ARRL Periodicals Archive - Search Results

A membership benefit of ARRL and the ARRL Technical Information Service

ARRL Members: You may print a copy for personal use. Any other use of the information requires permission (see Copyright/Reprint Notice below).

Need a higher quality reprint or scan? Some of the scans contained within the periodical archive were produced with older imaging technology. If you require a higher quality reprint or scan, please contact the ARRL Technical Inforamtion Service for assistance. Photocopies are $3 for ARRL members, $5 for non-members. For members, TIS can send the photocopies immediately and include an invoice. Non-members must prepay. Details are available at www.arrl.org/tis or email photocopy@arrl.org.

***QST* on DVD:** Annual DVDs are available for recent publication years. For details and ordering information, visit www.arrl.org/benefits.

Non-Members: Get access to the ARRL Periodicals Archive when you join ARRL today at www.arrl.org/join. For a complete list of membership benefits, visit www.arrl.org/benefits.

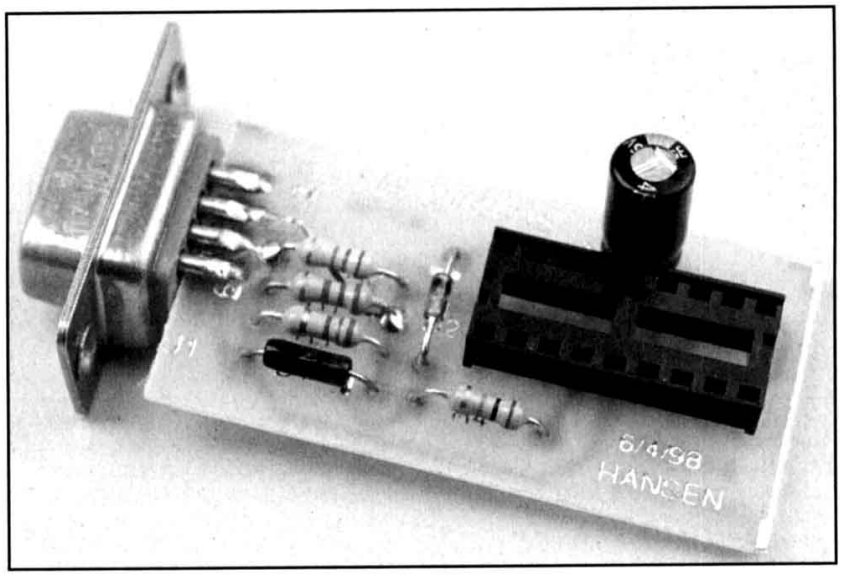
QST Issue: Oct 1998

Title: Using PIC Microcontrollers in Amateur Radio Projects

Click Here to Report a Problem with this File

By John A. Hansen, W2FS

Using PIC Microcontrollers in Amateur Radio Projects



These simple projects should whet your appetite to learn more about the little PIC microcontrollers you see so frequently.

Computers are now embedded in most consumer electronics products. Today, it's virtually impossible to buy a commercially made Amateur Radio transceiver that doesn't include at least *one* microprocessor. Recently, microprocessors have even begun to find their way into Amateur Radio home construction projects. In 1997, for example, nearly half the issues of *QST* included construction projects designed around several different computers on a chip![1-7] [*And we've got more in '98, too!*—Ed.][8-10]

The reason for this is simple: When using embedded microprocessors, it's possible to make equipment much more capable and much cheaper. And—contrary to the initial impression of many hams—embedded microprocessors make project design *easier*, not more difficult.

This article presents an introduction to using one of the simplest and cheapest microprocessors, Microchip Technology's Peripheral Interface Controllers—PICs—in Amateur Radio applications. I'll provide you with the necessary background to begin using PIC chips and give you some pointers on where to learn more about them. Most importantly, it will describe how to build a circuit that allows *you* to program PIC chips yourself — at a cost of less than $5! The next time you want to build a *QST* project that includes a PIC chip, if the source code is available, you'll be able to *program your own chip* instead of paying someone to do it for you!

What Embedded Microcontrollers Do

An embedded microcontroller can be thought of as a tiny computer that receives data, makes calculations or decisions based on that data and then acts in response. It interacts with the rest of the world through pins that can be configured as inputs *and* outputs. Configuring a pin as an *input* means that the microcontroller can *read* the pin to determine whether the voltage on it is high or low. When a pin is *high*, it means that 5 V is applied to it. When the pin is *low*, it means the pin is at ground potential. When a pin is configured as an *output*, it means that the microcontroller itself can make the pin high (+5 V) or low (0 V).

The microcontroller acts on a set of instructions (a *program*) that determine how the microcontroller converts these input signals into output signals. It is incredible the range of functions that these microcontrollers can perform! That's especially true when you consider that the chip's behavior is limited solely to finding out whether input pins are high or low, then setting output pins either high or low in response!

Inputs to the microcontroller might be pushbuttons (or arrays of buttons in a keypad), sensors of various types such as temperature, pressure, or acceleration (fed through an analog-to-digital converter [ADC]), or a serial or parallel data stream from any device capable of generating serial or parallel data (DTMF decoders, radio computer ports, PCs, etc). As long as the information can be presented to the microcontroller as a high or low signal on one or more input pins, the controller can recognize the information and perform predefined functions in response.

By using transistor switches or relays, the microcontroller outputs can switch external devices on and off, generate sounds, or send serial or parallel data to control other devices. Text or data from the microcontroller can be displayed on an LCD panel, or sent to a speech synthesis chip to be read aloud. Recent microcontroller projects in *QST* include a repeater controller (see Note 1), a CW IDer (see

[1]Notes appear on page 40.

