

## FEATURE ARTICLE

David Brobst

# Through the Looking Glass

## Taking a Look at the PIC16C5x Series

### Taking a Look at the PIC18Cxxx Series

If you're looking for high-end, look no further than Microchip's PIC18Cxxx family of 8-bits. Designed with the future in mind, its direction seems clear—with increased clock speed, hardware multiplier, and higher resolution A/Ds, this new unveiling is positioned as a low-end DSP solution. With intriguing features like the table read/write capability, David shows that the PIC18Cxxx is a valuable addition to the growing line of controllers.



In the early '90s, the venerable line of PIC16C5x products introduced a new way of thinking for 8-bit designs. Blazingly fast for its time, the PIC16C5x series was based on an EPROM one-time-programmable (OTP) program memory architecture. With its low-cost approach, small and medium quantity designs could benefit from the advantages of embedded intelligence.

The drawback to these first controllers was that they were bare bones—no interrupts, limited stack and subroutine memory, a paucity of onboard peripherals, and limited program memory size. Later, a mid-range line of controllers was introduced, the PIC16Cxxx family.

This line of controllers included interrupts, a larger stack, more data memory, much larger program memory, and an abundance of peripherals (A/Ds, timers, USARTs, parallel ports, serial ports, EEPROM, flash memory, and more). Once again, these controllers were aggressively priced

and available with short lead times and in reasonable quantities. The death knell of the masked-ROM-only applications had sounded, and other companies began to roll out lines of OTP controllers.

Next came the remarkable 8-pin microcontroller, the PIC12Cxxx family. These basic controllers were based on both the low-end and mid-range families and were priced under \$1. Because they required no external support circuitry and their six I/O pins were often more than enough for a single application, the PIC12Cxxx controllers were being spread around like mad, replacing special function chips (such as the venerable 555 timer) in designs everywhere.

Recently unveiled is a new family of microcontrollers that provide a glimpse into the future. The PIC18Cxxx family is Microchip's new foray into the high-end 8-bit market. While not perfect, these chips go a long way towards establishing a viable high-end presence for Microchip.

### PIC18CXXX CHIPS

The current offering of the PIC18Cxxx architecture is limited to the PIC18Cxx2 family. In keeping with the tradition of vertically scaled sub-families in controller lines, there are four chips in the PIC18Cxx2 family—PIC18C452, PIC18C442, PIC18C252, and PIC18C242. The number of I/O lines and amount of program/data memory are the only differentiating features among the four chips. Table 1 details some of the features for this line of controllers. Figure 1 shows a pinout of both the 28- and 40-pin chips.

After looking at Table 1 and Figure 1, a few things are blessedly obvious for anyone familiar with the mid-range family of PIC microcontrollers. First and foremost, the 28- and 40-pin devices in the PIC18Cxx2 family share

the exact same pinout as the 28- and 40-pin devices of the entire PIC16Cxxx family of microcontrollers. This means that current designs can potentially migrate to the more powerful chip without a costly hardware revision. At the very least, keeping the same pinout eases the design because all of the I/Os and special function modules are in familiar spots.

## MEMORY

A second welcome feature is the amount of both data and program memory available in the chips. Indeed, while the current Big Bertha of the PIC18Cxxx family is limited to 16 KB of program memory, the architecture has been designed with the future in mind. The device is capable of addressing 1 MB of instructions. In addition to that, the quantity of data memory is a welcome relief to anyone who has ever struggled to cram just one more variable into a program. Again, the amount of data memory in the introductory chips is just the tip of the iceberg. The data addresses have been set up to accommodate up to 4 KB of data memory in the future. Figure 2 shows the program memory for the PIC18Cxxx family.

A quick perusal of Table 1 shows that the PIC18Cxxx family has upped the ante over the mid-range controllers with increased clock speed, resolution A/D, serial support, and more special functions. Most of these features will be explored in detail later in the article.

Taking all of these features into consideration along with the literature provided for the PIC18Cxxx family, the direction Microchip is taking is clear. With the increased clock speed, hardware multiplier, and higher resolution A/Ds, the PIC18Cxxx family is placed as a low-end DSP solution. For example, it could be seen easily as a dedicated motor controller (lower-end DSP), but an advanced graphics card would be out of its league.

With its ample linear program memory and loads of data memory, the chip becomes more amenable to compilers and coding tools. The PIC18Cxxx family is more hospitable to C compilers than the earlier families of PIC microcontrollers. Indeed, a C compiler with libraries for the PIC18Cxxx family is already being provided. With all of this in mind, let's look at some of the new features in greater detail.

## DATA MEMORY

I'd like to review the details of some of the new peripherals or improvements on mid-range features available with the new PIC18Cxxx architecture. Both the good and the not-so-good will be discussed in this section.

