CIRCUIT CELLAR®
THE MAGAZINE FOR COMPUTER APPLICATIONS

Circuit Cellar, the Magazine for Computer Applications. Reprinted by permission. For subscription information, call (860) 875-2199, or www.circuitcellar.com. Entire contents copyright ©2001 Circuit Cellar Inc. All rights reserved.

FEATURE ARTICLE

by Robert Lacoste

CONTEST WINNER

The XY-Plotter

Drive High-Resolution LCDs For Less

Robert spent nearly 100 hours building his high-performance, LCD-based XY-Plotter. Now that he has written about the process, it should take you much less time to construct your own. Follow along as he shows you how to maximize your time and money when driving graphic LCD panels.

Implementing a graphical LCD is an excellent way to drastically change the look and feel of a project. You can transition from a classic technician-oriented, two-lined text LCD to a user-friendlier device. Unfortunately, graphic LCDs are resource-hungry devices, both in terms of memory and CPU power. So, you're forced to either create lovely minimalist designs with an intelligent LCD (with on-board LCD controller, processor, and memory, as described by Jeff Bachiochi in *Circuit Cellar* 150) or swap the usual microcontroller for a classic microprocessor, memory, display controller set.

Both options are expensive, and there doesn't seem to be another solution. For instance, the 240 × 320 pixel display used in this project eats one 4-bit nibble every 780 ns, and it needs a minimum of 10 KB of RAM just to store the displayed bitmap. Thus, it's impossible to drive it directly with a high-end PIC controller providing a 100-ns cycle when clocked at 40 MHz and 1536 bytes of RAM, right? Nothing useful can be done in less than seven assembly instructions per nibble, correct?

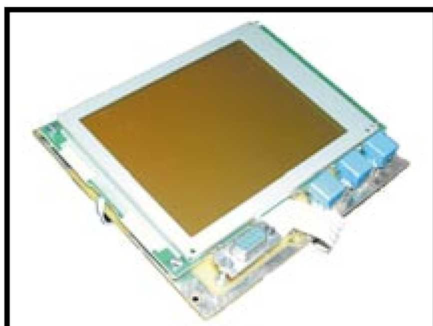

Photo 1—*The large 240 × 320 LCD is affixed to the PCB. The three control push buttons and the screen dump RS-232 connector are along the bottom edge.*

As you probably expect, this project proves that the impossible is possible with an optimized firmware design. You'll even learn that it's possible to use this minimalist concept for something useful!

PLOTTER BASICS

I got the idea for the XY-Plotter from an old spectrum analyzer sleeping in my garage. Despite the fact that the heavy analyzer's CRT display was dead, the radio parts worked well. So, from time to time, I used it with an oscilloscope as an output device. The arrangement was cumbersome and uncomfortable to implement. Consequently, I decided to repackage the analyzer in a smaller, prettier enclosure and design an LCD alternative to the CRT display. Because I wanted the ability to reuse the design, I chose to develop a generic display subsystem, the XY-Plotter (see Photo 1).

The XY-Plotter is an autonomous analog-like display with two main *x* and *y* inputs. Continuously scanning the two inputs, the plotter displays them on a real-time x-y graph by way of configurable modes (i.e., Sample, Maximum, Peaks, or Average) with Accumulate and Hold controls. Moreover, a set of analog and digital auxiliary inputs allow you to display configurable information on the screen such as center frequency, reference level, scan time, and so on (see Figure 1). Lastly, an RS-232 port dumps hard copies of the screen to a host computer.

As you can see in Figure 2, the XY-Plotter's overall architecture is simplistic: it contains nothing more than a PIC18F252, a few MCP6022 analog amplifiers, a low-cost LCD, and several other low-cost components. I built an integrated power supply using a MCP1541 precision voltage reference.

LCD TIMING REQUIREMENTS

I used an FTN reflective Epson ECM-A0635-2 LCD with a 240 × 320 pixel black and white screen (see Figure 3). The display is extremely dumb, and it should be supplied in real time with the required pixels. The host controller must send a new frame every 15 ms. Each frame includes 240 lines, and each line includes 320 pixels grouped into 4-bit nibbles.