Note 8) and a remote base controller (see Note 10). Each of these projects would have been *vastly* more complex, less capable and much more expensive if embedded microcontrollers had not been available. And think of it: We have only begun to scratch the surface of things that can be done with these ICs!

The Microchip PIC

Microchip Technology's series of PIC microcontrollers are among the most widely used by experimenters.[11] They are quite cheap and relatively easy to use. Microchip makes an assortment of these processors that hold differing program sizes and amounts of data. Some of these ICs include such features as on-board ADCs, serial ports, large numbers of input or output pins and multiple timers. To program these chips, you need a special programmer. To erase *most* of them, you need an ultraviolet eraser. In addition to the cost of the eraser (about $50), erasing one of these PICs takes about four minutes. So, if you are debugging a program by running it, checking how it functions and then making changes, getting to the finished project can be painfully slow.

Fortunately, Microchip has developed one series of chips that is *electrically* programmable and erasable. These chips can be programmed and erased over and over again—electrically. Not only does this make the process of programming and reprogramming easier, it also means the time and expense of using a UV eraser can be avoided.

Until recently, the most common version of these electrically programmable and erasable products was Microchip's 16C84. It has been largely superseded by their 16F84. In the 'F84, Microchip replaced the 16C84's EEPROM with flash memory. Operationally, both ICs are quite similar.[12] The 16F84 is nearly pin and code compatible with the 'C84, and has more room for data. It is also cheaper. In single quantities, the 16F84 costs about $6. If you buy 25 of them at a time, you can cut that cost by a third. Because of its ease of use, the 16F84 is an ideal chip to use when learning about PIC microcontrollers.

A Road Map to the 16F84

Let's dive right in with some projects—I'll explain things as I go along. Figure 1 shows a very simple circuit for experimenting with the 16F84 PIC: an LED and a piezo speaker. The PIC's pins labeled RB0-RB7 and RA0-RA4 are general input/output pins that the IC's program can specify as receiving or sending signals. In this simple circuit, we use only two outputs: RB0 and RB1. Through RB0, we can make the PIC light an LED. RB1 has been hooked to a small piezo speaker to allow experimentation with sound. The piezo speaker is an excellent choice for this application because it can be hooked directly to the output pin of the PIC to provide a fairly loud sound.

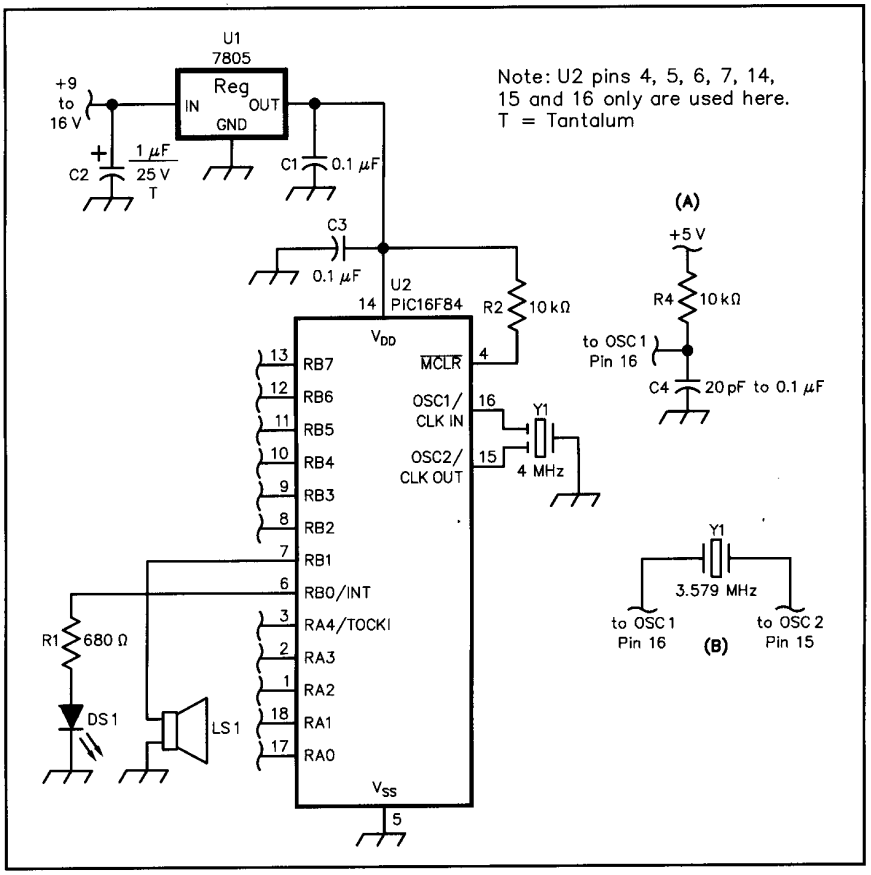


Figure 1—Schematic of the PIC sound-experimentation circuit. Unless otherwise specified, resistors are 1/4 W, 5% tolerance carbon-composition or film units; equivalent parts can be substituted. RS part numbers in parentheses are Radio Shack; DK numbers are Digi-Key.

C2—1 µF, 25 V tantalum (DK P2059; RS 272-1434)
LS1—Piezo speaker element (RS 273-091)
U1—7805 12 V, 1 A positive regulator (DK NJM7805FA; RS 276-1770)
U2—PIC16F84 microcontroller (see PIC Resources sidebar)
Y1—4 MHz resonator (DK PX400) or 3.579 MHz crystal (DK CTX049, HC-49 holder)

The rest of the parts in this circuit are needed in every PIC circuit to make it work. Positive 5 V is always applied to the V_{DD} pin and ground to the V_{SS} pin. Figure 1 shows how to use a 7805 voltage regulator to convert a 9 to 16 V supply to 5 V. An easy alternative is to buy a battery holder that holds four AA cells and use them to power the circuit. Although four AA cells deliver 6 V rather than 5 V, the PIC will handle that voltage level without difficulty.