Data memory was discussed briefly before, however, I would like to look at some of the unique features of the memory. Figure 3 shows the data memory of the PIC18C452. Although there are 1536 bytes of memory, they are set up in a paged manner. Each page of data memory is 256 bytes, and the

special function registers (registers that set up the peripherals) can be directly accessed without paging. So for many cases, the paging might not be a problem. However, the front page of the PIC18Cxx2 datasheet states a linear data memory space.

The memory is arranged in the memory map linearly, but accessing all of it directly still takes banking operations. This has been one of the major frustrations for all manufacturers with the entire line of microcontrollers. But with current design tradeoffs, it is hard to see how the paging could be eliminated without making the instruction bus too large and expensive.

## ADVANCED INDIRECT ADDRESSING

Hand in hand with the deep data memory, the PIC18Cxxx family provides three separate indirect addressing pointers, each with five modes of operation. The indirect address pointers are composed of two registers that are concatenated to form a 12-bit wide word that can handle the complete data space, so no paging is necessary when using indirect addressing (a blessed relief). The five modes of operation are: do nothing, post-decrement, post-increment, pre-increment, and add an offset. The three independent indirect pointers in conjunction with the operating modes make manipulating and tracking such things as arrays and large amounts of data manageable. Each of these operating modes is described in Table 2.

Feature/device	18C452	18C442	18C252	18C242
<b>Package</b>	40-pin DIP & SOIC 44-pin PLCC 40-pin TQFP	40-pin DIP & SOIC 44-pin PLCC 40-pin TQFP	28-pin DIP & SOIC	28-pin DIP & SOIC
<b>I/Os</b>	34	34	23	23
<b>Program memory</b>	16384 instructions	8192 instructions	16384 instructions	8192 instructions
<b>Data memory</b>	1536 bytes	512 bytes	1536 bytes	512 bytes
<b>Clock speed</b>	40 MHz	40 MHz	40 MHz	40 MHz
<b>A/D (10 bit)</b>	8	8	5	5
<b>Programmable brownout</b>	Yes	Yes	Yes	Yes
<b>Programmable low voltage</b>	Yes	Yes	Yes	Yes
<b>USART</b>	Yes	Yes	Yes	Yes
<b>SPI</b>	Yes	Yes	Yes	Yes
<b>I<sup>2</sup>C (Master)</b>	Yes	Yes	Yes	Yes
<b>PWM (10 bit)</b>	2	2	2	2
<b>Timers</b>	4	4	4	4
<b>Hardware multiply</b>	8 × 8 single instruction	8 × 8 single instruction	8 × 8 single instruction	8 × 8 single instruction

Table 1—Here are the 18Cxx2 controller features. In addition to the standard features, note the large amount of program and data memory.

