by Peter Best

The Ever-Shrinking µC

Six Pins and One MIP — If You Can See It!

The Fuzzball Rating System

To find out the level of difficulty for each of these projects, turn to Fuzzball for the answers.

The scale is from 1-4, with four Fuzzballs being the more difficult or advanced projects. Just look for the Fuzzballs in the opening header.

You'll also find information included in each article on any special tools or skills you'll need to complete the project.

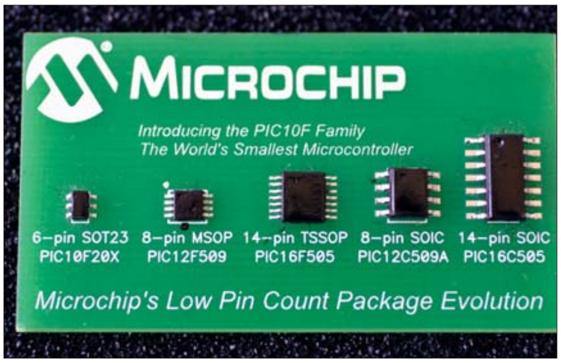
Let the soldering begin!

As you might imagine, I spend lots of my time reading technical books and electronic journals. Recently, to cut the ice, I've been reading about the exploits of men who flew and crewed B-17 bombers for the Eighth Air Force in World War II. If you recall your history, the flyers of the "Mighty Eighth" had one of the highest casualty rates in the war. Why? Because they were doing something that had never been done before.

On a daily basis, these men braved high altitude cold, bad weather, anti-aircraft fire, enemy fighters, and mid-air collisions with their own bombers to attempt to put their thin-skinned aircraft, tons of aviation gasoline, and bombs over a predetermined ground target in broad daylight.

Electric flying suits (a precursor to electric sleeping blankets), wool-lined flying jackets and shoes, and oxygen were essential to the survival of the crews while on their dangerous missions. I'm not going to take you flying on a B-17, but — like the B-17 flyboys — we will today be doing something no one else has done. Snap up that flight jacket and take a deep breath from that oxygen mask because — for the rest of this article — you will be breathing pure oxygen at high altitudes. We're going to fly some highly technical missions on Little Bits, which is powered by a pair of PIC10F206 engines.

Figure 1. Yep, the PIC10F206 is a tiny bugger, but it's still big enough to allow you to hang some wire-wrap wire off of its pins or solder it onto a set of SOT-23 copper pads.



The PIC10F20X Family of Microcontrollers

The PIC10F206 is the largest of the tiniest microcontrollers in the world. The PIC10F200 and PIC10F204 microcontrollers contain 256 words of program Flash and 16 bytes of SRAM. The PIC10F202 and PIC10F206 microcontrollers are loaded with double the program Flash of the PIC10F200 and PIC10F204 microcontrollers (512 bytes) and contain eight more bytes of SRAM (24 bytes).

The differentiator of the same-sized variants of the PIC10F20X family is the addition of an onboard comparator found in the PIC10F206 and PIC10F204 silicon. Both the PIC10F206 and PIC10F204 comparator modules are governed by an internal absolute voltage reference with all comparator inputs and outputs visible on their respective multiplexed I/O pins. The output of the comparator can also be configured not to be shown on the microcontroller I/O pin.

Each PIC10F20X microcontroller has four multiplexed I/O pins, which include three bi-directional I/O pins (GP0, GP1, and GP2) and one input only pin (GP3) when all of the I/O pins are configured for general-purpose I/O mode. All of the GPIO (General-Purpose Input Output) pins except GP2 can be configured with weak pull-ups and wake-up on change operation. Each I/O pin can source or sink 25 mA, which results in a total of 75 mA that can be sourced or sunk by the PIC10F20X microcontroller's I/O port.

Thus, the PIC10F20X family of microcontrollers can directly drive small resistive loads and LEDs.

Clocking for the PIC10F20X microcontrollers is provided internally. A 4 MHz internal clock supplies 1 µS instruction cycles that drive the PIC10F20X's processing engine. The internal clock is factory calibrated and is accurate to ±1%. Cycles from the internal clock can be used to drive the PIC10F20X's on-chip eight-bit timer. The eight-bit timer can also be configured to be driven externally via the T0CKI pin (GP2) or from the output of the comparator. Like the rest of Microchip's PIC microcontrollers, the PIC10F20X series also includes an integral WDT (Watchdog Timer), program Flash code protection, ICSP (In Circuit Serial Programming) capability, and sleep mode.

The typical PIC10F20X is very similar logically to the PIC12F508 and PIC12F509 microcontrollers, but — as you can see in Figure 1 — the PIC10F20X is a much smaller beast. With only four I/O pins, one would wonder what could be done with such a miniscule microcontroller. Take another deep breath — we're going up ...

Little Bits

Even though the PIC10F20X microcontrollers are tiny, there is no reason why the average Microcontroller Joe can't put them to use. Believe it or not, you can actually solder some wire-wrap wire to each of the PIC10F20X's six pins and breadboard the little bugger just like you would any other electronic part. On the other hand, you could also put together a specialized PIC10F20X printed circuit board that would include a regulated +5 VDC power source, an ICSP socket, and some LEDs with a breadboard

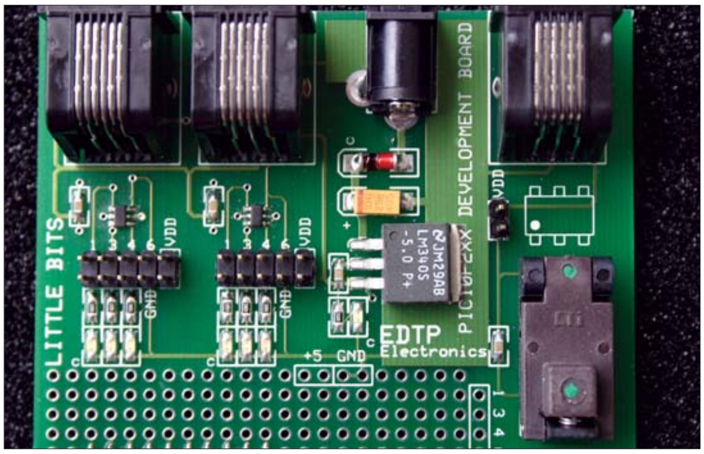
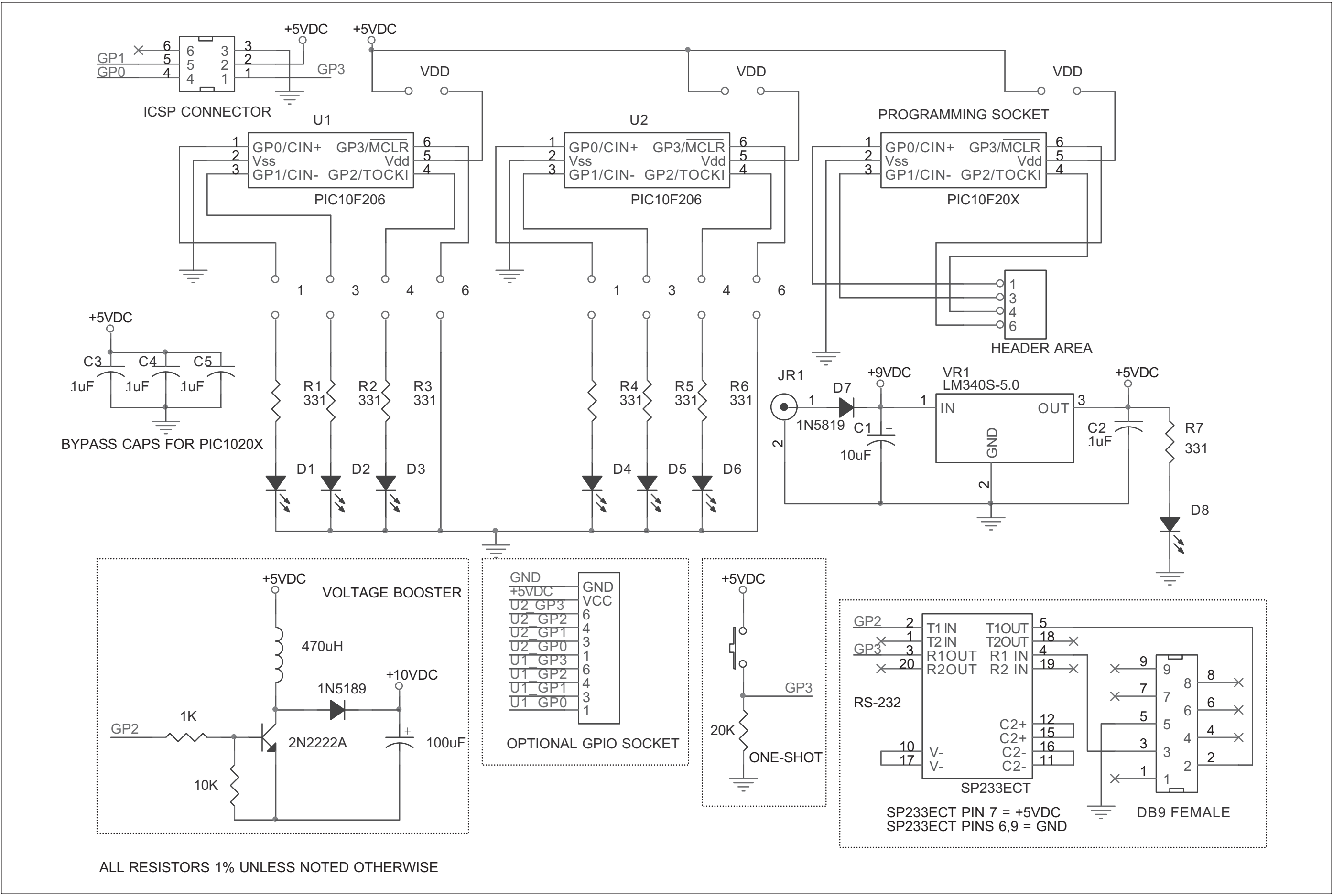