In addition to the 4-bit data input port, the controller must also supply three clocks: frame, line, and nibble. One new nibble must be delivered every 780 ns, which I arrived at via the following equation: 15 ms/240/(320/4). Note that for this project I used the display turned by 90° in Portrait mode

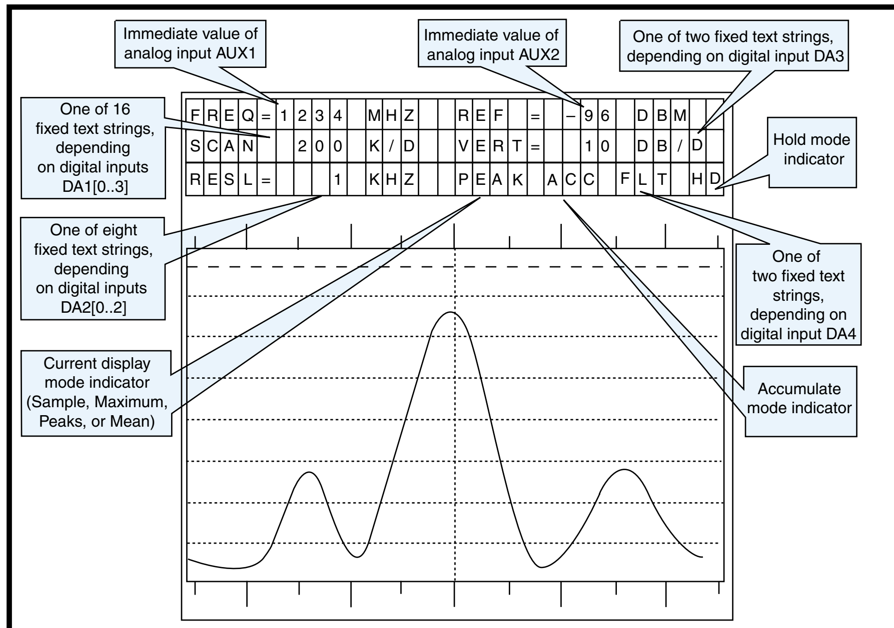

Figure 1—*The XY-Plotter screen displays a real-time x-y graph as well as three lines of configurable textual status information and real-time measurements.*

(320 pixels high, 240 wide), so the scan lines are vertical.

MICRO OF CHOICE

I chose the PIC18F252 microcontroller on the basis of certain project-specific criteria. First, I needed speed. The more instructions in these bloody 780-ns nibbles the better. I also wanted a significant amount of RAM. I didn't store the full bitmap but chose instead to store minimum, maximum, and sample values for each column already requiring 768 bytes. In addition, I needed a precision A/D converter and a large program memory for amassing the huge tables used in the design (including character bitmaps). Lastly, flash memory was necessary for configuring the display for each application.

One or two years ago, these requirements probably would have been impossible to fulfill, but, thanks to suppliers like Microchip, they are now easily satisfied, with the PIC18F*xx*2 product line in particular. The PIC18F252, for instance, has 1.5 KB of RAM and plenty of flash memory (32 KB).

GRASPING THE SCHEMATICS

Figures 4 and 5 are schematics of the XY-Plotter. Each analog input (X, Y, AUX1, and AUX2) is conditioned thanks to half of an MCP6022 dual rail-to-rail op-amp. Two 20-turn trimmers per input give you the ability to easily adjust the full-scale deviation as well as the DC offset for each channel. One of the channels, AUX2, even includes two inputs summed by the analog amplifier.

The values of the resistors used for each amplifier stage can be adjusted for each specific application to accommodate different input ranges and adjustment precision. It is not obvious how to design an amplifier stage with positive and negative offset adjustment without a negative power supply. Here's my trick: A fixed positive voltage, which is derived from a 0.6-V reference, is first subtracted from the input signal, and then a variable positive voltage is added to it, providing an offset that's either positive or negative. I used Excel to calculate the resistors.

The PIC is clocked by a 10-MHz crystal up-converted to 40 MHz thanks to the on-board PLL. The LCD is directly connected to the PIC I/O lines, whereas the auxiliary digital inputs, which are used to dynamically select the text for the screen, are either direct inputs of the PIC or multiplexed with LCD data lines (thanks to a firmware reconfiguration on the fly).