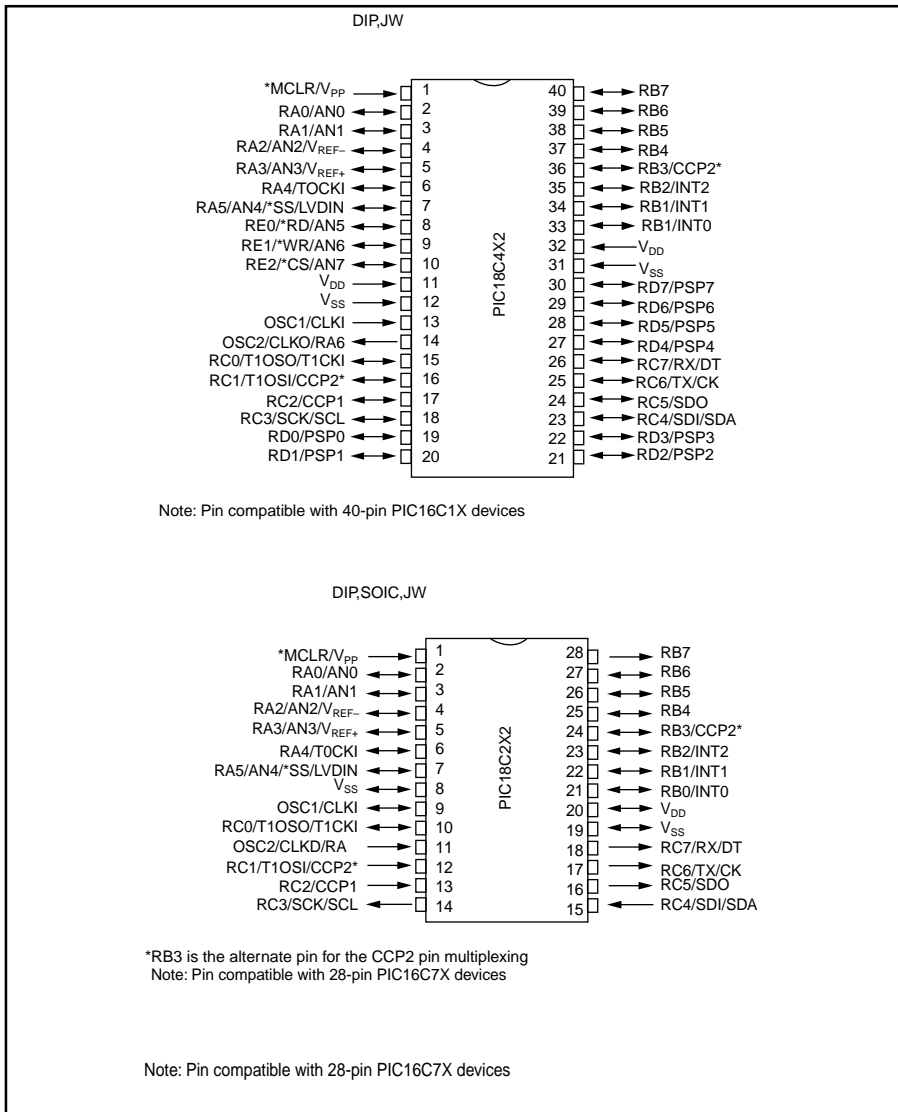


Figure 1

## DEEP AND ACCESSIBLE STACK

The PIC18Cxxx family has a 32-level hardware stack. Not only that, but the stack can be directly manipulated with new PUSH and POP instructions in the instruction set. In addition, the stack pointer is directly accessible as the STKPTR special function regis-

ter. The stack is mainly a program counter storage device for storing return addresses. However, it is possible to read from and write to the top value on the stack via special function registers.

This use is fairly limited and would be difficult at best to implement a

generic stack in the sense of storing and reading arrays of data and the like. A much better generic stack can be implemented using the indirect addressing registers.

## INTERRUPTS

The single level of interrupts in the mid-range family were a vast improvement over the no interrupts of the PIC16C5x family. However, there were still problems. Notably, only one interrupt could be serviced at a time, and more severely, you had to manually save all of the context-sensitive registers. The new PIC18Cxxx family sort of addresses this problem.

The PIC18Cxxx family has high- and low-priority interrupts. Each interrupt in the system can be manually set as a high- or low-interrupt priority. In addition, interrupt priority can be turned off. Unfortunately, this is really only a two-level priority system. So if there was a vital interrupt such as an absence of power, this would have to be the high-priority interrupt and all others would be low-priority interrupts.

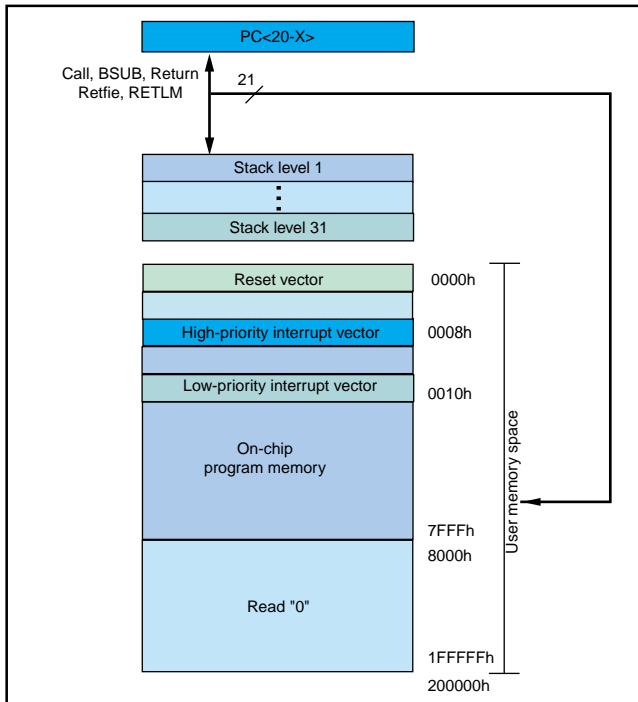
The other fix allows for fast interrupts. These automatically save the W, STATUS, and BSR registers. This saves the most important registers on a stack not accessible by you and returns them when the fast interrupt exits. Unfortunately this stack is only one layer deep, so a nested interrupt structure could not be supported with all interrupts running in fast mode.

## POWER-ON FEATURES

The PIC18Cxxx family includes a programmable brownout detect and

Operating mode	Description
Do nothing	After accessing the indirect data, the pointer is left as is. This is the current operating mode in the low- and mid-range PIC devices.
Post-decrement	After accessing the indirect data, the pointer is decremented by one. All carries between the two pointer registers are serviced internally.
Post-increment	After accessing the indirect data, the pointer is incremented by one. All carries between the two pointer registers are serviced internally.
Pre-increment	Before accessing the indirect data, the pointer is incremented. All carries between the two pointer registers are serviced internally.
Add an offset	The data in the indirect pointer is modified by the value in the working register (W) before the data access. The modification is a signed byte (2's complement) added to the indirect pointer, so the pointer can be offset by 127 to -128. The value in W and the pointer remains the same after the operation as before.