Figure 2. Little Bits is a combination of a regulated 5 VDC power supply and a pair of PIC10F206 microcontrollers. A third PIC10F206 can be programmed and run from the Wells-CTI SOT-23 programming socket position. Two banks of jumper-selectable LEDs are included to free you from having to pull out that logic probe. This photo also shows the dual-row 20-pin female header I added to allow easy access to the pair of Little Bits' PIC10F206 microcontrollers.

Figure 3. The ICSP connector in the schematic is common to all of the PIC10F20X microcontrollers. Jumper blocks allow for easy configuration of the Little Bits microcontrollers.



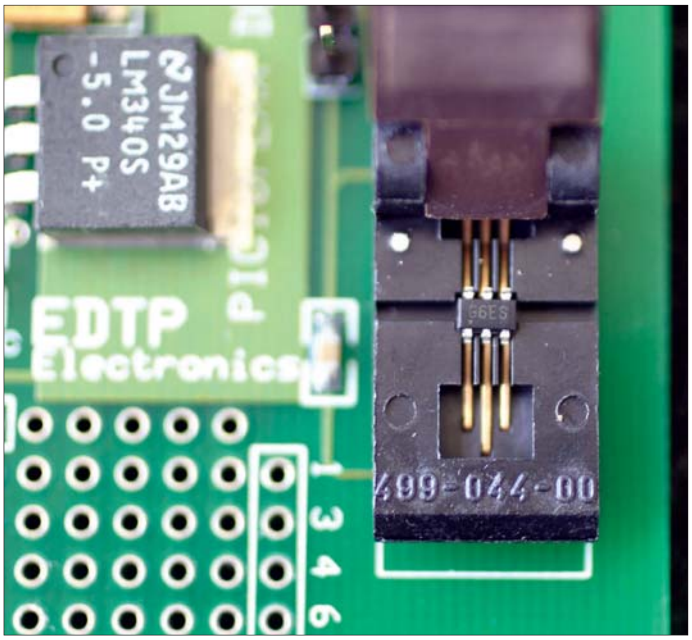


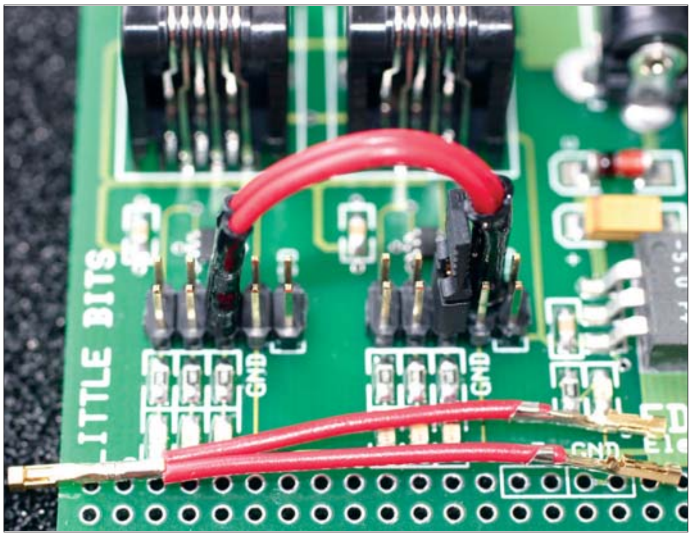
Figure 4. This is a view of a PIC10F206 in the jaws of the Wells-CTI programming socket. The orientation of the PIC10F20X parts inside the programming socket is silk screened on the Little Bits printed circuit board, just above the programming socket.

area. I went with Plan B and assembled the dual PIC10F206-based development board you see in Figure 2. The design you're looking at in Figure 2 is called Little Bits and it can program and run up to three PIC10F206 microcontrollers simultaneously.

As you can see in Figure 2, Little Bits is the classic implementation of the PIC10F206 microcontroller. Two of the PIC10F206 microcontrollers can be directly attached to Little Bits' onboard banks of LEDs via standard .1 inch jumper pin sets. There may be situations where you may not wish to run all of the PIC10F206 microcontrollers at the same time. So, power to each individual PIC10F206 is controlled by a VDD jumper, which also allows quick individual PIC10F206 or PIC10F20X power-on resets, depending on which PIC10F20X you have in the programming socket.

Speaking of programming sockets ... For those who need to program a PIC10F20X for use in an external circuit, Little Bits is equipped with a Wells-CTI SOT-23 programming socket and a third ICSP interface. The Wells-CTI socket position can also be used as yet another PIC10F20X hardstand, as the PIC10F20X I/O pins for the Wells-CTI position are brought out to a header pin area on the Little Bits Development Board. Take a close look at Figure 4 to get a view of the mechanics behind the Wells-CTI programming/burn-in socket.

Figure 5. The Y connector is pulling a signal from GP2 of the PIC10F206 running the TOGGLER program and feeding it into the input pin (GP3) of the PIC10F206, emulating an inverter. The two points of the Y allow the viewing of the source logic level changes. The PIC10F206 inverter's output is GP2 and is jumpered to an LED for easy viewing of the inverter output logic state changes.



Little Bits' Logic

The PIC10F206 was designed to be a utilitarian microcontroller that can stand in for standard logic ICs, generate and manipulate clocks, and perform small, programmable tasks. With some help from the HI-TECH PICC PIC10F20X C compiler, Microchip's MBLAB, and a Microchip MPLAB ICD 2, I'll show you how easy it is to apply some logic with the PIC10F206 and Little Bits.

Before we begin to code our Little Bits microcontrollers, there are some things on the hardware honey-do list that we need to finish up. As you can see in Figure 3, the two banks of LEDs that are common to the pair of permanently mounted PIC10F206s are separated from the PIC10F206's I/O pins by a jumper connection. To be able to see what state each particular I/O pin is in requires a jumper between the I/O pin and its respective LED. The problem is that, when an LED is connected to an I/O pin using a jumper block, there is no way to get the signal to and from the I/O pin/LED combination, as the jumper block is obscuring the I/O pin interface.

To circumvent that problem, I came up with a little Y connector that consists of a couple of pieces of standard hook-up wire terminated with three .025 inch square female terminal post pins (Jameco part number 100765CX). The idea is to connect the PIC10F206 I/O pin to the LED with the double point of the Y jumper assembly and feed or receive the I/O signal with the single point of the Y jumper assembly. To keep the Y cable points from coming into electrical contact with each other, I added a bit of shrink tubing to each of the exposed points of the Y connector. Y connectors with and without shrink tubing and their uses are shown in Figure 5.

Since I'm not pushing a single application here, we'll be adding auxiliary supporting hardware as we gain altitude with our PIC10F206 projects. So, rather than overload the Little Bits breadboard area, I added a dual-row 20-pin, .1 inch-spaced female connector (Jameco part number 70826) to my Little Bits to allow the association and disassociation of external hardware that is fitted with a matching set of dual-row male .025 inch header posts. This will come in handy when we teach the PIC10F206 to control external devices and communicate with the outside world.

PIC10F206 as an Inverter

Let's begin by casting the PIC10F206 as an inverter: the simplest of which is a 7404 or 74XX04, where XX can be HC, LS, etc. The typical logic inverter is a two-pin device that consists of an input and a complementary output. Since we have four I/O pins on a PIC10F20X, we can implement up to two inverter modules per microcontroller. So, let's choose GP3 — the input-only I/O pin — as our inverter #1 input and GP2, a bi-directional I/O pin, as our inverter #1 output.

The reason for choosing GP3 is rather obvious. The reason I chose GP2 as the output is because I can leave the MPLAB ICD 2 connected to Little Bits and still see the inverter code function via a LED patched into the GP2 output pin. The latest version of MPLAB IDE includes the ability to reset the target PIC and release the reset to the target PIC from within the MPLAB IDE. Therefore, I can write my C code, program the PIC10F206, and start and stop the PIC10F206's code execution all from the same MPLAB IDE window.