The PIC needs a clock source connected to pins **OSC1** and **OSC2**. We have a number of clock options. The simplest is an RC timing circuit (see insert A of Figure 1), but its clock rate is slow and it does not provide a high-stability timing source. If you simply want to light an LED, the RC circuit is fine, but realize that it is an inappropriate configuration for tasks that require more accurate timing (such as serial data communication). You can also use an external oscillator, but this is a more expensive solution than is really necessary. A crystal can be connected between the **OSC1** and **OSC2** (see insert B of Figure 1). Microchip recommends including small-value capacitors on both crystal leads, but most users have found this to be unnecessary. TV colorburst crystals (3.5795 MHz) are readily available, inexpensive and work well in this application.

The cheapest way to provide a stable clock, however, is to use a ceramic resonator (shown connected to U2 in Figure 1). These resonators are extremely small parts (smaller than crystals) and generally cost less than $1 each. (Make sure you get resonators with integrated capacitors.) To use a resonator, simply hook its outside pins to the PIC's **OSC1** and **OSC2** pins and the resonator's middle pin to ground.

Finally, you need to apply 5 V dc to the PIC's **MCLR** pin. You could connect 5 V dc directly to this pin, but it's better to apply it through a 10 kΩ current-limiting resis-

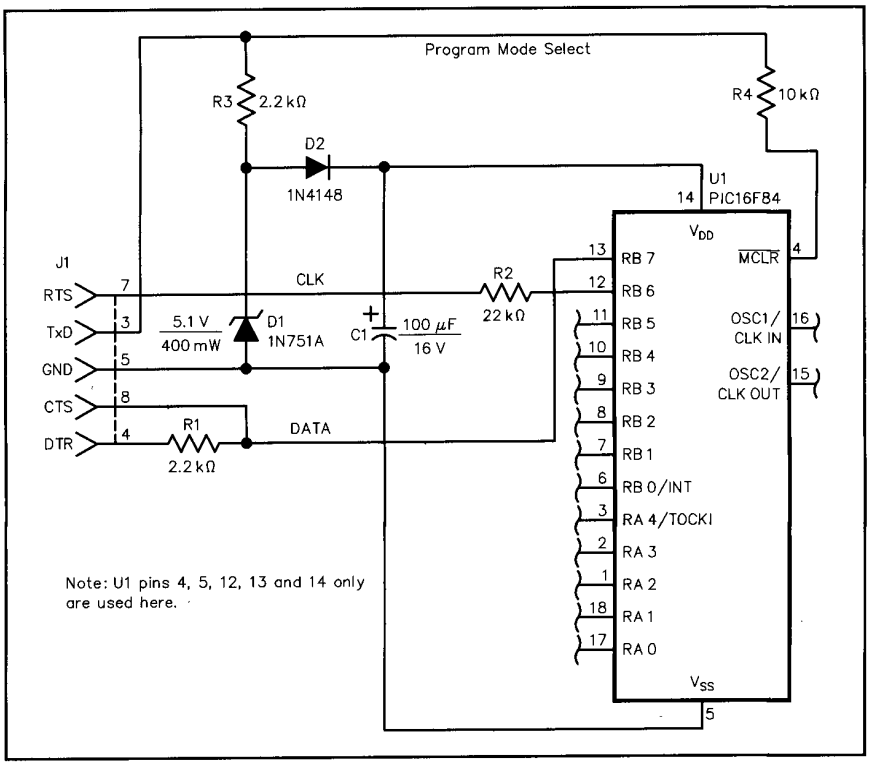


Figure 2—The simple PIC programmer schematic. Unless otherwise specified, resistors are 1/4 W, 5% tolerance carbon-composition or film units. Equivalent parts can be substituted. RS part numbers in parentheses are Radio Shack; DK numbers are Digi-Key.

C1—100 µF, 16 V electrolytic (DK P1119)
D1—5.1 V, 400 mW Zener diode (DK 1N5231BDICT; RS 276-565)
D2—1N4148 or 1N914 (DK 1N4148DICT; RS 276-1122)
J1—D89
U1—PIC16F84 microcontroller (see PIC Resources sidebar)

Table 1
An Assembly-Language Program to Create a Code Practice Oscillator (CPO.ASM)

```
        list        p=16F84
        __config    0x3FF3
portb   equ         0x06
        org         0x000
        movlw       0x00
        option
        movlw       0xFD
        tris        portb
start   btfsc       portb,4
        goto        start
        bsf         portb,1
        bcf         portb,1
        goto        start
        end
```

tor; then you can restart the PIC's program by simply shorting U2 pins 4 and 5. If you try this without the resistor, you'll create a short circuit across your power supply. You will find that 10 kΩ resistors are extremely common in PIC circuits. They are used to limit current flow. When you apply 5 V to a 10 kΩ resistor, a current of 0.5 mA flows (E/R = 5/10000 = 0.0005). This is sufficient current for the chip to detect the signal, offers minimal power supply drain and is nowhere near a current level that could damage the PIC. If you are going to do extensive work with PICs, buy 10 kΩ resistors in bulk. Because the PIC is a CMOS device, it is a good idea to tie each *unused* input pin to +5 V through a 10 kΩ resistor in your final design. For experimentation purposes, this isn't necessary.

I'll get to the programming of the PIC later on. (First, we need a programmer for the PIC.) When it comes time to build the circuit of Figure 1, I suggest you do so using a solderless breadboard (Radio Shack and other suppliers have them.) This will provide you with a test bed for experiments and trial designs for your circuits. Using this basic test bed, you will be able to build a circuit with any combination of inputs and outputs you would like. Once you have finalized your design, you can transfer it to a protoboard or a PC board.