Table 1



**Figure 2**—The program memory of the 18Cxxx family is set up in a linear manner and can accommodate up to 1 MB of instructions.

low-voltage detect. The brownout detect levels are set using the configuration word during programming. This is a nifty feature that allows for different brownout levels to be detected depending on the application. This brownout unit uses ~50  $\mu\text{A}$ , which is a welcome relief from the initial brown-out circuits of the midrange devices that gulped down an astounding 425  $\mu\text{A}$ .

The low-voltage detect is under software control and can be set using a special function register. It can be turned on and off at will and can generate an interrupt, so when a low-voltage condition happens, the event is automatically accounted for. The potential drawback to the low-voltage detect circuit is that it runs off of the internal power rails. There are pluses and minuses to this configuration.

On the minus side, there is no way to directly interface to an external voltage, only the  $V_{\text{DD}}$  of the controller. So in a battery system with a step-up regulator, the voltage is read off of the regulated side and not the batteries. One way around this shortfall is to add an A/D channel to measure an external voltage. On the plus side, using the internal rail enables a low-voltage detect without an I/O pin.

processors, it is not a quantum leap from the 5-MIPS performance of the mid-range devices.

## CLOCK SPEED

The PIC18Cxxx family has a built-in PLL circuit that multiplies the internal instruction clock by four to generate the internal instruction clock. With the PIC architecture, this means 10-MIPS performance is available when running off of a system clock of 10 MHz. Unfortunately, the PLL is not rated to run with a clock faster than the 10 MHz. So, while the 10-MIPS performance is a sizable improvement over the maximum speed of earlier

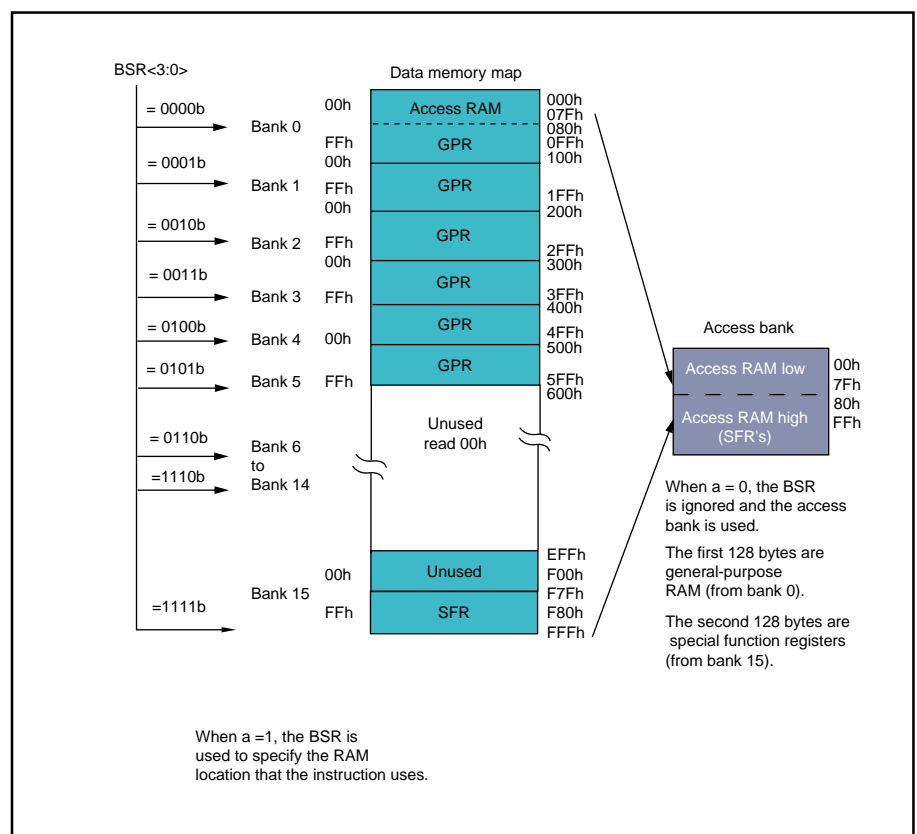
## 10-BIT A/D

The A/D unit on the PIC18Cxxx family is 10 bits. This is about 60 dB of signal-to-noise ratio for the A/D. This allows the controller to be used in higher precision systems than the mid-range PIC devices. The A/D functions just like the A/D units in the mid-range devices but with the added two bits. The results of the A/D span two 8-bit registers but can be justified to the MSB or LSB, depending on your needs.

The other interesting and useful feature of the A/D unit is that both the high and low references can be externally input. This allows the A/D unit to automatically take out an unwanted DC offset. For example, if a signal entered the A/D unit with a range of 2.5 to 4 V, the positive reference could be set to 4.096 V and the negative reference to 2.5 V to isolate the incoming voltage in the window of interest. This eases the demands on external conditioning circuitry on the front end of the A/D.