Every bit of code you'll see from now on will be C with a HI-TECH PICC flavor. The HI-TECH folks are way ahead of the curve

Listing 1. Chances are you'll never need to inspect the assembler code that the HI-TECH PICC C compiler generates. My purpose of showing it to you here is to point out how good the assembled C code really is.

```
//*********************************************************************
//*  HI-TECH C SOURCE CODE FOR INVERTER MODULE
//*********************************************************************

#include <pic.h>

__CONFIG(MCLRDIS & WDTDIS & UNPROTECT);

void main()
{
 TRIS = 0b11111011;      //GP2 = output : GP3 = input
 FOSC4 = 0;              //GP2 = I/O pin
 CMCON  = 0b11110111;    //comparator off
 OPTION = 0b11001111;    //pull-ups and wake-up off

 while(1)                //loop forever
 {
  if(GPIO & 0b00001000)  //look for a high on GP3
   GP2 = 0;              //GP3 was high
  else
   GP2 = 1;              //GP3 was low
 }//while(1)
}//main

//*********************************************************************
//*  RESULTING ASSEMBLER FROM HI-TECH C COMPILER
//*********************************************************************
     1                                 processor       10F206
     2                                 opt     pw 79
     3                                 psect   __Z18698RS_,global,delta=1
     4                                 psect
ctext0,local,size=512,class=ENTRY,delta=2
     5                                 psect   config,global,class=CONFIG,delta=2
     6                                 psect
text0,local,class=CODE,with=ctext0,delta=2
     7                                 psect   text1,local,class=CODE,delta=2
    18                                 psect   __Z18698RS_
    19   008
    20   008                           ;#
    21
    22                                 psect   config
    23   3FF   FEB                     dw      4075     ;#
    24
    25                                 psect   text0
    26   1F1                      _main
    27                              ;inverter.c: 33: TRIS = 0b11111011;
    28   1F1   CFB                     movlw   -5
    29   1F2   006                     tris    6
    30                              ;inverter.c: 34: FOSC4 = 0;
    31   1F3   405                     bcf     5,0
    32                              ;inverter.c: 35: CMCON = 0b11110111;
    33   1F4   CF7                     movlw   -9
    34   1F5   027                     movwf   7        ;volatile
    35                              ;inverter.c: 36: OPTION = 0b11001111;
    36   1F6   CCF                     movlw   -49
    37   1F7   002                     option
    38                              ;inverter.c: 38: while(1){
    39   1F8                      l3
    40                              ;inverter.c: 39: if(GPIO & 0b00001000)
    41   1F8   766                     btfss   6,3      ;volatile
    42   1F9   BFC                     goto    l5
    43                              ;inverter.c: 40: GP2 = 0;
    44   1FA   446                     bcf     6,2
    45                              ;inverter.c: 41: else
    46   1FB   BF8                     goto    l3
```

(continued ...)

(Listing 1, continued)

```
    47  1FC                         l5
    48                                 ;inverter.c: 42: GP2 = 1;
    49  1FC  546                            bsf      6,2
    50                                 ;inverter.c: 43: }
    51  1FD  BF8                            goto     l3
    52
    53                                      psect    text1
    54  0000

HI-TECH Software PICC Macro Assembler V8.05
Symbol Table                                                   Fri Aug 20 13:44:07 2004

          l3 01F8            l5 01FC          _main 01F1        start 0000
```

and at this writing have the only fully functional PIC10F20X compiler — C or BASIC. You may be asking yourself why I'm using C when assembler would be more compact and efficient. If you are indeed asking yourself this question, you're in for a surprise — breathe deeply and pull back the stick.

The C rendition of our PIC10F206 inverter code is shown in its entirety in Listing 1. The #include <pic.h> statement brings in all of the predefined names and locations that the HI-TECH PICC C compiler needs to associate the names and locations in our C source code to the PIC10F206 hardware. In fact, the pic.h include files bring in a specific set of definitions for the PIC10F206, which is defined in the very first line of the HI-TECH PICC C compiler-generated assembler listing. The __CONFIG statement sets up the PIC10F206 fuses. If you've ever written any PIC assembler, the only unfamiliar parameter is MCLRDIS, which configures the GP3 line as an I/O input. Otherwise, GP3 doubles as the MCLR pin.

The beginning of our inverter C code simply turns off all of the PIC10F206 frills and sets the PIC10F206's internal muxes to configure all of the PIC10F206's four I/O pins as GPIO. All of the inverter's working code lies between the while(1) braces. The while(1){} construct forms an endless loop. Within the endless loop, we are simply looking at GP3 and toggling GP2 in the opposite logical direction of what we see at GP3.

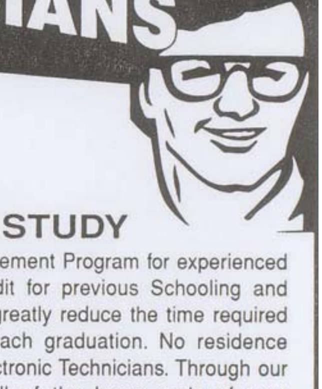
I've also provided the assembler listing that is generated by the HI-TECH PICC C compiler. I've eliminated the line numbers that did not contain any useful information for our discussion. The HI-TECH PICC C compiler collects like kinds of data in what are called psects or program sections. Without going into great detail, lines 3-7 and 18 define the psects. It's a logical process and you can see the names and types of program sections in the actual assembler code that are defined within the initial psect statements.

The HI-TECH PICC C compiler linker uses the psects to group like kinds of data into their appropriate memory areas of the PIC10F206. For instance, code in the psect text0 and psect text1 areas is placed in the PIC10F206's ROM (program Flash) area. Note that configuration fuse data is located in the CONFIG psect. Don't get too wrapped up about psects, as we're going to let the compiler automatically handle them for the PIC10F206.

Take a look at the very last line of Listing 1. You'll notice that the locations of labels l3 and l5 are listed. Also, note that the actual program code is placed into the PIC10F206 program memory, beginning at physical address 0x1F1, which is logically program location 0x0000. That may seem odd, but the HI-TECH PICC C compiler purposely puts things where they are for efficiency.

You've seen the mnemonics shown in the assembler listing beginning at _main in your own PIC assembler programs. Note that the binary numeric arguments in the C source are treated as negative 2's complement binary numbers in the assembler source statements. When using signed binary arithmetic, the most significant bit is used as a sign bit with 1 signifying a negative number and 0 indicating the number is positive. For instance,

ATTENTION! ELECTRONICS TECHNICIANS



EARN YOUR
B.S.E.E.
DEGREE
THROUGH HOME STUDY

Our Highly Effective Advanced-Placement Program for experienced Electronic Technicians grants credit for previous Schooling and Professional Experience, and can greatly reduce the time required to com-plete the program and reach graduation. No residence schooling required for qualified Electronic Technicians. Through our Special Program you can pull all of the loose ends of your electronics background together and earn your B.S.E.E. Degree. Upgrade your status and pay to the engineering level. Advance rapidly! Many finish in 12 months or less! Students and graduates in all 50 states and throughout the world! Established Over 50 Years! Write or call for free Descriptive Literature. (601) 371-1351

COOK'S INSTITUTE
OF ELECTRONICS ENGINEERING
4251 CYPRESS DRIVE, JACKSON, MISSISSIPPI 39212
• FAX: (601) 371-2619

Visit us TODAY on the World Wide Web at http://www.cooks.edu

Circle #40 on the Reader Service Card.

0b11111011 is the 2's complement representation of -5. If that is true, then the negative number 2's compliment rule states that — if I invert every bit within 0b11111011 and add 1 — I should be able to add the inverted result of the original number (0b00000101 or +5) to the original number (0b11111011 or -5) and end up with 0. Let's try that:

```
   11111011    original
+  00000101    flipped + 1
  100000000
```

What's up with that? My TI Voyage 200 in binary mode indicates the same answer, which is decimal 256. Remember, we're only working with eight bits. The carry of a 1 out to the ninth bit indicates that the result of the binary addition is positive. In this case, the binary addition result of 0 (0b00000000) is considered positive. No carry out of the ninth bit would leave the most significant bit at 1, indicating a negative result.

Let's get back to how the inverter code operates. The inverter assembler code is very tight. The C source statements are followed by their assembler counterparts. Do you think you could have written the assembler code from scratch any better than the C compiler did? For those of you who nodded your heads yes, let's add that second inverter.