Programming Hardware

Before you can use the PIC in the circuit of Figure 1, you must program it for the task you want it to accomplish. In some instances, the program might be written for you. Authors of many *QST* projects that use PICs (and other micros) make the source code available. So, even if you have no interest in writing your own PIC programs, it may be useful to have a PIC programmer to burn chips using code written by others.

There are a number of commercially made PIC programmers available, some of which cost more than $150. ITU Technologies makes a very nice programmer kit that will program a wide range of PICs. It's available for $39 ($59 assembled and tested) and comes complete with programming software to get you up and running quickly.[13]

For those who like to roll their own, Figure 2 shows an incredibly simple PIC 16C84/16F84 programming circuit. It is based on a design by Ludwig Catta, with a few changes made to ensure that all parts can be purchased at Radio Shack. The device is powered directly from your computer's serial port, so no other power supply is required. If all the parts are purchased new, this programmer still costs less than $5 to build. You will need a socket to hold the PIC while it is being programmed. You can use a standard 18 pin DIP socket if you're simply burning a single chip based on someone else's program code. However, if you're planning to use the programmer to do your own development work (requiring frequent insertion and extraction of the PIC), buy a zero-insertion-force (ZIF) socket to minimize the wear and tear on the chips.

Software Needed to Use the Programmer

The code (program) that is eventually loaded into the PIC consists of a series of hexadecimal numbers collectively called *machine language*. Generally speaking, it's extremely difficult to actually write programs in machine language, so virtually all designers use an intermediate step: They write the software in *assembler* or a *high-level language* (such as *BASIC* or *C*). They then use an *assembler* or *compiler* to convert the code to the machine language that the PIC can understand. You can usually determine which type of code is in a file by the file name's extension:

- HEX—machine language files that can be loaded directly into the PIC
- ASM—files written in assembler. They must be converted into HEX before they can be loaded into the PIC.
- C, H—files written in *C*. They must be compiled before they can be used. Depending on the compiler used, they might be converted into .ASM files, or directly to HEX.
- BAS—files written in *BASIC*. They must be compiled before they can be used.

For *BASIC* and *C* files, you cannot use just any compiler to convert them into HEX files. You must use a compiler *specifically designed* to generate instructions that can be read by the PIC.

The most popular tool used to convert

Table 2

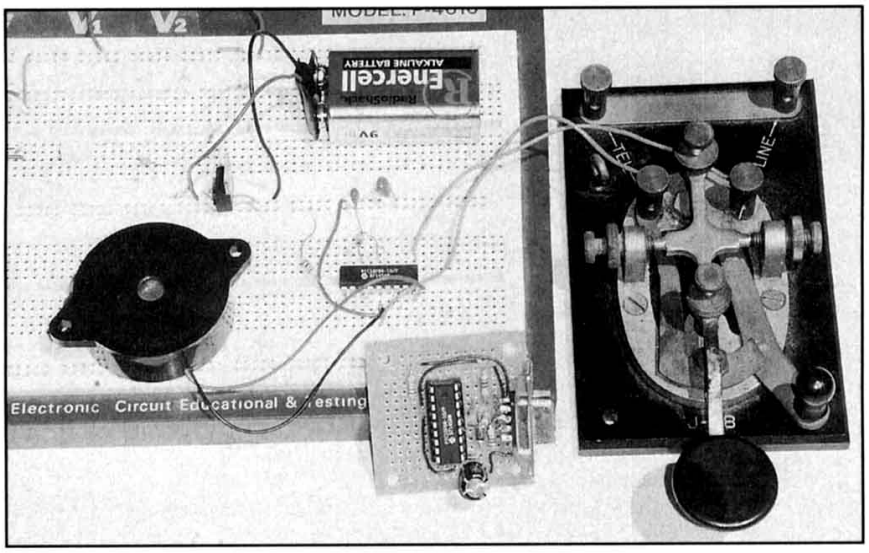
An Assembly Language Program for a PIC Morse Code Generator

```
            list    p=16f84
              radix   hex
              __config  0x3FF1
dahlen      equ     d'99'  ;  <==controls code speed..
                            ;increase
ditlen      equ     d'33'  ;  <==(though not over 255) for
            org     0x000       ;slower code, decrease for
            MOVLW   00          ;faster code. Make sure the
            MOVWF   0A          ;top number is three times
                                ;the bottom.
            GOTO    start
            NOP
            NOP
time        MOVF    10,W
            BTFSC   03,2
            GOTO    endtime
uptop       MOVLW   01
            MOVWF   0D
upagn       CLRF    0C
doagn       DECFSZ  0C,F
            GOTO    doagn
            DECFSZ  0D,F
            GOTO    upagn
            MOVLW   4A
            MOVWF   0C
upone       DECFSZ  0C,F
            GOTO    upone
            DECFSZ  10,F
            GOTO    uptop
endtime  RETLW  00

dah         MOVLW   01      ;subroutine to do a dah
            MOVWF   0E
agn3dah MOVLW           dahlen
            SUBWF   0E,W
            BTFSC   03,0
            GOTO    enddah
            BSF     06,1    ;turn on pin B1
            MOVLW   01
            MOVWF   10
            CALL    time    ;wait 1 millisecond
            BCF     06,1    ;turn off pin B1
            MOVLW   01
            MOVWF   10
            CALL    time    ;wait 1 millisecond
            MOVF    0E,W
            INCF    0E,F
            GOTO    agn3dah ;loop up to do it again.
enddah      MOVLW   3C      ;add a small delay
            MOVWF   10
            CALL    time
            RETLW   00

dit         MOVLW   01      ;subroutine to do a dit
            MOVWF   0E
agn3dit     MOVLW   ditlen
            SUBWF   0E,W
            BTFSC   03,0
            GOTO    enddit
            BSF     06,1    ;turn on pin B1
            MOVLW   01
            MOVWF   10
            CALL    time    ; wait 1 millisecond
            BCF     06,1    ;turn off pin B1
            MOVLW   01
            MOVWF   10
          CALL      time    ;wait 1 millisecond
            MOVF    0E,W
            INCF    0E,F
            GOTO    agn3dit ;loop up and do it again.
enddit      MOVLW   3C      ;add a small delay
            MOVWF   10
            CALL    time
            RETLW   00

lspace              0xB4    ;subroutine to make a
            MOVLW           ;letter space
            MOVWF   10
            CALL    time
            RETLW   00

start       CLRF    04      ;MAIN PROGRAM
                            ;STARTS HERE
            MOVLW   0xFD
            TRIS    6

top         CALL    dit     ;modify this code to
                            ;xmit the
            CALL    dah     ;CW you want to send
            CALL    dah
            CALL    lspace  ;lspace is a pause for
                            ;the space between
            CALL    dit     ;letters. Include it after
                            ;each letter.
            CALL    dit     ;As it appears here, the
                            ;IDer will
            CALL    dah     ;transmit W2FS
            CALL    dah
            CALL    dah
            CALL    lspace
            CALL    dit
            CALL    dit
            CALL    dah
            CALL    dit
            CALL    lspace
            CALL    dit
            CALL    dit
            CALL    dit

            MOVLW   01
            MOVWF   0E
                            ;this code programs the
                            ;delay between IDs.
loop        MOVLW   d'150'  ;<= 4 x this number =
                            ;number of seconds
            SUBWF   0E,W    ; between IDs when
                            ;using a 4 MHz resona-
                            ;tor.
            BTFSC   03,0
            GOTO    bottom
            MOVLW   10
            MOVWF   0F
again       MOVLW   0xFA
            MOVWF   10
            CALL    time
            DECFSZ  0F,F
            GOTO    again
            MOVF    0E,W
            INCF    0E,F
            GOTO    loop
bottom      GOTO    top
            END
```

