simple, I want to use an eight-bit data bus ROM with six inputs (five state, one button). Thus, 2^6 (64) states are required in the ROM. These states are expressed in Table 21. The reset on the input address latch is used to reset the state to zero on the power up. The button is assumed to be "up if a *1*" is returned and "down" if a *0* is returned.

This state table would then be converted into bits and burned into the ROM. An *x* means that both input states have the same result on outputs.

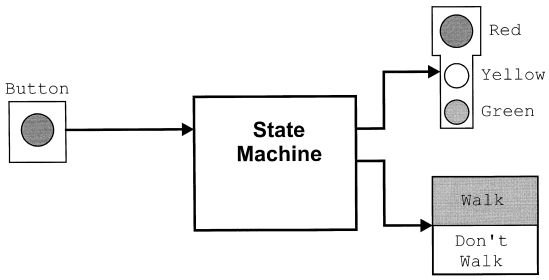


Figure 128 Traffic-light state machine block diagram

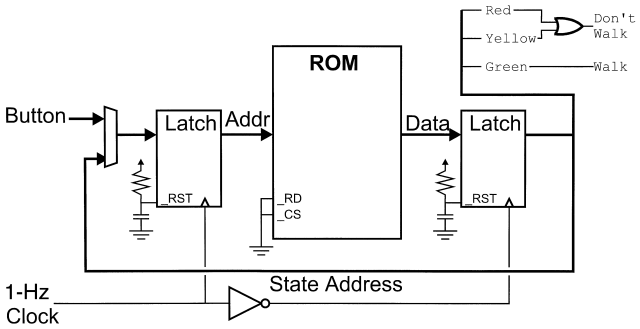


Figure 129 State-machine circuit for traffic lights

This application is reasonable to code and build, but a problem arises with very complex state machines (ones that require tens of inputs and hundreds of different states). These state machines are normally hard coded into an ASIC chip, rather than built with discrete parts, as shown for this application.

The reason to place it within a chip is to give more outputs and more states in a custom application. The depth and the width of the data in real applications better suited to custom chips that can have noncustom memories added much more easily than in the situation

TABLE 21 Example State Machine Rom Programming

	STATE	INPUTS BUTTON	STATE	OUTPUTS GREEN	YELLOW	RED	STATE	COMMENTS
	[0	1	0	1	0	0	0	Power up
Button	[0	0	0	0	1	0	1	
Press	[1	x	1	0	1	0	2	Reset
	[2	x	2	0	1	0	3	
	[3	x	3	0	1	0	4	no button
	[4	x	4	0	1	0	5	yellow on
	5	x	5	0	0	1	6	red on
	6	1	6	0	0	1	7	
	6	0	6	0	0	1	5	
	7	1	7	0	0	1	8	
	7	0	7	0	0	1	5	
	8	1	8	0	0	1	9	
	8	0	8	0	0	1	5	
	:							
	31	1	31	1	0	0	0	Timer done
	31	0	31	0	0	1	5	

where only commercial chips are used. The PICmicro® MCU's processor has a state machine built in to correctly execute instructions.

In this example, I used a state machine with a one-second clock. Obviously, problems are possible (such as the missed input if the button is pressed for less than one second and it isn't released after it is pressed) in this situation. This function makes state machines unattractive for rapidly changing inputs. Any kind of sophisticated real-time processing of inputs is simply not economical with the state machine. When I say "not economical," I am thinking in terms of the memory and properly programming the many states.

Although I do not recommend implementing a hardware state machine, this is not to say that software state machines should be avoided. In fact, software state machines are an effective programming tool. Several examples for the PICmicro® MCU are shown in the book.

Merging the Analog and Digital Worlds

When you start working with digital electronics (which microcontrollers and the PICmicro® MCU are a part of), you tend to try to fit everything you are doing from a digital perspective. That is to say that digital controls and I/O are used in situations where they aren't warranted (and definitely suboptimal). My favorite pet peeve and example of this is modern radio controls. I love having a digital frequency and volume readout, but buttons for tuning and volume control just don't work for me. This is especially true in a car where the lack of a knob means I miss the station or I can't "feel" the volume, except that it is either too loud or too soft.

Despite being a primarily digital device, the PICmicro® MCU can be used with analog I/O devices, as well as controlling a number of different interfaces that do not seem well suited for a digital-only microcontroller. Normally, analog data is presented as variable voltage levels.

Conversion of analog values into a voltage-level input is relatively easy. For example, potentiometers can be used as positional input devices and *Light Dependent-Resistors (LDRs)* can be used to convert a light level into a voltage level that can be determined by the PICmicro® MCU. Converting digital values into analog values is somewhat more difficult. However, this section presents the theory and practice behind doing this (along with a number of example circuits).

The ADC/DAC interfacing presented in this book is almost completely "steady state." Processing high-speed AC signals using digital systems falls under the science of control theory with the subbranch of *Digital Signal Processing (DSP)*. The PICmicro® MCU's processor does not have the capability of handling analog signals more than a few thousand hertz using DSP techniques.

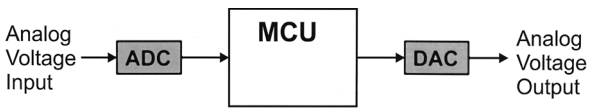


Figure 130 ADCs and DACs in an application

DIGITAL-TO-ANALOG CONVERTERS

One of the most obvious needs for interfacing a microcontroller to the "real" world is the need to provide an analog voltage output. Analog voltages are often seen as the best way to control meters, motors, and lights. In reality, *Digital-to-Analog Converters (DACs),* the circuits that are used to convert digital signals into analog voltages (their operation is shown in Fig. 130), are quite difficult to implement for many devices. As shown here and elsewhere in this book, there are digital means to perform the same functions that are much simpler and better suited for device control.

The most basic for of analog voltage output is the use of the voltage divider. This can consist of two resistors or a potentiometer (digital or otherwise) connected between a high voltage and ground with the wiper (or connection between the two resistors) outputting the analog voltage. The voltage-divider circuit is shown in Fig. 131. It illustrates three important aspects of DACs.

The first aspect is the V_{ref}. This is a known voltage of a given accuracy. Having a precise voltage reference is crucial in all DACs (and analog-to-digital converter) circuits to ensure that the output can be precisely predicted or measured.

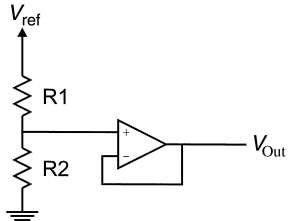
Along with having a precision voltage, the resistor values (R_1 and R_2 in Fig. 131) also have to be precision parts. A digital potentiometer might give this accuracy, but at increased cost and complexity. Now, there are some situations where standard 5% tolerance resistors are "good enough," but for most applications, a precision reference voltage, along with precision resistors are required.

Figure 131 also includes an analog "buffer" to provide drive current for the voltage-divider output. The graphic is for a "differential input" amplifier that uses the output voltage as negative feedback to ensure the output doesn't change because of changes in the load. All of the DAC circuits shown in this section include this buffer circuit because, although the DAC circuits can provide a set voltage, they cannot drive significant current loads.

In the voltage-divider circuit, any kind of current load will change the characteristic impedance of the voltage divider and the voltage output will change with it. To avoid any loading problems, the buffer should have as high an input impedance as possible.

Although the buffer circuit I have shown is very simple, in actuality, it can be very complex for high-current loads—especially for motors and other magnetic devices. In these cases, filtering is required to prevent inductive loading and kickback from affecting the output and possibly causing oscillations within the buffer and the device it is driving.

Despite these dire warnings, simple voltage dividers are useful circuits in many applications. One that you will see in many projects in this book is for a voltage reference for an LCD display contrast. In this application, the current required by the device is very small and a voltage divider can be used without the buffer. As well, the reference voltage



$$V_{Out} = \frac{V_{ref} \times R_2}{R_1 + R_2}$$

Figure 131 Voltage divider

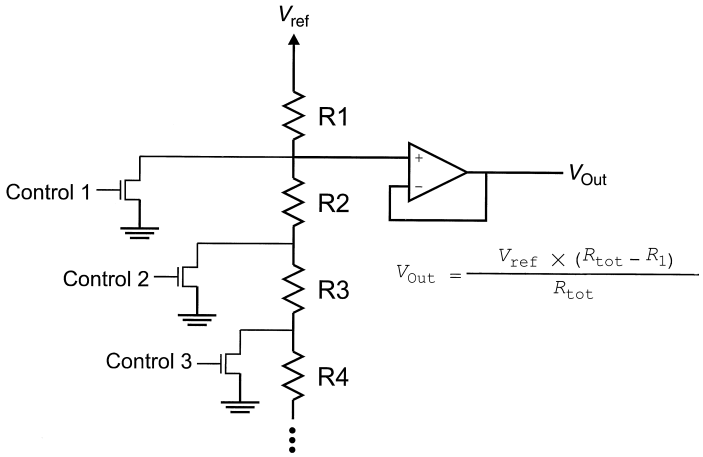


Figure 132 Multiple control voltage divider

does not need to be precise because the contrast is specific to the device and the user. Also, it is qualitative, rather than quantitative.

The voltage-divider circuit in Fig. 131 is only capable of outputting one voltage, which isn't very useful in many applications. Figure 132 shows a circuit that I use when I need to output different voltages in a circuit. In this circuit, when a transistor is turned on, the resistor "above" it is pulled to ground. This truncates the voltage-divider circuit and outputs a different voltage.

For example, if the transistor at Control 1 is turned on, the output voltage is 0 volts. When the transistor at Control 2 is turned on, the output voltage is $V_{ref} \times R_2/(R_1 + R_2)$. For the transistor at "Control 3", the voltage is characterized as $V_{ref} \times (R_2 + R_3)/(R_1 + R_2 + R_3)$. If no transistors are turned on, then the output is at V_{ref}. As more resistors are active, the output voltage approaches the general case of $V_{ref} \times (R_{total} - R_1)/R_{total}$.

The advantage of this type of voltage divider is that it can be easily implemented with a PICmicro® MCU in which each I/O pin can be put in a high-impedance state or pulled to ground (which avoids the need for a separate transistor at each control).

This method has some significant drawbacks, however. Most obvious is the difficulty in selecting resistor values for a large number of different voltage output levels. This is compounded by the circuit output only being a function of the highest turned-on transistor.

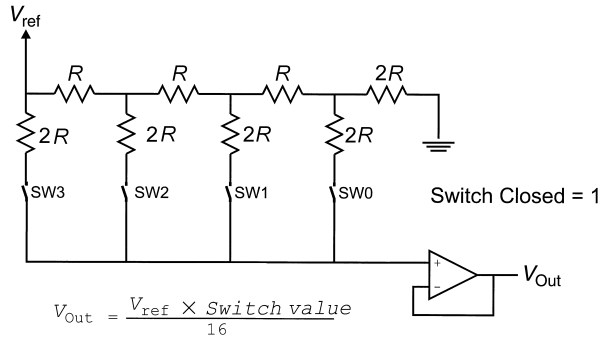


Figure 133 *R*/2*R* voltage-divider ADC

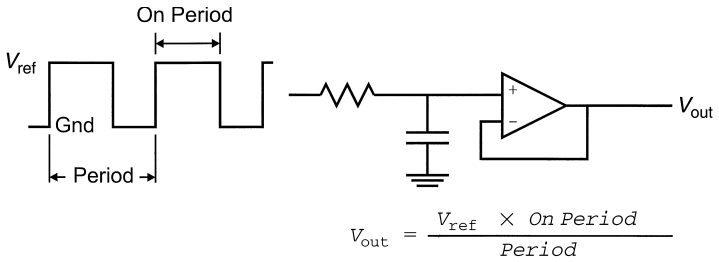


Figure 134 Pulse-width modulated analog voltage

Turning on multiple transistors will not provide an intermediate voltage output value. Despite these limitations, the circuit is useful when just a few analog voltages are required (such as providing composite-video voltage levels).

Figure 133 shows a voltage-divider circuit that can be used to provide a wide range of analog values (with intermediate values) quite easily. In this circuit, when the individual switches are closed, the hex fraction of V_{ref} is output. For example, closing SW3 and SW1 will result in V_{out} being equal to $V_{ref} \times 10/16$.

The largest drawback of this circuit for use as a DAC is the difficulty in finding appropriate analog switches that can be controlled by a digital circuit and allow current to pass through without affecting the operation of the circuit. This type of voltage divider is the circuit most often used inside "canned" DAC devices.

The last type of DAC presented is driving a PWM signal into a low-pass filter. In this circuit (Fig. 134), the capacitor resists the extremes of the digital signal input and smoothes it out into an analog voltage.

PWM signals are widely used in controlling and powering electronic devices, just not using this method. The problem with this circuit is that no matter how good the filtering that is provided, there will always be some ripple in the output voltage, which can cause problems. A much better way of using a PWM is to drive a control with a constant voltage-level signal and using the PWM's duty cycle divided by the period to provide an average power level that is proportional to the desired analog voltage output.

With each of the DAC methods presented in this section, I have also listed a number of significant drawbacks to each method in different applications. In this section and the book, I have tried to provide the applications where each of the different methods are best suited and demonstrate how they can be implemented using the PICmicro® MCU.

ANALOG-TO-DIGITAL CONVERTERS

Although converting digital data into analog voltages can be somewhat difficult, going the other way can be easier—especially with hardware built into the PICmicro® MCU and other microcontrollers. Also, many single chips are dedicated to providing analog-to-digital conversion that can be easily interfaced to the PICmicro® MCU. When I describe ADC conversion in this book, I am primarily describing measuring steady-state voltages, rather than rapidly changing signals. As shown later in this section, it is difficult to accurately measure rapidly changing waveforms.

The most basic way of checking an analog voltage is to compare it against a known reference voltage. The aptly named *comparator circuit* performs this comparison action. Figure 135 shows the comparator's electronic symbol, as well as its response to a fluctuating input.

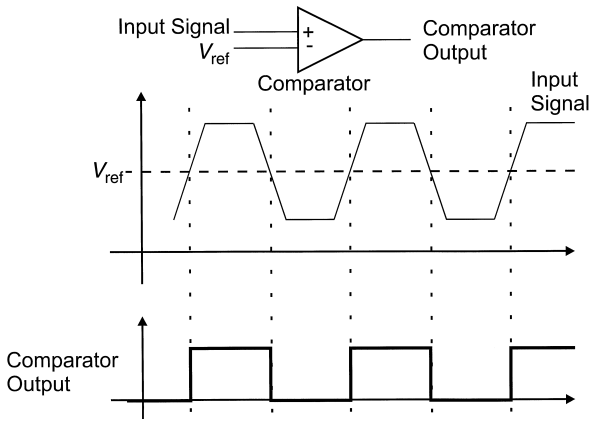


Figure 135 Comparator response

The input circuitry of a digital device is actual a voltage comparator with the compare voltage set to a logic-family-specified threshold voltage. This aspect of digital inputs is used with the PICmicro® MCU elsewhere in this book to provide analog voltage measurements for the PICmicro® MCU.

In the comparator, the output is a *1* if the voltage is greater than the reference voltage. If a second comparator is used, with a voltage window (provided by a voltage-divider circuit), as shown in Fig. 136, a test for a signal being within a window can be produced.

In the thermostat circuit of Fig. 136, Test Out is active if the voltage in is between high and low and could be used to implement an electronic thermostat.

Another application of the comparator is to gang a number together to test multiple conditions and return a digital representation for the voltage (Fig. 137).

The priority encoder will return the binary number for the most-significant comparator, where the comparator's $+ > -$ output is true.

This circuit is known as a *flash ADC* and has the advantage of being very fast (hence its name), but it is also very expensive to implement. This is especially true for an ADC that has a large number of bits. This is because of the difficulty in building the precision resistors needed for the reference voltages and the need to redrive the analog voltage to be accurate across all the comparators. For an eight-bit accuracy ADC, a minimum of 256 resistors and comparators are required.

Another common type of ADC is the integrating ADC. This device uses a sweep generator, which provides a voltage ramp, which is compared against the incoming voltage

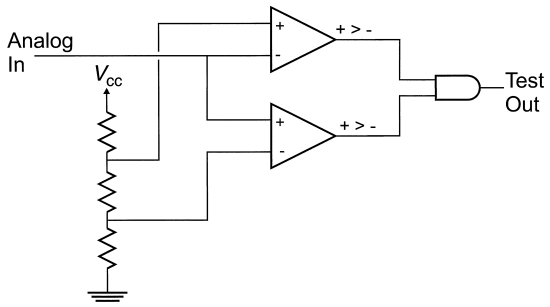


Figure 136 Thermostat using comparators

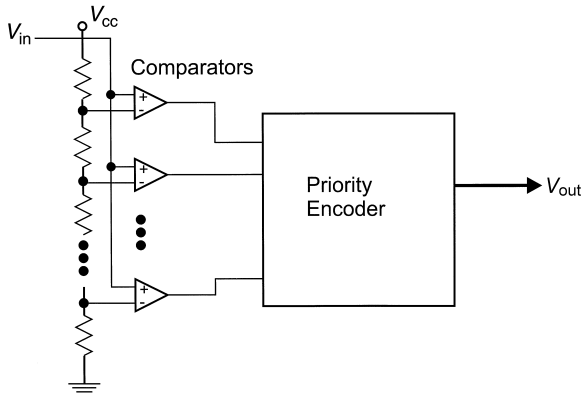


Figure 137 Flash analog-to-digital comparator

when the ramp is started. A timer is also started, which stops when the comparator output is more than the voltage in (Fig. 138).

This circuit is very simple and much more cost effective than the flash ADC, but does take a relatively large amount of time to execute. Often 20 μs (or more) are required for a voltage conversion.

This time leads to some interesting problems when sampling changing voltages. For example, Fig. 139 shows an arbitrary frequency saw-tooth wave going into an integrating ADC, which samples the signal at a much lower frequency.

As can be seen in Fig. 139, the nature of the sawtooth is completely lost and the values read are essentially meaningless. There are methods for interpreting this data, but these fall into the realm of DSP, which is beyond the material of this book (and is really beyond the mathematical capabilities of the PICmicro® MCU).

This inability to process high-speed analog signals is why I suggest that you only work with steady-state analog signals.

In the integrating ADC, you should always ignore one or two least-significant bits of the result. This is because of jitter in the operation of the input signal, the comparator, timer (and its driver circuit), and the sweep generator. If eight-bit accuracy is required in an ADC, then a 10-bit ADC or more should be used to avoid this problem with the least-significant bits of the result being able to be ignored and still have an eight-bit result.

You might get into situations where you want to measure voltages that are higher than

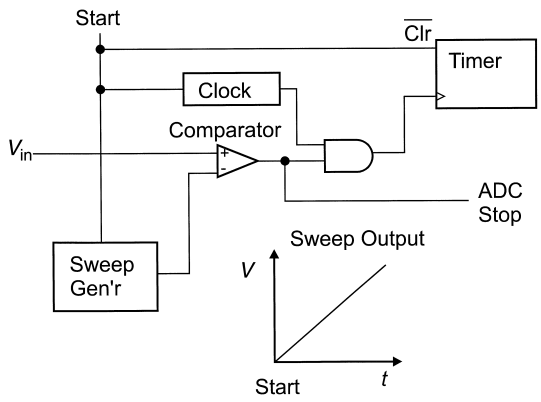


Figure 138 Integrating analog-to-digital converter

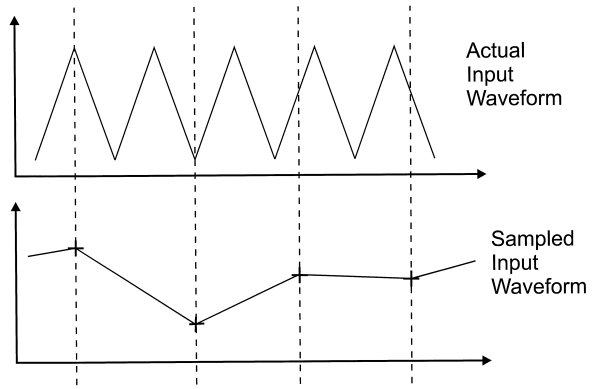


Figure 139 Missed sample read waveform

the ADC is capable of measuring. Normally, an ADC is only capable of measuring from ground to its input powering voltage (+5 volts normally). In these cases, a voltage divider (Fig. 140) is used. Precision resistors (1% tolerance) should be used in the voltage divider to ensure there is no significant change in the accuracy of the ADC.

The final point to make about ADCs is to be sure you that have a good common ground between the voltage input and the ADC. Not having a good ground will result in noise or ground shifts in the ADC input circuitry compared to the input source, which can lead to errors in the measured voltage.

Oscillators

As you become more sophisticated with digital electronics, you'll find that you will be required to create your own oscillators for your applications. Oscillators themselves are very easy to set up, but might have some unusual problems that you will have recognize and debug. A "classic" oscillator circuit is shown in Fig. 141.

This circuit will not start oscillating until the resistor/capacitor delay network has reached the "zero/one" threshold of the NAND gate. When this happens, the NAND gate will essentially become an inverter and allow the oscillator to operate. This resistor/capacitor delay network is used to allow the input voltage to stabilize before the crystal starts to oscillate.

The crystal (often abbreviated as *XTAL* or *Y*) delays the passage of a signal for a precise period of time. This delay will keep the NAND gate's signal at a specific state until the signal causes the input state to change, resulting in a change in output state. The in-

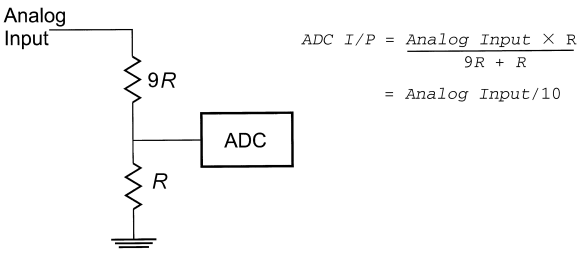


Figure 140 10-to-1 voltage-divider ADC input

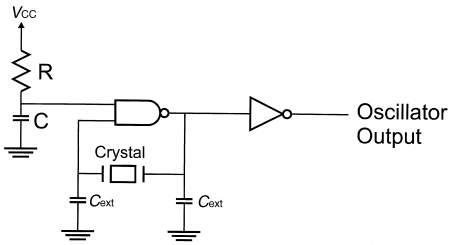


Figure 141 Classic oscillator circuit

verter in the circuit is used to buffer the clock and prevent any external loads from affecting its performance.

The two C_{ext} capacitors are used to condition the signal so that the output square wave is at the correct period. These capacitors are designed to counteract the effects of resistances and inductances built in the circuit and the wiring used to implement it. These values are very small, never greater than a few tens of picofarads. The C_{ext} capacitor values are normally specified by the circuit/oscillator chip/crystal manufacturer for use in the application at a specific clock frequency.

Along with crystals, ceramic resonators are used in the majority of the experiments and projects in the book. A ceramic resonator does not have the costly quartz of the crystal and is much more robust (i.e., better able to withstand physical shocks). Ceramic resonators are often cheaper than crystals, although not as accurate. Ceramic resonators are accurate to about 0.5%, whereas crystals are generally accurate to 0.02% (or better). Ceramic resonators were virtually unknown five years ago, but are reasonably easy to find now in most electronic stores.

Some ceramic resonators have built-in C_{ext} caps, which further simplify their use in an oscillator circuit. Be sure that the built-in caps are at an appropriate value for your application. This can be an issue with the PICmicro® MCU, which has specific capacitor requirements for applications.

The obvious change (optimization) that can be made to the oscillator circuit is replacing the NAND gate and R/C delay network with a simple inverter (Fig. 142).

This circuit will work without any problems almost all of the time. The problem is that it might not start oscillating 100% of the time. Slow initial voltage ramps could cause problems with the circuit entering a Metastable state so that it is unable to start on its own. The resistor/capacitor network with the NAND gate in the original oscillator circuit will allow voltage and the logic state of the NAND output to settle before the oscillator can start. The resistor/capacitor delay time constant should be chosen to be at least three times the expected power voltage rise time.

In some situations, you might find that the clock will not start up properly—even though the circuit was wired correctly and the correct values are used. In these cases, the circuit needs a bit more impedance to the crystal to "kick start it" (Fig. 143).

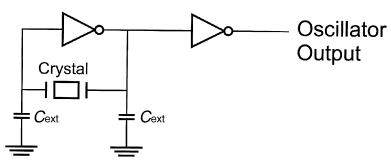


Figure 142 Simplified oscillator circuit

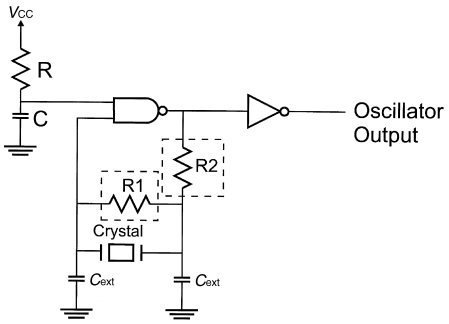


Figure 143 Oscillator circuit with kick-starting resistors

R_1 provides a parallel resistance to start up the OSC and is generally quite high (1 MΩ or more). R_2 provides a series resistance of several hundred ohms to lower the impedance driven by the output of the NAND gate. Adding these resistors should only be done when different C_{ext} values have been tried and the oscillator circuit does not start or run properly.