The dual-inverter C source in Listing 2 is quite a bit different than our simple single inverter code. GP0 has been assigned as the input for Inverter #2 with GP1 acting as the inverter #2 output. The rest of the code is simply looking at GPIO and testing for every possible logical combination of the input pins: GP0 and GP3. The result of the switch (GPIO & 0b00001001) statement determines which case statement will execute. All

Listing 2. It would defeat the purpose of using C, but you could actually write your application in C and then look at the generated assembler to figure out how to write that same application more efficiently with assembler.

```
//**********************************************************************
//*  HI-TECH C SOURCE CODE FOR DUAL INVERTER MODULE
//**********************************************************************
void main()
{
 TRIS = 0b11111001;             //GP3 = input #1 : GP2 = output #1
                                //GP0 = input #2 : GP1 = output #2
 FOSC4 = 0;                     //GP2 is an I/O pin
 CMCON  = 0b11110111;           //comparator off:pullups off:wakeup off
 OPTION = 0b11001111;           //prescaler assigned to WDT

 while(1)                       //loop forever
 {
  switch(GPIO & 0b00001001)
  {
   case 0b00000000:             //inverter #1 input = LOW
    GP1 = 1;                    //inverter #2 input = LOW
    GP2 = 1;
    break;
   case 0b00000001:             //inverter #1 input = LOW
    GP1 = 0;                    //inverter #2 input = HIGH
    GP2 = 1;
    break;
   case 0b00001000:             //inverter #1 input = HIGH
    GP1 = 1;                    //inverter #2 input = LOW
    GP2 = 0;
    break;
   case 0b00001001:             //inverter #1 input = HIGH
    GP1 = 0;                    //inverter #2 input = HIGH
    GP2 = 0;
    break;
  }//switch
 }//while(1)
}//main

//**********************************************************************
//*  RESULTING ASSEMBLER FROM HI-TECH C COMPILER
//**********************************************************************
                                       processor      10F206
                                       opt     pw 79
                                       psect   __Z18698RS_,global,delta=1
                                       psect
ctext0,local,size=512,class=ENTRY,delta=2
                                       psect   config,global,class=CONFIG,delta=2
                                       psect
text0,local,class=CODE,with=ctext0,delta=2
                                       psect   text1,local,class=CODE,delta=2
                                       psect
temp,global,ovrld,class=BANK0,space=1,delta=1
                                       psect   __Z18698RS_
        00C
        00C                            ;#

                                       psect   config
        3FF   FEB                      dw      4075    ;#

                                       psect   text0
        1D8                         _main
                                     ;dual_inverter.c: 52: TRIS = 0b11111001;
        1D8   CF9                      movlw   -7
        1D9   006                      tris    6
                                     ;dual_inverter.c: 53: FOSC4 = 0;
        1DA   405                      bcf     5,0
                                     ;dual_inverter.c: 54: CMCON = 0b11110111;
        1DB   CF7                      movlw   -9
```

(continued ...)

(Listing 2, continued)

```
1DC  027                  movwf   7        ;volatile
                      ;dual_inverter.c: 55: OPTION = 0b11001111;
1DD  CCF                  movlw   -49
1DE  002                  option
                      ;dual_inverter.c: 57: while(1){
1DF  BF6                  goto    l6
                      ;dual_inverter.c: 58: switch(GPIO & 0b00001001)
1E0                   l7
                      ;dual_inverter.c: 59: {
                      ;dual_inverter.c: 61: GP1 = 1;
1E0  526                  bsf     6,1
1E1  BE3                  goto    L1
                      ;dual_inverter.c: 62: GP2 = 1;
                      ;dual_inverter.c: 63: break;
1E2                   l8
                      ;dual_inverter.c: 64: case 0b00000001:
                      ;dual_inverter.c: 65: GP1 = 0;
1E2  426                  bcf     6,1
1E3                   L1
                      ;dual_inverter.c: 66: GP2 = 1;
1E3  546                  bsf     6,2
                      ;dual_inverter.c: 67: break;
1E4  BF6                  goto    l6
1E5                   l9
                      ;dual_inverter.c: 68: case 0b00001000:
                      ;dual_inverter.c: 69: GP1 = 1;
1E5  526                  bsf     6,1
1E6  BE8                  goto    L2
                      ;dual_inverter.c: 70: GP2 = 0;
                      ;dual_inverter.c: 71: break;
1E7                   l10
                      ;dual_inverter.c: 72: case 0b00001001:
                      ;dual_inverter.c: 73: GP1 = 0;
1E7  426                  bcf     6,1
1E8                   L2
                      ;dual_inverter.c: 74: GP2 = 0;
1E8  446                  bcf     6,2
1E9  BF6                  goto    l6
1EA                   l30004
1EA  20A                  movf    btemp+2,w
1EB  643                  btfsc   3,2
1EC  BE0                  goto    l7
1ED  F01                  xorlw   1
1EE  643                  btfsc   3,2
1EF  BE2                  goto    l8
1F0  F09                  xorlw   9
1F1  643                  btfsc   3,2
1F2  BE5                  goto    l9
1F3  F01                  xorlw   1
1F4  643                  btfsc   3,2
1F5  BE7                  goto    l10
                      ;dual_inverter.c: 75: break;
1F6                   l6
1F6  206                  movf    6,w      ;volatile
1F7  E09                  andlw   9
1F8  02A                  movwf   btemp+2
1F9  06B                  clrf    btemp+3
1FA  20B                  movf    btemp+3,w
1FB  643                  btfsc   3,2
1FC  BEA                  goto    l30004
1FD  BF6                  goto    l6

                          psect   text1
0000
                          psect   temp
```

(continued ...)

of the possible logical combinations of the inverter pair outputs are found within the four case statements.

The assembler code for our dual inverter is a bit more hairy, as well. The big picture is that the code immediately jumps to label l6 and gathers the states of the inverter input pins. The code then jumps to label l30004, where the inverter logic is performed. Depending on the outcome of the logic computation, the program then jumps to the appropriate case statement — beginning at label l7 — and sets the inverter pair output pins.

I've taken the liberty of placing the compiler build output at the bottom of Listing 2 to show you where everything is located within the PIC10F206. Note the psect temp and the reservation of four bytes of memory beginning at SRAM location 0x008. Do you still think you can write better assembler code than that produced by the HI-TECH PICC C compiler? If you're still nodding yes, you had better turn on that electric flying suit. We're gaining altitude.

I think you have the idea now. You're all checked out on the PIC10F controls and we're flying level. That flight suit is pretty warm right now, but I don't want your feet to get cold. So, go aft and find that pair of wool-lined flight boots because, next time, we're going to be gaining even more altitude. So far, we've looked at the PIC10F, a hardware platform called Little Bits, the HI-TECH C Compiler that drives them, and an inverter application.

Next time — as we pull the yoke back — we'll fly through coding a two-input and three-input AND gate. I'll also describe how to construct various other logic gates using the PIC10F microcontroller. When we've flown through logic gate altitude, I'll show you how to code a D FLIP-FLOP. The air will be thin, but we'll keep climbing and write some PIC10F pulse generation code. When you think the contrails can't get any thicker, pull out that Digital Filter Development Board because I'm going to pair it with the Little Bits Development Board to show you how easy it is to code RS-232 routines that run on the world's smallest microcontroller. **NV**

Resources

HI-TECH Software
HI-TECH PICC C compiler
www.htsoft.com

Microchip
MPLAB ICE 2000
PIC10F206
MPLAB IDE
www.microchip.com

EDTP Electronics, Inc.
Little Bits
Digital Filter Development Board
www.edtp.com

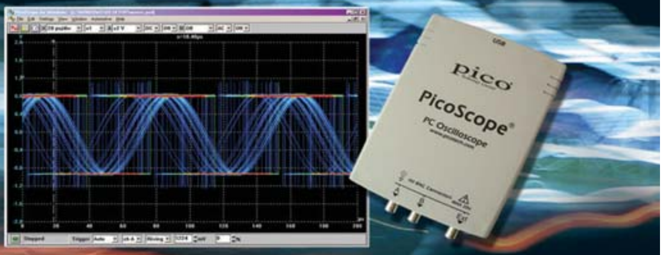
(Listing 2, continued)

```
   137   008                          btemp
   138   008                                ds      4


Psect Usage Map:

Psect     | Contents                     | Memory Range
---------|-----------------------------|--------------------
init      | Initialization code          | $0000 - $0000
end_init  | Initialization code          | $0001 - $0001
text      | Program and library code     | $01D8 - $01FD
vectors   | Reset vector                 | $01FE - $01FE
temp      | Temporary RAM data           | $0008 - $000B
config    | User-programmed CONFIG bits  | $03FF - $03FF

Memory Usage Map:

Program ROM   $0000 - $0001  $0002 (     2) words
Program ROM   $01D8 - $01FE  $0027 (    39) words
                             $0029 (    41) words total Program ROM
Bank 0 RAM    $0008 - $000B  $0004 (     4) bytes total Bank 0 RAM
Config Data   $03FF - $03FF  $0001 (     1) words total Config Data

Program statistics:
Total ROM used       41 words  (8.0%)
Total RAM used        4 bytes  (16.7%)
```

PicoScope 3000 Series PC Oscilloscopes

The PicoScope 3000 series oscilloscopes are the latest offerings from the market leader in PC oscilloscopes combining high bandwidths with large record memories. Using the latest advances in low power electronics, the oscilloscopes draw their power from the USB port of any modern PC, eliminating the need for mains power.