Creating a PIC-Based IDer

Table 2 contains the assembly-language code that makes a 16F84 PIC generate Morse code. The code is designed to work with the circuit of Figure 1, using a ceramic resonator or a 3.5795-MHz colorburst crystal. Audio output is obtained at pin RB1. The program is designed to be easy to modify to insert your choice of call sign, Morse code speed and transmit interval. To change the Morse code speed, alter the lines labeled "dahlen" and "ditlen." To change the ID interval, change the number between the apostrophes in the line labeled "loop." Change the lines that start with the label "top" to change the call sign (or other text) that is to be transmitted by the chip. Follow the pattern shown in the program and make sure you put an "lspace" after the end of each letter. If you plan to transmit text other than just a call sign, you can do this by adding extra "lspace" instructions to create longer delays between the words. When altering this code, be sure to keep the label "top" on the same line as the first code element to be sent.

If you type in the program yourself, you may leave out the comments on each line by dropping the semicolon and the text that follows it. Blank lines can either be left in or deleted as you prefer. Alternatively, you can download this file from the ARRL's FTP site and edit it. The program may look long, but it only uses about 10% of the program capacity of a 16F84.



CPO on a breadboard—keyed by one of the famous J-38s. A perfboard version of the Ludwig Catta (Ludi) PIC programmer is in the foreground.

assembly language files to HEX files is available from Microchip itself. Called *MPASM*, it is easy to use—best of all, it is *free*! You can obtain *MPASM* from Microchip's Web site (see note 14). You can write the assembly language instructions using any editor that handles plain ASCII text (such as *Windows' Notepad* or DOS's *Edit*), then use *MPASM* to compile the code into a HEX file.

You also need software to load the HEX file into the PIC. The best program I've found for doing this is called *PIX* (see note 15). It supposedly is a DOS program, but I've had trouble getting it to run when my computer is booted in MS-DOS! However, *PIX* runs fine when I run it in a DOS box within *Windows* (that's a switch!). To get *PIX* to work with the programmer of Figure 2, first edit the *PIX.CFG* file that comes with the program. Find the two lines that say:

Port=LPT1

and

Programmer = Shaer

Place a semicolon at the start of each line. Then find the line that says:

;Programmer=Ludi

and remove the semicolon. Next, add a line that says:

Port=COM*x*

where *x* is the number of the serial port to which your programmer is connected.

It is important to use a *short* serial cable to connect your programmer to your computer. In building the programmer in Figure 2, I simply glued a female DB-9 connector onto the programmer itself and then plugged the programmer directly into the serial port on my computer. In any case, do not use a cable longer than about a foot.

When you run the program with the simple PIC programmer hooked up to your serial port, you will probably see a dialog box that says, **MODEM DETECTED** or **NO/BAD HARDWARE. NOT TRUE CONTINUE.** Highlight the answer **YES** and press **ENTER**. The program will then start. This program allows you to read and write code to PICs and erase PICs. It will even disassemble the HEX code to show you the assembly language instructions. Pressing function key **F3** allows you to load your HEX file into the *PIX* program. When you press the **F9** key, the program loads into the PIC.

In summary, you use an ASCII text editor to write your assembly language instructions and save the file with an ASM extension. Then, use *MPASM* to assemble the ASM file into a HEX file. Finally, use *PIX* to load the HEX file into your PIC. To test your program, simply plug the programmed PIC into the completed target circuit and apply power. It should automa-

PIC Resources

Books

Benson, David, *Easy PIC'n: A Beginner's Guide to using PIC 16/17 Microcontrollers* (Kelseyville, California: Square 1 Electronics, 1996). It's a good introduction to PIC chips and assembly programming.