## HARDWARE MULTIPLIER

The PIC18Cxxx has an onboard hardware,  $8 \times 8$ , unsigned, multiplying



**Figure 3**—The data memory of the 18C452 has ~1.5 KB and is configured for ease of use.

circuit. The multiplier takes only one instruction cycle to perform the multiplication. Although not a 32-bit MAC unit in a DSP, this hardware multiplier can significantly speed up the response time of the controller to outside stimulus. Table 3 states the comparison between firmware-implemented multiplication and the hardware implementation of the PIC18Cxxx family.

As can be seen in Table 3, a 16 × 16 signed multiply implemented with the 8 × 8 hardware multiplier enjoys about a seven-times speed increase over the firmware implemented version. In many embedded systems, there can be many multiplies per loop, so the speed variation can make the difference between getting the job done and failing.

## TIMERS

The PIC18Cxxx has added four timers. Each of these timers can be configured as 16-bit timers with numerous post-scaler and pre-scaler options.

One of the long-standing issues with the low- and mid-range PIC devices is the sharing of a pre-/post-scaler between TMR0 and the watchdog timer. By assigning the scaler to one peripheral, the other peripheral is left high and dry and finds itself bereft of a scaler of any type. This means that, if the scaler is assigned to the watchdog timer, TMR0 will find itself flying along at a 1:1 ratio with the instruction clock. This often makes the two peripherals mutually exclusive at worst, or a hodgepodge of scaler switching at best. The PIC18Cxxx rectifies this long-standing problem by giving the WDT its own post-scaler.

Now for the bad news. Unfortunately, the WDT post-scaler is found in the configuration fuses and can only be set once during programming. After set, it cannot be reliably changed for all instances. However, even with the

post-scaler in the configuration fuses, the WDT/TMR0 combination in the PIC18Cxxx is a vast improvement over the kludge that preceded it in the low- and mid-range devices.

## CCP/PWM

The CCP/PWM modules on the PIC18Cxxx family are essentially the same as the CCP/PWM modules on the mid-range PICmicro family, however, there is one intriguing new feature. The CCP2 pin can be assigned to two different I/O pins, RC1 or RB3. This assignment is done during programming in the configuration fuses, so although it cannot be changed on the fly, it can ease PCB layout and the like.

## USART

The PIC18Cxxx family contains a full function USART, much like the mid-range PIC devices, with an additional feature. The PIC18Cxxx USART allows 9-bit transmission and reception, with the ninth bit (MSB) acting as an address signifier. With this configuration, the USART in the PIC18Cxxx can be set up to generate an interrupt only when it receives a byte with the ninth bit set. If the ninth bit is set, the program can look at the byte and see if the eight LSBs match the address of the PIC18Cxxx chip.

In a multi-drop system (many devices sharing the same serial line), this addressing scheme is a great first step to increasing the throughput and reliability of the communication channel. Listing 1 shows an assembly code fragment that demonstrates addressing setup and use in the PIC18Cxxx.

## I<sup>2</sup>C MASTER

Almost from the beginning, controllers supported slave mode I<sup>2</sup>C communication. However, this was of limited use because most embedded systems have a single controller that acts as a

master and talks to I<sup>2</sup>C peripherals such as EEPROM, real-time clocks, and A/Ds. So, the hardware I<sup>2</sup>C port sat dormant on many a mid-range PIC device while the controller used I<sup>2</sup>C devices and communicated to them with a bit-banged firmware solution. Fortunately, the new PIC18Cxxx series rectifies this gaffe and provides a hardware I<sup>2</sup>C master solution.

The new I<sup>2</sup>C peripheral fully supports all master modes of the Philips I<sup>2</sup>C specification in the form of Master Synchronous Serial Port (MSSP). Multi-master mode, 7- and 10-bit addressing, clock arbitration, and 400-kHz mode are supported by the new MSSP. I<sup>2</sup>C is an involved communication protocol, and it is easy to lose the forest through the trees. However, it is possible to fully realize the benefits of the I<sup>2</sup>C port on the PIC18Cxxx. Listing 2 gives some sample code that shows how easy it is to implement I<sup>2</sup>C communication. Note that the code is basic and does not test for all of the various error conditions.

## TABLE READ/WRITE

For those of you who are not familiar with the PIC17Cxxx series of controllers, perhaps the most intriguing new feature on the PIC18Cxxx family is the table read/write capability. This allows the PIC18Cxxx to directly read from and write to its own program memory. Almost surely, the most important aspect of the read capability is that it allows the PIC18Cxxx controllers to perform program memory integrity checks versus a checksum. This allows the PIC18Cxxx controllers to periodically monitor their program memory and ensure that the code is working properly and has not been changed.