- **High performance: 10GS/s sampling rate & 200MHz bandwidth**
- **1MB buffer memory**
- **Advanced display & trigger modes**
- **Compact & portable**
- **Supplied with PicoScope (oscilloscope/spectrum analyser) & PicoLog (data acquisition) software.**



PicoScope	3204	3205	3206
Bandwidth	50MHz	100MHz	200MHz
Sampling rate (repetitive) Sampling rate (single shot)	2.5GS/s 50MS/s	5GS/s 100MS/s	10GS/s 200MS/s
Channels	2+Ext trigger	2+Ext trigger/Sig gen	2+Ext trigger/Sig gen
Oscilloscope timebases	5ns/div to 50s/div	2ns/div to 50s/div	1ns/div to 50s/div
Timebase accuracy	50ppm	50ppm	50ppm
Spectrum ranges	0 to 25MHz	0 to 50MHz	0 to 100MHz
Record length	256K	512K	1MB
Resolution accuracy	8 bits / 3%		
Ranges	10mV to 2V/div		
PC Connection/Power supply	USB2.0 (USB1.1 compatible)		

Tel: 585 385 1750
www.picotech.com/scope215

pico® *Technology Limited*

Circle #55 on the Reader Service Card.

by Peter Best

The Ever-Shrinking µC — Part 2

Six Pins and One MIP — If You Can See It!

If, after reading Part 1 last month, you've been wondering why there are two PIC10F206s on the Little Bits Development Board, here's part of that answer: The inverter pair we just implemented can be tested by simply moving jumpers carrying the desired logic levels between the inverter inputs and watching the inverter outputs on the LEDs. Instead of swapping around jumpers, the second PIC10F206 can be used as stimulus for the first PIC10F206. I wrote a small piece of code called TOGGLER that does nothing but count from 0 to 7 continually. Using jumpers, I feed the output of the PIC10F206 running TOGGLER to the input of the inverter pins of the PIC10F206 running the inverter code. As the count progresses, all of the possible inverter input combinations are provided to the inputs of the inverter pair we realized with the other PIC10F206. The C source for TOGGLER is shown in Listing 3.

I think you have the idea now. So, I've included some C code for a two-input AND gate in Listing 3, as well. There's always more than one way to skin a cat when coding. I've also included an optimized version of the two-input AND gate code in Listing 3 for your approval. From the example code I've presented, you should now be able to create many other logic gates, including a three-input AND gate, an EXCLUSIVE OR/NOR gate, and an OR/NOR gate.

For instance, to fabricate a NAND gate, all you have to do is invert the output levels within the case statements of the AND gate code. An OR gate can be fabricated from the AND gate code by simply changing the output within the case statements to 1 for any case statement that contains a 1 as its argument. To get the NOR function, invert the output of the OR gate within the OR gate case statements.

You can also emulate clocked logic with the PIC10F206 as well, provided that the clocked logic module you're emulating doesn't have more than three output pins. A clocked logic block that immediately comes to mind is the D flip-flop. When clocked, the Q output of a D flip-flop will follow the logic level of the D input with the NOT-Q output complementing the logic level of the D input.

Compile the D flip-flop code in Listing 3 and jumper the D and NOT-Q GPIO pins together. This will divide the incoming clock by 2 and produce the divided clock on the Q output pin. Feed the CLK

***Listing 3**. Remember, the PIC10F206 has a 1 mS instruction cycle time. So, although the logic will work as designed, the logic blocks we emulate with the PIC10F206 won't be as fast as the real thing.*

```
//*********************************************************************
//*  HI-TECH C SOURCE CODE FOR TOGGLER MODULE
//*********************************************************************
void main()
{
 unsigned int x,y;

 TRIS = 0b00001000;      //GP3 input : all others output
 FOSC4 = 0;              //GP2 is an I/O pin
 CMCON  = 0b11110111;    //comparator off:pullups off:wakeup off
 OPTION = 0b11001111;    //prescaler assigned to WDT
 while(1)
 {
  ++GPIO;                //increment output pin set GP0, GP1, GP2
  for(x=0;x<0xFFFF;++x)  //delay by incrementing y 65534 times
   ++y;                  //increment y
 }//while(1)             //loop forever
}//main TOGGLER

//*********************************************************************
//*  HI-TECH C SOURCE CODE FOR 2-INPUT AND GATE MODULE
//*********************************************************************
void main()
{
 TRIS = 0b11111011;      //GP2=output:all other GPIO=input
 FOSC4 = 0;              //GP2 is an I/O pin
 CMCON  = 0b11110111;    //comparator off:pullups off:wakeup off
 OPTION = 0b11001111;    //prescaler assigned to WDT

 while(1)
 {
  switch (GPIO & 0b00000011)
  {
   case 0b00000000:      //GP0 = LOW:GP1=LOW
    GP2 = 0;             //GP2 = LOW
    break;
   case 0b00000001:      //GP0 = HIGH:GP1=LOW
    GP2 = 0;             //GP2 = LOW
    break;
```

(continued)

input of your PIC10F206 D FLIP-FLOP divide-by-2 emulator from one of the three PIC10F206 TOGGLER outputs. You'll see the division of whatever TOGGLER output clock you feed to the CLK input on the Q output of the D flip-flop emulated by the second PIC10F206. Figure 1 (see Part 1 in last month's issue) physically depicts all of the logic we've emulated or talked about so far. The view from way up here is tremendous, isn't it?

Pulse Generation and Signal Conditioning With Little Bits

The 555 is a wonderful device. However, it has its shortcomings, as it programs in an analog fashion rather than a digital one. For instance, it takes a few choice components and some steering diodes to get a true %50 duty cycle pulse train directly from the output of a 555. With the small bit of C in Listing 4 and a single PIC10F206, I've created a 60 Hz pulse train with a 50% duty cycle.

The beginning of the code is very similar to our logic examples, except that the OPTION argument value has now assigned the prescaler to TMR0 (Timer 0) by clearing bit 3 of the OPTION register. Bits 0, 1, and 2 set the prescaler value to 1:64, which instructs TMR0 to increment once every 64 instruction cycles.

The meat of the 60 Hz code is centered on the TMR0 instructions. First, the TMR0 register is loaded with 0x80. After a couple of synchronization cycles, TMR0 begins to count upwards from 0x80. Visualizing this in binary, the first count will be 0b10000001. The second count will be 0b10000010 and so forth until the count reaches 0b11111111 and then rolls over to 0b00000000.

When the count rolls over to 0b00000000, the most significant bit is no longer set and the while(TMR0 & 0x80); becomes false, allowing the code to fall through to the next statement, which is GP2 = 0;. At this point, the code shifts the logic level of GP2 to low and spins in the TMR0 loop, just as it did when GP2 was high. You can alter the frequency of the TMR0-generated pulse train by toying with the three least significant bits of the OPTION register. You can also change the frequency by altering the value loaded to TMR0 and then checking for that value in the following while statement.

(Listing 3, continued)

```
   case 0b00000010:      //GP0 = LOW:GP1=HIGH
    GP2 = 0;             //GP2 = LOW
    break;
   case 0b00000011:      //GP0 = HIGH:GP1=HIGH
    GP2 = 1;             //GP2 = HIGH
    break;
  }//switch
 }//while(1)
}//main

//*********************************************************************
//*  HI-TECH C SOURCE CODE FOR 2-INPUT AND GATE MODULE OPTIMIZED
//*********************************************************************
void main()
{
 TRIS = 0b11111011;      //GP2=output:all other GPIO=input
 FOSC4 = 0;              //GP2 is an I/O pin
 CMCON  = 0b11110111;    //comparator off:pullups off:wakeup off
 OPTION = 0b11001111;    //prescaler assigned to WDT

 while(1)
 {
  switch (GPIO & 0b00000011)
  {
   case 0b00000011:      //GP2=HIGH only if both GP0 and GP1 are HIGH
    GP2 = 1;
    break;
   default:
    GP2 = 0;
    break;
  }//switch
 }//while(1)
}//main

//*********************************************************************
//*  HI-TECH C SOURCE CODE FOR D FLIP-FLOP MODULE
//*********************************************************************
void main()
{
 TRIS = 0b11111100;              //GP0=Q:GP1=NOT-Q:GP2=CLK:GP3=D
 FOSC4 = 0;
 CMCON  = 0b11110111;
 OPTION = 0b11001111;
                                 //power-up clear operation
 while(GP2);                     //wait for CLK to go LOW
 GP0 = 0;                        //set Q
 GP1 = 1;                        //clr Q

 while(1)
 {
  while(!GP2);                   //wait for CLK to go HIGH
  switch (GPIO & 0b00001000)
  {
   case 0b00000000:              //D = 0
    GP0 = 0;                     //clr Q
    GP1 = 1;                     //set NOT-Q
    break;
   case 0b00001000:              //D = 1
    GP0 = 1;                     //set Q
    GP1 = 0;                     //clr NOT-Q
    break;
   }//switch
   while(GP2);                   //wait for clock to go LOW
  }//while(1)                    //loop forever
}//main
```