Benson, David, *PIC'n Up the Pace: PIC 16/17 Microcontroller Applications Guide* (Kelseyville, California: Square 1 Electronics, 1997). Essential reading if you are serious about learning assembler. If you plan to use a *C* or *BASIC* compiler, you need not have this book.

Predko, Myke, *Programming and Customizing the PIC Microcontroller* (New York, McGraw-Hill, 1998). Somewhat more advanced, this book contains a good discussion of the PIC architecture and lots of projects.

Peatman, John B., *Design With PIC Microcontrollers* (New York: Prentice Hall, 1997). This is a well written college-level text on PIC microcontrollers.

PIC Programmers

PIC-1A—Parallel-port programmer for all 16x6/16x7/16x8 chips. Available from ITU Technologies, 3704 Cheviot Avenue, Suite 3, Cincinnati, OH 45211, tel 888-448-8832, 513-661-7523, fax 513-661-7534; e-mail **sales@itu-tech.com**; Web site **http://www.itu-tech.com**. Kit: $39; wired and tested, $59. Add $9 for a ZIF socket. I have found this firm's customer service to be excellent.

For about $20, you can build more complex programmers that are similar to the ITU unit. See, for example David Tait's Web pages:

http://www.man.ac.uk/~mbhstdj/files/index.html
http://www.man.ac.uk/~mbhstdj/piclinks.html

Compilers

PCM, a *C* compiler available from Custom Computer Services, Inc, PO Box 2452, Brookfield, WI 53008; tel 414-781-2794 ext 35; **http://www.ccsinfo.com/picc.html**; price, $99.

Hi-Tech's PIC *C* compiler. Although this compiler is too expensive for most of us ($850), a working demo is available for free that will compile small projects. HI-Tech Software, LLC, Suite 105, 7830 Ellis Rd, Melbourne, FL 39204; tel 800-735-5717; **http://www.htsoft.com/**.

PicBasic Compiler, a *BASIC* compiler available from microEngineering Labs, Inc, Box 7532, Colorado Springs, Colorado 80933; tel 719-520-5323; **http://www.melabs.com/mel/**. Price: $99.95.

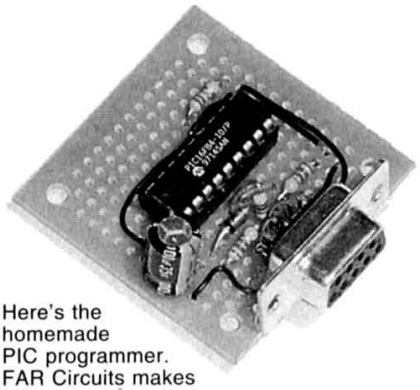
PIC Chips and Other Parts

Digi-Key Corporation, 701 Brooks Ave S, Thief River Falls, MN 5601-0677; tel 800-344-4539; **http://www.digikey.com**. Digi-Key also stocks ceramic resonators and ZIF sockets.

ITU Technologies, see information above.

JDR Microdevices, 1850 South 10th St, San Jose, CA 95112-4108; tel 800-538-5000; **http://www.jdr.com**.

[Author's note: This is not an exhaustive list of resources. There is a wide range of resources available for PICs including complete development environments that sell for over $2000. I have focused here only on those resources that are within the budget of a typical amateur.—*John Hansen, W2FS*]



Here's the homemade PIC programmer. FAR Circuits makes PC boards for this project and the others in this article.

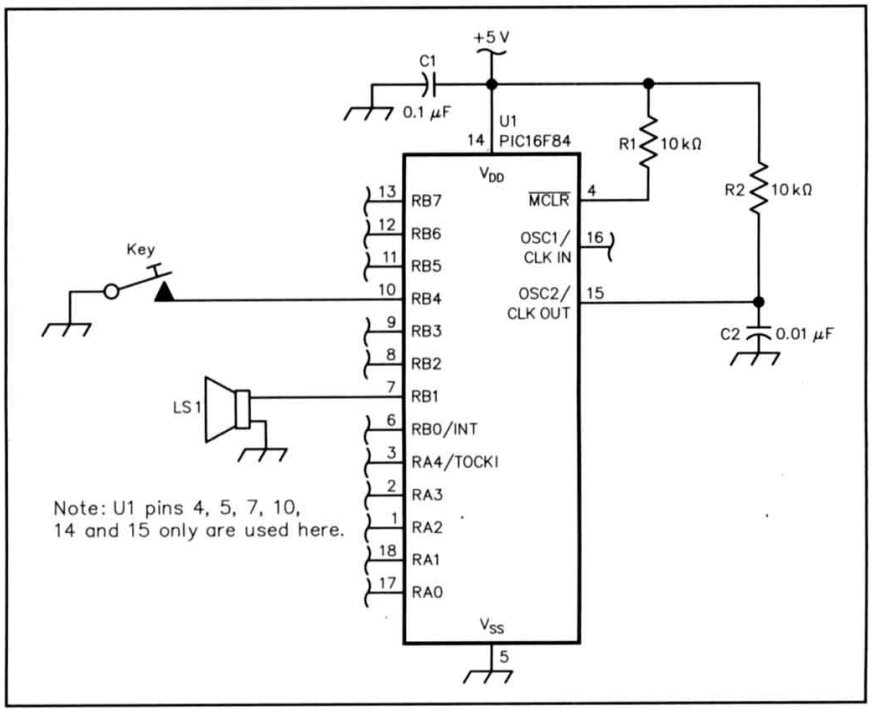


Figure 3—Code-practice oscillator schematic using a PIC microcontroller. Unless otherwise specified, resistors are 1/4 W, 5% tolerance carbon-composition or film units. Equivalent parts can be substituted.