This brings me to another point when debugging oscillator circuits that don't seem to be working. During circuit debugging, it might appear when checking oscillator with a logic or oscilloscope probe that the oscillator is working. Adding the impedance of an oscilloscope (or logic) probe to an oscillator circuit might cause it to start running. When checking an oscillator circuit, always probe at both sides of the crystal and its output. I've seen a lot of technicians who suspect a bad oscillator, only to find that when they probe it, it appears to be working.

Both sides of the crystal must be checked with a probe on it. If on one side, the oscillator seems to be working, then operation of the circuit should be checked to see if the probe has caused it to start working. Chances are the additional capacitance of the probe has caused the circuit to start working. To fix this problem, a larger capacitance should be added to the side where the probe had started the oscillator working.

Looking at the oscillator circuit, you might wonder if it can be further simplified by eliminating the crystal (or ceramic resonator) altogether and feeding back the inverters output directly into its input. This will work very well and is known as a *ring oscillator* (Fig. 144) with the frequency output being a function of the delay within the feedback inverter. The problem with this circuit is that the output is usually at a very high frequency and different physical chips (of the same type) can have output frequencies differing by up to 100% or more.

I'm really mentioning the ring oscillator because it is possible to have one any time that an output is fed back into an input in an inverted state. The circuit shown in (Fig. 145) has the possibility of becoming a ring oscillator if the unspecified NAND gate inputs are driven high.

Creating a ring oscillator inadvertently is easy to do and sometimes hard to debug, depending on the application.

Along with crystals and ceramic resonators, RC networks can also be used to provide custom delays in an oscillator circuit (Fig. 146).

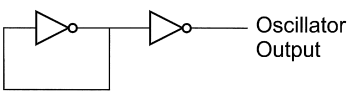


Figure 144 Ring oscillator circuit

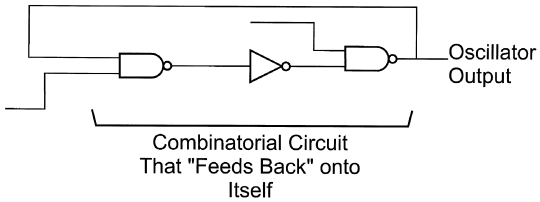


Figure 145 Inadvertant ring oscillator

In this circuit, the resistor limits the current flow to the capacitor. When the charge in the capacitor (and the voltage across it) reaches the threshold of the inverter's input, the output charges the capacitor, then discharges through the resistor. This circuit works very well and is very robust, but the inverter should have a "Schmidt Trigger" input to allow the cap to charge/discharge away from a single threshold value that would cause it to oscillate very quickly. Notice that in this circuit, the RC constant equations do not apply; you will probably have to determine the values experimentally yourself.

AC Considerations

When I got out of university, I was complacent in the thought that digital electronics, except for very high-speed signals, was not something I would have to worry about moving into the alternating current world. I did not enjoy Electromagnetics and Antenna Theory, so I felt that I could safely leave them behind me as I just concentrated on digital (computer) circuits. The error of this assumption was hammered home to me when I worked on memory *SIMMs (Single Inline Memory Modules)* about 10 years ago. In failure analyzing the four megabit DRAM memory chips that were on these parts, we discovered that the chip was not just four million plus digital circuits, but over eight million networked capacitors, each with ability to affect other circuits and data in the chip.

As a result of this work, I have turned a much keener eye to the issue of analog issues and how they relate to digital circuits. This section describes many of the issues relating to working with digital circuits in an analog world. For the most part, what I present should not be that surprising or cause you to drastically change the way you build circuits, but it can help you to avoid problems later.

When you look at a piece of wire, how do you see it? You might think of it as just a resistor (Fig. 147), but, in fact, it is approximated by the reasonably complex network shown below the "better approximations." To best model the line, multiple resistors, capacitors,

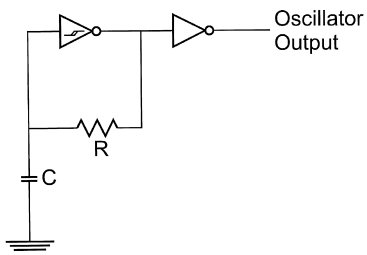


Figure 146 Enhanced ring oscillator with RC period specification

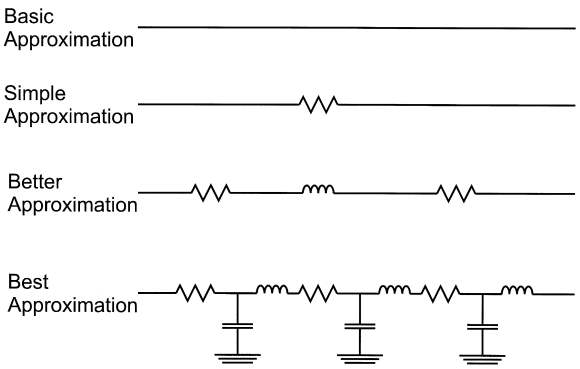


Figure 147 AC approximations of simple wire

and inductors have to be put together (as shown at the bottom of Fig. 147). In fact, to be totally accurate, an infinite number of these devices should be included in the analog.

Thankfully, for the digital circuits we will be working with, this level of detail is not required at the speeds and frequencies used with the PICmicro® MCU, if normal parts and assembly techniques are used. If you intend to use unusual materials or different assembly methods, then you might have some problems.

All wire connections have what is known as *characteristic impedance*. This impedance is an approximation of the line "resistance" frequency. The coax label used for your TV set has a characteristic impedance of 75 Ω. Most computer circuits are designed for driving signals on a 55-Ω impedance line (which is the nominal impedance of lines in a PC board card). Using wires that are unusually large, long, or coiled should be avoided to keep the opportunities for changing the characteristic impedance of the lines in the circuit to a minimum.

When the characteristic impedance is changed, then the signals on the line will be affected. If too much capacitance or inductance is added, the digital signals will be "rounded" or "ring" (Fig. 148). Rounding of edges delays the speed in which digital signals are propagated in a line, and ringing could result in multiple edges being recognized by the circuit. Ringing in some PICmicro® MCU inputs could cause the device to reset itself unexpectedly. The easiest way to avoid these problems is the use the shortest interconnect wires as possible.

Another wiring problem to watch out for is discontinuities in the line. A *discontinuity* is any time the characteristic impedance of a line is changed. Along with changes in the wiring, problems can be incurred by "tees" or "stubs." Discontinuities cause reflections in the line, which can be picked up as extra input edges or can overload the output as it tries to compensate.

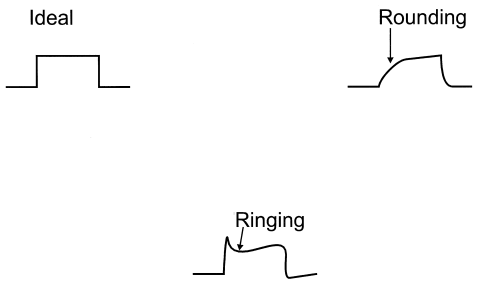


Figure 148 Problems with real signals

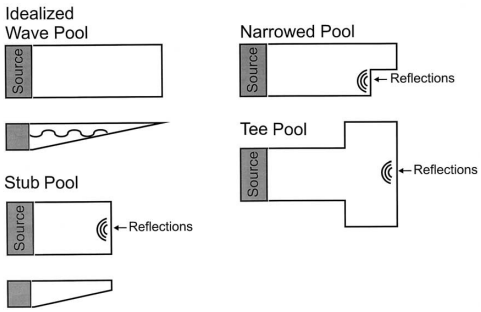


Figure 149 Analogs of electrical reflections using water

To show how reflections are caused, I like to think of a wave pool. In a normal situation (Fig. 149), the wave travel from the source to the end without any reflections. As is also shown in Fig. 149, if there is an abrupt narrowing, or the pool splits off or passes waves to dead ends, reflections will occur.

To avoid these problems in your application, the following precautions must be taken. To keep the characteristic impedance the same, keep your wiring as short as possible and do not use anything other than standard lines (20- to 32-gauge copper wire) and materials (tin/lead solder and copper connectors). Do not coil your wire and keep other lines as far away as possible.

When laying out your circuits avoid putting tees or stubs into the lines. As is shown in Fig. 150, a driver at one end of a net is the best with no connections hanging off into space. Figure 150 shows how a connection between three chips can be done badly or well.

Notice in the "good layout" of Fig. 150, when I have shown the line turning 90° that I have "rounded" the edge by putting in a 45° turn. Going back to the wave-pool analogy, this is the same as going from a 90° bend in the waves (signal) to a gentler one that has fewer reflections back to the source (Fig. 151).

Although not a matter of consideration in the PICmicro® MCU circuits presented in this book, the speed of light and signal propagation becomes a factor in high-speed digital circuits. Electrical signals travel through copper at about a third of the speed of light. The general rule is that a signal travels one foot (30 cm) in one billionth of a second (a nanosecond). In some circuits, great pains are taken to ensure all connections are identical lengths to avoid disparity in signal reception at receivers.

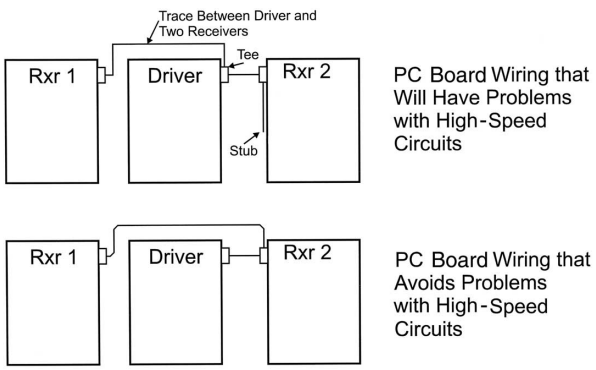


Figure 150 Avoiding reflections in PC Boards

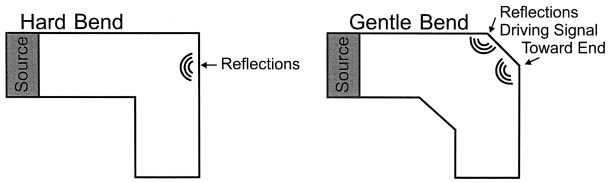


Figure 151 Avoiding problems with bends in PC Boards

It is important to put decoupling capacitors on all active components in your circuits. As chips switch states, the changes in current requirements (because of CMOS switching currents or load changes) can cause induced signals on other lines or power nets. To minimize any opportunity for disruption, 0.01- to 0.1-μF "decoupling" capacitors should be put across the V_{cc} and GND lines of each chip, as close to V_{cc} as possible.

Depending on your experience, you might have seen some of the issues presented in this section. When working with the PICmicro® MCU the likelihood of seeing AC problems, except for decoupling and oscillator, is quite small. But is you work with higher frequencies (which I feel start at 50 MHz), you should follow these rules:

1. Keep connections as short as possible.
2. Do not coil wires.
3. Use standard TTL electronic parts and wiring. For fast operations, use FPGAs or ECL Logic.
4. Do not place tees or stubs in your circuits.
5. Keep parallel circuits the same length.
6. Use decoupling capacitors on all active circuits.

These rules become absolutely imperative for successful operation of your applications. This signal integrity often requires automated tools to check out circuit layouts to ensure they perform properly under all conditions.

RC DELAYS

Even though I am reluctant to use analog circuits in digital (and PICmicro® MCU) applications if I can help it. I use a lot of analog resistor-capacitor combination networks for causing delays in circuits. These delays can be useful for resetting processors, timing oscillators, debouncing buttons, and providing timing delays in applications. In the book, I use RC delays for each of these purposes. This section provides the background behind the circuit operation.

The RC delay is normally wired as shown in Fig. 152. The resistor limits the current to the capacitor so the capacitor takes time to charge to a logic input's threshold. This is shown in Fig. 153 with the input stop function driving an RC delay and the resulting exponential output.

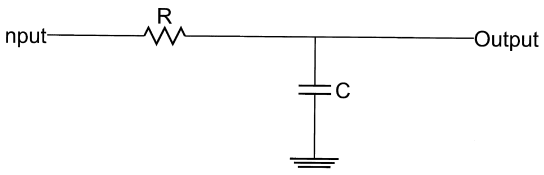


Figure 152 Basic RC delay network

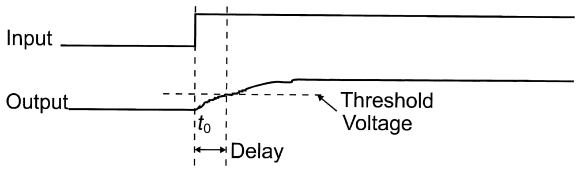


Figure 153 Basic RC delay response

The output waveform is governed by the formula:

$$V_{out} = V_{in} (1 - e^{-tRC})$$

The product *RC* is known as the *RC time constant*. This value is used to determine the time required for the RC delay circuit to reach a specific voltage point after time *t*.

The time to reach a CMOS threshold voltage can be approximated to:

$$T_{delay} = 2.2 \times RC$$

When I wrote this section originally, I went through some of the mathematics that derive this formula, as well as the reasons for this approximation, but they complicate things. A number of provisos keep this value approximate (explained in the following section) and not exact.

To show how this formula is used, I use the microprocessor reset circuit shown in Fig. 154 as an example.

When power is applied to the circuit, the Clock needs time to start up and stabilize before it can drive the microprocessor reliably. The RC delay circuit on the _Reset pin provides this delay. If a 10-k resistor and a 0.1-μF capacitor are used, the delay can be approximated to:

$$\begin{aligned} \textit{Reset delay} &= 2.2 \times RC \\ &= 2.2 \times 11\ \text{k}\Omega \times 0.1\ \mu\text{F} \\ &= 2.2 \times 10\ (10^3) \times 0.1\ (10^{-6})\ \text{seconds} \\ &= 0.0022\ \text{seconds} \end{aligned}$$

So approximately 2.2 ms after V_{cc} is applied to the MPU circuit, the reset will go high and allow the MPU to start running. For a 1-MHz oscillator, this is 2,200 cycles, which is usually enough for an oscillator to stabilize before the microprocessor starts running.

RC delays can also be used as button debounce circuits. If a switch is fairly "clean," then an RC delay circuit can be put in line to filter out the bounces using the circuit shown in Fig. 155.

The waveforms below the circuit show how the RC delay smoothes out the noisy input and the Schmidt trigger input helps to filter out any large bumps that might escape. For a

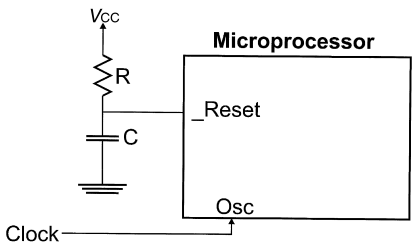


Figure 154 Microprocessor reset with RC delay

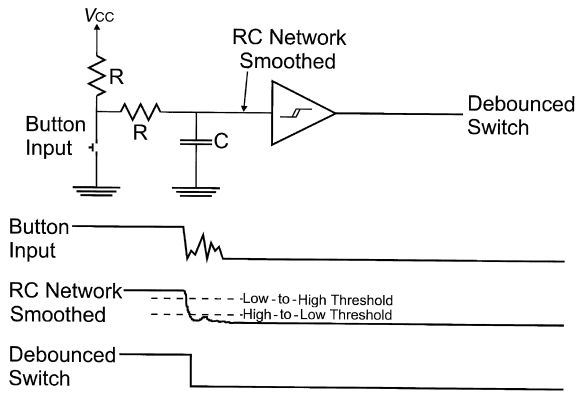


Figure 155 Debounce circuit using RC input

20-ms delay, a resistance of 100 k and a capacitance delay of 0.1 μF should be used. The pull-up value should be relatively small to minimize its impact on the RC delay circuit.

There are a number of reasons why RC delays can be only thought of as being approximate. The first is the actual tolerance of the parts. Five or 10% accurate resistors are very common and cheap. On tantalum capacitors, values can be up to 50% percent from the specified value, with electrolytic and other types being more accurate, but still having large tolerances. The biggest factor in the error of the computed time delay to the actual is in the chip input threshold voltages and outputs.

I realize that earlier in this appendix, I stated that inputs should have infinite impedance and outputs should have zero. For the button debounce example, the high has an input impedance of 10 k and an output impedance of 0 Ω. This will mean the button press and release-time delays will be different.

Cooling

The subject of cooling can be very important to PICmicro® MCU designs, although nowhere as crucial as the cooling requirements for a PC, in which modern Pentium processors can dissipate more than 50 watts of heat. Some parts in your PICmicro® MCU applications that you will have to be sure that you understand whether or not you have to provide features and options that will help cool them. The parts that you will have to most actively cool are voltage regulators, and motor and relay drivers. These devices can dissipate several watts of power—enough to cause the parts to become hot enough to burn you, cause thermal shut down, or even unsolder themselves from the board.

In this section, I'm going to talk about passive cooling, which does not involve any parts that aid cooling and have to be powered themselves. Instead, I describe the need for cooling, along with two methods of "passively" cooling parts.

As I have described in the previous sections, DC power is defined by the equation:

$$P = V \times I$$

where V is the voltage across the device and I is the current throughout it. In a linear regulator or bipolar motor driver, this equation can be used directly. For example, in a

7805, which has nine volts input and is sourcing one amp, the power dissipated by it is defined as:

$$\begin{aligned} P &= V \times I \\ &= (9 \text{ volts in} - 5 \text{ volts out}) \times 1 \text{ amp} \\ &= 4 \text{ volts} - 1 \text{ amp} \\ &= 4 \text{ watts} \end{aligned}$$

For MOSFET motor drivers, there is no defined voltage drop, instead, there is an internal resistance. In this case, the power equation replaces *I* with the Ohm's Law equivalent:

$$\begin{aligned} P &= V \times I \\ &= V \times (V/R) \\ &= V^2/R \end{aligned}$$

So, for an N-channel MOSFET driver that has an on resistance of 0.5 Ω with a 1/4-volt drop in a motor circuit, the power dissipated is:

$$\begin{aligned} P &= V^2/R \\ &= (1/4 \text{ V})^2/0.5 \ \Omega \\ &= 1/8 \text{ watt} \end{aligned}$$

This example shows two things: MOSFET transistors can be much more power efficient in providing motor control than bipolar devices, which have on resistances on the order of 10 Ω. Secondly, Ohm's Law and the other basic DC laws should always be kept in the back of your mind.

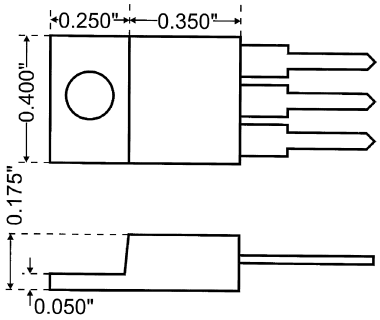
With the power dissipated by each component in a circuit calculated (and you should take step for every circuit), you should determine whether or not components require a heatsink. My general rule is that if more than 0.5 watt is not going to be dissipated by the part itself, then it should have a heatsink connected to it.

This statement is a bit hard to understand because it takes into account the ability of the component's package to act as a heatsink itself. A heatsink is a metal device, which is used to pass heat from a component into the surrounding environment (normally, the air naturally circulating around the component). You have probably seen heatsinks; a good example of one is the finned cylinders of an air-cooled motorcycle engine. These fins pass heat from the device (whether it is a V register or a single cylinder) and transfer it to the surrounding air.

For devices that operate at room temperature, (20 to 25°C or 68 to 77°F), I use the general rule that one square centimeter of heatsink area is required for each watt of heat to dissipate. The important part of this statement is the term *surface area*. *Surface area* refers to the amount of area of the heatsink that is exposed to the outside environment. For the situation where you have a 78.5 voltage regulator in a TO-220 package, which is bolted to a PC board, the surface area available to act as the heatsink is the exposed sides of the part to the surrounding air (Fig. 156).

According to my calculations, the surface area of the TO-220 package is 3.452 cm^2, which means the package itself can dissipate 3.45 watts of power. Going back to the voltage regulator example above (4 watts output), the amount of power, which is not accounted for is (4 to 3.45 watts) 0.55 watts and will have to be dissipated by a heatsink.

The term *surface area* is also an important concept to remember because it affects how a heatsink is designed. Figure 157 shows two different types of heatsinks, both of which



Total Area =
Top Side +
Left/Right Sides +
Ends
$= (0.4 \times 0.6) +$
$2 \times [(0.35 \times 0.175) +$
$(0.25 \times 0.05)] +$
$2 \times (0.4 \times 0.175) +$
(0.4×0.050)
$= 0.240 + 2 \times 0.061 +$
$0.013 + 2 \times 0.070 +$
0.020 Inch^2
$= 0.535 \text{ Inch}^2$

$0.535 \text{ Inch}^2 = 3.452 \text{ cm}^2$

Figure 156 TO-220 surface-area calculation

take up the same amount of space, but the one with the cut outs have a lot more surface area.

By adding cutouts to a heatsinks irregular surface, you can double or triple the surface area of a heatsink quite easily.

As you learned in grade school, black is the best color for heat transfer. Commercial heatsinks use black anodized aluminum with a rough surface, which provides more surface area than on a smooth surface and the black color allows better heat transfer.

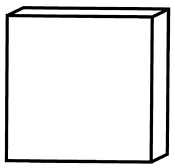
Most heatsinks are designed to operate in an open-air environment or in a cabinet that has holes to allow heat to leave via convection. If an opening is not provided, then heat can build up within the product and cause the electronics to fail or "age" prematurely.

If an air path is not available, then the components can be bonded to large heatsinks. When using large heatsinks, you should be aware of heat conductance and heat capacity. Heat capacity is the amount of heat that a sink can absorb. It is directly related to the mass of the heatsink.

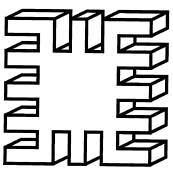
Heat conductance is the most important parameter as it relates to how quickly heat is transferred through the sink. As Fig. 158 shows, a good heat conductor will have the most even heat distribution through the heatsink.

Ideally, in a solid heatsink, the heatsink passes the heat to an air or water source, which will transfer the heat out of the heatsink. Otherwise, the heatsink will have to be designed so that it does not become saturated during operation and between high load operations, the heatsink will cool sufficiently to prevent any kind of heat build up from happening and prevent the part from working at the specified temperature.

Physical heatsinks add cost to a project, as well as often increasing the height of the components. If you have an application, which has modest (less than 5 watt) dissipation



1 cm $\times$ 1 cm $\times$ 0.5 cm
Heatsink Plate
3 cm^2 Surface Area



1 cm $\times$ 1 cm $\times$ 0.5
Cut Heatsink Plate
5.370 cm^2 Surface Area

Figure 157 Heatsink surface-area comparison

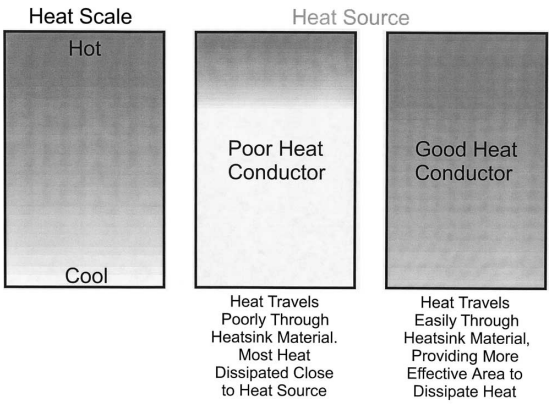


Figure 158 Heatsink material heat conduction

needs, the PC board that the application is mounted on can be used as the heatsink. For a TO-220 package, the heatsink, put on the card's backside could be used (Fig. 159).

When implementing this type of heatsink, an outside PC board layer must be used with a direct metal (i.e., no solder mask) interface should be used between the topside heatsink interface and the part. When a backside heatsink is provided, then additional vias should be built into the card to help transfer heat from the topside to the bottom.

For this type or heatsink, instead of using the general rule of one cm^2/watt, I use 0.5 cm^2/watt for planning the amount of area required in the PC board. This derating is used to reflect the lower airflow that can be expected across the heatsink. This might seem like a significant derater, but remember that each square inch has 6 1/4 square centimeters, dissipating five watts using this method requires less than two square inches.

Printed Circuit Boards (PC boards)

As I was proofreading this appendix, I realized that I have not given you an introduction to the *printed circuit board (PC board)*. Although there are a lot of different things to know about PC boards, I felt it would be important to introduce you to the basic concepts of the two-layer printed-circuit board, its features, and how it is manufactured (Fig. 160).

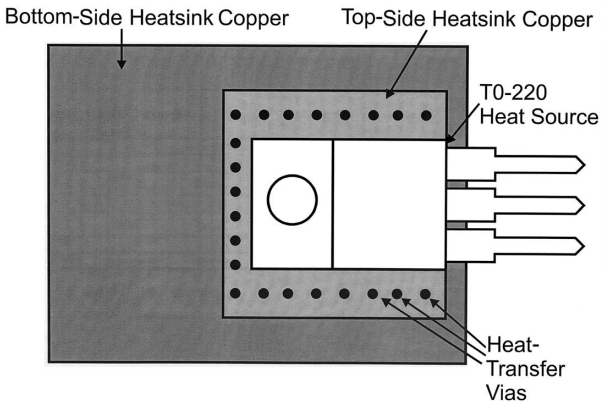


Figure 159 Using PC Board for TO-220's heatsink

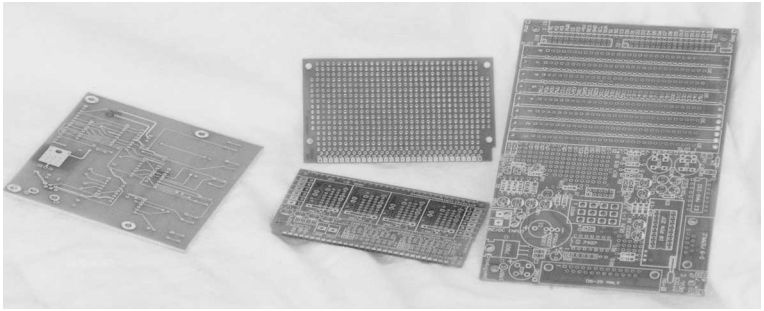


Figure 160 Different PCBs

I will also describe multilayer (more than two layer) boards—even though you will probably not work that much with them as you develop your PICmicro® MCU application development skills.