Listing 4. I'm sure there are some really weird analog things that you can do with a 555, but the PIC10F206 can hold its own against the 555 when it comes to handling and generating pulses.

```
//**************************************************************************
//*  HI-TECH C SOURCE CODE FOR PULSE GENERATOR MODULE
//**************************************************************************
void main()
{

 TRIS = 0b00001000;      //GP3 input : all others output
 FOSC4 = 0;              //GP2 is an I/O pin
 CMCON  = 0b11110111;    //comparator off:pullups off:wakeup off
 OPTION = 0b11000101;    //prescaler assigned to TMR0 @ 1:64

 while(1)                //loop forever
 {
     GP2 = 1;            //GP2 = HIGH
     TMR0 = 0x80;        //load TMR0
     while(TMR0 & 0x80);//count from 0b10000000 to 0b00000000

     GP2 = 0;            //GP2 = LOW
     TMR0 = 0x80;        //load TMR0
     while(TMR0 & 0x80);//count from 0x80 to 0x00
 }//while(1)
}//main

//**************************************************************************
//*  HI-TECH C SOURCE CODE FOR LED DIMMER MODULE
//**************************************************************************

void main()
{
 unsigned char x,y,z;

 TRIS = 0b00001000;      //GP3 input : all others output
 FOSC4 = 0;              //GP2 is an I/O pin
 CMCON  = 0b11110111;    //comparator off:pullups off:wakeup off
 OPTION = 0b11000010;    //prescaler assigned to TMR0 @ 1:8

 y = 0x04;               //initialize y
 z = 0x80;               //initialize z

 while(1)                //loop forever
 {
 do                      //outer do loop
 {
    x=0xFF;

  do                     //inner do loop
  {
     GP2 = 1;            //GP2 = HIGH
     TMR0 = y;           //load TMR0
     while(TMR0 & y);    //count for y time

     GP2 = 0;            //GP2 = LOW
     TMR0 = z;           //load TMR0
     while(TMR0 & z);    //count for z time
   }while(--x);          //do inner loop until x is decremented to 0

    y *= 2;              //multiply y x 2
    z /= 2;              //divide z by 2
  }while(y);             //do outer loop until y = 0

  y = 0x04;              //reinitialize y
  z = 0x80;              //reinitialize z
 }//while(1)
}//main
```

(continued)

It stands to reason that, if we can control the frequency of a pulse train generated by TMR0, we can also control the duty cycle of that pulse train. A %50 duty cycle means that every cycle has equal high and low logic levels with respect to time. If we apply that voltage to an LED, it will be on for half the time and off for half the time. If the frequency is high enough, the LED may appear to be dimmer than it would seem to be when full voltage is applied to it. If we switch between on and off fast enough, our eyes and mind will fool us into thinking that the LED is really never turning off. We can use this phenomenon to our advantage with the second code listing you see in Listing 4.

Let's work our way through the LED dimmer code inside out, beginning with the inner do loop. The GP2 = 1 code is exactly the same as our 60 Hz pulse train generator except that the 0x80 hard-coded value is replaced with a variable of y. The y variable is initialized to a value of 0x04 in the beginning of the code sequence. Recall that the TMR0 register increments every instruction cycle. So, the y count begins at 0b00000100 and ends at 0b00001000 when the TMR0 register rolls over to 0b00001000 from 0b00000111.

Looking at the GP2 = 0 code, it is exactly the same as our previous 60 Hz code for GP2 in a low logic state with the only exception being the variable z holding the 0x80 value, which was loaded right after the y value. What all of this means is that, initially, the high part of the cycle is much smaller than the low part of the cycle with respect to time, which, in turn, says that the LED will initially be off longer than it is on and will appear to be very dim. Each duty cycle period is alive as long as x is not equal to zero. The while(—x) decrements x with each pass through the inner do loop. The outer do loop initializes x, doubles y, and halves z at the completion of each pulse train cycle.

Since y is the variable that determines the high level time of each cycle and z is the variable that determines the low level time of each cycle and the values are approaching each other from the opposite directions, when y is equal to 0b10000000, z will be equal to 0b00000010. At this point, the LED will be at its brightest, since the high part of the cycle (y) will be much longer than the low part of the cycle (z). The visual effect you will see is the LED going from dim to bright continually.

Delays and pulse trains can also be created with the PIC10F206 by utilizing code similar to that which is used in TOGGLER (Listing 3). Delays that last for seconds can easily be achieved by looping on a for construct like the one used in the TOGGLER code (available at **www.nutsvolts.com**).

Take a look at the voltage booster circuit in Schematic 1 (see Part 1 in last month's issue). The 2N2222A alternately builds and collapses the field formed by the inductor. The steering diode routes the inductor's energy into the 100 µF capacitor. The build-up and collapse of the magnetic field is caused by switching the transistor on and off rapidly with a pulse train provided by a PIC10F206. The code (part of Listing 4) is identical to the 60 Hz pulse train code, except that the TMR0 prescaler is set for 1:2 and the duty cycle of the pulse train is heavily biased to the logic low level. This may, at first, seem backward. However, the energy from the inductor is transferred to the capacitor when the transistor is turned off. When the transistor is on, the inductor is allowed to ramp up a charge.

By rapidly switching the inductor on and off, we are able to feed the inductor's energy through the diode and charge the output capacitor to a voltage that is higher than the input voltage at the inductor. The little circuit you see in Schematic 1, in combination with the voltage booster code in Listing 4, generates about +10 VDC across the output capacitor. You can obtain a much higher voltage by tweaking the pulse train's duty cycle.

(Listing 4, continued)

```
//**********************************************************************
//*  HI-TECH C SOURCE CODE FOR VOLTAGE BOOSTER MODULE
//**********************************************************************
void main()
{

 TRIS = 0b00001000;      //GP3 input : all others output
 FOSC4 = 0;              //GP2 is an I/O pin
 CMCON  = 0b11110111;    //comparator off:pullups off:wakeup off
 OPTION = 0b11000000;    //prescaler assigned to TMR0 @ 1:2

 while(1)                //loop forever
 {
  GP2 = 1;               //GP2 = HIGH
  TMR0 = 0x04;           //load TMR0
  while(TMR0 & 0x04);    //count from 0b00000100 to 0b00001000

  GP2 = 0;               //GP2 = LOW
  TMR0 = 0x80;           //load TMR0
  while(TMR0 & 0x80);    //count from 0x80 to 0x00
 }//while(1)
}//main
//**********************************************************************
//*  HI-TECH C SOURCE CODE FOR ONE-SHOT MODULE
//**********************************************************************
void main()
{
 unsigned int x,y;
 TRIS = 0b00001000;      //GP3 input : all others output
 FOSC4 = 0;              //GP2 is an I/O pin
 CMCON  = 0b11110111;    //comparator off:pullups off:wakeup off
 OPTION = 0b11000111;    //prescaler assigned to TMR0 @ 1:256

 GP2 = 0x00;             //clear GP2
 while(1)                //loop forever
 {
  while(!GP3);           //wait for GP3 to go HIGH
  TMR0 = 0x80;           //debounce the switch closure
  while(TMR0 & 0x80);    //loop for 35mS using TMR0 with prescaler @ 1:256
  GP2 = 1;               //set GP2
  for(x=0;x<0xFFFF;++x)  //delay for one-shot time
   ++y;
  GP2 = 0;               //clear GP2
  while(GP3);            //wait for button release:GP3 to go LOW
  TMR0 = 0x80;           //debounce button release
  while(TMR0 & 0x80);    //loop for 35mS
 }
}
```

In addition to generating pulses, the PIC10F206 can also be programmed to condition pulses. Let's use some TMR0 code we've already written and create a one-shot timer with a built-in switch debouncer. We'll only need a push-button switch and a resistor, as shown by the one-shot section of Schematic 1.

The C code is straightforward. GP3 is the input from the switch/resistor combination. When the switch is open, GP3 is held low by the 20K resistor. The while(!GP3) loops waiting for GP3 to go high. When the switch is closed, GP3 goes high and the TMR0 debounce code is executed; 35 mS later, GP2 goes high and the one-shot delay loop runs. When the one-shot delay loop falls through, GP2 is cleared to a logic low level and — if the switch is still depressed — the while(GP3) statement loops until the push-button is released. The switch is again debounced when released and the one-shot-switch-debounce process repeats from the beginning statement (while(!GP3)).

Now you can see that handling pulse trains and delays with the PIC10F206 is easy to do. So far, we have enabled a crude PWM (Pulse Width Modulation) function with our LED dimmer code and the one-shot code conjures up lots of other possibilities. The same one-shot code could be used to enable a pulse stretcher or pulse shrinker. A missing pulse detector is yet another coding possibility. The bottom line is that precious microcontroller CPU cycles can be offloaded to the PIC10F206 as it can perform mundane tasks, such as debouncing switches and conditioning incoming signals.