Lastly, the ubiquitous MAX232 does what it's intended to do. It should be noted that I included an in-circuit programming header just in case; however, I haven't had to use it thanks to Microchip's boot loader firmware. All of the programming was accomplished though the serial port.

POWER SUPPLIES

The power supply is a significant part of the design (see Figure 5). First, I needed a clean 5 V. I was already using all of the PIC's analog inputs, so I couldn't configure its ADC in external-reference mode. I still needed a stable reference for the analog-to-digital conversions. After experiencing a few headaches, I decided to use the PIC in its 0- to 5-V reference mode and to provide a well-stabilized 5 V. I implemented a high-precision MCP1541 voltage reference and built a discrete power supply around a low-drift LMC6462 op-amp. The second part of the op-amp is used to get the 0.6-V reference drawn on by the offset circuitry.

The LCD was hard to deal with because it needed both a –24-VDC input (for the display itself) and a 100-VAC power for the EL backlight. To limit the number of power inputs, I went with a small 5- to ±12-VDC con-

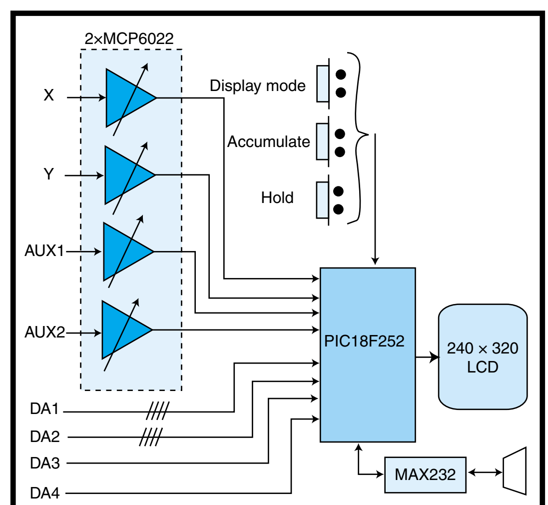

Figure 2—*The XY-Plotter's hardware design is simple. The PIC18F252 manages everything including LCD pixel generation in real time. A couple of Microchip MCP6022 rail-to-rail op-amps were used to scale the analog inputs.*

verter to generate the –24 V switched by two transistors under PIC control. I couldn't find a ready-made DC/AC converter for the backlight in time, but it wasn't an issue. I built a pretty one with a small 220/12-V transformer driven by a NE555 timer. Done.

PROTOTYPE ASSEMBLY

I built a simple PCB for this project (see Photo 2). All the components fit easily because I wanted the size of the PCB to be identical to the LCD. Note that the front panel components, including the push buttons and RS-232 connector, are soldered on the bottom. All of the trimmers are easily accessible with a screwdriver, because they are laterally shifted from one to the other.