The basic PC board consists of a fiberglass board with copper circuits attached to it. Holes (known as *vias*) are drilled into the board to allow *Pin-through Hole (PTH)* parts to be soldered to the backside, with the PC board protecting the component from the heat of the solder. The copper circuits embedded on the card are known as *traces* or *lands*. This book uses the term *trace* to describe each circuit passed between the vias or surface-mount component pads on the card. Normally, the vias have a layer of copper inside of the holes to make soldering the component pins within them more reliable and provide a current path between the PC board's top side and bottom side traces.

Most PC boards, once components have been placed on them and their leads or pins are inserted into the vias are run over molten solder. This provides a high-speed method to solder all the components to the board. To minimize problems with the solder wave shorting traces together or burning the fiberglass PC board, a layer of a plastic called *solder mask* is stenciled onto the PC board.

Surface-Mount Technology (SMT) components are soldered to the traces on the top and bottom sides of the PC boards. The pads that the SMT components are soldered to have openings in the solder mask in which they can access the copper traces on the PC board. Later, this appendix describes the issues behind adding SMT components to a PC board.

When a PC board is cross sectioned, it will look like the diagram shown in Fig. 161.

PC boards are manufactured from a PC board *panel*, a piece of fiberglass with copper sheets bonded to the top and bottom sides for a two-layer board or on just the bottom side for a single-layer board. This panel is first drilled with the vias that were specified in the PC board information (Gerber) files. If the panel is a two-layer PC board, it is then immersed into a bath of electroless copper where the copper inside the vias is "grown."

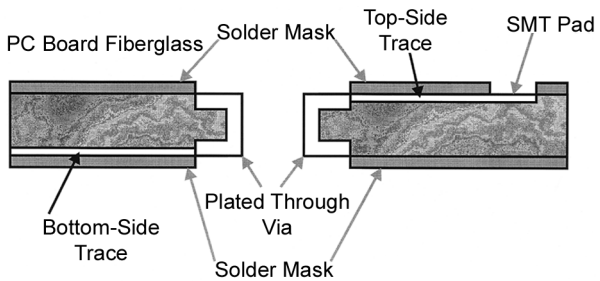


Figure 161 PC board cross section

Next, the board is coated with a chemical known as *photoresist*. This material changes state when exposed to light and then developed (much like a photographic negative is chemically developed). Depending on the type of photoresist, the areas of the board that were exposed to the light are either eaten away by a chemical etchant or left untouched. The circuits are put onto the board by placing a mylar mask over the board and exposing the combination to light. The mask has the pattern of the circuits printed on it and will either allow or prevent the light from passing through it to the photoresist.

When the photoresist has been exposed and developed, the PC board panel is then put into an etchant bath. An *etchant* is a chemical that removes the copper from the panel. After the panel is removed from the etchant bath, it is essentially ready to have components soldered to it. For "quick-turn" boards, this is generally the product that you will receive.

For standard PCBs, after the copper is etched, a layer of solder mask is put on the top and bottom of the board, and then removed in the locations where components are to be soldered to the board. The solder mask is removed using a process that is very similar to that of etching the copper off the board with a layer of photoresist used to specify where a chemical etchant will operate. Once this is done, a silkscreen of placement, part, directions, or information can be printed on the card.

In some cards, specifically ones that will be plugged into card-edge connectors, a layer of gold might be bonded to some of the exposed copper. This will allow the card to be plugged in or out without the copper oxidizing and no longer providing a low-resistance path between the connector and the circuits on the PC board. This is really all there is to manufacturing PC boards.

Multilayer PC boards are built by bonding multiple two-layer PC boards together. Although being quite a bit more expensive, they do offer advantages in terms of circuit density (extra layers means more area for circuits), as well as provide large ground and power planes. The ground and power planes are also useful heatsinks for high-power consumption devices. The cross section of a four-layer multilayer board is shown in Fig. 162.

The bonding material for the two-layer constituent PC boards is a layer of fiberglass known as *prepreg*. The PC boards are compressed with the prepreg sheets in between them in very precise presses that heat up the PC board to speed up the prepreg curing and provide a vacuum to draw away any excess resin or gasses.

The PC boards that are used in multilayer PC boards have not been drilled. Also notice that the final top and bottom layers are not etched when the PC boards are pressed

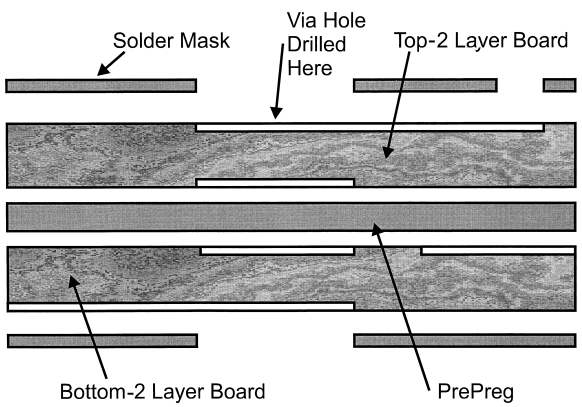


Figure 162 Four-layer multiple-layer PC board cross section

together. Once the various PC boards have been attached with prepreg and cured, the vias are drilled into the card and plated through. Once this is done, the top- and bottom-side circuits are etched and then solder mask and silkscreen are applied.

Multilayer PC boards are much more complex to work with than single- or double-layer PC boards. Although single- and double-layer PC boards can be successfully designed and worked with very little experience or knowledge, multilayer PC boards do require quite a bit of science to design and solder components to properly. When you first start designing your own PC boards, I suggest that you keep to a single- or double-layer PC board and try to find resources that can help you understand the various issues of working with multilayer PC boards.

ESD

If you are familiar with modern electronics, you will know that static electricity can be a major concern that could potentially affect the operation of your PICmicro® MCU applications and even damage the semiconductor circuits within them. In this section introduces the concept of *ElectroStatic Discharge (ESD)* and the damage it can do, as well as the measures that can be taken to avoid it.

Static electricity should not be a new concept to anyone. I'm sure that everyone has shuffled across a synthetic fiber carpet and touched a door handle and received a shock at one time or another. This series of actions consist of building up the static electrical charge (shuffling across the carpet) and discharging the energy (the blue spark and the snap that you hear). The energy discharge is caused by electrons at one potential jumping to an object of a different potential. After the discharge, both objects are at the same electrical potential and the opportunity for the charge to move between the objects is greatly diminished.

The spark and the current flow are known as *electrostatic discharge* and have some characteristics that you should be aware of. First, the actual current flow between the objects is minuscule. Often, they are on the order of millionths of coulombs of charge, but the voltage potentials are very large. For you to feel the ESD shock, a potential of at least 2,500 volts must be produced.

Silicon semiconductors can be damaged by electrostatic discharges of a 100 volts of potential and billionths of coulombs of charge. The surge from the electrostatic discharge can pierce gates and doping junctions (Fig. 163), degrading the operation of the transistor that received the charge. ESD causes microholes in the silicon, which can allow invalid current flows, changing the operation of the transistor. As the transistor is hit with repeated ESD events, the surface area of the holes increases, degrading the transistor's operation.

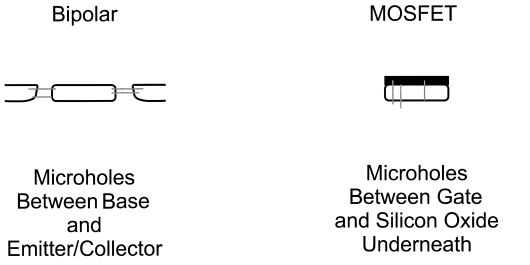


Figure 163 ESD damage to transistors

One popular method of reducing ESD damage is to place clamping diodes on I/O pins (Fig. 164). When the ESD surge is passed to the I/O pin, it is shunted through the diodes the same way a "kickback" surge is shunted through clamping diodes. These diodes greatly reduce the opportunity for ESD damage.

I have found the PICmicro® MCU to be particularly immune to damage to ESD, but this does not mean it cannot be damaged and execution upsets are possible. To avoid any problems from of ESD, you can do a number of things to minimize the opportunity of ESD shocks from reaching the PICmicro® MCU and application circuits.

The most basic thing you can do is to use an anti-static wrist strap any time that you handle electronic components. By connecting the strap to your wall outlet's earth ground, you will protect your circuits from being damaged by any static charges that build up on you.

The earth ground can normally be found as the center screw in a wall outlet. Before connecting the wrist strap to the earth-ground screw, use a three-pin socket tester to ensure that the wall socket is wired correctly. Although a 1-MΩ resistor is built into the wrist strap and ESD cable, it is not a good idea to wire your wrist to 110 VAC.

Wrist straps (and the coiled cables that come with it) cost just a few dollars and are available from most electronic stores.

A better solution is to also have an antistatic mat that is connected to earth ground that the wrist strap can attach to. A picture of my set up is shown in Fig. 165.

When you are not working with electronic projects, place them in ESD "static-safe" bags or containers. This will minimize the opportunity for the casual handling of parts causing problems and damaging parts. Large numbers of these bags can be bought for just a few dollars. If you're cheap (like me), you can save the ESD bags that come with chips that you've ordered and re-use them for completed projects.

A fairly easy ESD precaution is to ensure the air is continually humidified. For the North American Southwest, a "swamp cooler" is definitely a necessity in the summer. In Northern climates, a humidifier should be added to your furnace. The moister the air is, the lower the opportunity for a static charge to build. Ideally, the relative humidity should be 60 to 70% to prevent static build up, but not have condensation forming.

Once your project is finished, it should be put into an enclosed box. Not only will this prevent static-filled fingers from causing ESD damage to the circuit, but it will also protect those same fingers from shocks from the high-voltage circuits of the application.

These precautions are pretty rudimentary. If you are going to be creating a professional lab, then you might want to invest in ESD smocks, shoes, tiling, and other specialized apparatus to prevent the product you are working with from being damaged.

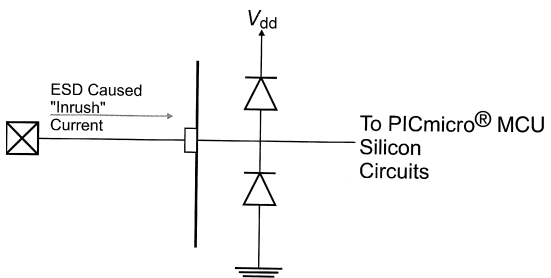


Figure 164 ESD clamping diodes

Figure 165

Electronic Design Systems

The first circuits that you design yourself on paper will probably look like meaningless scribbles on a piece of paper that has been worn thin by multiple erasures. When I look back at the first circuits I did (including some done years later), I find I can barely read them because of incomplete erasures, margin notes, and miscellaneous calculations (a scrap of paper wasn't near by). Your first designs will probably look just the same if you don't opt to draw the circuits out using a PC-based modern design system.

Depending on how deeply you get into circuit design, you might want to buy yourself schematic capture and PC-board layout tools. These two programs are designed to help you design and lay out your applications on a PC board. There are a lot of different vendors out there, each one claiming that their tools are the best. As well, you will find people on the Internet that will swear by one type of tool over another.

Although a variety of schematic capture and PC-board layout tools are on the market today, each designed to make the chore of designing circuits and PC board boards to build them on, I have been disappointed by the quality of low-cost tools. This is not to say that the inexpensive tools aren't designed well, but they often crash or lock up the PC. They also may have a limitation on the number of nodes (or "pins") you can work with (to keep the price low).

In higher-priced tools (costing up to tens of thousands of dollars per year for a "seat"), the problems of the low-cost tools are much less prevalent and the tools can make you more ef-

ficient in designing your circuits and laying them out onto a PC board. Additional features, such as built-in simulators, allow you to try out your circuits before building prototypes. Autorouters, which find the best path for the circuits, are also common in these tools.

If you know of any integrated schematic capture and PC board layout tools that are reasonably inexpensive ($300 or less), well integrated with Microsoft Windows (which means that they can use the clipboard and import and output common graphics files), and are reliable (no crashing or system hang ups), please let me know. All of the low-cost tools that I have looked at (meaning downloaded and bought) have had problems that have caused me to keep looking. I cannot recommend any low-cost system.

SCHEMATIC-CAPTURE TOOLS

This and the following sections, introduce a schematic-capture and PC-board layout tools, give you some pointers on what kind of features to look for and how to decide what best meets your needs.

The schematic capture tool automates the task of drawing out your circuit. The application is really a graphics program with a very narrow focus with what it can do. Along with providing the ability to put down standard graphics and draw lines between different parts of them, a schematic capture tool will also produce netlists that are used to wire the application. A typical schematic capture tool was used for the schematics presented in this book. The window that I had while developing the YAP-II is shown in Fig. 166.

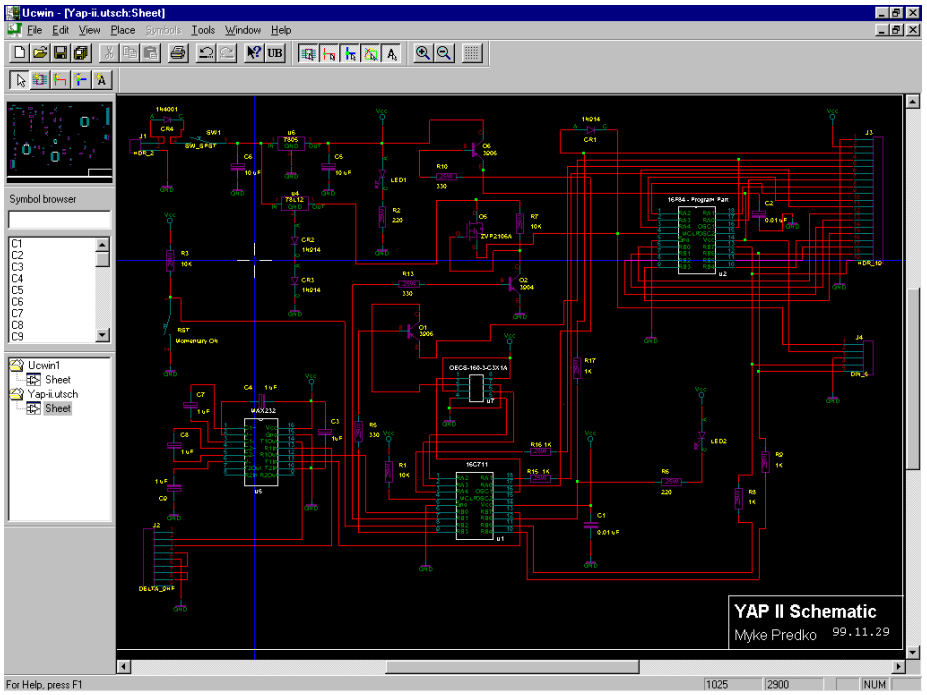


Figure 166 Example schematic-capture tool

To start designing a circuit using a schematic capture tool, first all of the parts are laid out onto the sheets that are used for the application. For the PICmicro® MCU applications presented in this book, I always started with the central core circuit and worked my way out from there. Next, I add connectors and started filling in the blanks, wiring the PICmicro® MCU and the peripheral circuits to the connectors or displays.

When I say "filling in the blanks," I mean putting in the wiring between the different components. Each wire is known as a *net* and it can consist of several components wired together in one circuit. As a general rule, in a net, there can only be one driver at a time or else bus contention will occur when two drivers are outputting different values. During bus contention, the actual voltage value is generally referred to as being *undefined,* with the actual value being dependent on the electrical characteristics of the two drivers.

Many schematic capture tools claim to have a very wide range of parts built in and you will never need to add your own. Don't ever believe this claim: literally hundreds of thousands of different parts could be used in your application and literally hundreds more are becoming available each week. For this reason, you will have to be able to develop your own prototypes for parts.

Creating your own prototypes is usually not a terrible experience because most schematic capture tools provide a library manager (Fig. 167), which shows a PIC16F877. This tool will allow you to create your own parts to add to the schematics.

When I create parts, I generally design them on a pin-per-pin basis (Figs.166 and167) so that the wiring to the part can be observed directly. In many predefined libraries, parts are

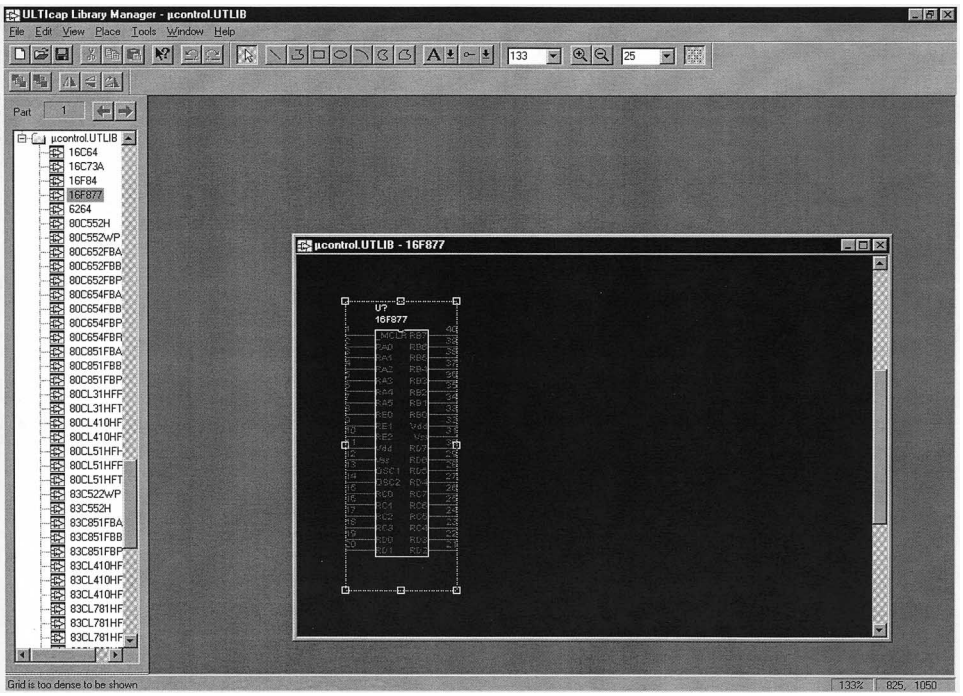


Figure 167 Example library manager

laid out according to functional blocks or conventions that can be hard to follow when you are creating your own parts. Usually, these conventions attempt to follow logic standards or use conventions (the most common is inputs on the part's left side and outputs on the part's right). To be honest, I find that I never seem to have the time to properly follow the conventions, so using the part pinout usually works best for me.

Along with putting individual lines (or "nets") on schematics, you can also specify *busses,* which are collections of nets that are used for a similar purpose. Busses in schematic capture tools generally follow the same rules as busses in circuit descriptions: there are data busses, address busses, etc. The reason for having busses in schematic capture tools is to keep the schematic from getting too "messy" and unreadable.

Developing a circuit schematic should follow the same rules as software development, with intermediate levels saved so that when problems are discovered, fixing them does not involve major changes to the schematic. I generally work through several versions of a schematic before indicating that it is ready for creating a netlist and laying out a PC board.

If a circuit is very complex, you might wish to place it on multiple sheets of paper. Most schematic capture tools have the capability of linking the nets together to form a coherent circuit. Using multiple pages can be an effective method of tracking changes to the circuit. Thus, you can go back and fix specific problem areas, rather than go through the entire schematic. This process is analogous to linking multiple object files together into an application.

Many schematic capture tools have the capability of creating a netlist from the schematic. This process involves checking the pin (or "node") types in a net and ensuring that they are compatible. In most schematic capture tools, you will see the following types of pins:

1. Power
2. Input
3. Output
4. Bidirectional
5. Tristatable (not the same as bidirectional)
6. No connect
7. Unspecified

When the schematic capture tool checks the pins, it is ensuring that no I/O pins connected to power, no connect pins are in fact no connect, and the other pins are connected to something. The tool might also go ahead and wire power and ground in a circuit automatically, based on the pin specifications. These are known as *design rules,* which might be changed according to the requirements of the application.

If any errors occur in what the schematic capture tool sees in how the circuit is wired together, the program will return an error and ask you for your input. Instead of changing the default design rules, you will probably want to allow variances in specific cases when the rule check/netlist create takes place.

Once the schematic capture tool has checked all of the pins' connections and established that nothing seems invalid, it will go ahead and create a *netlist,* which is list of each net in the circuit and the connections that are made for them. The netlist is passed to the PC board layout tool to design the circuit into a PC board.

As indicated at the start of this section, a great deal of differences occur among

schematic-capture packages. At the very low end, you can get a set of standard device symbols, which can be embedded into a standard graphics package (such as CorelDRAW!, PowerPoint, or Freelance) to allow a circuit to be built from there. As you work your way up, you will get more features, larger libraries, and move complex rule checking and circuit simulation (including the ability to test such devices as programmed PICmicro® MCUs and applications in circuits).

The ability of these features to run correctly seems to be more of a function of the cost of the schematic capture tool rather than anything else. A number of schematic capture tools are on the market in the sub-$500 range, each with varying features and varying stability. When I mention "stability," I am talking about applications that run without failing, have all their features work correctly at the same time, and are able to save the current circuit at any time.

If you are a hobbyist, trying your hand at circuit design for the first time, these problems and modest features will probably not significantly hinder you. Keeping the applications simple will probably make your life a bit easier if you don't have to learn many functions (most of which will probably not be required for your initial applications).

On the other hand, if you are going to make your living from designing circuits, then you will probably want to investigate what is available on the market and find out how well it is received in terms of stability and operation. You will find that the more expensive a schematic capture tool is, the better it is received. Tools costing $10,000 a year (or more) per "seat" are given the best reviews.

Netlists When the circuit is entered into the schematic capture tool software and has been checked against the built-in "rules," the schematic-capture software will produce a file detailing how the circuit is wired together. This is known as a *netlist*. There are many different formats for this file, but once you understand what you are being shown, you should be able to decode the information for yourself.

A common format for circuit wiring is to have the net name followed by the components connected to the net and their pin numbers. Part of a netlist could look like that shown in Table 22.

TABLE 22 Example Circuit Netlist

```
GND
  U1.8, U1.11
  U2.4
  R3.2
  Q1.3
  C1.2

DO
  U1.9
  U2.5

$$$01
  U1.10
  U2.6
  C1.1
```

The net names are usually picked up by the schematic capture tool from the component pin names, connector names, bus names, or explicit labels given to the net by the circuit designer. If there is no way for the schematic capture tool to label it using names in the circuit, then it will produce one for itself.

The first line of the example specifies the net name of the power ground (GND). The next line indicates that U1 has pins 8 and 11 connected to the net. The subsequent lines list the pins on U2, R3, Q1, and C1, which are all connected to ground.

The next line, D0, indicates the start of a new net that connects U1 pin 9 and U2 pin 5 together.

$$$01 is a net name generated by the schematic capture tool for a net that wasn't labeled when the circuit was designed. This net connects different pins of U1, U2, and C1 together. To really understand this net, you might have to go back to see what the different pins connected to it are doing. Once this is done, the net can be explicitly labeled according to the signals it is carrying.

The netlist can be passed to PC board layout software or used as a reference to wire the circuit yourself using wire breadboards, wire wrapping, or point-to-point wiring. The netlist is almost always "human readable," so it can be checked by the circuit designer or used to wire a prototype of the circuit.

PC BOARD LAYOUT TOOLS

Once you have your circuit designed, the parts are specified, and you have a net list, you can sit down and create your own printed circuit board. Many people have played around with kits that give you tape, markers, and chip layouts to manufacture their own cards. Radio Shack (among others) has been selling these kits for years. The biggest problem after figuring out how to safely dispose of the etchant (which is toxic waste under any definition) is that these kits only allow one card to be built after quite a bit of effort.

A much better method of creating PC boards is to use an automated program, which will allow you to electronically design the PC board with traces and holes specified automatically for you. Once this is done, you can either send your layout information (known as a *Gerber File*) to a PC board manufacturing house, or you can plot it out yourself and have the PC board built by using the automated layout tool. The PC board's design is automatically saved and more boards can be built from it, instead of the single board of the original method.

Like schematic capture tools, PC board layout tools are application-specific graphic programs. Lines can be drawn on a board along with component hole patterns, but working with complex graphics or shading is not possible. The output options are quite limited with printer or plotter output of each PC board layer possible.

A basic PC board layout tool (such as Protel's EasyTrax, which is available for downloading over the Internet) provides the simple ability of placing a component and routing traces to it across the different layers. Figure 168 shows the PC board design for a resistor connected to a PICmicro® MCU.