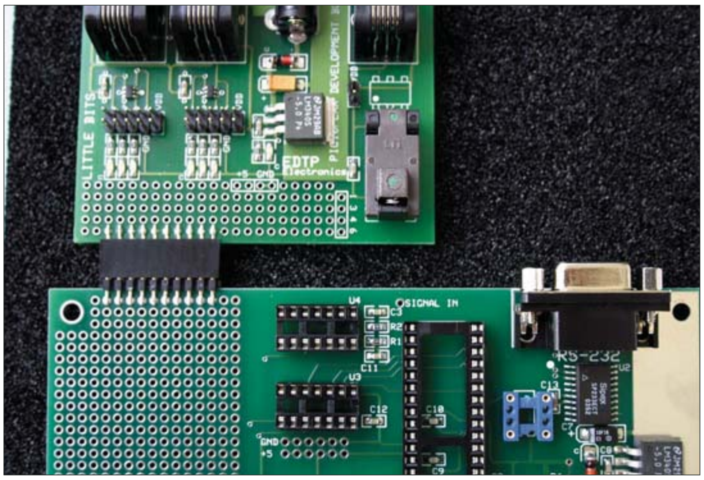


Figure 6. The Digital Filter Development Board is teamed up with a Little Bits to complete the realization of a PIC10F206 bit-bang serial port.

Communicating With Little Bits

There are no USARTs (Universal Synchronous Asynchronous Receiver Transmitters) or UARTs (Universal Asynchronous Receiver Transmitters) contained within the PIC10F206 silicon. So, there's no native PIC10F206 serial communications functionality. Just because the specialized UART hardware doesn't exist doesn't mean we can't implement a software PIC10F206 UART of our own.

In fact, we can do just that and — thanks to the HI-TECH PICC C compiler — we won't have to write any of the serial communications drivers from scratch. All of the bit-bang serial driver code that comes with HI-TECH PICC C compiler is shown in Listing 5. You can study the code in detail if you wish. However, the only things you have to know about the serial driver are the serial functions that you will use when applying the driver code. To send a character, use the putch function. To receive a character, use the getch function. A optional function called getch echoes the incoming character.

This is where the 20-pin female connector I added to my Little Bits comes into play. As you can see in Figure 6, I've called upon a Digital Filter Development Board to aid the Little Bits in getting its serial port operational. I've wired the Digital Filter Development Board's SP233ECT RS-232 IC into one of the PIC10F206 microcontrollers on Little Bits via a 20-pin male header on the Digital Filter Development Board. Only four connections are necessary, with power and ground being givens. Look again at the beginning of the code in Listing 5. You'll see that I've designated GP2 as the transmit pin and GP3 as the receive pin and specified a baud rate of 9600 bps. I've detailed the PIC10F206-to-Digital Filter

Seetron Serial LCDs

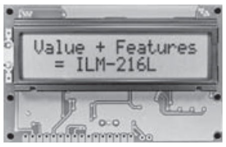
Interface a sharp LCD display to your BASIC Stamp® or other micro-controller project with ease. No-solder wiring harnesses and easy mounting kits available too. See www.seetron.com today.

- 3.2 x 1.4 in. supertwist LCD
- 2400/9600 baud serial
- Low (≈2mA) current draw
- Great with BASIC Stamps®

$45 BPI-216N

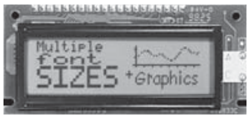
- 3.2 x 2 in. backlit LCD
- 1200-9600 baud serial
- Advanced protocol, 4 switch inputs
- EEPROM for configuration settings
- Favorite for OEM applications

$49 ILM-216L



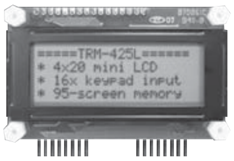
- 3.2 x 1.4 in. graphics LCD
- 2400/9600 baud serial
- Font and 15 screens in EEPROM
- Easily draw points, lines, screens

$99 SGX-120L



- 3 x 2 in. supertwist LCD
- 1200-9600 baud serial
- ESD-protected, 4x4 keypad input
- Store up to 95 screens in EEPROM

$119 TRM-425L



Scott Edwards Electronics, Inc.
1939 S. Frontage Rd. #F, Sierra Vista, AZ 85635
phone 520-459-4802 • fax 520-459-0623
www.seetron.com • sales@seetron.com

More displays available, including bright VFDs. See www.seetron.com

Circle #117 on the Reader Service Card.

Wireless - 40 Miles - $299

New 1 Watt 9XTend Stand-Alone RF Modem

The best price-to-performance value on the market

FEATURES	MaxStream's 9XTend	Low Cost Competitor	High Cost Competitor
Price	★ < $300	★ < $300	✖ > $1,200
Max. Range line-of-sight	✖ Up to 40 Mi.	✖ Up to 20 Mi.	★ Up to 60 Mi.
Data Throughput	★ 115 kbps	✖ 57 kbps	★ 115 kbps
Encryption	★ 256-Bit AES	✖ 56-Bit DES	✖ None
Max. Sensitivity	★ -110 dBm	✖ -99 dBm	★ -110 dBm
Interference Immunity	★ YES	✖ NO	★ YES

Call for aggressive volume discounts & development specials.

RS-232/485

MaxStream
www.maxstream.net
toll-free 866-765-9885

Circle #123 on the Reader Service Card.

Development Board hardware connections in the RS-232 box of Schematic 1.

Now we're ready to send some characters. I set up a Tera Term Pro session on my personal computer and tuned it in for 9600 bps. I then wrote the C code you see at the bottom of Listing 5. The code sends the string NUTS & VOLTS continually. The string I defined at the beginning of my code is actually stored in the PIC10F206's program Flash area and is automatically terminated with a null or zero character by the HI-TECH PICC C compiler.

I've pulled the psect that details how the characters in the string are stored and placed it under my code so you can see how the HI-TECH PICC C compiler handles strings in the Flash memory area. Putting the string in the program Flash memory area is a good thing, as the PIC10F206 doesn't have enough SRAM to hold the string and do other SRAM related things at the same time. Pretty clever, huh?

The null character that signifies the end of the string comes in handy, as I can simply test for it as I send characters

Sources

HI-TECH Software
HI-TECH PICC C compiler
www.htsoft.com

Microchip
MPLAB ICE 2000
PIC10F206
MPLAB IDE
www.microchip.com

EDTP Electronics, Inc.
Little Bits
Digital Filter Development Board
www.edtp.com

Listing 5. This code takes up almost half of the PIC10F206's program Flash. That still leaves enough room to make effective use of the serial port this code creates.

```
// *
// *    Serial port driver (uses bit-banging)
// *    for 16Cxx series parts.
// *
// *    IMPORTANT: Compile this file with FULL opti-
mization
// *
// *    Copyright (C)1996 HI-TECH Software.
// *    Freely distributable.
// *    Adapted for use with the PIC10F20X by Peter
Best

#include<pic.h>
__CONFIG(MCLRDIS & WDTDIS & UNPROTECT);

//Tunable parameters
//Transmit and Receive port bits
#define SERIAL_PORT     GPIO
#define SERIAL_TRIS     TRIS
#define TX_PIN          2       //GP2
#define RX_PIN          3       //GP3

//Xtal frequency
#define XTAL    4000000

//Baud rate
#define BRATE   9600
```

(continued)

Be an FCC LICENSED ELECTRONIC TECHNICIAN

Learn at home in your spare time.

The United States of America
Federal Communications Commission
General Radiotelephone

Earn up to $100 an hour and more!

No previous experience needed!

You can earn more money if you get an FCC License!

Not satisfied with your present income? Add prestige and earning power to your electronics career by getting your FCC Government License.

The Original Home-Study course prepares you for the "FCC Commercial Radiotelephone License" at home in your spare time.

This valuable license is your professional "ticket" to thousands of exciting jobs in: Communications, Radio-TV, Microwave, Maritime, Radar, Avionics & more...you can even start your own business!

No need to quit your job or go to school. This proven "self-study" course is easy, fast and low cost!

GUARANTEED TO PASS – You get your FCC License or your money will be refunded.

Call for FREE facts now!

(800) 932-4268 Ext. 220

www.LicenseTraining.com

COMMAND PRODUCTIONS
FCC LICENSE TRAINING - Dept. 220
P.O. Box 3000 • Sausalito, CA 94966
Please rush FREE details immediately!

or mail coupon today

Name ______________________________

Address ____________________________

City ____________________ State ______ Zip ________

Circle #135 on the Reader Service Card.