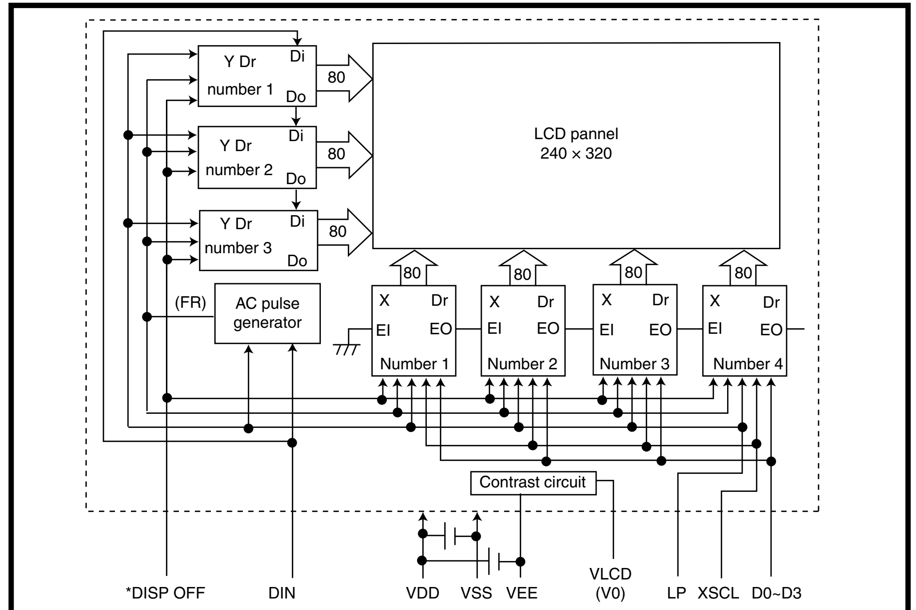

Figure 3—*The EPSON ECM-A0635-2 display doesn't include anything more than lines, columns, registers, and drivers. The host controller must send pixels with strict timing requirements and supply frame, as well as line and pixel clock signals.*

FIRMWARE DESIGN

The hardware side of this project was straightforward, so if you're imaging that the firmware was more difficult, you're right. Figure 6 illustrates the overall architecture. In order to comply with the requirement of seven instructions per nibble, I didn't use an interrupt. I built a fully sequential program flow.

A main loop is executed every

Figure 4—*The XY-Plotter's power supply isn't included in this schematic. An MCP6022 analog amplifier, with scale and offset controls, scales each analog input. Some of the microcontroller's I/O lines are multiplexed to limit the I/O count requirement.*

15.8 ms. It starts with a frame-batch routine that manages the push buttons, and more importantly reads the auxiliary inputs (analog and digital) and generates the text that will be displayed in the first lines. The text is stored as ASCII characters in RAM using 90 bytes (3 × 30).

If you want to study the binary-to-decimal conversion routine, which I found on the 'Net, refer to the Resources section at the end of this article. The frame-batch routine also manages the UART by way of a simple protocol. Then a loop is executed for each of the 240 columns in the display. At each iteration, a line-batch routine is first executed. This routine reads and manages the *x* and *y* analog values (storing *y* minimum, maximum, and sample values for each *x* value in three 256-byte RAM areas). The display blanking (i.e., *y* is not stored when *x* is reducing) is also managed.

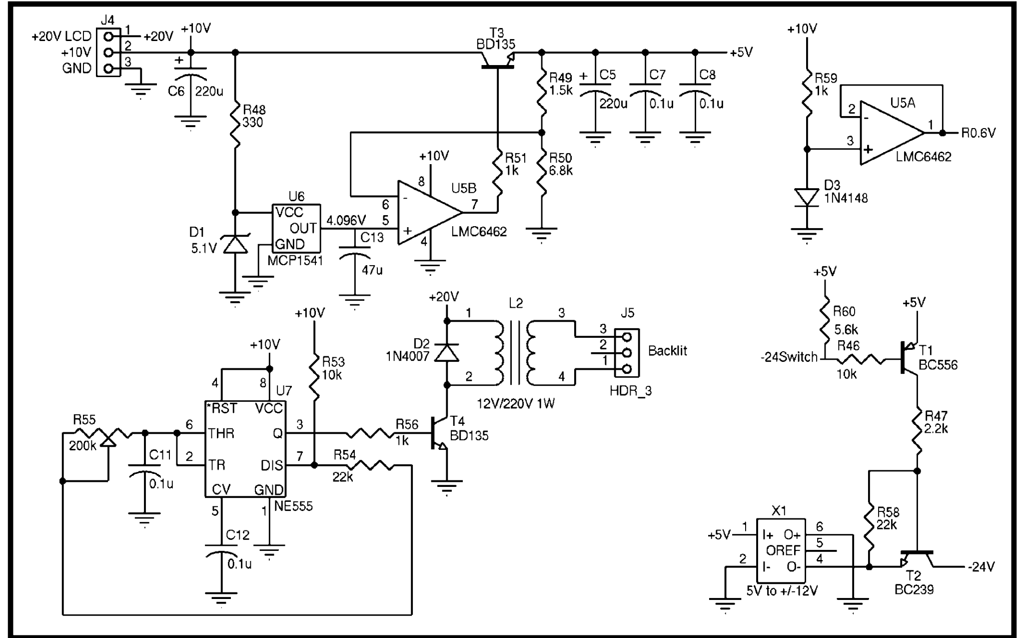

Figure 5—*The power supply includes four independent subsystems, one of which is the main 5-V regulator, which I built using a high-precision Microchip MCP1541 reference. I used a 5- to –24-V converter for the LCD. A homemade converter supplies the backlight voltage (100-V AC). Lastly, note that a 0.6-V reference is provided for offset control.*

The last step is tricky. For each line, the firmware must generate the nibbles to send the LCD on the fly. It must first send the nibbles corresponding to the graphic area (the back of the screen depicted in Figure 1) and then the ones for the three text lines at the top. Now let's discuss the details.