For mission-critical systems, this is an absolute must. Additionally, the read capability can be used to read the configuration fuses and ID locations directly. These could then be reported directly to a querying unit for inventory purposes and system verification. Listing 3 shows the table read command being used to read some of the con-

Multiplication type	Cycles/program length with hardware multiplier	Cycles/program length with firmware implementation
8 × 8 unsigned	1/1	69/13
16 × 16 unsigned	24/24	242/21
16 × 16 signed	36/36	254/52

Table 2—The indirect memory modes of the 18Cxxx family encourage creative memory manipulation and data storage.

figuration bytes.

More intriguing is the write capability. Basically, the PIC18Cxxx chip can modify its program space with executable code. Note that with the initial PIC18Cxxx offerings, the program space is still EPROM-based and inherently one-time-programmable. In the PIC18Cxxx, any non-programmed byte is all ones (FF hex). A one can always be changed to a zero, but a zero can revert to a one only through UV exposure. So if program space already has executable code, writing over the program space with the table-write command would simply destroy the code at best and make some new unintended instruction at worst.

With this limitation in mind, there is still a myriad of uses for the table write command that spring to mind. Because the EPROM memory is OTP, any table write is going to store information in a nonvolatile manner. Thus, the table write command could be used for maintenance logs, serial numbers, time stamps, and so on. In addition, if the reset code was appropriately designed, the table write command could be used to patch existing code or provide field upgrades.

This all sounds great in theory, however, implementation is a bit more complicated. The first major problem with the table write command is that it requires 12 V on the /MCLR line. This is referred to as the programming voltage ( $V_{pp}$ ). The tricky part for the table write command is providing the 12 V<sub>pp</sub> on \*MCLR when needed and then the typical reset voltage during

other times. This is a minor problem and easily overcome with a small amount of inexpensive circuitry. Figure 4 shows a circuit that can be used to accomplish this.

Although these initial voltage and one-time writes are limiting at the moment, it is easy to see where this functionality can lead. On the drawing board for the PIC18Cxxx family are flash memory versions of the part. When these become available, the voltage and memory reuse problems should disappear. Listing 4 shows some code that implements a table write to program memory. A pushbutton ends the programming cycle and wakes up the controller.

## CURRENT STATUS

As mentioned earlier, the PIC18Cxx2 family is currently available. Microchip has just rolled out the first devices of the PIC18Cx58 series of chips. These are an extension of the base PIC18Cxxx architecture with integrated CAN 2.0B bus support. When these new PIC18Cxxx chips hit the streets, the PIC18Cxxx family will have as many members as the PIC17Cxxx family.

Because of the programming advantages offered by the PIC18Cxxx family, most of the growth for the high-end controllers will most likely be taking place in the PIC18Cxxx family.

By perusing the hits and misses discussed in this article, a trend can be found. As a group, most of the misses are things that can be worked around or are esoteric. In short, the architectural misses are minor and can be worked through without too great a headache. The hits are, for the most part, bonuses of the chip itself. They are things that imply Microchip's devotion during the chip specification and design process to achieving a better microcontroller family than is currently offered.

There is some design migration that the PIC18Cxxx family might capture as mid-range designs mature and features are added, but the bulk of designs will most likely have to come from other sources, notably engineers using low-end 16-bit processors or other high-end 8-bit controllers.

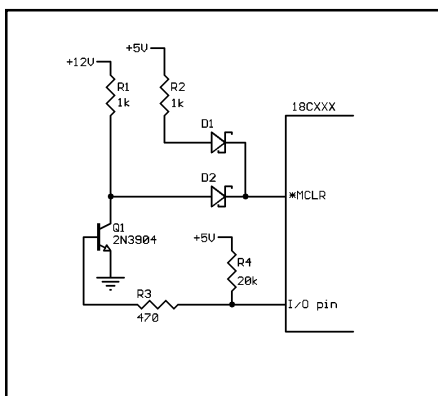
On the balance, the PIC18Cxxx family is a strong entry into the expanding line of controllers. It will be interesting to watch what happens in the next 18 to 24 months with the PIC18Cxxx family to see if it has as large of an effect on the high-end 8-bit market as the PIC12Cxxx, PIC16C5x, and PIC16Cxxx families had on the low-end market. ■

*David Brobst is a founding partner of Solutions Cubed, an embedded control design coompany. Some of Solutions Cubed's recent projects include deep sea ROVs, a communication protocol adapter, a PID controller for a DC servo motor, and instrumentation. You can reach David's company web site at [www.solutions-cubed.com](http://www.solutions-cubed.com).*

## SOURCES

IR Module  
Sharp Electronics Corp.  
(360) 834-8611  
Fax: (201) 529-8425  
[www.sharp-usa.com](http://www.sharp-usa.com)

CA 3082, MAX 455, buzzer, seven-segment digit  
JDR Microdevices  
(800) 538-5000  
(408) 494-1400  
Fax: (408) 494-1420  
[www.jdr.com](http://www.jdr.com)



**Figure 4**—The table-write functionality can be realized with a simple circuit and inexpensive components.

Circuit Cellar, the Magazine for Computer Applications. Reprinted by permission. For subscription information, call (860) 875-2199, [subscribe@circuitchellar.com](mailto:subscribe@circuitchellar.com) or [www.circuitchellar.com/subscribe.htm](http://www.circuitchellar.com/subscribe.htm).