In this example, the resistor connection is placed on the bottom layer of the PC board. Most of the applications I have presented in this book consist of double-layer PC boards. These boards consist of copper traces on the top and bottom of the boards with "vias" passing signals between them. Figure 161 shows a two layer with a via passing a trace from the bottom side to the board to the top side.

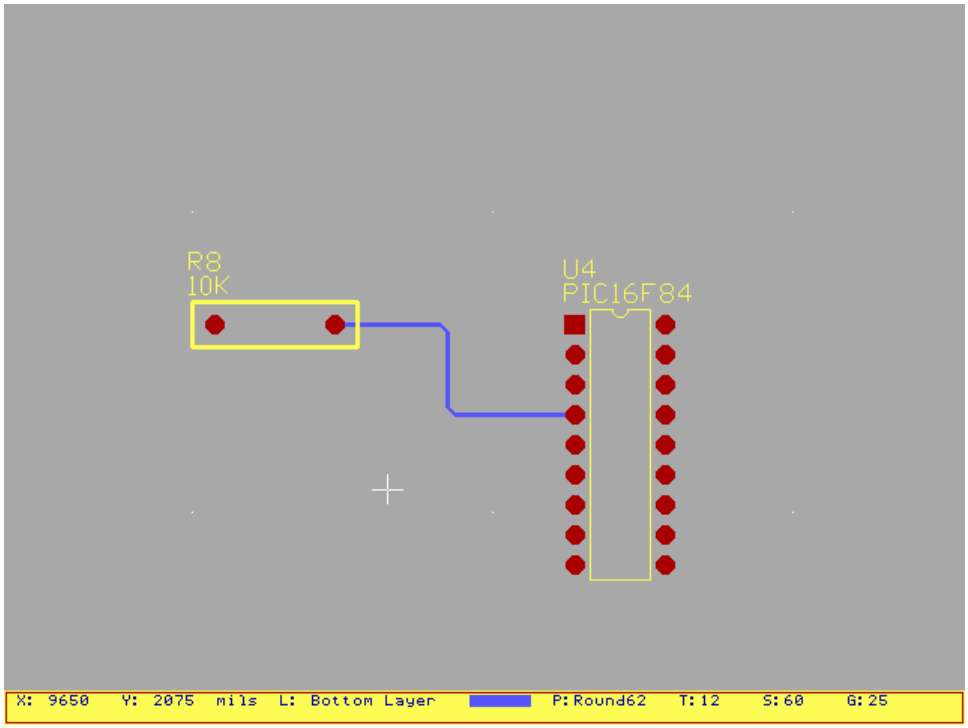


Figure 168 Wiring a resistor to a PICmicro® MCU

A *via* is a hole drilled through the card that is copper plated inside to pass signals through the card. Vias are also used for pin-through-hole parts to be soldered. The PC board layout tool must be able to differentiate between the top layer and the bottom layer (in EasyTrax, the top layer is red and the bottom layer is blue). Looking at the resistor/PICmicro® MCU example in Fig. 168, a via could be placed between their connections to allow other traces to pass by the trace (Fig. 169).

When you first start working a PC board layout tool, you'll probably discover that the connectors and many chips you want to use are not entered into the system. Although the information required for many parts is available on the Internet, many are not. This makes it important to have the capability to add you own parts to the library. After you have done a few designs, you will discover that you have built up a good set of connectors and other parts that you use all the time.

Traces can be put on the board in one of two ways. The first is manually, with parts and traces laid down by hand. This might seem time consuming, but I recommend starting this way because it will give you an idea of what is required for board design.

The second is to allow an "auto router" to lay out the board from the net list after you have specified the component placements. This process should be painful, but it isn't unusual to find out that an auto router requires more work on your part than wiring the card manually. Once the components are placed, you will probably find that the auto router cannot figure out how to put down individually traces. To fix this problem, you will have to go back and move components or specify the position of various tracks in an effort to solve the "impasses."

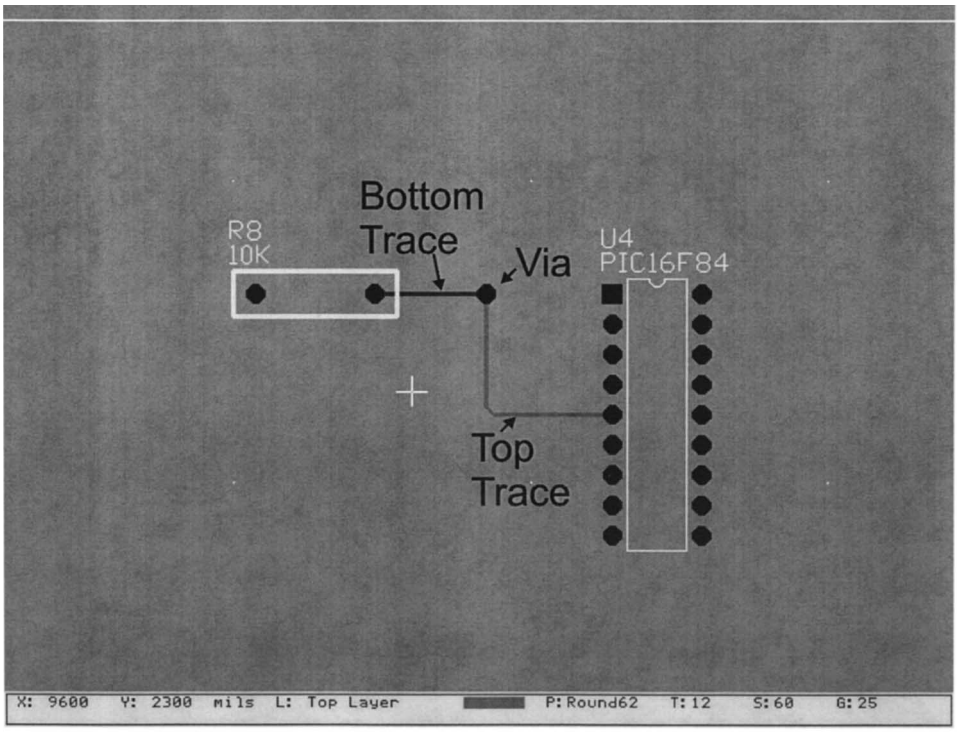


Figure 169 Changing layers using a via

These problems are exacerbated by the use of two-layer boards and the fact that most inexpensive auto routers are not very good. Very high end (license cost of $10,000 or more per year) tools do not have a lot of these problems, but layout tools with auto routers that cost less than $15,000 can be very difficult to work with.

After considerable searching, I have not been able to find an inexpensive auto router that works the way that I would want it to. I have always been tempted to write one for myself just to see how difficult it is and what are the issues for doing it.

When you first work with an auto router, you will probably find that it is very frustrating trying to learn the tool, properly specify component layouts, and then visualize how traces are placed on the board. When you persevere, you will have a new layout tool that will help you more efficiently create your own applications.

Remember to save temporary versions of your designs. Personally, I always stop, save, and copy to a new file at these steps:

1. Board dimensions specification.
2. Connector replacement.
3. Component placement.
4. Power trace wiring.
5. Connector wiring.
6. Circuit wiring.

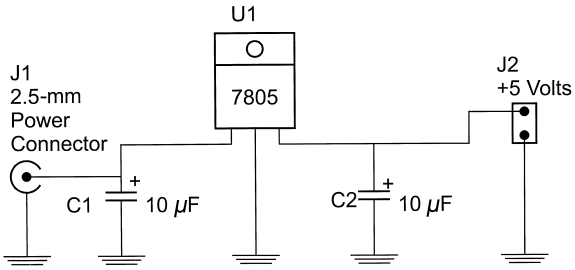


Figure 170 Simple circuit for demonstrating Gerber files

I have found that these points are optimal and allow me to go back and make changes without losing all of the prerequisite work.

Gerber files In many areas of electronics, early standards have had a way of staying around long after they make sense. When people talk about this, they usually point out something like RS-232, which was first developed in the 1950s and has somehow managed to stay in the forefront for almost 50 years. The methods used to specify printed circuit boards is probably a better example of this, where information required to build cards was first specified at the same time as RS-232. Unlike RS-232, which has been sped up and improved, the information specifications for printed circuit boards are still done exactly the same way as they were done when the data was transferred on paper tape rolls, rather than attached to e-mail.

The basic format for data transfer is the *Gerber file* and you will hear the term *Gerbers* when the design information of printed circuit boards is mentioned. *Gerbers* is actually a collective term, describing the design, aperture, drill, and tooling files used to build a PC board. To demonstrate how these files work, I have created the simple power-supply application shown in Fig. 170.

This is a pretty straightforward 7805-based power supply circuit. I spent a few moments and created the single-sided board shown in Fig. 171 using Protel's EasyEdit. The design is very simple, but it actually contains all the elements that are required for specifying a printed circuit board that can be built by a commercial company.

Once I finished with EasyEdit, I created the Gerber files using EasyPlot. From EasyPlot, I created a top-side overlay Gerber file, a bottom-side solder mask Gerber file, and a drill file. When these files were created, I had an aperture and tool file already made up to use with the application.

When PC boards were first manufactured, they used a tool called a *photoplotter*. This tool would expose PC boards that had been sprayed with a light-sensitive emulsion to light, according to the data in the Gerber files that specified how the card was designed.

The Gerber files consist of a series of commands for the photoplotter to move a head to and expose the light to the card or turn it off before moving. The Gerber file that EasyPlot produced for the backside of this simple power supply is given in Table 23.

I realize that this file seems very complex and almost impossible to translate back to Fig. 171, but it is actually very simple and easy to work through. The asterisks in the file are used to delineate the commands to the photoplotter. Taking the first line, I can break the commands up into:

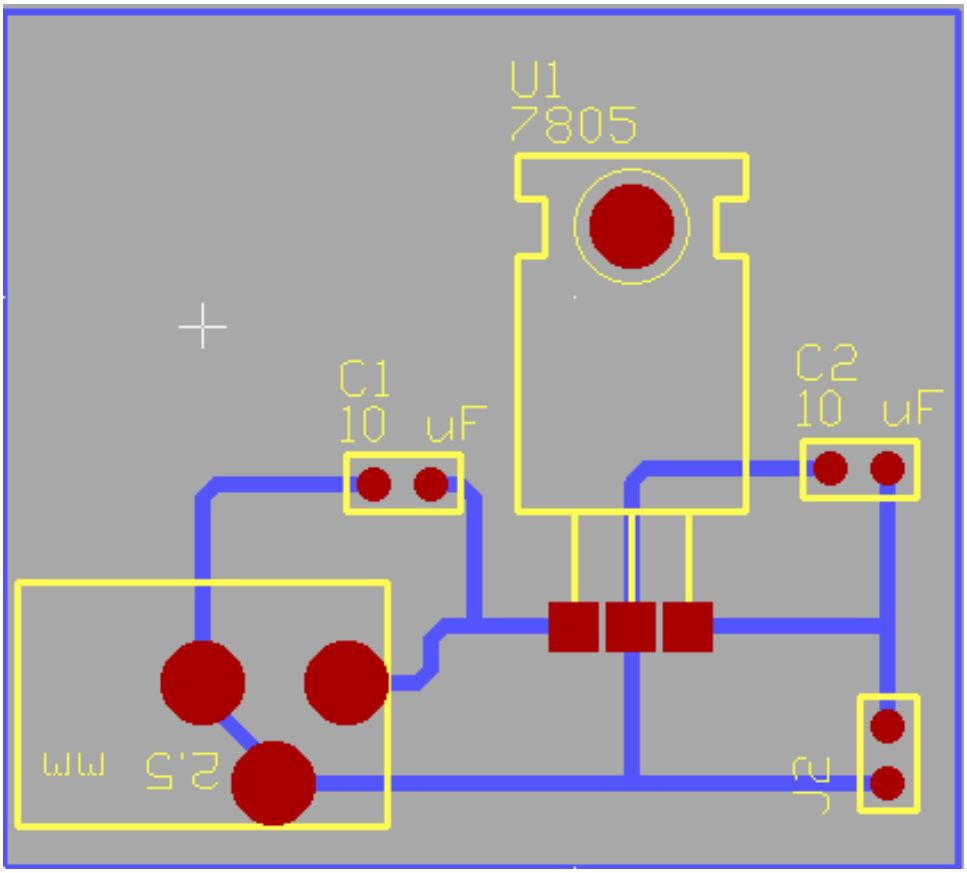


Figure 171 Layout of simple circuit

```
X0Y0
X0Y375
D13
X60Y185
D01
X60Y185
X80Y185
D02
X80Y125
D01
X80Y185
D02
```

which are used to instruct the photoplotter where to put its light over the card. The first instruction sends the photoplotter to the origin of the card, which is at its bottom left corner. Next, the photoplotter is instructed to move to X = 0, Y = 375. The 375 is actually in thousandths of an inch, so the photoplotter is being instructed to move its head to the coordinate 0″, 0.375″.

The next command is an aperture command. The original photoplotters had a wheel of different apertures that could have light passed through them to expose the printed circuit

TABLE 23 Example Circuit Gerber File

X0Y0*X0Y375*D13*X60Y185*D01*X60Y185*X80Y185*D02*X80Y125*D01*X80Y185*D02
*X20Y125*D01*X80Y125*D02*X0Y145*D01*X20Y125*D02*X0Y145*D01*X0Y225*D02*X
0Y225*D01*X20Y245*D02*X20Y245*D01*X80Y245*D02*X120Y125*D01*X120Y125*X12
0Y245*D02*X120Y245*D01*X200Y245*D02*X120Y245*D01*X200Y245*D02*X120Y185*
D01*X120Y245*D02*X120Y185*D01*X160Y185*D02*X120Y185*D01*X160Y185*D02*X1
20Y125*D01*X120Y185*D02*X120Y125*D01*X200Y125*D02*X240Y125*D01*X240Y125
*X240Y245*D02*X240Y245*D01*X300Y245*D02*X300Y245*D01*X320Y225*D02*X320Y
205*D01*X320Y225*D02*X300Y185*D01*X320Y205*D02*X240Y185*D01*X300Y185*D0
2*X240Y185*D01*X260Y185*D02*X260Y185*D01*X320Y125*D02*X360Y125*D01*X360
Y125*X420Y125*D02*X420Y125*D01*X440Y145*D02*X440Y145*D01*X440Y165*D02*X
420Y185*D01*X440Y165*D02*X380Y185*D01*X420Y185*D02*X380Y185*D01*X420Y18
5*D02*X420Y185*D01*X440Y205*D02*X440Y205*D01*X440Y225*D02*X420Y245*D01*
X440Y225*D02*X360Y245*D01*X420Y245*D02*X360Y245*D01*X380Y245*D02*X380Y1
25*D01*X380Y245*D02*X480Y125*D01*X480Y125*X480Y245*D02*X480Y245*D01*X56
0Y245*D02*X480Y245*D01*X560Y245*D02*X480Y185*D01*X480Y245*D02*X480Y185*
D01*X520Y185*D02*X480Y185*D01*X520Y185*D02*X480Y125*D01*X480Y185*D02*X4
80Y125*D01*X560Y125*D02*X600Y125*D01*X600Y125*X600Y245*D02*X600Y245*D01
*X660Y245*D02*X660Y245*D01*X680Y225*D02*X680Y205*D01*X680Y225*D02*X660Y
185*D01*X680Y205*D02*X600Y185*D01*X660Y185*D02*X600Y185*D01*X620Y185*D0
2*X620Y185*D01*X680Y125*D02*X840Y125*D01*X840Y125*X900Y125*D02*X900Y125
*D01*X920Y145*D02*X920Y145*D01*X920Y165*D02*X900Y185*D01*X920Y165*D02*X
860Y185*D01*X900Y185*D02*X860Y185*D01*X900Y185*D02*X900Y185*D01*X920Y20
5*D02*X920Y205*D01*X920Y225*D02*X900Y245*D01*X920Y225*D02*X840Y245*D01*
X900Y245*D02*X840Y245*D01*X860Y245*D02*X860Y125*D01*X860Y245*D02*X980Y1
25*D01*X980Y125*X1020Y125*D02*X960Y145*D01*X980Y125*D02*X960Y145*D01*X9
60Y185*D02*X960Y185*D01*X980Y205*D02*X980Y205*D01*X1020Y205*D02*X1020Y2
05*D01*X1040Y185*D02*X1040Y145*D01*X1040Y185*D02*X1020Y125*D01*X1040Y14
5*D02*X1080Y205*D01*X1080Y205*X1160Y205*D02*X1120Y145*D01*X1120Y145*X11
20Y245*D02*X1120Y145*D01*X1140Y125*D02*X1140Y125*D01*X1160Y145*D02*X120
0Y205*D01*X1200Y205*X1280Y205*D02*X1240Y145*D01*X1240Y145*X1240Y245*D02
*X1240Y145*D01*X1260Y125*D02*X1260Y125*D01*X1280Y145*D02*X1340Y125*D01*
X1340Y125*X1380Y125*D02*X1320Y145*D01*X1340Y125*D02*X1320Y145*D01*X1320
Y185*D02*X1320Y185*D01*X1340Y205*D02*X1340Y205*D01*X1380Y205*D02*X1380Y
205*D01*X1400Y185*D02*X1400Y145*D01*X1400Y185*D02*X1380Y125*D01*X1400Y1
45*D02*X1440Y125*D01*X1440Y125*X1440Y205*D02*X1440Y185*D01*X1440Y205*D0
2*X1440Y185*D01*X1460Y205*D02*X1460Y205*D01*X1480Y185*D02*X1480Y165*D01
*X1480Y185*D02*X1480Y165*D01*X1480Y185*D02*X1480Y185*D01*X1500Y205*D02*
X1500Y205*D01*X1520Y185*D02*X1520Y125*D01*X1520Y185*D02*X1680Y125*D01*X
1680Y125*X1680Y245*D02*X1680Y125*D01*X1680Y245*D02*X1680Y125*D01*X1760Y
125*D02*X1820Y125*D01*X1820Y125*X1840Y125*D02*X1800Y145*D01*X1820Y125*D
02*X1800Y145*D01*X1800Y185*D02*X1800Y185*D01*X1820Y205*D02*X1820Y205*D0
1*X1840Y205*D02*X1840Y205*D01*X1860Y185*D02*X1860Y145*D01*X1860Y185*D02
*X1840Y125*D01*X1860Y145*D02*X1840Y125*D01*X1860Y145*D02*X1860Y145*D01*
X1880Y125*D02*X1920Y145*D01*X1920Y145*X1920Y205*D02*X1920Y145*D01*X1940
Y125*D02*X1940Y125*D01*X1960Y125*D02*X1960Y125*D01*X2000Y165*D02*X2000Y
165*D01*X2000Y205*D02*X2000Y105*D01*X2000Y205*D02*X1980Y85*D01*X2000Y10
2100Y165*D01*X2120Y185*D02*X2100Y205*D01*X2120Y185*D02*X2060Y205*D01*X2
100Y205*D02*X2040Y185*D01*X2060Y205*D02*X2040Y145*D01*X2040Y185*D02*X20

TABLE 23 Example Circuit Gerber File

```
40Y145*D01*X2060Y125*D02*X2060Y125*D01*X2100Y125*D02*X2160Y125*D01*X216
0Y125*X2160Y205*D02*X2160Y165*D01*X2160Y205*D02*X2160Y165*D01*X2200Y205*
D02*X2200Y205*D01*X2220Y205*D02*X2220Y205*D01*X2240Y185*D14*D02*X1675Y
375*D01*X1675Y375*X1675Y1875*D02*X1675Y1875*D01*X1675Y1875*X0Y1875*D02*
X0Y1875*D01*X0Y1875*X0Y375*D02*X0Y375*D01*X0Y375*X1675Y375*D19*D02*X155
0Y525*D01*X1550Y525*X1100Y525*D02*X1100Y525*D01*X1100Y525*X1100Y550*D02
*X1100Y550*D01*X1100Y550*X1100Y800*D02*X1100Y800*D01*X1100Y800*X1100Y10
50*D02*X1100Y1050*D01*X1100Y1050*X1125Y1075*D02*X1125Y1075*D01*X1125Y10
75*X1450Y1075*D02*X1550Y1075*D01*X1550Y1075*X1550Y625*D02*X1550Y800*D01
*X1550Y800*X1200Y800*D02*X1000Y800*D01*X1000Y800*X775Y800*D02*X775Y800*
D01*X775Y800*X750Y775*D02*X750Y775*D01*X750Y775*X750Y725*D02*X750Y725*D
01*X750Y725*X725Y700*D02*X725Y700*D01*X725Y700*X600Y700*D02*X500Y525*D0
1*X500Y525*X350Y675*D02*X350Y675*D01*X350Y675*X350Y700*D02*X350Y700*D01
*X350Y700*X350Y1025*D02*X350Y1025*D01*X350Y1025*X375Y1050*D02*X375Y1050
*D01*X375Y1050*X650Y1050*D02*X750Y1050*D01*X750Y1050*X800Y1050*D02*X800
Y1050*D01*X800Y1050*X825Y1025*D02*X825Y1025*D01*X825Y1025*X825Y800*D02*
X1075Y525*D01*X1075Y525*X500Y525*D02*X1075Y525*D01*X1075Y525*X1100Y525*
D22*D02*X1088Y811*D01*X1088Y811*X1088Y788*X1111Y788*X1111Y811*X1088Y811
*X1100Y800*D02*X1011Y811*D01*X1011Y811*X988Y811*X988Y788*X1011Y788*X101
1Y811*X1000Y800*D02*X1188Y811*D01*X1188Y811*X1188Y788*X1211Y788*X1211Y8
11*X1188Y811*X1200Y800*D23*D02*X1450Y1075*D03*X1550Y1075*D03*X1550Y625*
D03*X1550Y525*D03*X750Y1050*D03*X650Y1050*D03*D30*X600Y700*D03*X475Y525
*D03*X350Y700*D03*X1100Y1500*D03*D02*M02*
```

card with varying shapes and sizes of light. The *D##* command is used to instruct the photoplotter what to do with the operation. The basic aperture commands are:

```
D01 - Light On for the Current Aperture
D02 - Light Off
D03 - Flash the Light On and Off
```

These are coupled with the different apertures available to the photoplotter. The list of apertures that I typically use are stored in the file:

```
V115 Aperture File
;
; Myke Predko
;
; 96.11.11
;
;
; <draft code>  Code assigned to this aperture e.g. D14
;               Note that this code must start with 'D'
;               followed by the code number.
;
; <shape>       Can be any of the following shapes:
;                 CIRCULAR
;                 RECTANGULAR
;                 SQUARE
;                 OCTAGONAL
```

```
;                ROUNDRECT
;                CROSSHAIR
;                MOIRE
;
; <xsize>         size of aperture in the x direction in
;                mils (thous).
;
; <ysize>         size of aperture in the y direction in
;                mils (thous).
;
; <hole size>    size of hole in aperture in mils (thous).
;                zero if no hole in aperture.
;
; <use>          specifies what the aperture can be used for.
;                There are three possible settings:
;                 LINE can only be used to draw lines
;                 FLASH can only be used to flash pads
;                 MULTI can be used for either
;                 blank defaults to MULTI setting
;
D11 CIRCULAR     40  40  0
D12 SQUARE       10  10  0
D13 CIRCULAR     10  10  0
D14 CIRCULAR     12  12  0
D15 CIRCULAR     15  15  0
D16 SQUARE       20  20  0
D17 CIRCULAR     20  20  0
D18 CIRCULAR     25  25  0
D19 CIRCULAR     30  30  0
D20 SQUARE       50  50  0
D21 CIRCULAR     50  50  0
D22 SQUARE       62  62  0
D23 CIRCULAR     62  62  0
D24 CIRCULAR     70  70  0
D25 CIRCULAR     75  75  0
D26 CIRCULAR     85  85  0
D27 CIRCULAR    100 100  0
D28 CIRCULAR    110 110  0
D29 CIRCULAR    125 125  0
D30 CIRCULAR    150 150  0
D31 CIRCULAR    200 200  0
D32 CIRCULAR    250 250  0
D33 CIRCULAR     40  40 10
```

For each aperture, a shape and size is specified, along with an optional card hole size. When these apertures are selected, the resulting line drawn by the photoplotter using the commands from the Gerber is used to make the circuits on the card. Along with the aperture commands (D codes), there are also G (Go codes) and M (Machine codes) commands. Other than the D codes, the only other command you should be concerned with is the M02 command, which indicates the end of the Gerber plot.

Using the aperture information, the first line can be decoded as in Table 24.

This is all there is to Gerber files used to command the traces built into a printed circuit board. Photoplotters are no longer used in modern board manufacturing, but the Gerber files are still used to make up photographic "films" that are used to control the exposure of light to a photosensitive printed circuit board so that the circuits can be etched onto them.

When a printed circuit board is specified, a Gerber file (and, optionally, a specific aperture file) for each copper layer of the board, any solder masks and the silkscreen layers of

TABLE 24 Decoding of Gerber File Actions

COMMAND	ACTION
X0Y0	Move Photoplotter head to Origin
X0Y375	Move Photoplotter head to X=0″, Y=0.275″
D13	Select Aperture D13 (Circle, 0.010″ in Diameter)
X60Y185	Move head to X=0.060″, Y=0.185″
D01	Turn on Light
X60Y185	Move Head to X=0.060″, Y=0.185″
X80Y185	Move Head to X=0.060″, Y=0.185″
D02	Turn off Light
X80Y125	Move Head to X=0.080″, Y=0.125″
D01	Turn on Light
X80Y185	Move Head to X=0.080″, Y=0.185″
D02	Turn off Light

the board. For a traditional two-sided board with top-side silk-screening, a minimum of five Gerber files have to be sent to the board manufacturer.