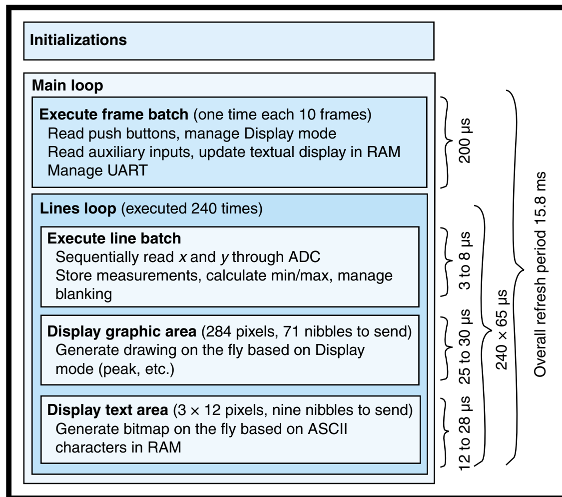

Figure 6—*The most critical section of this chart, which shows the architecture and timing of the firmware, is the graphic display routine. Basically, 71 nibbles must be sent in 30 µs, giving 422 ns per nibble or four PIC instructions per nibble (even at 40 MHz).*

GRAPHIC DISPLAY

How can you generate the graphic display on the fly? The fixed parts (e.g., borders and scales) are easily sent to the LCD with the proper timing. The graph is built in real time from the minimum, maximum, and sample values. It also depends on the display mode (see Figure 7).

The LCD is used in Vertical mode (320 pixels high), so the scan is vertical, too. Thus, the successive nibbles sent to the LCD correspond to successive vertical blocks of four pixels. In order to generate them, an optimized algorithm is implemented based on another trick: For each column, there is only one black line surrounded by whites. First, the black line's two extremities (ystart and ystop) are calculated based on the operating mode. Then, a loop sends an optimal number of fully blank nibbles followed by (depending on the ystart and ystop values) precalculated bitmaps that correspond to the different situation and are stored in a precalculated table as well as full black or full white nibbles in good quantity. For reasons of efficiency, the flash memory-based table is cached at startup in a RAM page. Figure 8 provides visual description of the algorithm.

TEXT DISPLAY

The three text lines are also generated on the fly based on the ASCII characters that are stored in RAM. For this purpose, a specific character bitmap was precalculated and stored in flash memory. The table gives the successive nibbles to send to the dis-

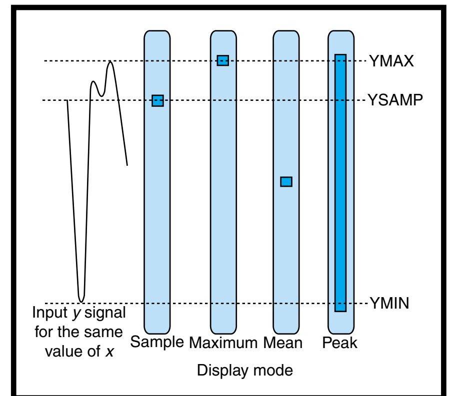

Figure 7—*Four display modes are supported by the XY-Plotter. Sample mode simply plots the first* y *value acquired for each* x *value. The Maximum mode plots the highest* y *for a given* x*. The Mean mode isn't in fact a true mean; it simply displays the midpoint of the minimum and maximum values. Last but not least is Peak mode, which displays a line showing all of the* y *values measured for a given* x*.*