LS1—Same as used in the circuit of Figure 1.

U1—PIC16F84 microcontroller (see PIC Resources sidebar)

tically begin executing its program.

Building Your First Program

Now that we can program the PIC, we need a program to make it do something. Learning to write assembly-language programs for the PIC is no trivial matter. My intention here is not to provide a tutorial in assembler, but to give you a sense of what it is like.[16]

The circuit of Figure 3 (quite similar to that of Figure 1) can be used to construct a simple code-practice oscillator (CPO). Sure, there are simpler ways to build a CPO than using a computer-on-a-chip, but this application does provide a good introduction to programming PICs. Furthermore, you can build this project using only two resistors, two capacitors, the 16F84 PIC and a piezo speaker, making this just about as inexpensive a way to build a CPO as the more traditional methods. The CPO circuit of Figure 3 uses an RC timing circuit for the clock. The key is connected between pin RB4 and ground. The speed at which the processor runs can be altered by changing the value of C2. Any capacitance value between 220 pF to 0.01 µF will work, with lower capacitance values resulting in a higher-pitched oscillator.

Using your text editor, enter the code given in Table 1 and name the saved file CPO.ASM. Enter the code in three columns as shown. You don't need to have the exact spacing between columns shown above, but you do need to have your code in three columns. Note the two underscores preceding the word "config." In Table 1, they appear as one long underline.

The first line of code tells the assembler which PIC is being used. In the second line, the internal configuration of the clock and the timers is set up. Here we specify that we are using a RC circuit for the clock. If we wanted to use a colorburst crystal or a ceramic resonator, we would have specified 0x3FF1.

The third line makes the code easier to read. The internal location for the Port B I/O lines (RB0 through RB7) is 0x06. This line of code specifies that we will call this location "portb" instead. The "org" instruction in the next line says where the program should start; in this case, is at the first instruction entered in the chip (at location 0).

The next two lines enable internal pull-up resistors on all of the Port B pins that are used as inputs. This means that 5 V is applied to each of these pins through a current-limiting resistor. This happens inside the chip and requires no external components. As a result, each of the input lines sees a 5 V signal, unless you short the pin to ground. When the line is grounded, the pin no longer sees 5 V. This is the mechanism used for keying the CPO. When you key the CPO, it momentarily connects pin RB4 to ground. The program detects that this pin is low, and generates a tone as long as the pin is grounded.

The next two lines determine which of the PIC's pins will be inputs, and which will be outputs.[17] The PIC assumes the pins are inputs, unless it is specifically told to make them outputs. In this case, the value FD (11111101 in binary) changes RB1 to an output. The next line is labeled "start" so that the program can loop back up to this line when it needs to. Once it gets to this point, all the program does is repeatedly execute the remaining lines of the program (except the end statement). This forms what is called an *infinite loop* because the program just continues doing this until you shut off the power to the microprocessor. In most programming environments infinite loops are avoided at all costs—they are often the things that cause computers to "hang" when they occur unintentionally. With PICs, however, infinite loops are very common. They are used whenever you want the PIC to continue running the same program over and over again until the power is shut off.

The line labeled "start" says that if pin four on Port B (RB4) is low (grounded), skip the next line of code. That next line just sends the program back to "start." Thus, if pin RB4 is high (+ 5 V), the program continues to alternate between these two lines of code until pin RB4 is grounded. When pin RB4 is grounded, the instruction to go back to "start" is skipped. The next two lines of code take pin RB1 high (bsf) and low (bcf) again. Then the program goes back up to "start." The effect of this is that if pin RB4 is grounded, the program causes pin RB1 to alternate between 5 V and ground at the same rate that the clock of the microprocessor is running. This produces a rectangular wave (not quite square) at about 700 Hz. Because we are using an RC circuit to clock the chip, it runs at a relatively low frequency. If we had used a 4 MHz crystal instead (which runs the microprocessor clock at 1 MHz), it would have been necessary to insert additional delay instructions to slow down the rate at which pin RB1 alternates between high and low. A CPO that runs at 1 MHz is not very useful! You won't hear it!

That's all there is to it. After you have saved the CPO.ASM file, use *MPASM* to

compile it into a HEX file using the command: **MPASM CPO**

If any errors are reported during the compiling process, it means you have mistyped something. By viewing the *CPO.ERR* file, you can find out which lines contain the errors. After the program assembles without error, use *PIX* to load the HEX file into your 16F84 PIC.

When you get this project running, try changing the code to lower the tone. Or, try having the PIC light an LED (as in Figure 1) *and* generate a tone. Each time you change the program, you need to rerun *MPASM* to reassemble the code and reload it into the chip with *PIX*. By experimenting with variations of this basic circuit, you can better understand how assembly-language instructions work.

Isn't There an Easier Way?

Of course, there is an easier way to do this. You can use a high-level language such as *C* or *BASIC* to write your program. For that, however, you need a compiler that converts your *BASIC* or *C* code into assembler or machine language. The cheapest of these compilers costs about $100. However, there are indications that some shareware compilers are beginning to come into the market. I've had very good luck with the Custom Computer Services, Inc compiler called *PCM*. It costs just under $100, and has built-in routines to make serial communication particularly easy. If you are going to do a lot of work with PICs, spending the money on a compiler may be worth considering.

Time to PIC Up Your Toys

What I enjoy most about PICs is that they allow me to combine my interests in hardware and software development. The circuits themselves are much easier to design because most of the heavy lifting is done by the code inside the PIC, or by specialized chips that can easily be mated to the PIC. The software isn't all that hard to write either, especially if you use a higher-level language and a compiler. This makes it possible for those of us with little or no formal background in either hardware or software engineering to do some amazing things with these inexpensive chips. Give it a try! You will find experimenting with these new toys to be an enormous amount of fun!