(Listing 5, continued)

```
//Don't change anything else
#define SCALER                     10000000
#define ITIME                      4*SCALER/XTAL    // Instruction cycle time
#if BRATE > 1200
 #defineDLY             3                  // cycles per null loop
 #defineTX_OHEAD        13                 // overhead cycles per loop
#else
 #defineDLY             9                  // cycles per null loop
 #define TX_OHEAD                  14
#endif
#define RX_OHEAD        12                 // receiver overhead per loop

#define DELAY(ohead)               (((SCALER/BRATE)-(ohead*ITIME))/(DLY*ITIME))

static bit      TxData @ (unsigned)&SERIAL_PORT*8+TX_PIN;        // Map TxData to pin
static bit      RxData @ (unsigned)&SERIAL_PORT*8+RX_PIN;        // Map RxData to pin
#define INIT_PORT                  SERIAL_TRIS = 1<<RX_PIN        // set up I/O direc-
tion

void putch(char c)
{
 unsigned char  bitno;
 #if BRATE > 1200
  unsigned char dly;
 #else
  unsigned int  dly;
 #endif

 INIT_PORT;
 TxData = 0;                       // start bit
 bitno = 12;
 do
 {
  dly = DELAY(TX_OHEAD);// wait one bit time
   do                              // waiting in delay loop
    while(--dly);
  if(c & 1)
   TxData = 1;
  if(!(c & 1))
   TxData = 0;
  c = (c >> 1) | 0x80;
 }while(--bitno);
 NOP();
}

char getch(void)
{
 unsigned char  c, bitno;
 #if BRATE > 1200
  unsigned char dly;
 #else
  unsigned int  dly;
 #endif

  for(;;)
  {
   while(RxData)
    continue;           // wait for start bit
   dly = DELAY(3)/2;
   do                   // waiting in delay loop
    while(--dly);
   if(RxData)
    continue;           // twas just noise
   bitno = 8;
   c = 0;
```

(continued)

out of the PIC10F206 serial port. Once I encounter a null, I know that I have sent the entire string and I send a carriage return and line feed combination. A simple delay loop is executed and the bit-bang serial process sends another NUTS & VOLTS message.

Good Things in Small Packages

Okay, let's begin our descent and land this thing. As you have seen, lots of useful things can be done with a tiny PIC10F206, its four I/O lines, and a good C compiler like the HI-TECH PICC C compiler. The air is really clear at 23,000 feet. You have seen for yourself that using a C compiler with a tiny PIC like the PIC10F206 is not necessarily a bad thing.

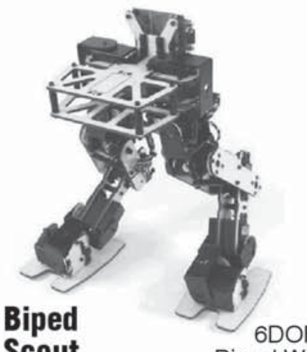
Whether you code in assembler or C, the PIC10F20X series of microcontrollers is a blast to work with. I'm sure you'll want to try your hand at some tiny applications, as well. So, I'll make all of the Little Bits code in the listings available for download from the *Nuts & Volts* FTP server (**www.nutsvolts.com**).

For those of you who want to melt some solder around a PIC10F206, either the Wahl Iso Tip portable soldering iron with a 7566-100 micro tip or a Metcal Soldering Station with a SSC-645A soldering element is perfect for the task. If you don't want to roll your own Little Bits, you can get a kit of parts or an assembled Little Bits unit from EDTP Electronics (**www.edtp.com**). **NV**

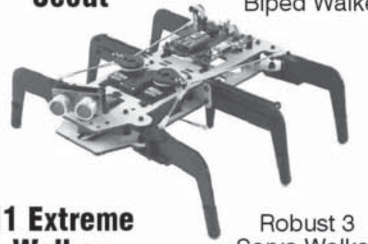
(Listing 5, continued)

```
   do
   {
    dly = DELAY(RX_OHEAD);
    do                   // waiting in delay loop
     while(--dly);
    c = (c >> 1) | (RxData << 7);
   }while(--bitno);
   return c;
  }
}

char getche(void)
{
 char c;

 putch(c = getch());
 return c;
}

//***********************************************************************
//*  HI-TECH C SOURCE CODE FOR RS-232 MODULE
//***********************************************************************

const char * string = "NUTS & VOLTS";

void main()
{
 unsigned char x;
 unsigned int y,z;

 FOSC4 = 0;
 CMCON   = 0b11110111;
 OPTION = 0b11001111;

 while(1)                        //loop forever
 {
  x = 0;                         //initialize character index
  do
  {
   putch(string[x++]);           //send character indexed by x: increment x
  }while(string[x]!= 0);//look for null character at end of string
  putch(0x0D);                   //send carriage return
  putch(0x0A);                   //send line feed
  for(y=0;y<0xFFFF;++y)          //delay for a while
   ++z;
 }
}
//***********************************************************************
//*  HOW THE STRING IS STORED IN FLASH
//***********************************************************************
   248                                    psect   strings
   249  018                        u19
   250  018  84E                          retlw   78      ;'N'
   251  019  855                          retlw   85      ;'U'
   252  01A  854                          retlw   84      ;'T'
   253  01B  853                          retlw   83      ;'S'
   254  01C  820                          retlw   32
   255  01D  826                          retlw   38      ;'&'
   256  01E  820                          retlw   32
   257  01F  856                          retlw   86      ;'V'
   258  020  84F                          retlw   79      ;'O'
   259  021  84C                          retlw   76      ;'L'
   260  022  854                          retlw   84      ;'T'
   261  023  853                          retlw   83      ;'S'
   262  024  800                          retlw   0
```

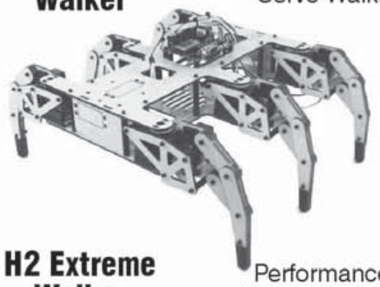
www.lynxmotion.com



Biped Scout — 6DOF Biped Walker



H1 Extreme Walker — Robust 3 Servo Walker



H2 Extreme Walker — Performance 12 Servo Walker

Lynx 6 Robotic Arm — Five DOF Arm

4WD2 Rover — Articulating Chassis

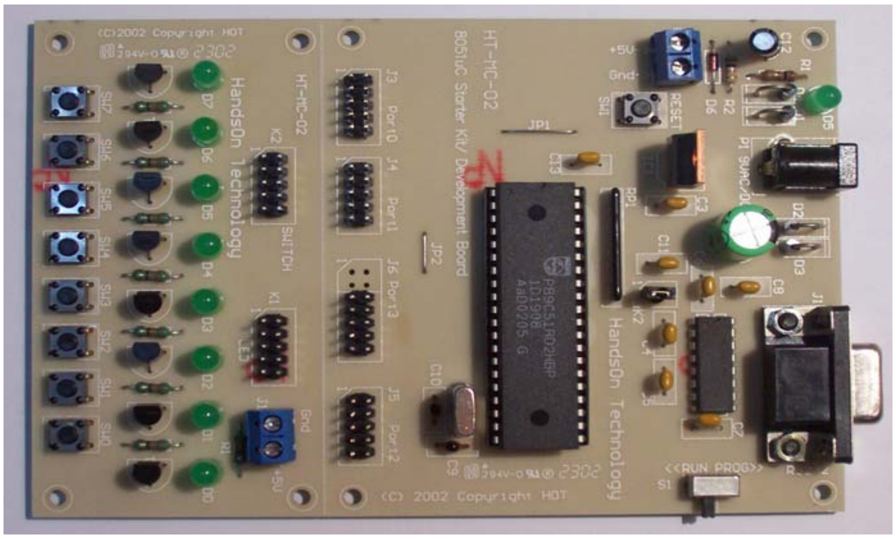
We have many more unique robot kits. Our robots feature:

- Precision Laser-Cut Lexan
- Preassembled Electronics
- Custom Aluminum Components
- Injection Molded Components
- Very High Coolness Factor

Toll Free: 866-512-1024
Web: www.lynxmotion.com

Low Cost 8051µC Starter Kit/ Development Board *HT-MC-02*

HT-MC-02 *is an ideal platform for small to medium scale embedded systems development and quick 8051 embedded design prototyping.* ***HT-MC-02*** *can be used as stand-alone 8051µC Flash programmer or as a development, prototyping and educational platform*



Main Features:

- 8051 Central Processing Unit.
- On-chip Flash Program Memory with In-System Programming (ISP) and In Application Programming (IAP) capability.
- Boot ROM contains low level Flash programming routines for downloading code via the RS232.
- Flash memory reliably stores program code even after 10,000 erase and program cycles.
- 10-year minimum data retention.
- Programmable security for the code in the Flash. The security feature protects against software piracy and prevents the contents of the Flash from being read.
- 4 level priority interrupt & 7 interrupt sources.
- 32 general purpose I/O pins connected to 10pins header connectors for easy I/O pins access.
- Full-duplex enhanced UART **–** Framing error detection Automatic address recognition.
- Programmable Counter Array (PCA) & Pulse Width Modulation (PWM).
- Three 16-bits timer/event counters.
- AC/DC (9~12V) power supply – easily available from wall socket power adapter.
- On board stabilized +5Vdc for other external interface circuit power supply.
- Included 8x LEDs and pushbuttons test board (free with ***HT-MC-02*** while stock last) for fast simple code testing.
- Industrial popular window ***Keil*** C compiler and assembler included (Eval. version).
- Free ***Flash Magic*** Windows software for easy program code down loading.

PLEASE READ ***HT-MC-02 GETTING STARTED MANUAL*** BEFORE OPERATE THIS BOARD
INSTALL ACROBAT READER (AcrobatReader705 Application) TO OPEN AND PRINT ALL DOCUMENTS