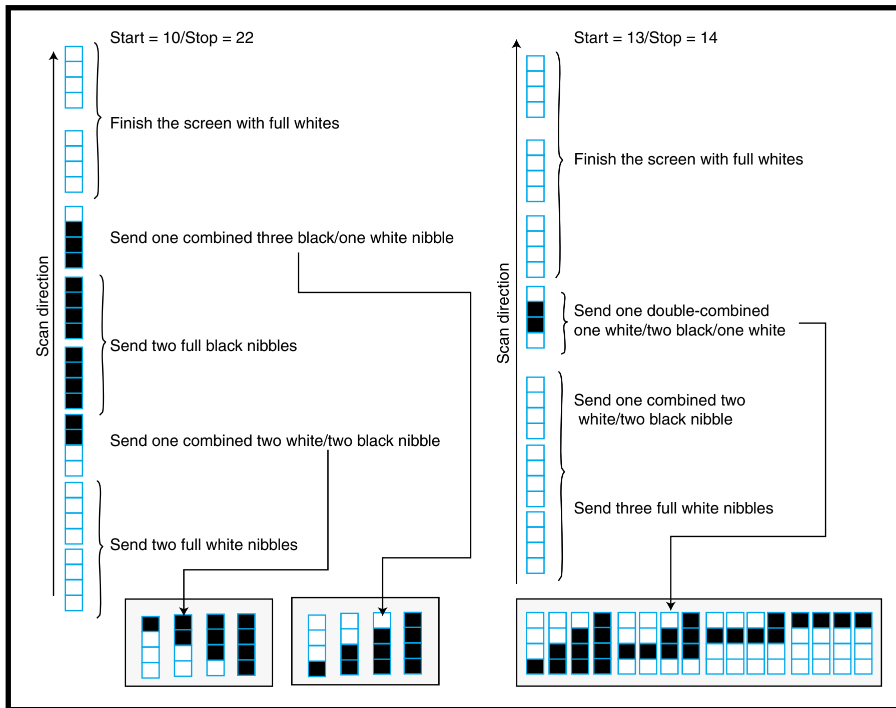

Figure 8—*The 4-bit nibbles are generated in real time for each of the LCD's scan lines based on the position of the first and last black pixel on that column. The firmware first calculates how many 0000s must be sent, and then two things can happen: If all the black pixels to draw are in the same nibble, then a combined white/black/white nibble is extracted from a table and sent to the display (on the right). Otherwise, one white/black transition nibble is sent, followed by the required number of full black nibbles, and followed by one black/white transition nibble (on the left).*

play for each character (from back to top and from left to right). Each character is encoded in an 8 × 12 pixel bitmap, giving 30 (240/8) characters per line.

A significant overhead is needed at the start of each character (first scan line out of the eight) in order to precalculate the different pointers. I built this unusual character bitmap table in Excel, starting with a standard 8 × 12 bitmap I found on the Internet.

LINE-BATCH ROUTINE

The line-batch routine manages the acquisition of the *x* and *y* analog values as well as the storage of the minimum, maximum, and sample values in RAM. I built the routine as a five-stage step machine (see Figure 9). Each step corresponds to a different acquisition sequence.

You can't lose time with this architecture. A full pair of *x* and *y* values is acquired every 260 µs (4 × 65µs), which produces a satisfactory 3.8-kHz update rate. Depending on the scan rate you apply (i.e., the frequency of the saw-tooth applied on the *x* input), two modes are automatically executed. If the scan rate is lower than 15 Hz (3.8 kHz/256) or the scan time is higher than 7 ms per division (1/15 × 10) using the usual scope vocabulary, then more than one *y* value is acquired for each *x* value per scan, enabling functionality such as minimum, maximum, and peaks.

If the scan speed is higher (up to 500 Hz), an equivalent time-sampled display is generated, and minimum, maximum, and peak measurements are only available in Accumulate mode (i.e., no resetting of the minimum and maximum between scans).