Along with the Gerber files, the drill information has to be created as well. This is normally created by the design system that creates the Gerber files and uses a similar data format. For this power supply card, EasyPlot produced the drill file shown in Table 25.

The same way the Gerber file needs an aperture file, drill files need a tool file. In the drill file in Table 25, the *T02*, *T03*, and *T06* commands are for different drill sizes. The tool file that goes along with the drill file is:

```
Drill Sizes

This file contains the references to the Drill File (T105A.DRL).

T02 - #65 Drill (0.0350") - For PTH
T03 - #53 Drill (0.0595") - For Berg Pins/Connectors
T04 - #44 Drill (0.0860") - For TO-220 Mounting
T06 - 1/8" Drill (0.125") - For Tooling Holes/DB9F Holes
```

Once the files have been created, it is always a good idea to check them to be sure that they are valid. I use GCPREVUE (a free tool that can be downloaded from the Internet). Figure 172 shows the bottom-side solder pattern for the board displayed in this section.

At the same time as I was proofreading the manuscript for this book, I discovered ViewMate by Lavenir. This is a Microsoft Windows-based application program that can be downloaded free of charge from **http://www.lavenir.com**.

This application is somewhat more user friendly and follows standard Windows conventions.

TABLE 25 Example Gerber Drill File

```
M48
T02F00S00
T03F00S00
T06F00S00
%
T02
X0075Y00675
X0065
X0145Y007
X0155
Y0025
Y0015
T03
X011Y00425
X012
X01
T06
X006Y00325
X0035
X00475Y0015
X011Y01125
M30
```

Soldering

Arguably, the most basic skill required for working with electronics is the ability to solder components, wires, and PC boards together. Soldering experts have spent literally years understanding heat flows and chemical operations. This section introduces hand soldering and the issues associated with it.

Soldering is the joining of two metal parts by melting a metal material (known as *solder*) between them so that the atoms of the various metals combine, forming a mechanically strong, low-resistance electrical bond (joint). This operation (Fig. 173) shows how the two metal parts are linked by the solder.

The most common type of solder used today is a tin-lead mixture, with silver optionally added to it. Solder is designed to melt at a temperature far below that of the parts (normally

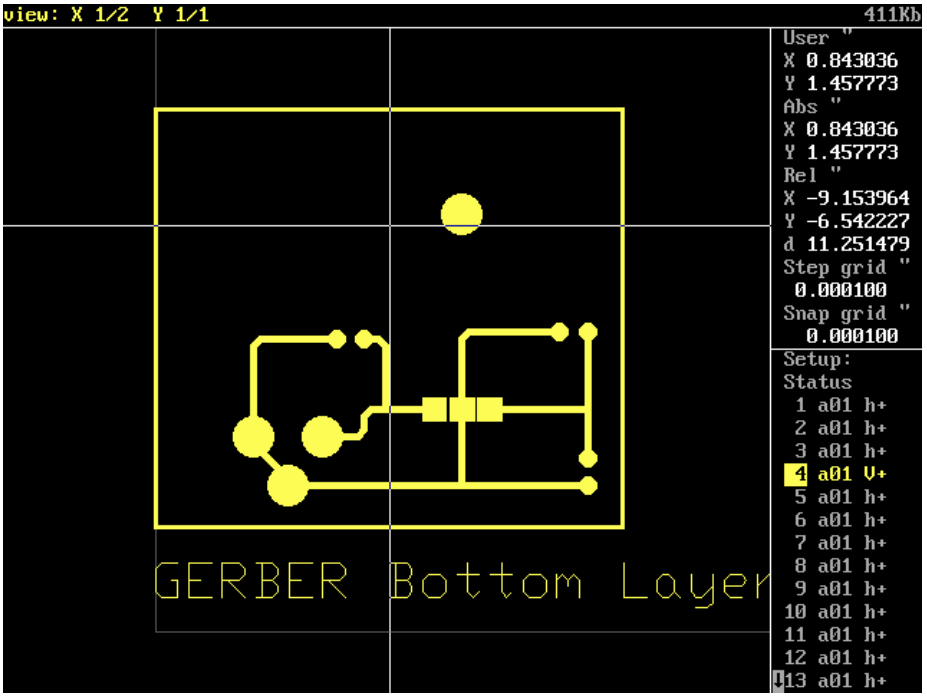


Figure 172 Simple circuit Gerber check

copper) that it will connect. The lower melting temperature of solder is to ensure that the metal parts to be joined are not deformed and any devices connected to them (such as PICmicro® MCU's) are not damaged.

When solder is applied to a metal, an *intermetallic* layer is formed, which is an alloy of the solder and the metal part. Ideally, this intermetallic layer should be as thin as possible to allow easy re-melting of the solder for reworking and for the best electrical characteristics.

With this in mind, I recommend that you buy a temperature-controlled soldering station, which can be purchased for as little as $50. Having a temperature-controlled station will give you the control to solder high-quality joints in a way that works best for you.

Along with the soldering station, you should have as small a tip as possible, a "solder sponge," and rosin-core solder. With these tools, you will be able to solder the circuits shown in this book and most other electronic circuits.

Plumbing soldering irons, although performing the same function, are not appropriate because they provide too much heat. It might sound ridiculous, but I have been approached

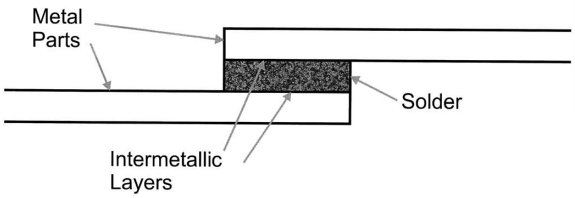


Figure 173 Solder joint features

by quite a few people asking for help when they've burned a hole through a PC board with an iron that worked fine on a 3″ copper drain pipe the week before.

The tip should be kept clean and shiny by wiping it on the damp solder sponge. Placing the solder against the hot tip should cause it to melt, give off some smoke, and flow over the tip. If there are parts of the tip where solder doesn't flow to, then replace the tip. Never file down the tip. If it has lost its shininess, then replace it. The tip is specially coated. When this shininess (the tin coating) is gone, the tip is useless.

To solder two pieces of metal together, hold the hot tip against both pieces of metal to heat them both up, wait a couple of seconds, and then touch the solder to one of the metal surfaces (Fig. 174).

The solder should flow between the two pieces of metal and there should be a curl of smoke. The smoke is the flux built in the solder. *Flux* is a weak heat-activated acid that cleans away any oxides on the metal surfaces before the solder melts to it.

Plumbing solder, like plumbing soldering irons, is not appropriate to use. The flux built into plumbing solder is much more aggressive and will literally eat away at electronic circuits. In case you haven't guessed it, no plumbing materials should be used in electronic assembly.

After the solder has flowed, remove the solder and the iron. After a few moments, the solder should harden into a smooth, shiny surface with a negative fillet. If it doesn't, reapply the soldering iron and try again. Good solder joints are shown in Fig. 175.

Bad soldering joints have a dull "crinkly" finish and are known as *cold solder joints*.

When you first start soldering, expect to make a mess and potentially damage (burn) a few joints. With a bit of experimentation with the heat control of your soldering iron, you will be soldering like a pro in no time.

Notice that although soldering is not a terribly dangerous operation, there are a few things to watch out for. First, the iron is hot. Test it by melting solder on it or making the damp solder sponge sizzle. Do not, like a former employee of mine, hold it up to your lips to see if it is hot.

Soldering is a manufacturing process that involves some moderately dangerous chemicals. Work in a well-ventilated room and don't smoke while soldering (hot cigarette smoke and lead fumes generate cyanide). Keep the solder away from children and wash your hands after soldering.

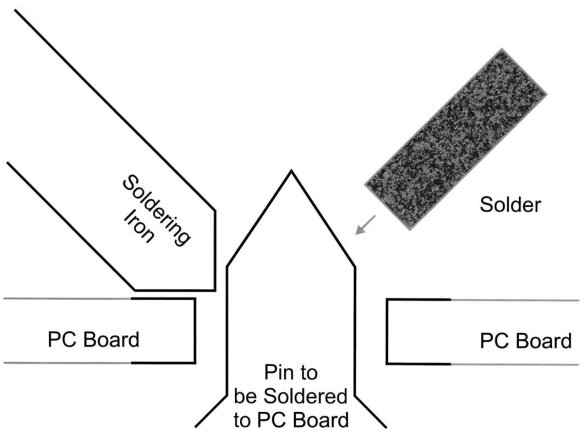


Figure 174 Creating a solder joint

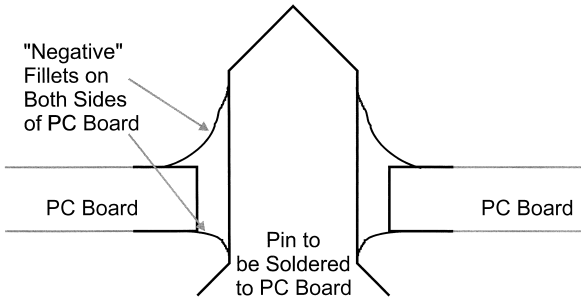


Figure 175 Good solder joints

I can summarize soldering with three simple rules:

1 Shiny is good.
2 Plumbing is bad.
3 Keep everything clean.

Project Assembly Techniques and Prototyping

In this book contains well over 50 different hardware experiments, tools, and projects that you can build for yourself. These applications all required to be wired on something. Deciding on what is the appropriate method of building the circuits will have some pretty profound implications on the time required to build the application, its cost, and whether to not the circuit will work.

Most applications are fairly low-speed (frequency) circuits and to maximize the reliability of the circuits, I have kept the clock frequencies and data rates as slow as possible. As you get into higher speeds, different techniques will alter how signals actually propagate in the circuit.

When I've drawn logic waveforms, I have shown the signals as idealized waveforms (Fig. 176).

If you were to look at an actual signal on a line, chances are you would see something like the oscilloscope picture in Fig. 177.

You can see that the edges are actually rounded and that ringing is in the middle of the signal. This rounding and ringing is because of the in-line resistance, capacitance, and inductance of the actual wiring between components, along with the component's characteristics. To minimize these effects, the obvious solution is to slow down the circuit speeds as much as possible so that the edge rounding and level rise time becomes insignificant.

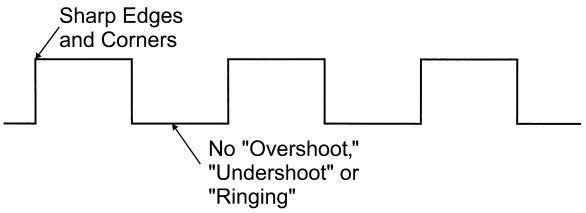


Figure 176 Idealized clock signal

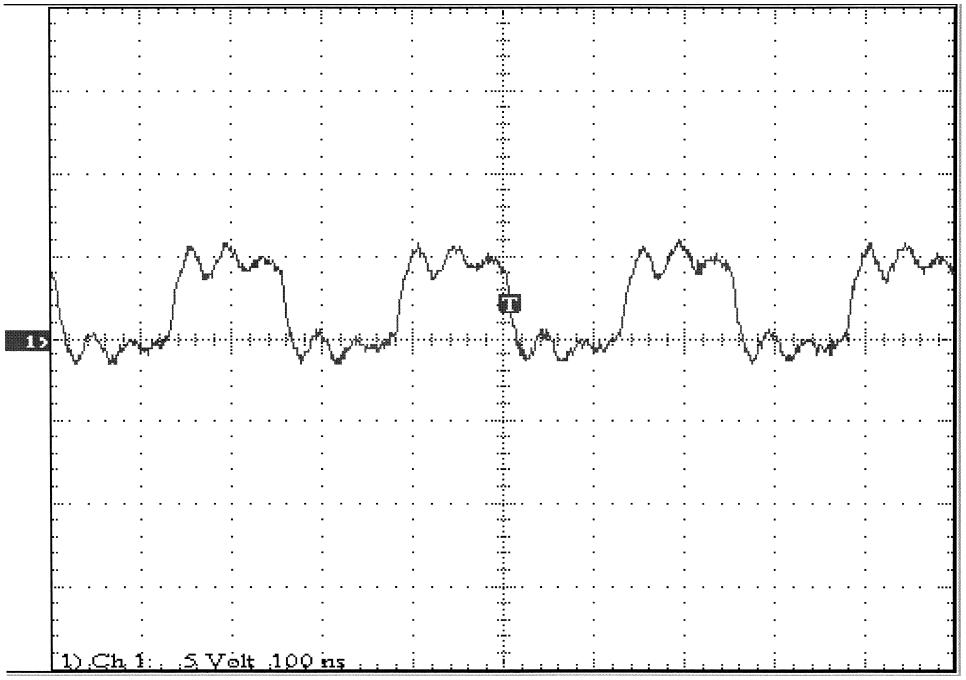


Figure 177 Actual clock signal

As I review the different methods of building circuits, I'll describe these issues, along with the out-of-pocket cost, reliability, and time spent using each method.

BREADBOARDS

Probably the easiest way to create test applications is to use a *breadboard*. This product consists of spring-loaded holes, which will grip a component and provide connections to other holes so that circuits can be built out of connected components. In this book, I have designed the experiments and many of the products to use breadboards for simple build up and tear down of PICmicro® MCU circuits (Fig. 178).

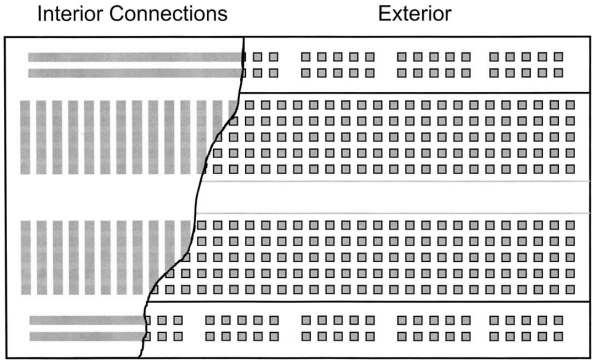


Figure 178 Breadboard with interior connections shown

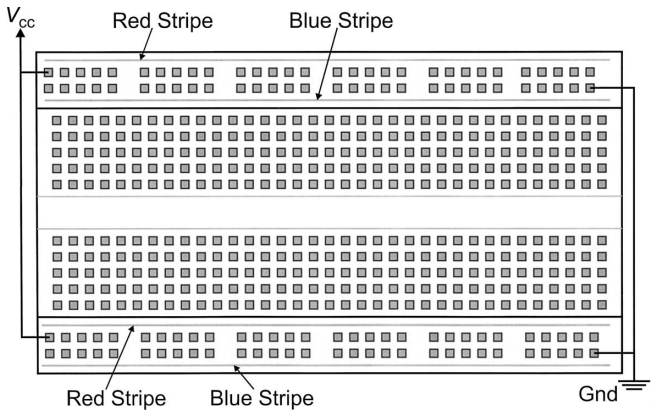


Figure 179 Breadboard power connection

The breadboard consists of two sets of connected pins for the DIP packages. Each row of pins along with two "columns" across the top and bottom are interconnected (Fig. 179).

To make power distribution easier, most breadboards are shipped with bus bars across the top and bottom. These rails connect using plastic tabs and receptacles and lock the parts together. Multiple rails and breadboards can be connected together to create different sizes of prototype assembly areas.

The rails are available with one or two "common strips." I normally use the ones with two rows and place V_{cc} and Gnd on both sides of the breadboard. These rails might be married with red and black (or blue) stripes to indicate that the pins are common across them. Figure 180 shows how this is done.

If you look at the example application +5-volt power-supply circuit (described earlier in this appendix), you'll see that I've included two rows of IDE 0.100″ connectors. This allows the power supply to be attached to the breadboard without any wiring by plugging it into the common strips to provide power to the breadboard and application.

Wiring circuits is accomplished by selecting rows of pins and connecting them together.

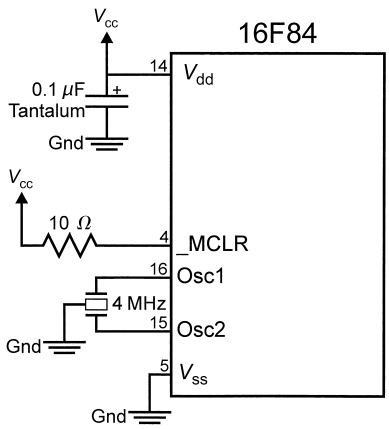


Figure 180 16F84 core circuit

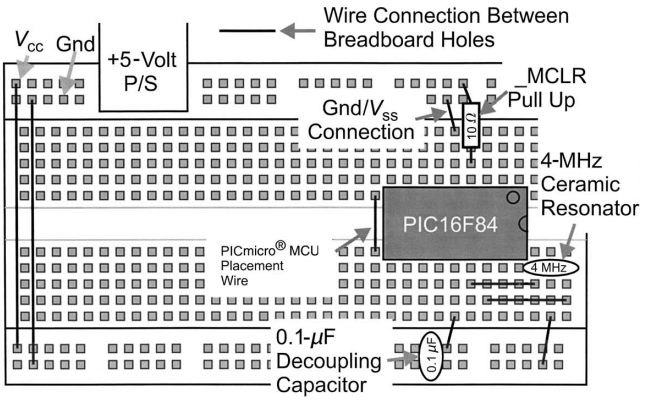


Figure 181 16F84 breadboard core circuit

For the PICmicro® MCU example circuit that is used for experiments, I use the circuit shown in Fig. 181.

This can be wired into a breadboard (Fig. 182).

This layout looks quite simple, but there are quite a few things to notice. First, notice that I placed the 16F84 to the edge of the circuit with pin 1 facing outwards and a "place-mat wire" across the first rows of pins past the end of the chip. When a breadboard is used, I expect to pull the 16F84 out for reprogramming by prying it out and slipping a small screwdriver underneath it. The wire is used to locate the PICmicro® MCU when I have re-programmed it and I'm putting it back.

Notice that I placed the ceramic resonator "ahead" of the PICmicro® MCU. This allows the RA0 and the RA1 I/O pins to be connected elsewhere on the breadboard. For example, if two LEDs were added to the circuit, the core circuit diagram looks like Fig. 183.

The breadboard layout would become that shown in Fig. 184.

This circuit is quite simple, but as the wiring becomes more complex, the difficulty in making the accurate connections becomes more difficult. Fairly large gauge wire is used

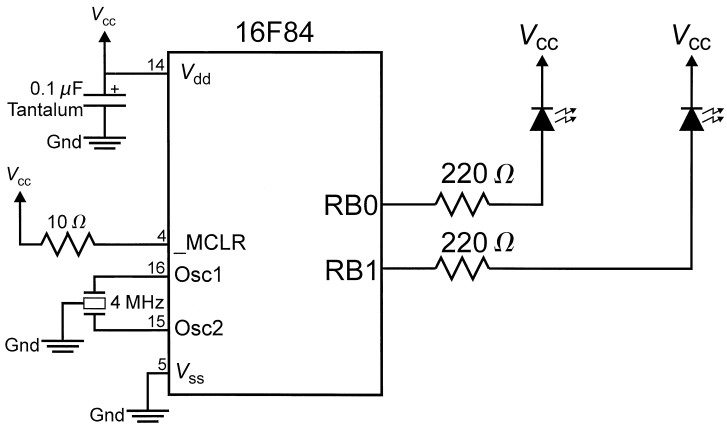


Figure 182 Two LEDs added to the 16F84 core circuit

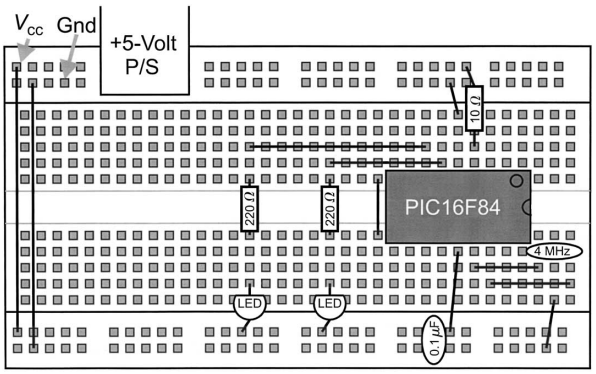


Figure 183 16F84 breadboard core circuit with LEDs

for the connections (18- to 20-gauge solid-core wire). Because it must be insulated, more than two or three wires in a run can look messy and difficult to trace between points on the breadboard.

This messiness can be alleviated somewhat by planning the layout of your application to keep the number of wires in an area to a minimum. By choosing different color wires, signals also can be traced easier.

Of the different project prototyping methods I presented here, breadboarding is the easiest to create circuits on. It is also the method that can have the greatest problems with noise, parasitic capacitances, and crosstalk because of the large connections in close proximity of each other. For this reason, I try to keep the clock speed of PICmicro® MCU projects built on breadboards to no more than 4 MHz. I have run circuits on breadboards at higher speeds, but 4 MHz is a good clock speed in which the signals will be just about perfect.

Figure 184 Hand tools used for prototyping

The tools required for breadboard (and other types of) prototyping include clippers, wirestrippers, screw drivers and so on, as shown in Figure 184.

WIRE WRAPPING AND POINT-TO-POINT WIRING

When I'm talking to people about things I've had to replace over the years, I usually focus on how many PCs I've owned. I should probably describe the number of wirewrap tools I've owned. For some reason, I have gone through 50 or more wirewrap tools in the last 20 years. This is in contrast to my still having the same pliers and clippers that I bought when I was a teenager. I tend to wear out or jam wire wrap tools at a rate of two or three a year, probably because I use them so often when building prototype circuits.

I tend to primarily wirewrap my initial prototypes or, if they are very simple, use point-to-point wiring. I find that using these methods produce a reasonably robust circuit that can be fairly easily modified and enhanced.

The basic tools needed for wirewrapping and providing point-to-point wiring are a wirewrap tool, a wire stripper (which is often integrated into the wire wrap tool), 32- to 28-gauge wire and sockets. Reasonable-quality wirewrap tools and wire can be purchased from "Radio Shack" for just a few dollars.

Motor-driven wire-wrap tools and precut and stripped wire lengths are available from many vendors. Although this increases your efficiency, it also dramatically increases your costs. If you are starting out, I recommend that you buy as cheap tools as possible and try to determine whether or not wirewrapping is something you want to do a lot of.

As the name implies, *wirewrapping* consists of wrapping a wire around a post to make an electrical connection. Figure 186 shows a sample wirewrap joint.

To make the joint, 3/4″ to 1″ of wire is stripped and placed in a wirewrap tool. The tool itself is placed over the pin to the connected and turned until all the bare wire has been wrapped around the pins post. Figure 187 shows the wirewrapping operation in more detail.

The wirewrap tool's wire hole is normally designed for placing a short length of insulated wire, so there is no chance of adjacent pins being shorted together. The wirewrap socket has very long pins leading from them to which the wire can be soldered. The post should have a square cross section because this makes a better connection with the wrapped wire.

Some unscrupulous dealers will try to sell you sockets with round posts. These sockets will not work as well as square ones. Chances are that the wire will slide right off them. Square wirewrap sockets are the most expensive sockets, but if you scrimp on this area, you will pay later.

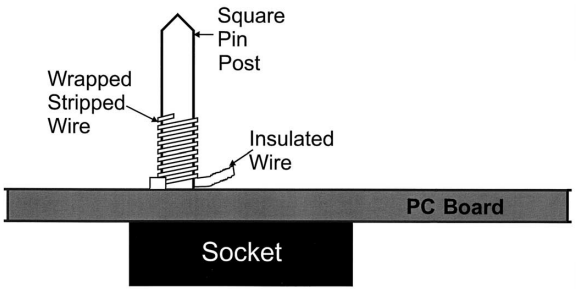


Figure 185 Wirewrap electrical joint

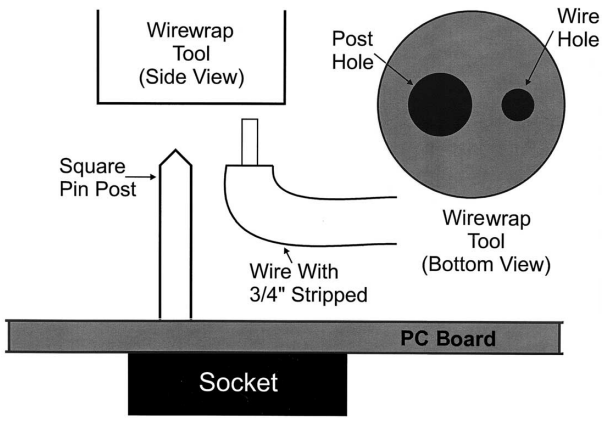


Figure 186 Tools for making wirewrap joints

Some sockets have decoupling capacitors built in. These capacitors are wired to the top right and bottom left corners, this wiring arrangement makes them unsuitable for PICmicro® MCUs, which do not have power pins in these two corners.

When circuits are to be wirewrapped, some time should be spent determining where the sockets are to go. With a little foresight, you can make a big difference in how much work is required to wire the board.

The net list produced by the schematic capture tool is used for listing out how the wiring is to be done. For example, from a schematic capture tool, you might have the signals:

```
Bit0
U1.6
U2.11
U5.7
```

These pins are wired together in one continuous string, not in a kind of triangular shape, where each pin is wired to the other two. Just one path should be used at all times to avoid inductive loops that pick up "noise."