## Listing 1

```
;
;After saving the context, check for the interrupt.  If it is the receive
;interrupt, then check the address.  If the address matches the address of the
;18CXXX found in the ADDRESS register, then someone is trying to communicate
;to the 18CXXX, so receive the rest of the information, without addressing.
;
Interrupt_Check
    btfss PIR1,RCIF
    goto Interrupt_Continue
IC_Receive_Interrupt
    movf RCREG,W                ;Check address
    subwf ADDRESS,W
    btfss STATUS,Z             ;If Z=1 then address match
    goto Interrupt_Continue
IC_18CXXX_Addressed
    bcf RCSTA,ADDEN            ;No address detection for remainder
;
;Continue with whatever other housekeeping is necessary in the interrupt.
;
Interrupt_Continue

INT_Low_Routine
    retfie

;
;
;*****
Main
;
;Setup the USART for nine bit receive with addressing mode interrupt
;enabled.
;
USART_Init
    movlw H'60'                ;9 bit transmission, transmission on
    movwf TXSTA                ; low speed
    movlw H'F8'                ;Serial port on, 9 bit reception,
    movwf RCSTA                ; receive on, addressing enabled
    bsf PIE1,RCIE              ;Enable receive interrupt, default
    movlw H'CO'                ; low priority
    movwf INTCON
;
;After this point the USART is continuously receiving, however it will only
;generate an interrupt if the MSB of the received 9 bit word is set.  So
;continue with normal code.
;
Code_Continue
```

## Listing 2

```
;*****
;I2C_RECEIVE_BYTE - This routine receives one byte of data from the I2C bus.
;    Called From:      Main_Loop
;    Registers Used:   PIR1, SSPBUF, SSPCON2
;    Subroutines Called: None
;    Enabled Interrupts: None
;    Stack Depth:     1
;*****
I2C_RECEIVE_BYTE
    bsf SSPCON2,RCEN          ;Turn on a receive
I2C_RB_Wait
    btfss PIR1,SSPIF         ;See if byte received
    goto I2C_RB_Wait
```

(continued)

**Listing 2** —Continued

```

        bcf    PIR1,SSPIF          ;No spurious interrupt
        movff SSPBUF,POSTINCO     ;Store byte received
I2C_RECEIVE_BYTE_end
        return
;
;
;*****
;I2C_SEND_BYTE - This routine sends one byte of data out on the I2C bus.
;   Called From:      Main_Loop
;   Registers Used:   PIR1, SSPBUF
;   Subroutines Called: None
;   Enabled Interrupts: None
;   Stack Depth:     1
;*****
I2C_SEND_BYTE
        movwf  SSPBUF
I2C_SB_Loop
        btfss  PIR1,SSPIF          ;See if byte sent
        goto  I2C_SB_Loop
        bcf    PIR1,SSPIF          ;No spurious interrupt
I2C_SEND_BYTE_end
        return
;
;
;*****
Main
;
;First setup the MSSP port for I2C operation.
;
I2C_Setup
        clrf   SSPSTAT              ;No slew rate control
        movlw  H'2B'                ;MSSP on, I2C master mode
        movwf  SSPCON1
        clrf   SSPCON2              ;Leave I2C idle to begin with
;
;If sending I2C data use the following loop.  The data to send is assumed to
;be preloaded in a buffer that has the total number of bytes to send tallied
;in TEMP1.  The indirect pointer 0 (FSR0) is assumed to point to the top of
;the buffer.
;
I2C_Send
I2C_Send_Start
        bsf    SSPCON2,SEN          ;Send start bit
I2C_SB_Wait
        btfss  PIR1,SSPIF          ;See if start bit has been sent
        goto  I2C_SB_Wait
        bcf    PIR1,SSPIF          ;No spurious
I2C_Send_More
        movf   POSTINCO,W           ;Move byte to send to W, increment FSR0
        call  I2C_SEND_BYTE
        decfsz TEMP1,F              ;See if sent all bytes
        goto  I2C_Send_More
I2C_Stop_Bit
        bsf    SSPCON2,PEN          ;Send stop bit
I2C_Stop_Wait
        btfss  PIR1,SSPIF          ;See if stop bit sent
        goto  I2C_Stop_Wait
        bcf    PIR1,SSPIF
;
        goto  Program_Continue
;
;If need to receive I2C data use the following loop.  To receive data, three
;bytes must first be sent to tell a slave device to send information to the
;18CXXX.  After that receive data.  The first three bytes to send are assumed
;to be in a buffer with FSR0 pointing at the first byte.  When receiving data
;the data is loaded into the buffer using indirect addressing.  The number of
;bytes to receive are assumed to be in TEMP1.  The indirect pointer 0 (FSR0)