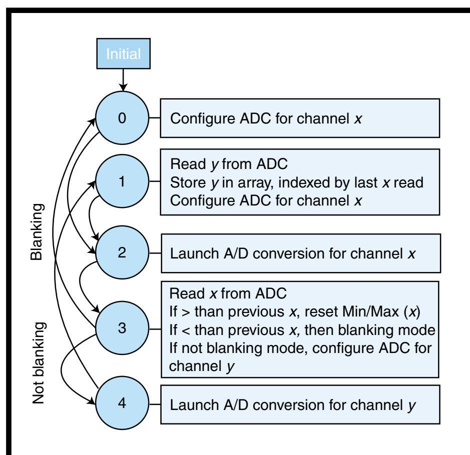

Figure 9—*The acquisition of the* x *and* y *analog values is managed thanks to a five-state machine executed each time the line-batch routine is called (each 65 µs). This allows you to comply with the PIC ADC timing (precharge, conversion, and then read) without losing any time.*

OPTIMIZATION TIPS

Optimizing the firmware's cycle count requires a huge effort. For instance, one of its basic tasks is to send *N* pulses to the LCD's nibble clock input. A loop already needs five cycles to do this, but remember that you have time for less than seven instructions per nibble, and the firmware has more to do than simply send clock pulses! So, you'll need additional optimization techniques like code expansion and the calculated `goto` procedure (see Listing 1).

I used the calculated `goto` technique extensively. Basically, I was manually unrolling the code like an optimized compiler does (or tries to do). For instance, I used a long calculated `goto` table to select the specific line-generation algorithm for each column in the display (e.g., graduations, plain line, ordinary curve column, etc.). The result is a strange assembly listing to read but an interesting one to write!

Another tip is to copy, at startup, the combined pixel table from flash memory and paste it in RAM. An indirect access to RAM is quicker than a table read from flash memory.

MEMORY REQUIREMENTS

The aforementioned firmware optimizations are memory hungry. Fortunately, with 32 KB of flash memory it's not an issue. My firmware currently uses only 10 KB.

I used the PIC18F252's entire RAM. Three pages at 256 bytes each were used to store the respective minimum, maximum, and sampled *y* value for each *x* value. One page was devoted to the storage of the ASCII text, although only 90 bytes were actually needed. One last 256-byte page was used to store the bitmap patterns. That left 256 bytes for general-purpose variables. All in all, that's 1536 bytes.

DEVELOPMENT PROCESS

The project was developed with the MPLAB environment and simulator. I also used Microchip's boot loader firmware (AN851) to burn flash memory, which is an interesting feature even if firmware improvements are welcome. In particular, no on-chip debug facility is currently provided (e.g., breakpoints), but I'm sure they'll be in the next version.

Also note that the AN851 boot loader doesn't provide an automatic reentry facility. As soon as an application firmware is downloaded and activated, there's no way to reactivate the bootloader without specific user-supplied application code (like simultaneously pressing the three keys at power-up). This is well documented in the literature but more secure solutions exist (e.g., timeout).

I wasn't lucky enough to have a full-featured ICE for the processor, so I wanted to avoid hundreds of burn and test cycles. I started by developing the critical code (e.g., the pixel generation algorithm) on a PC in C—just to validate the algorithm itself. Then, I developed the full firmware with MPLAB, keeping a structured approach to facilitate the validation.

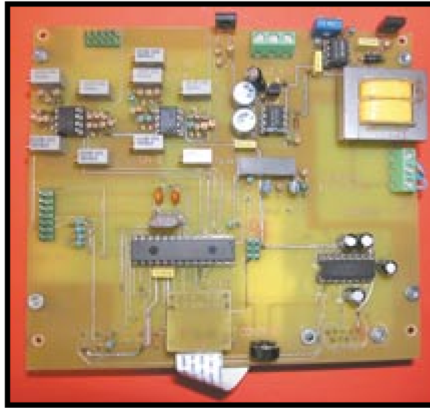

Photo 2—*The analog front end is on the upper left with its nine trimmers, the power supplies are on the top right, and the PIC is in the middle. The PCB has plenty of empty space because the LCD's dimensions dictated its size.*

Later, I implemented a bottom-up approach. I simulated 100% of the software with small stub routines in an effort to execute each routine individually. Note that I was still using MPLAB and didn't have a target system at that point. I even kept a source listing and ticked all of the assembly lines to be sure to go across each of them.

I used the MPLAB stopwatch to verify the timings. When everything seemed fine under the simulation framework, I went to the target processor. That approach proved successful. My first burned firmware was not free from bugs, but I got a working display with the first burned file!