Be aware of two other points. First, wire the pins in the order that is given in the net list. In many cases, this will be extremely inefficient in terms of wiring (which is why I suggested that you think about how the chips are arranged).

Second, the multiple pins are wired in a *daisy chain*. Figure 188 I shows two methods of wiring daisy chains. The first is probably the most obvious way to do it, but it can be a problem when you have to unwrap a wire. The second (right) *stacked wiring method* is recommended to ease reworking the circuit. In the stacked wiring method, if a net is to be rewired, then a maximum of three wires are changed for each pin to avoid leaving a wrapped pin or having to unwrap all of the pins.

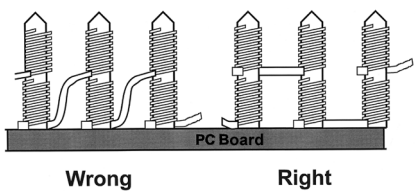


Figure 187 Daisy-chaining wirewrap joints

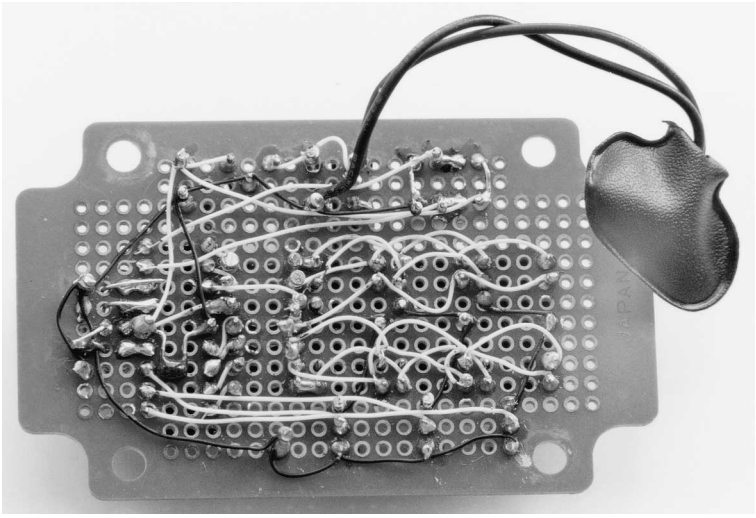


Figure 188 Backside of digital thermometer

In any case, no more than two wires should be wrapped onto each post. The pictures show fairly neat wraps with straight wires. With a bit of practice, you can be doing this as well. The problem that trips most new wirewrappers up is keeping track of the pins; it is important to remember that you are working with the mirror image of the pins and they are essentially reversed from how they look on the top side. You can buy plastic templates that fit over the pins before wrapping to help you keep the wires straight.

At an electronics store, you will find that along with simple wirewrapping tools (which will often look like a pen or a block of aluminum with black anodized steel posts pressed into it), "hand" and electrically powered wirewrap tools will be available. These tools will have pistol grips and cost $200 (or more). These tools will make the work quite a bit easier, but before investing in them, you should decide how much you want to wirewrap. Even if an electric tool makes it 100 times easier, wirewrapping is still an unreasonable amount of work.

Wirewrapping individual pins is quite easy, but it is easy to get bogged down in large projects. For example, the YAP-II has about 140 nets. If you assume that each net will require 1.5 wires and each wire requires one and a half to two minutes to wrap, the total time required is somewhere between five and a half and seven hours of effort.

This probably seems like a long time, but it is actually a pretty good estimate. Professional wirewrappers quote a half-minute per wire, but they tend to have literally years of experience. As well, discrete parts (resistors, capacitors, and transistors) require a lot more time than DIP pins to place on a board and wire.

In contrast, getting a quick-turn PC board designed and built will be much cheaper in terms of time and possibly out-of-pocket expenses for sockets, special connectors, and carriers for discrete parts. Having a quick-turn PC board could be less expensive. On the plus side, the design can be easily replicated or modified.

Point-to-point wiring is a particularly onerous method of wiring prototype circuits. Instead of wrapping pins with very thin wire, the pins are soldered to the wire directly. Al-

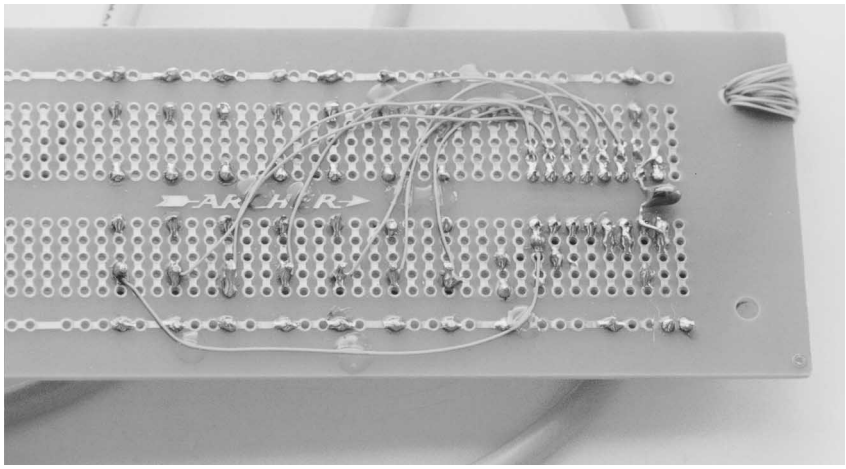


Figure 189 Backside of PIC12C673 fan interface

though I seem to be the only person dumb enough to use this method of circuit prototyping, useful skills can be developed when you have to modify or debug a circuit.

The problem with point-to-point wiring is that this "fiddly" work is not always reliable. Many projects, such as the digital thermometer (Fig. 189), were put together using point-to-point wiring. I personally find point-to-point wiring to be about as fast as wirewrapping, but it avoids the need for keeping wirewrap sockets on hand and the problem of trying to Figure out how to handle discrete components.

Point-to-point wiring does make sense in cases like the PIC12C673 fan controller, where only a few wires were required (Fig. 190). Note that in this circuit, I used "Radio Shack" prototyping cards with etched copper busses to make the wiring easier.

When doing point-to-point wiring, I always try to look for opportunities where I can solder a wire to a trace without having to solder it to a pin (which is much more difficult). This skill is important for reworking PC boards: by baring a cut trace, a wire can be added to another net. *Reworking* a trace is most easily accomplished by first scraping off any solder mask left on the board. Next, the trace is cut in two places using an X-Acto knife and then rubbing a hot soldering iron to remove the trace (Fig. 191). The cuts are best made

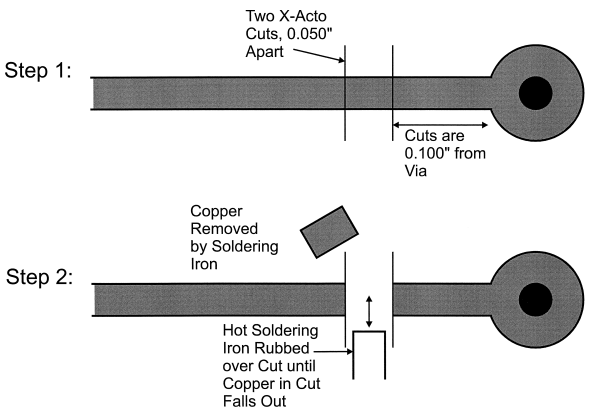


Figure 190 Cutting a PC board trace

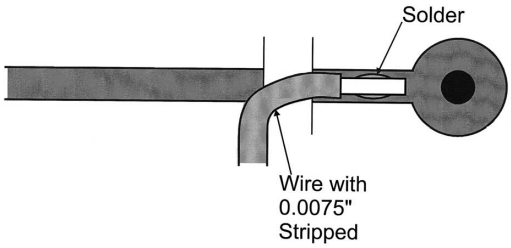


Figure 191 Adding a wire to a cut trace

close to a pin or a via on the PC board, but not so close that there is no trace left to solder a wire to it. After the cuts are made, the wire is soldered to the trace by the cut (Fig. 192).

Notice that I cut both ends of a trace. This is done to avoid long stubs, which can cause problems with the operation of the circuit because of reflections from the open line. Doing the second cut usually just takes seconds and avoids any potential problems on the net.

Being able to reliably rework boards using point-to-point soldering techniques comes in very useful with quick-turn prototypes (Fig. 193).

The two reworks shown in Fig. 193 were done before the board was assembled. In these cases, I was able to wrap the wires around a pin or insert them into a via, which made the job of adding the wires much easier. If I had to add the wires after the board was assembled and the components wired, then I would have used the methods described in this section.

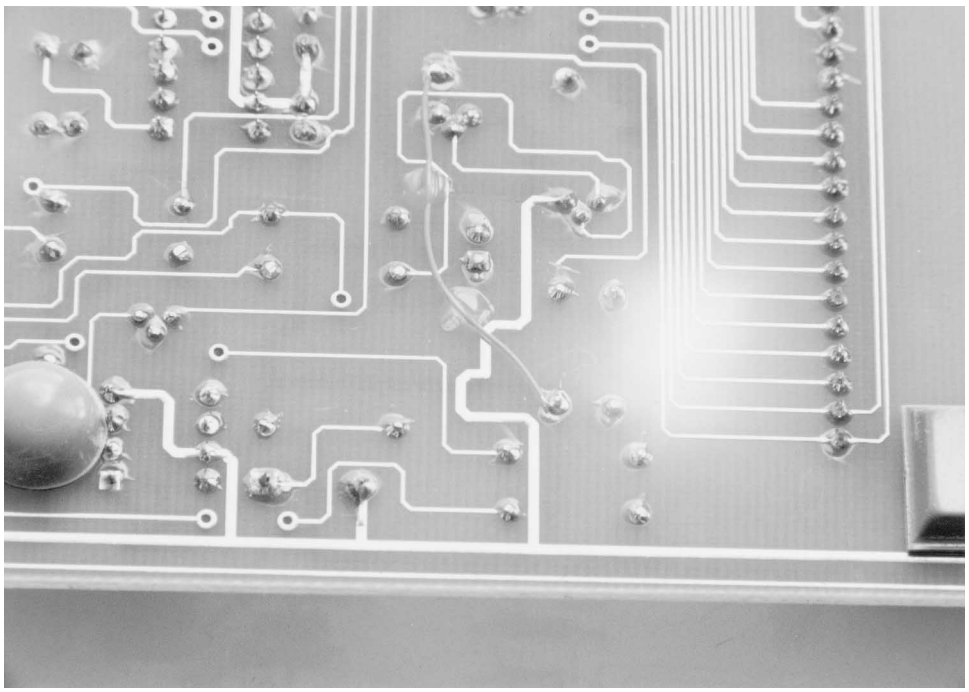


Figure 192 Fast turn prototype board with traces cut and new wire "net" added

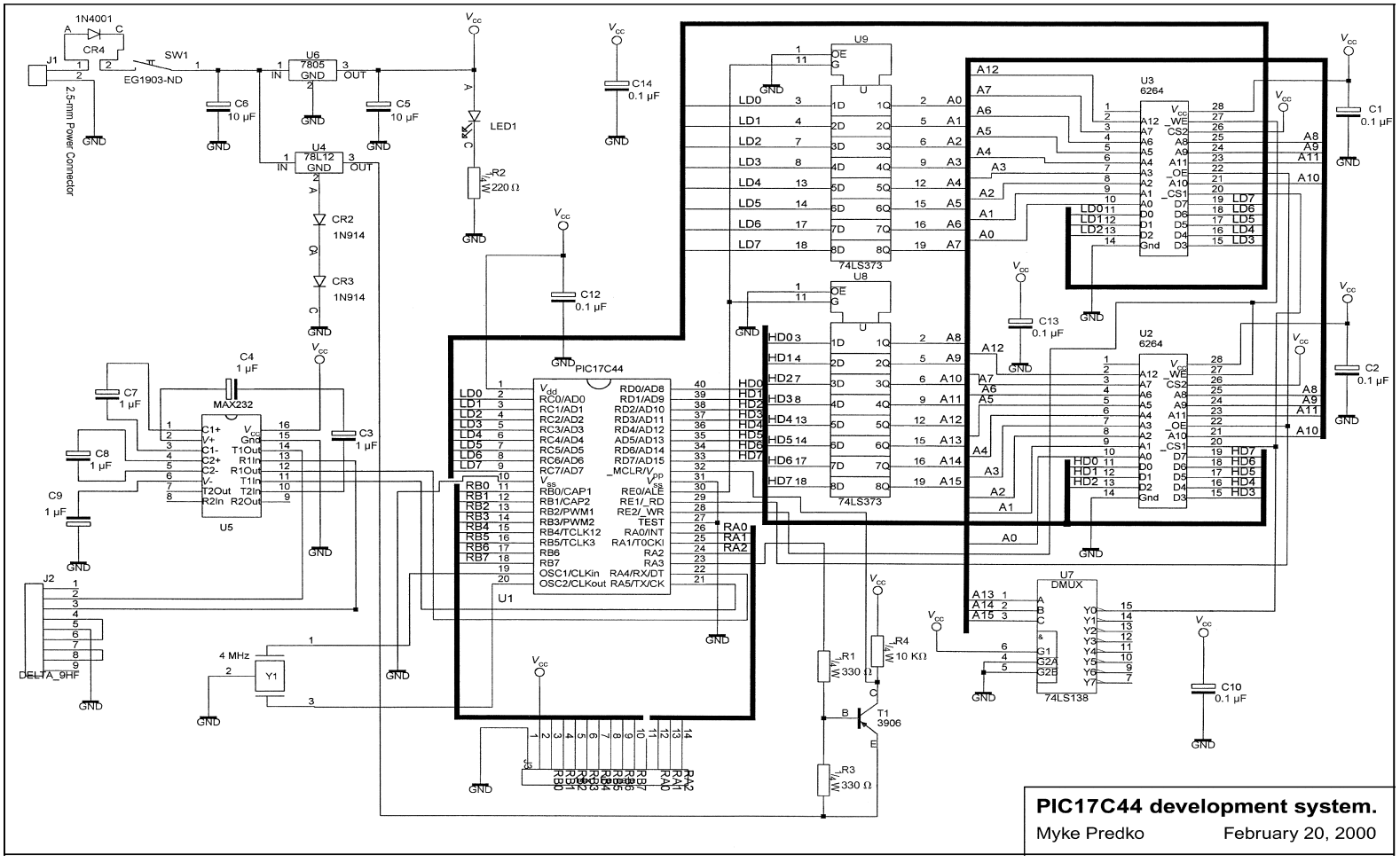


Figure 193 Pic17C44 development system

Embedded PC Boards

It probably seems unusual to say that an embedded PC board could be used as a prototyping circuit method, but it can be a very cost and time effective way to build your initial circuits. As shown in this section, you can use a few tricks to create PC boards for your projects that take a similar amount of time and costs than more traditional methods. The result is circuits that are much easier to assemble and debug.

As I work through this section, I show how I might design a prototype card and what sorts of features I would design into it to make it suitable for prototyping and then as a production part. The schematic capture and layout tools that I used are Ultimate's ULTICap and Protel's EasyTrax, respectively. EasyTrax is available for free of charge for downloading off the Internet. For the circuit, I want to create a relatively complex circuit in which I can connect two CMOS SRAM memory chips to a PIC17C44 with an RS-232 interface. Along with this, I want to have the ability to try out PIC17Cxx ICSP so that I have added a reset control circuit on the card as well. The goal of the exercise is to come up with a prototype PC board for this application that will allow me to experiment with external SRAMs or other peripheral devices.

The circuit that I started with is shown in Fig. 194. The reason why I say "started with" is because as I laid the circuit out onto a PC board and made a few changes to the original circuit to simplify the wiring that I had to do. As I work through the PC board development, I describe these changes.

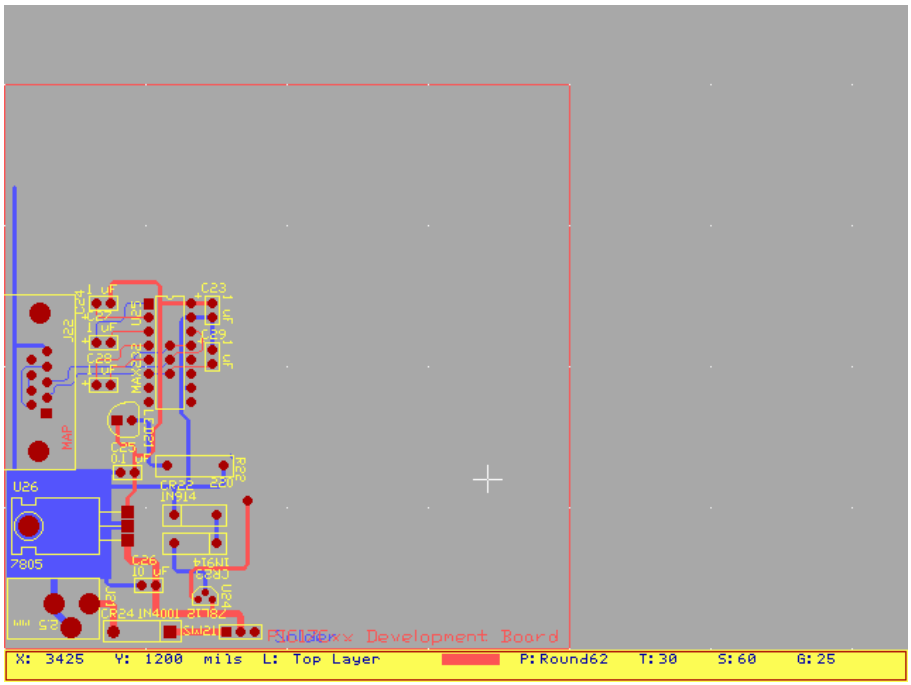


Figure 194 Start of prototyping. Power and RS-232 interface copied from YAP-II PC board design

In Fig. 194, notice that I have directly copied the power supply (for +5 volts and +13.4 volts) and RS-232 interface of the YAP-II and EMU-II. I did this because it works and it eliminated the need for me to create my own circuits. I also copied in this area when I laid out the PC board to avoid the need to add it to the circuit as well.

With the circuit designed and peripheral functions passed to a prototyping connector, I was ready to start laying out the PC board. I decided on a 4″-by-4″ card size to avoid having to squish the circuit onto the card (although, some portions of the board will become quite congested with circuitry). Figure 195 shows an EasyTrax screen shot of the start of the prototype PC board layout with the RS-232 interface and power supply copied from the YAP-II and EMU-II.

Next, I put down the major parts that I want to use in the circuit. Going back to the schematic, these parts are the PIC17C44 microcontroller, the 74LS373 buffers, the two 6264 SRAMs, and the 74LS138 decoder. When I first layout a circuit, I first try placing the functional blocks together (Fig. 196). Also, when I place the circuits, I ensure that all the components are orientated in the same direction and that the programmable part (the PIC17C44) can be easily removed from the circuit for reprogramming.

In Fig. 196, the parts are laid out as to be as far away from each other as possible to make my wiring easier. As well, I have also arranged the 6264s to be in line to see if this makes the "bus" (address and data) wiring easier. When I initially lay out the components, I work through what I consider to be a small "thought" experiment and visualize how each component will interconnect with others. As a result of this thought experiment, I realized that U2 and U8 (and U3 and U9) should be paired together to take advantage of the com-

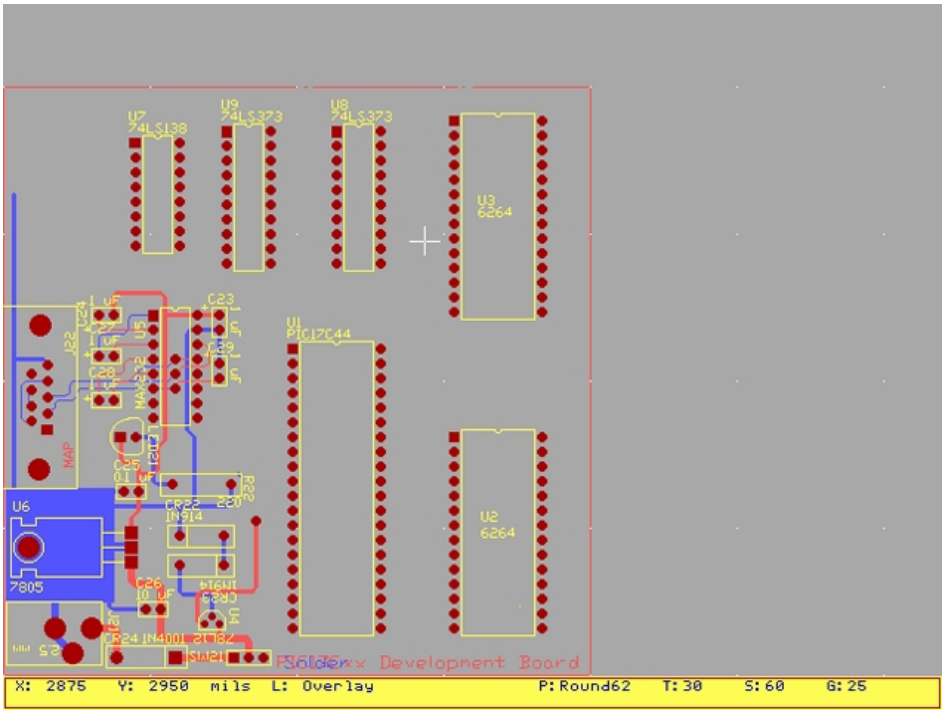


Figure 195 Prototyping: Adding DIP components to a circuit

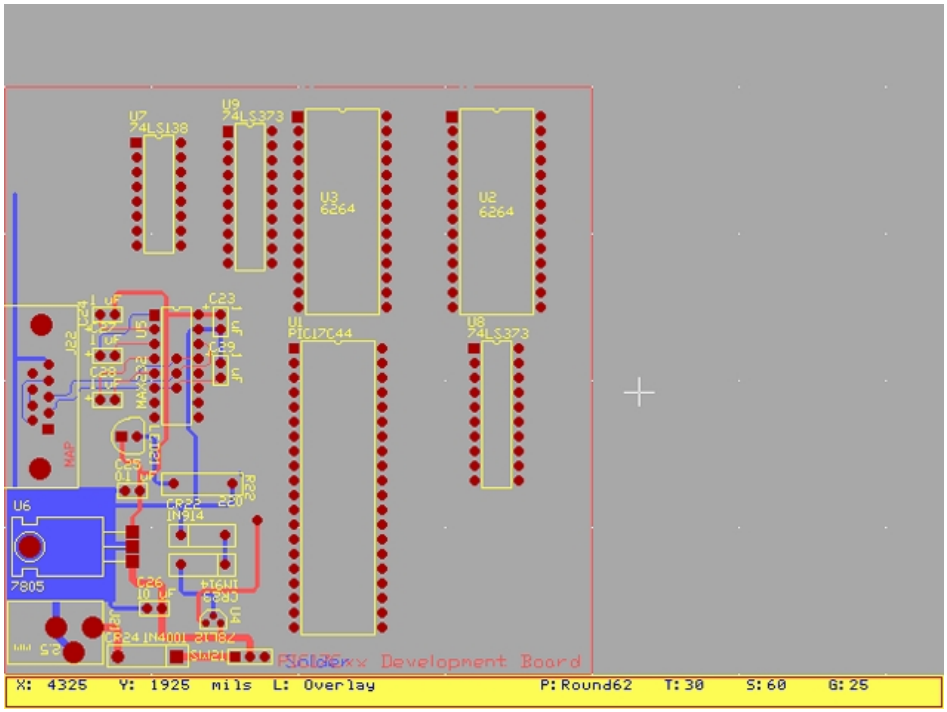


Figure 196 **Prototyping: Moving DIPs to take advantage of busses**

mon multiplexed address and data busses that will be wired to each chip. The resulting design is shown in Fig. 197.

Each time I change the PC board design significantly, I also save the PC board design as a different file name (usually I add a two- or three-digit number to the end of the filename for this purpose). This way if I make any mistakes, I can go back, identify the problem, and fix it.

Now, I can make the "basic" connections of power, I/O, reset, and clocking. These traces and any extra components needed to support them are easily laid down with as many traces as possible put on the bottom side of the board. The power traces are given a 0.030″ width (the signal traces are 0.012″ wide) and the V_{cc} tracks are put on the top side of the card to ensure that they remain physically separate from the bottom side. The updated card is shown in Fig. 198.

If you look carefully in Fig. 198, you should be able to find three errors. This is common, especially with large chips, like the PIC17C44, which has a lot of pins on one side. These problems will be fixed as I add more wires to the PC board and discover things where they shouldn't be. You will also notice that I moved the 74LS138 to be closer to U8 (I missed this when I did the original "thought experiment").

Depending on what you want to do with the PC board, you might consider yourself finished. If wirewrap sockets are going to be used to make the interconnections on the board between the chips, you could hand wire the circuit very easily. Many people stop at this point to avoid incurring the costs of unneeded PC board masks, but personally, I don't believe that this saves a lot in the long run.