```

(continued)

Listing 2— Continued

```

;is assumed to point to the top of the buffer.
;
I2C_Receive
I2C_Receive_Start
    movlw  H'02'                ;Send the first three bytes
    movwf  TEMP2
    bsf    SSPCON2,SEN          ;Send start bit
I2C_Rec_Wait
    btfss  PIR1,SSPIF          ;See if start bit has been sent
    goto   I2C_Rec_Wait
    bcf    PIR1,SSPIF          ;No spurious
I2C_Rec_More
    movf   POSTINC0,W           ;Move byte to send to W, increment FSR0
    call   I2C_SEND_BYTE
    decfsz TEMP2,F              ;See if sent all bytes
    goto   I2C_Rec_More
I2C_Repeat_Start
    bsf    SSPCON2,RSEN
I2C_RS_Wait
    btfss  PIR1,SSPIF          ;See if repeated start bit has been sent
    goto   I2C_RS_Wait
    bcf    PIR1,SSPIF          ;No spurious
I2C_RS_Address
    movf   POSTINC0,W
    call   I2C_SEND_BYTE
;
;Have now sent the setup, so receive the number of bytes necessary.
;
    movff  TEMP1,TEMP2          ;Load number of bytes to receive
    movlw  H'03'                ;Reset to top of buffer
    movwf  FSR0L
I2C_Receive_Loop
    call   I2C_RECEIVE_BYTE
    bcf    SSPCON2,ACKDT        ;Assume send an ACK
    movf   TEMP2,W
    btfsc  STATUS,Z             ;If Z=1 then stop receiving
    goto   I2C_Receive_Stop
I2C_Receive_Continue
    decf   TEMP2,F              ;Keep track of bytes received
    bcf    SSPCON2,ACKDT
    bsf    SSPCON2,ACKEN        ;Send ACK
I2C_IRC_Wait
    btfss  PIR1,SSPIF
    goto   I2C_IRC_Wait
    bcf    PIR1,SSPIF          ;Ensure no spurious
    goto   I2C_Receive_Loop
I2C_Receive_Stop
    bsf    SSPCON2,ACKDT        ;Send NAK
    bsf    SSPCON2,ACKEN
I2C_IRS_Wait
    btfss  PIR1,SSPIF
    goto   I2C_IRS_Wait
    bcf    PIR1,SSPIF          ;Ensure no spurious
I2C_IRS_Stop_Bit
    bsf    SSPCON2,PEN          ;Send stop bit
I2C_IRS_SB_Wait
    btfss  PIR1,SSPIF
    goto   I2C_IRS_SB_Wait
    bcf    PIR1,SSPIF          ;Ensure no spurious

    goto   Program_Continue
;
;The rest of the program goes here
;
Program_Continue

```

### Listing 3

```
;
;This code fragment simply reads the first two configuration bytes and stores them in
;TEMPO and TEMP1 respectively. Pointer incrementing was not used to avoid clouding the
;issue.
;
Table_Read
    movlw  H'00'                ;Set at beginning of configuration
    movwf  TBLPTRL              ;          bytes
    movlw  H'00'
    movwf  TBLPTRH
    movlw  H'30'
    movwf  TBLPTRU
    tblrd*                       ;Read 1st configuration byte
    movff  TABLAT,TEMPO         ;Store 1st configuration byte
    incf  TBLPTRL,F            ;Goto next configuration byte
    tblrd*                       ;Read 2nd configuration byte
    movff  TABLAT,TEMP1        ;Store 2nd configuration byte

Code_Continue
```

### Listing 4

```
;
;This code fragment writes 2 bytes (1 16 bit memory location) to the program memory
;of the 18CXXX. It assumes that the address to write to is in TEMPO:TEMP2 to start with.
;In addition, the bytes to write are assumed to be in TEMP3:TEMP4. A falling edge
;RBO interrupt is used to wake the device out of sleep.
;
Main_Loop
    bcf  INTCON2,INTEDG0        ;INTEG0 = so falling edge
    bsf  INTCON,INTOIE         ;RBO will wake up
    bcf  INTCON,INTOIF
    bsf  RCON,LWRT             ;Enable table writes
    bcf  PORTA,Table_Program    ;Put /MCLR to +12V
    movff TEMPO,TBLPTRU         ;Go to start of programming block
    movff TEMP1,TBLPTRH
    movff TEMP2,TBLPTRL
ML_1st_Word_Write
    movff TEMP3,TABLAT          ;Assume always start at even address
    tblwt*+                     ;Post increment
    movff TEMP4,TABLAT
    tblwt*+
    nop                          ;Device goes to sleep, will wake up from
    nop                          ;          falling edge RBO

Code_Continue
```