Notes

[1]Jeff Otterson, N1KDO, Peter Gailunas, KA1OKQ, Richard Cox, N1LTL, "Build a $60 Talking Repeater Controller," *QST*, Feb 1997, pp 37-40.

[2]Bryan H. Suits, WB8WKN, "DTMF/LT Decoding Made Easy," *QST*, Apr 1997, pp 34-36.

[3]Jay Craswell, WB0VNE (now W0VNE), "The PortaPeater," *QST*, Apr 1997, 37-39.

[4]Neil Heckt, "A PIC-Based Digital Frequency Display," *QST*, May 1997, pp 36-38.

[5]Lee Richey, WA3FY, "The CycleMaster," *QST*, Sep 1997, pp 37-42.

[6]Gary M. Diana, Sr, N2JGU, and Bradley S. Mitchell, WB8YGG, "TiCK-2—The Tiny CMOS Keyer 2," *QST*, Oct 1997, pp 42-45.

[7]Ron Alspaugh, W6NKS, "A Computer Keyboard CW Encoder," *QST*, Dec 1997, pp 32-35.

[8]Bob Anding, AA5OY, "A PIC of an IDer," *QST*, Jan 1998, pp 36-38.

[9]Steven C. Hageman, "Build Your Own Network Analyzer—*Part 1*," *QST*, Jan 1998, pp 39-45; *Part 2*, Feb 1998, pp 35-39.

[10]John Hansen, W2FS, "An Inexpensive, Remote-Base Station Controller Using the Basic Stamp," *QST*, May 1998, pp 33-37.

[11]Microchip Technology, 2355 West Chandler Blvd, Chandler, AZ 85224-6199; tel 602-786-7200, fax 602-899-9210; **http://www.microchip.com**.

[12]Because these chips are so similar, any project you find that uses a 16C84 (such as the Talking Repeater Controller; see Note 1) can be built with a 16F84.

[13]ITU Technologies, 3704 Cheviot Ave, Suite 3, Cincinnati, OH 45211; tel 888-448-8832, 513-661-7523, fax 513-661-7534; e-mail **sales@itutech.com**; Web site **http://www.itutech.com**.

[14]MicroChip Technology, **http://www.microchip.com**. At that site, you will also find a free program called *MPLAB*. It is a complete development environment including an editor, a simulator and an assembler.

[15]Available at **http://home5.swipnet.se/~w-53783/**.

[16]For a good beginner's tutorial in assembler for the PIC, see David Benson, *Easy PIC'n: A Beginner's Guide to using PIC 16/17 Microcontrollers*, (Kelseyville, California: Square 1 Electronics, 1996).

[17]Microchip considers the *option* and *tris* instructions outdated. However, they will work in most applications. Because the alternative methods for configuring the I/O pins and pull-up resistors are somewhat more complicated, in the interest of simplicity, the first approach is selected for this project.

Formerly licensed as WA0PTV, John Hansen, W2FS, has been an Amateur Radio operator since 1966. His interests include satellites, digital communication and "homebrewing." For a number of years, John was the editor of The AMSAT Journal, *and is the 1993 recipient of the ARRL Atlantic Division Technical Achievement Award for his work on digital-satellite-gateway software. John maintains a career as a Professor of Economics at the State University of New York to provide enough income to purchase radio equipment and PICy things. You can contact John at 49 Maple Ave, Fredonia, NY 14063; e-mail* **hansen@fredonia.edu**.

Photos by John Martinson, WB2WXN QST

New Products

CONQUERING COMMUNICATIONS INTERACTIVE TEXT

◊ Delmar Publishers offers *Conquering Communications*, by H. Paul Shuch, N6TX. Described as an interactive textbook for students of electronic communications, the software allows students to learn at their own pace. Its four units (called Basic Blocks, Mastering Modulation, Deciphering Detection, and Probing Propagation) teach the principles of analog modulation, demodulation, signal processing and propagation.

It is designed for hams who want to expand their electronics knowledge, as well as for lower-level college students. Each chapter consists of tutorials, review questions, simulations, student interactions and an end-of-chapter quiz.

Shuch has written more than 150 articles and books on such subjects as UHF communications, amateur satellites, and the search for extraterrestrial life. He is executive director of the SETI League Inc, and a former professor of electronics.

Price: $38.95. For more information, contact Delmar Publishers, Box 15015, Albany, NY 12212; tel 800-865-5840; **http://www.delmar.com/**.

MFJ'S RAPIDBATTERY CHARGER FOR KENWOOD H-Ts

◊ MFJ's Model 641K *RapidBattery* Charger handles NiCd and NiMH battery packs for Kenwood H-Ts using MFJ's exclusive *RapidBattery* technology to determine whether the battery is fully charged—eliminating overcharging and helping to extend battery life.

Switching from quick charge to trickle charge modes automatically, the '641K uses interchangeable charging slots to eliminate charging and voltage mishaps that could destroy your batteries. The unit ships with one charging slot (choose one from your battery pack from MFJ's catalog). Additional slots are available for Kenwood PB-6, 7, 8, 9, 11, 30, 32, 33, 34, KNB-5, 5A, 6, 6A, 7, 7A and 9A battery packs. The '641K features microprocessor control and an LED charge-status indicator.

Price: $49.95. The charger is backed by MFJ's "No Matter What" one-year limited warranty. For more information, see your local Amateur Radio products dealer or contact MFJ, PO Box 494, Mississippi State, MS 39762; tel 800-647-1800; fax 601-323-6551; e-mail **mfj@mfjenterprises.com**; **http://www.mfjenterprises. com**. QST