If wirewrapping at this point was going to be used, the total number of wires required is

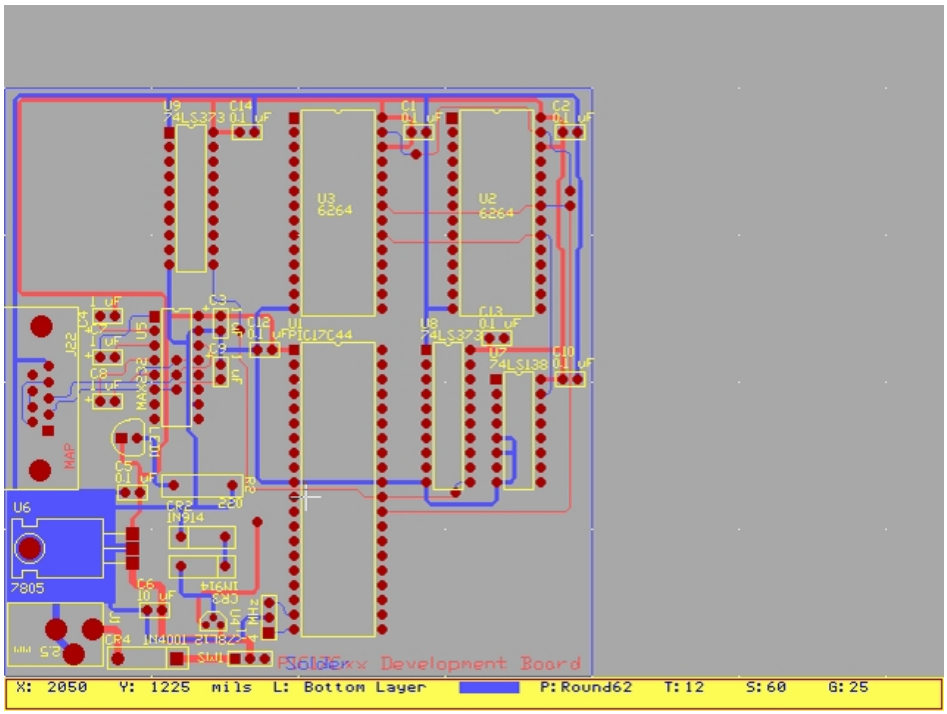


Figure 197 Prototyping: putting power-control signals onto the PC board

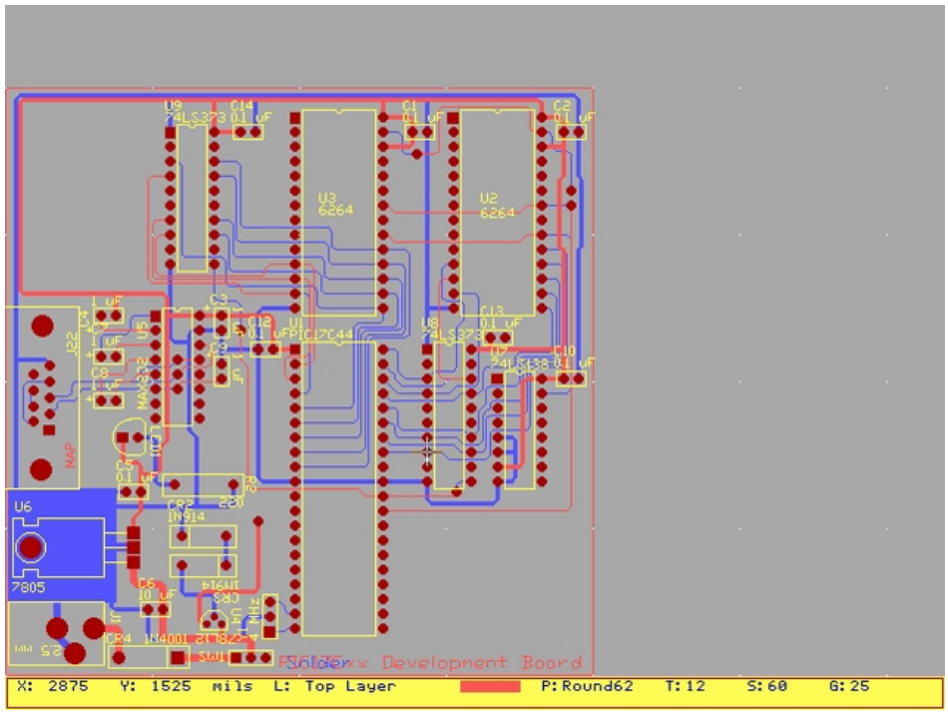


Figure 198 Prototyping: adding multiplexed address/data bus on card

about 100. Using the calculations presented earlier in this appendix about prototyping, it would take about two hours to wirewrap it. In actuality, it would probably be significantly less because all the wirewraps are "easy" ones that require no special connections to power or discrete components to add to the board. Because it took me about 30 minutes to get to this point, the total time to build the prototype circuit will be about three hours.

Because I am this far and I don't like wirewrapping or point-to-point wiring more than 25 wires or so, I would continue and finish the board with all the traces connected properly. Initially, I would want to do the multiplexed traces (from the PIC17C44 to the 74LS373s and 6264s). These are the basic connections between the PIC17C44 and the 16-bit bus. I tried to put as much on the backside of the PC board to make reworking (i.e., fixing the board) as simple as possible. Figure 199 shows the 16-bit, multiplexed address, and data bus connected on the board.

When I was making the connections in Fig. 199, I was not able to follow the schematic exactly. The differences are in the two 74LS373s (U8 and U9). To simplify the wiring and minimize the need for vias, I rerouted some of the signals going to the '373s. When I made these changes, I documented them and then updated the schematic.

With the bussed connections made, I then proceeded to make the connections to the address lines from the 74LS373 buffers and the 74LS138 address selector chip. In Fig. 200, you can see the result. Notice that some of the traces that were put down earlier in the design had to be changed to allow "throughways" for the common address signals between the two 6264 SRAMs.

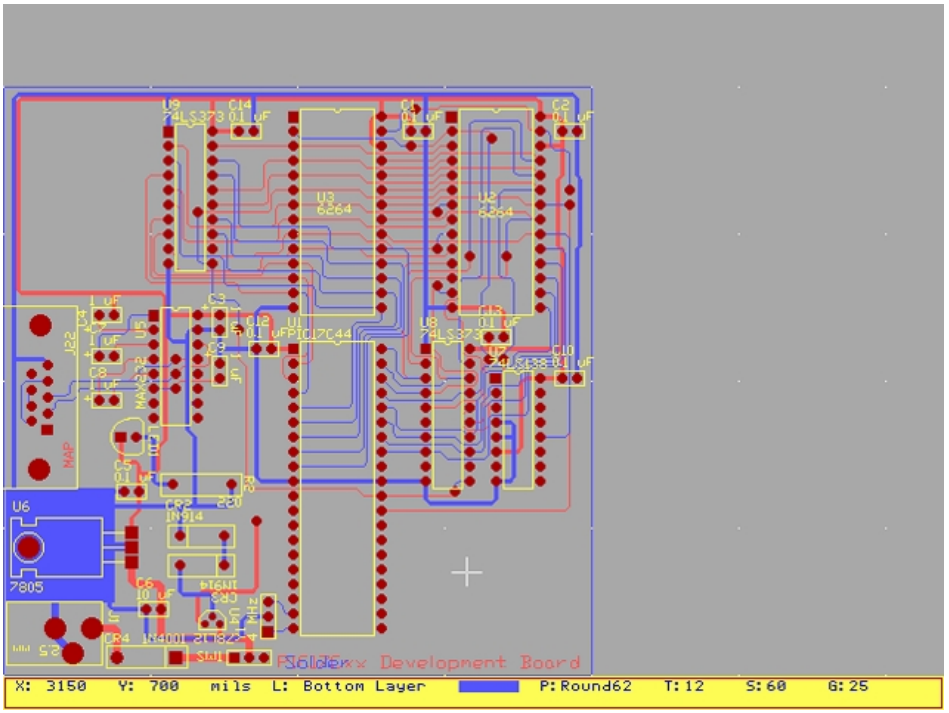


Figure 199 Prototyping: adding buffered address to SRAM on card

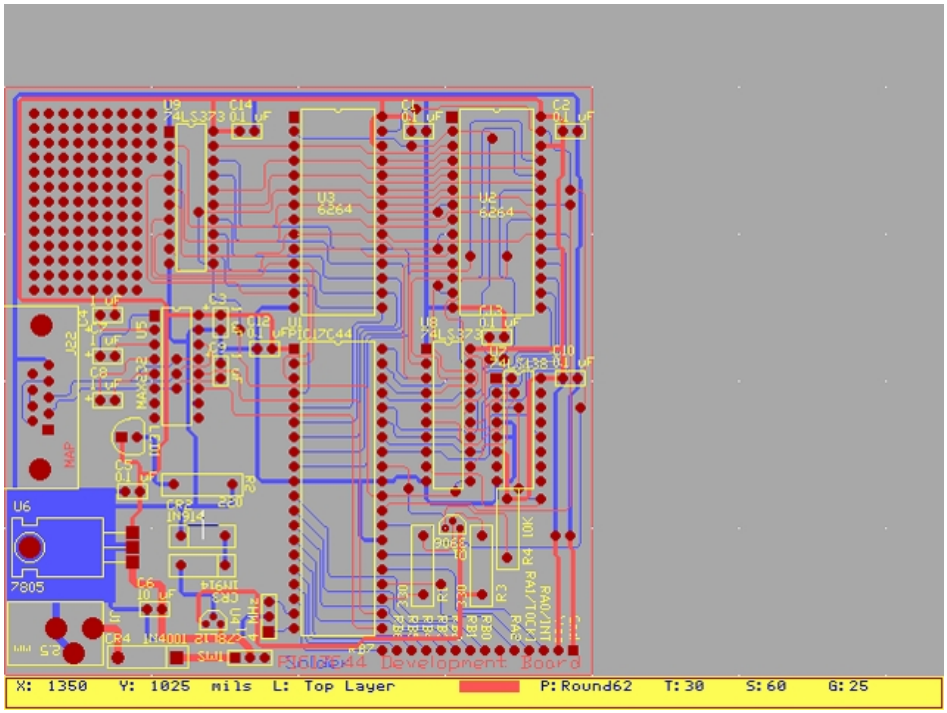


Figure 200 Finishing off the board with prototyping areas and a PORTA / PORTB connector

With the 6264 SRAM's address data and control lines connected, the majority of the work to wire the board is finished. To finish it off, I added three parts to the PC board design. The reset circuit is designed to allow code to write to the PIC17C44's internal EEPROM and was taken from the Microchip *In-Circuit Serial Programming Guide* (Microchip document DS91015B). As well, I added a 14-pin prototype connector to give me easier access to the PORTB and unused PORTA I/O pins and a "prototyping" area that uses the unused space in the top left corner of the board (Fig. 201).

Putting down the reset circuit was very simple (as would be expected with a single transistor and three resistors). Of a bit more difficulty was putting down the multiple vias. I had to relocate some of the power traces in the area. The final result is a 13-by-8 area in which sockets and components can be inserted into the board to try out new circuits. Adding a prototyping area like this is something that I always try to do when developing first-off or time-sensitive boards. Having an area in which I can add new circuits to an application has saved my reputation on more than one occasion at work when application requirements have changed (of course it's never *my* fault). Adding these vias to the board generally doesn't add to the cost of the final PC board.

When I wired the PORTA and PORTB connector into the bottom right corner of the board, I discovered that it would be much easier if I reversed the PORTA connections with the PORTB. This change was made and the prototype PC board design completed. This should not be regarded as a significant change to the board because the connector was put on to make access to the advanced functions of the board simpler.

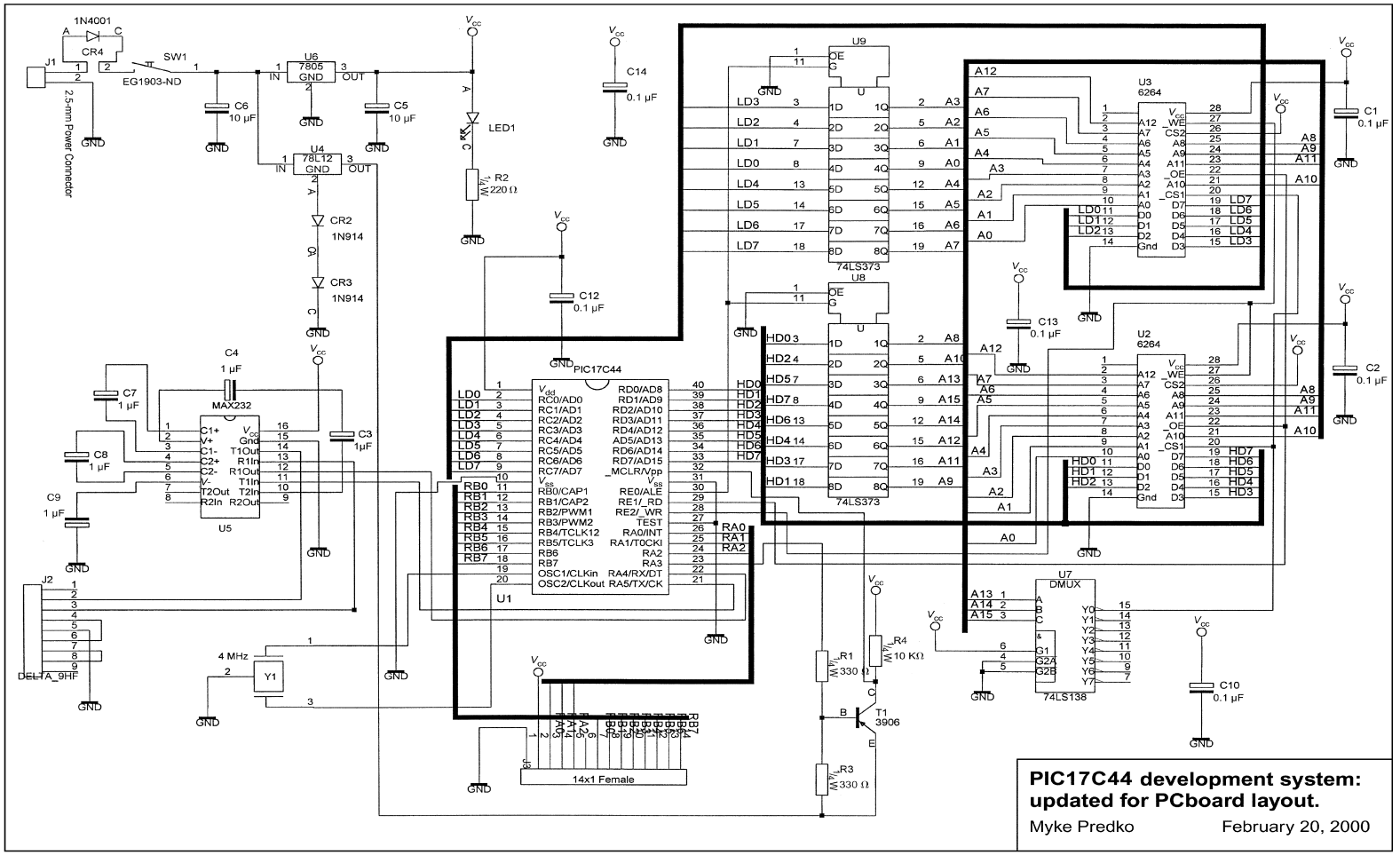


Figure 201 PIC17C44 development system: updated for PC board layout

With all the changes made to the circuit documented, I went back and updated the schematic. The final schematic is shown in Fig. 202 and just took a few moments to update.

The total time to design this PC board was about two hours. Building the circuit requires about 30 minutes. Although this is only about two thirds of the time required to wirewrap a prototype, by developing a PC board prototype board, the board can be updated, fixed, or even replicated as often as you need it. The investment in developing the board is not lost and will pay for itself more and more as you make additional copies.

In terms of costs, the difference between a wirewrapped prototype and a PC board is not as significant as you would probably expect. If you were to assume that a "quick-turn" prototype PC board house (like AP Circuits) produces boards at a cost of $4.50 per square inch (for two-sided boards with plated-through holes and no solder mask or silk-screening), the total board cost would be about $72.00. When I build circuits like this, I usually only socket the expensive parts like the PICmicro® MCU and the SRAMs. Everything is soldered to the board to avoid socket costs (which can be as much as the "glue" logic chips). The total cost for the three sockets will probably be about $5, for a total of about $77 for the PC board.

For wirewrapping, the socket prices are about $2 each for 16- and 20-pin wirewrap sockets, and $4 dollars each for the 28- and 40-pin wirewrap sockets. If sockets with internal decoupling capacitors are used, the socket costs will at least double. A good-quality prototyping card will cost a minimum of $5 (and could be as much as $20). Wirewrap wire will be about $5. Finally, there will be the issue on how to connect discrete components and the RS-232 connector. Using a Scotchflex connector for this function will run you about $10 in small quantities. The total cost for the wirewrapped solution will range anywhere from $40 to $75, take you longer to assemble than the PC board, and only result in a maximum of one copy of the board for your investment.

In this calculation, I haven't included such PC board issues as shipping and handling, taxes, and other incidentals (the price does include mask development, though), so there might be some differences in the actual price when you finally get the boards. For wirewrapping, I have not included the time required to build the prototype. If you consider that professional wirewrappers get about $25 per hour, you can see that from an out-of-pocket perspective, an embedded PC board is the better deal.

Looking at the $4.50 cost per square inch PC board quick-turn assembly costs (production costs are considerably less), you might feel like you could minimize the prototype costs even more by building the PC boards yourself. Kits are available that provide every-

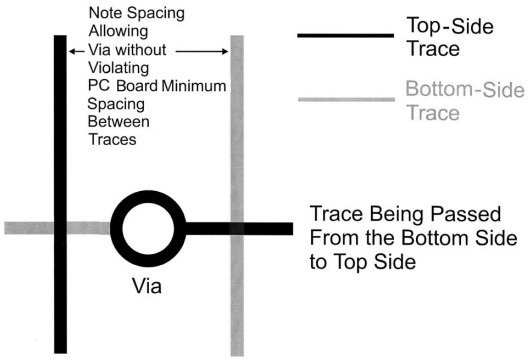


Figure 202 Vias to pass a trace between PC board layers

thing that you need for making your own prototype PC boards for less than $20, which will put the costs of a PC board below that of wirewrapping.

I would like to discourage you from building your own PC boards unless you are going to do it for a large number of boards and are willing to make the investments in the proper tools to do the job. The kits are difficult to work with and often require that you draw out your design with a pen on the backside of the board. This can be simplified by buying component "masks," but this adds to the costs. Another issue is that these kits only provide single-sided boards in them. With dual-sided boards, there is no way to add plated-through vias. PC-board drill bits are quite costly and easily broken.

A new option is a copper sheet that can be printed in a laser printer with the design of the circuit and then ironed onto a piece of phenolic or fiberglass PC board material. Once the copper is ironed onto the board, it is then etched as if it were a regular copper board. If the PC board is already predrilled with holes at 0.100″ centers, this can be a quite easy proposition to build your own boards. This method avoids having to draw the traces onto the copper of the PC board or expose the PC board with a mask and develop it before etching.

The only issue with this method is that although two sheets can be used to lay on top of each other, they cannot be connected using plated-through vias or holes. When laying out the board using these sheets of printed copper, you should plan for vias to be placed in between the components and small pieces of wire soldered between the sides to bridge them. No top-side copper pads should be used at component interfaces because this will be difficult to solder reliably.

Last, the chemical etchant that is provided with the kits is a toxic substance and must be disposed of as "hazardous waste." Never pour etchant down a drain. Not only is it illegal, but the chemicals will dissolve your home's pipes.

This is not to say that it isn't interesting or fun to make your own PC board boards. I have done it a number of times over the years and you do end up with a good sense of satisfaction when you are finished. It's just that there is a lot more to it than buying a $10 kit at "Radio Shack." Your time, money, and energy would be better spent designing, debugging, and building circuits of your own.

Circuit layout rules One of the most neglected aspects of an application is how a prototype or even a product's circuits are laid out. Like many aspects of application design, the circuit design and layout is a result of developer experiences and preferences. The success of a circuit design, like the software and overall concept, is a result of the time spent planning out how the circuit is built before laying it out onto a PC board. This section reviews some points and tips that I have discovered makes a circuit board layout successful.

These rules are similar for prototypes that are built into a breadboard, hand-wired together, or built into an embedded circuit card. Some of these points are more applicable to specific methods than others, but they are all good to remember.

1 Be sure that all parts can fit on the board. This might seem basic, but I have seen (and unfortunately designed) a few boards where the carrier isn't large enough for all the parts. The result is chips glued "dead bug" on the backs of others or cards that have "daughter cards" that are held together by solder and wire.

2 Put the components on the board in the following order:

- **A** Connectors
- **B** Programmable parts/chips
- **C** Decoupling capacitors
- **D** Oscillators
- **E** Discrete components
- **F** Remaining parts

When you follow this order, you will be putting the connectors and programming parts where they can be easily accessed for programmable parts (for example, PICmicro® MCUs). Placing them on the edges of the board will allow them to be easily removed for reprogramming. Decoupling capacitors must be as close to chips as possible. Once these parts are finished, the positions of the remaining ports can be chosen.

3 When making a PC board, lay out the power tracers first using 0.030″ to 0.050″ width traces (normal signal traces are 0.010″ to 0.012″ wide). Once these are done, you can do the signal traces. Notice that I always put positive voltage traces on the topside and negative (ground) voltage traces on the bottom side of the board. Also, work to only have one path for voltage and ground. Avoid loops in the traces, which can pickup electronic noise.

4 When planning your wiring, do power first and be sure that nothing is "blocked." Never lay a power trace over another that will prevent signal traces from following through. It is important to remember to always stagger them so that traces can pass between them (Fig. 203)

5 Another good idea for embedded boards (as well as hand-wired boards) is to put as many signal traces on the backside as possible to allow tracing of signals and reworks with components in place.

6 When wiring a board, keep decoupling capacitors as close to the PICmicro® MCU's V_{dd} and other chips V_{cc} as possible. The distance to end ground (V_{ss}) is less important. When I'm hand-wiring boards, I'll solder the decoupling capacitor to the backside of the chip socket. You can also buy sockets with decoupling capacitors built in, which eliminates the need for this wiring.

7 Don't plan on getting everything right the first time. For example, the El Cheapo programmer underwent four circuit designs and three PC board designs until I was happy enough with it to be included in this book.

If you wire a circuit incorrectly, remember that although problems are maddening, they are never the end of the world. There is nothing that can't be fixed or reworked. You can fix the problem in the next pass of the board. Yes, you don't want to do any extra work,

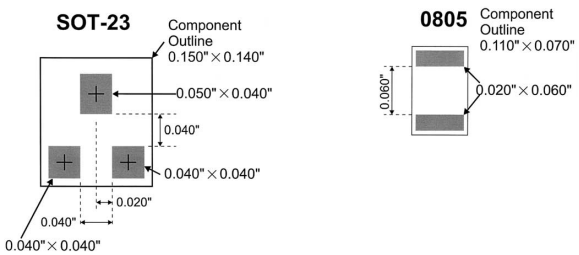


Figure 203 SOT-23 and 0805 SMT pad design

but chances are you will have learned something that will make it easier to prevent doing again in the future.

SMT PROTOTYPING

Sometimes you will be stuck designing a prototype circuit for a very small enclosure, which necessitates the use of *Surface-Mount Technology (SMT) components*. In other cases, such as the S7600A web server chip, SMT parts are the only types available. To develop prototype circuits for the situations, you will have to design your own PC board (although some prototyping systems are available). This isn't terribly hard to do if you follow the rules outlined in this section.

As noted earlier in the book, SMT components are soldered to the surface of the card, rather than through a hole (via) in the card. When SMT parts are soldered to the PC board during production, a "paste" of solder and flux is "printed" onto the card and the component pins pushed into the paste. When all parts are on the board, the assembly is run through an oven to melt the solder and attach the components to the board.

For resistors, capacitors, and transistors, I follow the IPC standard pad layout for the different components. To allow easy hand assembly and reworking, I only use 0805 chip parts and SOT-23 packages for SMT diodes and transistors. Figure 204 shows how the pads are laid out:

These patterns can be loaded into your PC board design system if they are not already there.

DIP parts are a bit more difficult to work with and dependent on the package type. The general rule that I use is to add 0.020″ to the length of the pin, outside of the component and make the width the pin spacing the component pin spacing minus 0.020″. This is shown for a PIC12C508 in a small outline (SM) package in Fig. 205.

When placing SMT parts on a PC board design—especially if you are using a quick-turn PC board supplier, do not run traces between the pads. Instead either run them outside of the part of connect them to vias leading outside of the part outline on a different layer.

When soldering SMT parts, I use the following trick. I put solder on one of the pads, next I "tack" one of the device's pins to this pad. With the part held down, I go around the board with the soldering iron, soldering down the other pins and finally resolder the pin that was tack soldered to the PC board originally.

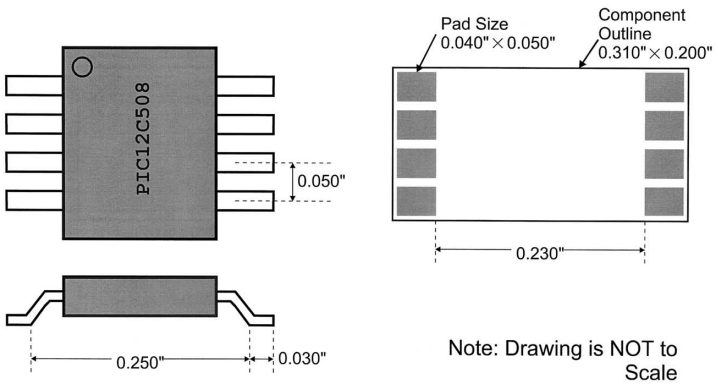


Figure 204 PIC12C508 SOIC SMT pad design

Figure 205 Digital Multi-Meter (DMM)

To remove components, a hot air gun (which can be quite expensive) can melt (or reflow) the solder, allowing the component to be removed. SMT parts can be removed very carefully with a soldering iron and "solder sucker" or you can cut the leads off with an X-Acto knife and then remove the pins individually with a soldering iron. If you are just starting out, I recommend using the knife approach; it is cheap and not very frustrating. In fact, it can be somewhat satisfying.

If SMT-packaged PICmicro® MCUs are going to be used in the application, be sure that the ICSP pins are passed to a connector on the card to ensure you can program/reprogram the part later. This is the only thing that you should beware of when laying out SMT parts on a card. Other than ensuring that the ICSP pins are available, you should be able to add SMT parts as if they were *Pin Through Hole (PTH)*.

Notice that if you are going to take a SMT card from the prototype stage to production, you should be sure that you keep mixing of SMT and PTH parts to a minimum. Ideally, you should have a card that is either all SMT or all PTH to minimize the costs of producing the card. You will find that a hybrid card of SMT and PTH parts will cost much more than a single technology card for a contract manufacturer to build and might require equipment that is not available at all vendors. Going to an all-SMT card, even though SMT connectors can cost more than PTH ones, will generally be your cheapest option for contract manufacturing of any kind of volume.

Test Equipment

When working with any electronic circuits, I recommended that at the very least, you should have a DMM capable of reading voltage, current, and resistance. These simple

functions will allow you to work through your applications and understand the reasons why the circuit doesn't work as you expect.

Because DMMs can be purchased with more than these simple functions, you can buy a number of other pieces of equipment to make testing and debugging your electronic circuits easier. A fully equipped electronics laboratory for developing PICmicro® MCU applications can be set up for less than $2000, but literally hundreds of thousands could be spent to create a laboratory that can work through any possible part or application problem at signal speeds running up to gigahertz frequencies.

This appendix finishes with introductions to some different types of test equipment and the problems they are best suited for helping out with. Elsewhere, the book describes how situations can be set up to find problems with applications and how the tools are used to find them.

DIGITAL MULTIMETERS

Earlier in this appendix, I introduced the concepts of how ammeters and voltmeters work. These electromechanical devices have been replaced by small hand-held *Digital MultiMeters (DMMs),* which are cheaper, more accurate, easier to read, and can perform a number of other measurements as well (Fig. 206). Digital multimeters can be bought for as little as $20.

The output of a DMM is a three or four digit numerical display. In many devices, the measurement is selected via a switch on the DMM and the display's decimal point moves over. If the value is too large for the display, something like a *1* in the left-most digit will be displayed with the other digits will be blanked out. Table 26 lists what would be shown on a four-digit display for different resistance measurements.

Notice that the ranges start at *2*, rather than *1*, as you would probably expect.

This is a common convention, left over when displays were more expensive and the single-digit *1* added extra flexibility for minimal cost.

Some digital multimeters have up to six digits, but I must stress that as you work with

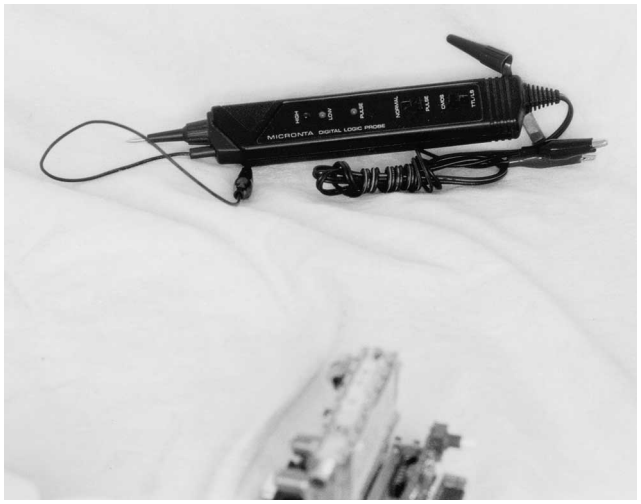


Figure 206 Logic probe

TABLE 26 Digital Multi-Meter Output Examples

RESISTANCE	RANGE	OUTPUT	COMMENTS
220 Ω	200	1	Value too Large for Display
220 Ω	2k	0.220	220 Ω = 0.22k
220 Ω	20k	0.022	

the digital multimeters for your own PICmicro® MCU or digital electronics projects, never use more than three digits (and ideally not more than two). The extra accuracy is not needed and it adds a lot to the cost of the device.

Each digit represents a power of ten. For a three-digit display, the value is supposedly accurate to one part in a thousand. Greater than one part per thousand accuracy is only very rarely required in very specialized cases. If this level of accuracy is required, precision power supplies, crystals, and components would be used alone with a specially calibrated digital multimeter. It might be interesting to see differences between devices at 10 millionths of a volt, but this accuracy is not practical for any PICmicro® MCU applications that I can think of.

Digital multimeters are not fast-response devices. You might find that it can take as long as 10 seconds before the display stabilizes on a value. This time to stabilize means that the digital multimeter is not capable of measuring changing digital signals, unless the signal changes once every few seconds. If this is not possible, you would be better off using one of the other tools described in the following sections of this chapter.

Along with the ability to measure current, voltage and resistors, digital multimeters are available with the following features and capabilities:

1. Perform autoranging while measuring a parameter.
2. Measure capacitance.
3. Measure temperature.
4. Measure a bipolar transistor's "beta."
5. Perform diode checks.
6. Measure frequency.

These features are nice to have, but not crucial for the projects in this book. If you don't currently have a digital multimeter but an analog meter, I highly recommend that you scrap the analog meter and buy yourself a digital multimeter to help you understand what is happening in your circuit. If you don't currently have a digital multimeter, then you shouldn't be reading this. You should go out, buy one of the cheaper ones, and learn what you can do with it.

LOGIC PROBES

Along with a digital multimeter, I consider a logic probe to be a basic tool for anyone developing digital circuits. A *logic probe* is a pencil-like device (Fig. 207) that can be used to check different parts of a board and return a visual/audio signal, which indicates

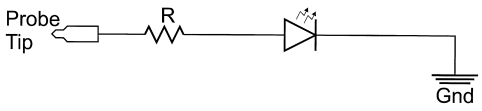


Figure 207 Basic logic probe

whether or no a signal is logic high, low, tri-stated, or not connected to a driver or a pulled-up or pulled-down input).

Usually, two controls are on a logic probe. The Pulse/Continuous switch will cause an LED to flash when a changing signal is encountered or a single LED will light for a continuous level. Along with an LED, many logic probes have a speaker in them that will provide beeps and tones so that you can tell what is going on without looking at the LEDs. I highly recommend getting the speaker audio output.

The other switch is TTL/CMOS, which selects the voltage threshold that should be used. Normally TTL is 1.4 volts and CMOS is 2.5 volts.

Good-quality logic probes can be bought for as little as $20.00. I recommend you buy one, rather than try to build one your self. You might think you could build one yourself with a resistor and LED (Fig. 208).

The circuit in Fig. 208 this has three problems:

1 It doesn't detect high or low outputs, just high.
2 If a circuit doesn't have enough current capacity, the LED won't light.
3 No indication on level transition.

To detect high, low, and tri-state/no connection, another LED will have to be added to the circuit. If the single LED is not lit, then the pin could be attached to a low, tri-state, or low-current high output. To fix this, voltage comparators, capable of driving a LED, should be used in the circuit (Fig. 209).

In this circuit, if the probe is driven high or low by a pin, one of the LEDs will light. A switch-selectable "threshold reference" voltage divider could be put in for a TTL/CMOS selection.

The circuit as it stands will not detect open conditions; for that feature, some kind of MOSFET sensor is required. A P-channel MOSFET could by put in to sense when the logic level is low (Fig. 210)

When the probe pin is low, the P-channel MOSFET will turn on, passing current to the LED. To add a transition output, a single shot should be triggered by each of the LED

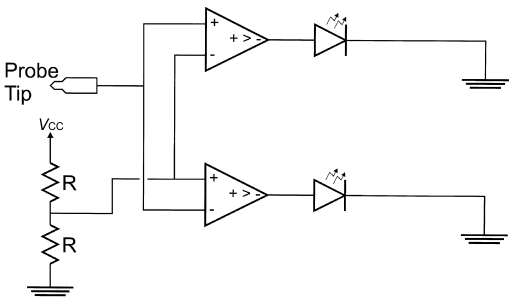


Figure 208 Logic probe capable of returning high/low.

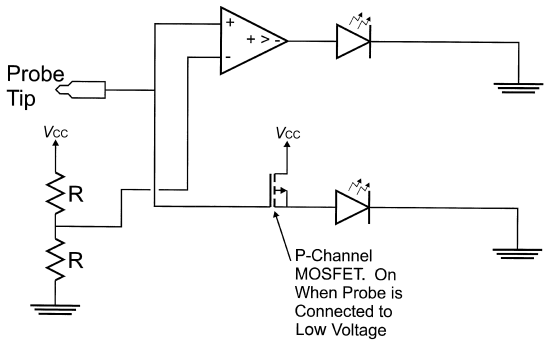


Figure 209 Enhanced high/low sensing logic probe

drivers. This will cause a third LED to flash when the LED drivers become active (Fig. 211)

Looking at this, you're probably confident that you could build this circuit easily. You probably could, although there might be some problems with getting the circuit to fit in a small enough package to be useful. As well, if you did attempt to build one yourself, it would probably cost as much in parts as buying a completed unit.

For these reasons, I recommend that you avoid the headaches of building one and just buy one ready made. As unlikely as this sounds, the Radio Shack logic probe contains all the features I've listed here and is quite inexpensive ($20), making it the device you should look at first. I've owned three of them over the years. They are as good or better than other logic probes costing twice as much (or more) from more "prestigious" manufacturers.

OSCILLOSCOPES

If you are expecting a tax return, a grant, or a stock dividend, put off that purchase on a wide-screen TV or a new PC and look to buy yourself an oscilloscope to help you work on the PICmicro® MCU. No single tool will give you the capabilities of an oscilloscope to understand what is going on in your application. I have owned the Tektronix TDS-210 (Fig. 212) for the past four years and I have to say that the experiments and projects in this book would not have been possible without it.

There are two different types of oscilloscopes. The most basic one is the analog oscilloscope that starts a sweep generator when a voltage trigger level is reached. The sweep generator causes a CRT electron beam to move across the screen. The continuing signal is

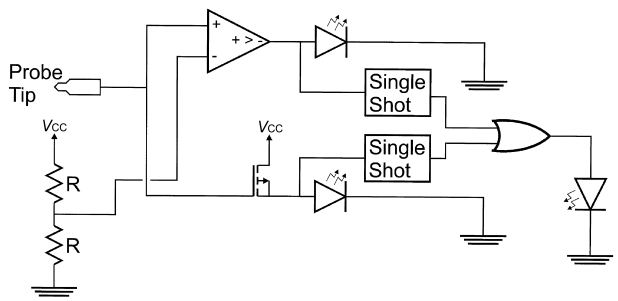


Figure 210 Complete high/low sensing logic probe

Figure 211 TDS-210 digital oscilloscope

"drawn" on the circuit screen, deflected up and down proportionately to the voltage of the input signal (Fig. 213).

The circuit screen is marked off in *graticules,* which indicate the time between features on the screen and their magnitudes. The oscilloscope itself is calibrated so that these values can be read off the screen simply counting the graticules.

Multiple signals can be displayed by alternating the position of the single electron beam. One way is to "alternate" the source, displayed each time that the oscilloscope is triggered. The second method is to change which source is being displayed as the beam sweeps (known as *chop*). Figure 214 shows how these methods work and appear on an oscilloscope display.

Neither method is "perfect" for looking at multiple signals. In either case, important data could be lost or not be visible.

There are two problems with the analog oscilloscope when it comes to working with PICmicro® MCU (and other digital) circuits. First, each time the electron beam sweeps,

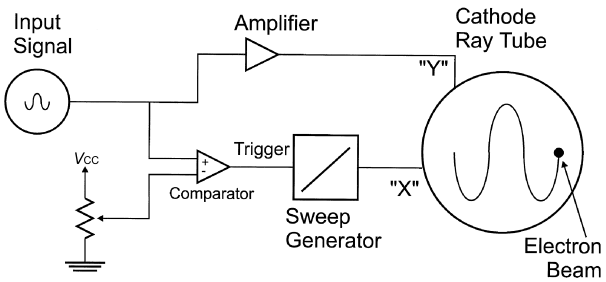


Figure 212 Basic analog oscilloscope block diagram

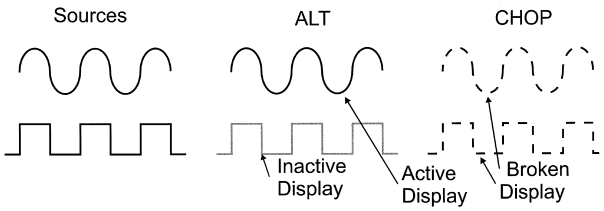


Figure 213 Basic analog oscilloscope operating modes

the signal on the display is very dim and fades quickly. If the intensity is turned up, the phosphors on the backside of the CRT could be damaged. The second problem is, it is very easy to miss single events in multiple sources because the Alt and Chop displays miss the changes. These problems get worse with faster sweep speeds and multiple signals.

Over the years, some manufacturers have come up with ways to store signals and improve phosphor performance, but there are still a lot of problems using an analog oscilloscope with digital signals. Analog oscilloscopes are excellent for very low speed, repeating signals, such as you would find in your stereo with a reference signal, but they are not helpful for digital signals.

Instead of using analog oscilloscope for digital applications, I would recommend using a digital oscilloscope, more properly known as a *Digital Storage Oscilloscopes (DSO)* or *digitizers*. These oscilloscopes save incoming signal values digitally and display then in a similar format as an oscilloscope. The block diagram for a digital oscilloscope is shown in Fig. 215. In this circuit, the incoming circuit is digitized by the flash ADC and stored into memory. When the triggered event has finished, the data is read from memory and displayed on the oscilloscope's display.

The obvious advantage to a digital oscilloscope is its ability to capture and display an event's multiple signals without any data being lost or being difficult to observe. It also has the added advantage of being able to transfer the data from the digital oscilloscope to a PC without having to take a picture of the screen and then digitizing it (as you would with an analog oscilloscope). The oscilloscope pictures shown in this book were transferred via RS-232 from my TDS-210 oscilloscope and stored into .TIF format files for publishing.

Events can also be displayed more easily in a digital oscilloscope because many models have the capability to sample continuously. When the trigger point is reached, the counter just continues to the end of the sample. This is shown as throughout the book with the various oscilloscope pictures that are presented in it.

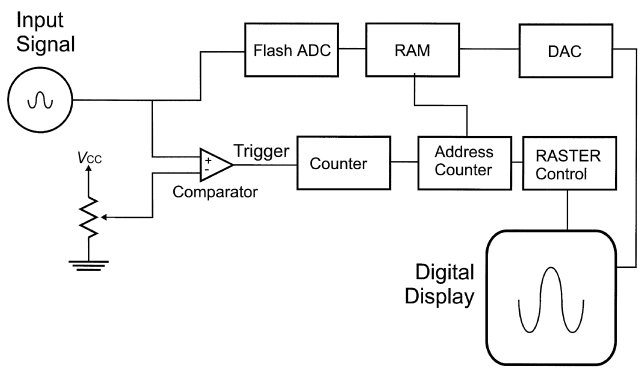


Figure 214 Digital oscilloscope block diagram

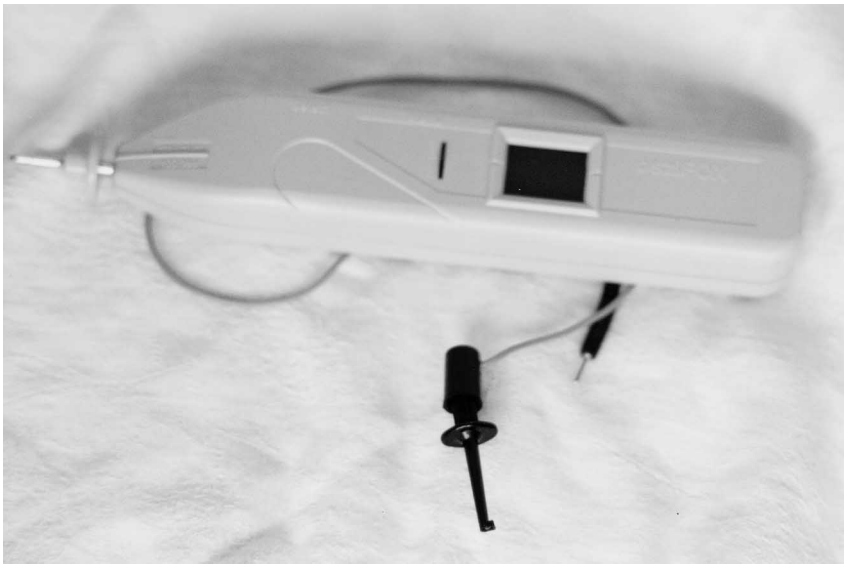


Figure 215 OsziFOX hand-held digital oscilloscope

The drawback of a digital oscilloscope is its cost. When I was in university, all oscilloscopes were analog. Once, in a demonstration, a radar lab was opened up and the class was shown a digital oscilloscope. It was a big event and the professor responsible for it had barriers put up to keep inquisitive fingers from damaging the instrument. This was probably a reasonable precaution for a $50,000 piece of equipment. A couple of years later, when I first started working, the cost of digitizing storage oscilloscopes was still several times that of an analog oscilloscope—high enough to make it difficult to justify their purchase.

Today, a digitizing oscilloscope is two or three times the cost of a comparable analog oscilloscope. Relatively small devices, like my handheld oscilloscope (Fig. 216), can be bought for $100 or so.

Despite the higher costs, I highly recommend that you get a digital oscilloscope over an analog one. The digital oscilloscope can make virtually all of the same measurements of the analog oscilloscope, but the ability to monitor individual events will make it worth its weight in gold. Even reasonably slow digital oscilloscopes (the OsziFOX hand-held oscilloscope shown in the photograph can capture signals at a frequency up to 5 MHz) are more useful when working with digital circuits than analog oscilloscopes that can run 10 times faster.

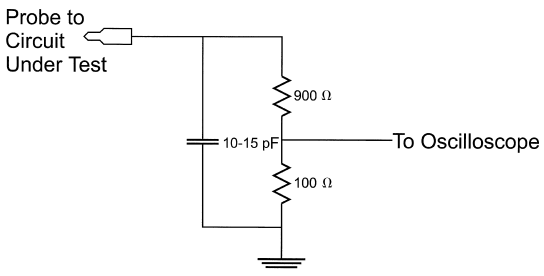


Figure 216
Oscilloscope 10× probe circuit

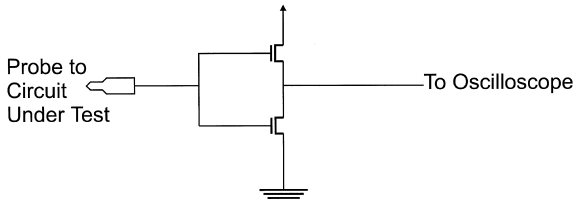


Figure 217
Oscilloscope FET probe circuit

You will have to work through and understand a few issues before you are comfortable with an oscilloscope. First is understanding how probes work. You might expect that oscilloscope probes are actually straight-through wires into the oscilloscope circuitry. Although this is possible, most oscilloscopes have 10× probes, which provide a high input impedance to the circuit. Built into the probe is a 10:1 voltage divider (Fig. 217). The nominal impedance is 1-Mê resistance and probably 10- to 15-pF capacitance. This low probe impedance will allow an oscilloscope to monitor a low-impedance circuit without affecting its operation significantly.

Depending on the equipment, you might not have a compensated display (i.e., instead of the signals coming at 0 to 5 volts for digital logic, they are displayed at 0 to 0.5 volts).

Along with 10× probes, there are cases where any kind of load at all on the circuit can affect its operation. To avoid this, MOSFET transistor probes (known as *FET probes*), shown in Fig. 218, are available.

FET probes are quite expensive, (usually starting at a few hundred dollars) and are designed for monitoring very high speed, impedance-controlled circuits and nets. None of the applications presented in this book come close to requiring them.

The last issue regarding oscilloscopes is "framing" the event you want to observe. It is probably obvious to say that it is important that the entire event should be shown on the oscilloscope display. This is easier said than done when you first start working with the oscilloscope. To make it easier on yourself, work with repeating signals as much as possible and play with the oscilloscope's controls until you are happy. As time goes on, you will find that you are better able to Figure out how to set up the oscilloscope so that you can see exactly what is happening very quickly. When you first start working with an oscilloscope, don't be surprised if it takes 15 minutes or more to set up each 'scope shot.

To make this a little more comfortable and avoid causing your arms cramping while

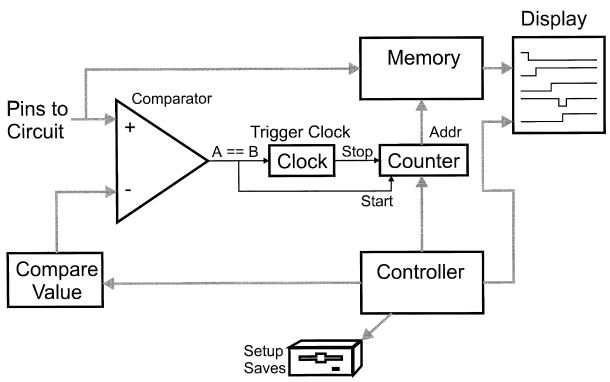


Figure 218 **Logic analyzer block diagram**

holding the probe and resetting the PICmicro® MCU or pressing a button to initiate an event, buy some "chip clips." With this tool, you can fasten the probes to the circuit, rather than having to hold onto them.

For SMT parts, you might want to solder short wires to the board and clip the oscilloscope probes to them. This is much easier than trying to hold something to the board for a long period of time. It avoids my number one annoyance with SMT parts: counting pins when the probe has slipped off.

LOGIC ANALYZERS

A logic analyzer borrows a lot of the technology from the digital oscilloscope described in the previous section. The logic analyzer (Fig. 219) uses the incoming multiple-line data pattern for the trigger and then loads its memory with the bit data until it is full. Logic analyzers tend to be quite expensive, although there is a trend today by manufacturers to include logic-analyzer capabilities with a digital oscilloscope.

Instead of recording digital representations of analog data, digital logic values are recorded at various points in the circuit. Logic analyzers are much more difficult to use within a circuit because of the amount of connections to be made and the need to label signals on the display and create a pattern to compare against to trigger the sample. To help cut down on the need to re-enter data, most logic analyzers have a disk or nonvolatile memory built into them to save specific "setups."

Logic analyzers can present data in two different ways. A graphical logic display, like Fig. 219, is useful in looking at signals and comparing them to simulator output or even circuit drawings.

The second type of display is a "state display" and a "clock" is used to strobe data into the memory, rather than saving the data according to a logic-analyzer internal clock. This type of display is best suited for monitoring the execution of an application. If the data and addresses are captured, it can be a fast and inexpensive way to create a processor emulator/tracer.

The two most crucial parameters or the logic analyzer is the speed at which data can be processed at and the "depth" of memory behind each pin. The need for fast memory should be obvious.

Depth relates to how much data can be stored and, more importantly, how much you can "zoom" in on a problem. In Fig. 219, channels SignalC1, SignalC2, and SignalC3 are sup-

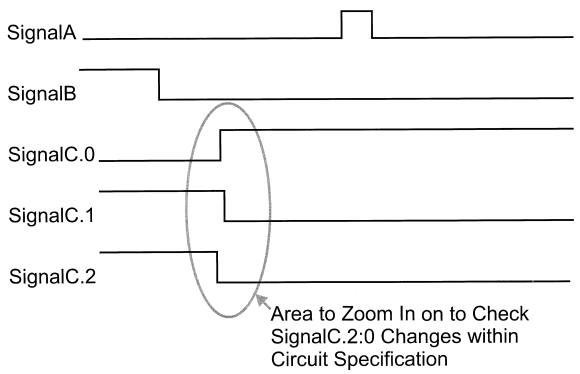


Figure 219 Logic analyzer display

posed to all change at the same point. To confirm this, you would want to check these transitions by "zooming in" to see what is actually happening. A logic analyzer with insufficient depth would not be able to display any data, except at the current resolution. To go to a higher resolution, another picture would have to be taken.

Before you get out your checkbook and start shopping for a logic analyzer, I must say that if you are going to be working with the PICmicro® MCU, then this will probably be the least-useful test instrument described in this book. Logic analyzers are best suited for applications that have external memory and I/O. If you are looking at what types of test equipment that you should buy, here is a list of items in priority order:

1. Digital multimeter
2. Logic probe
3. Oscilloscope
4. PICStart Plus or Promate II programmer
5. PICmicro® MCU emulator
6. Logic analyzer

From this list, the first one or two tools (the DMM and logic probe) will be sufficient to work through most of the applications in this book and help you to develop your own. As you gain more experience in electronics in general and the PICmicro® MCU specifically, you will be able to choose the tools that are best suited for your particular style of development. With this knowledge, you should be able to create a suite of tools that meet all your requirements.