The RS-232 helped a great deal during the final debugging steps. In fact, rather than having to develop a specific protocol for each project, I used an easy and powerful method.

First, the UART firmware dumps the RAM's content on the RS-232 port per the host's request. Following this, software on the PC side is able to grab interesting information based on the RAM content (e.g., rebuilding something like a screen hard copy). But the most interesting point is that the same feature is invaluable during the debugging steps!

PROBLEMS SOLVED

Strangely, my firmware generated a serious problem: some of the LCD's columns were darker than others, and it was dependent on the operating mode and input signals. It took me several nights of thinking before I realized that this was because of the slightly different CPU time spent between columns. Because the LCD is dumb, its buffers had stayed open longer on the columns, and they gave a darker display.

As always, when you can clearly define a problem, the solution tends to be straightforward. For this particular problem, I simply configured one of the on-board timers and waited until precisely 65 µs had been spent on each column. Problem solved.

IMPROVEMENTS TO COME

The fully operational XY-Plotter prototype demonstrates that the concept actually works. The screen is refreshed 70 times per second and doesn't flicker. The A/D management, graph generation, and textual display

Listing 1—*I used this coding technique to meet the strict timing requirements of the project. The routine sends a configurable number of pulses to the LCD clock input with less than three PIC instructions per pulse on average! Try to do it with a classic loop.*

```
;Send W pulses to the XSCK line (W = 0 to 60). Execution duration: 100
;ns × (2xW + 20) for W < 60. Average with W = 20 (worst case) giving
;three instructions/pulse (300 ns).
************************************************************************
send_upto60_pulses     ;Limited to 60 because of page boundary
input in tmp_send_w_pulses
       movf    tmp_send_w_pulses,W
       sublw   .60      ;Calculate 2 × (60 - w)
       rlncf   WREG
       rlncf   WREG
       movwf   tmp_send_w_pulses
       goto    pulsesaligned
pulsesnotaligned
       org     (1 + high pulsesnotaligned)*.256
                        ;Must start on a page boundary
pulsesaligned
       movlw   high pulsesaligned
       movwf   PCLATH  ;High byte of new PC should be defined
movf   tmp_send_w_pulses,W
       addwf   PCL,F   ;Jump to next instruction if W = 0 (60 pulses)
       bsf     PORTB,RB_LCDXSCL_BIT    ;pulse 60
       bcf     PORTB,RB_LCDXSCL_BIT
       bsf     PORTB,RB_LCDXSCL_BIT    ;pulse 59
       bcf     PORTB,RB_LCDXSCL_BIT
       bsf     PORTB,RB_LCDXSCL_BIT    ;pulse 58
       bcf     PORTB,RB_LCDXSCL_BIT
                                       ;etc…
       bsf     PORTB,RB_LCDXSCL_BIT    ;pulse 02
       bcf     PORTB,RB_LCDXSCL_BIT
       bsf     PORTB,RB_LCDXSCL_BIT    ;pulse 01
       bcf     PORTB,RB_LCDXSCL_BIT
       retlw   0        ;Must be in the same page as the first one
```

are perfect in every mode.

It took me roughly 100 h to complete this project. I still have a couple of bugs to correct but nothing too critical. A few more nights of work, and the plotter will be embedded in my new spectrum analyzer.

This project clearly demonstrated the power of low-cost microcontrollers. In addition, it proved that efficient debugging requires a good simulator. I also learned that LCD backlight high-voltage generators are harmful, but that's another story.

I have a long list of future improvements, one of which is PC-based configuration software to customize the display for new applications (e.g., modification of the textual information). That will be easy thanks to the flash memory-based PIC I used.

Developing this useful project was extremely fun. I hope reading about it was fun too! ■

Robert Lacoste lives near Paris, France. He has 15 years of experience working on innovative real-time software and embedded systems. Specialized in cost-optimized mixed-signal designs, he has won over a dozen international design contests. Robert currently manages his own design and consulting company. You can reach him at rlacoste@alciom.com or www.alciom.com.

PROJECT FILES

To download the code, go to ftp.circuitcellar.com/pub/Circuit_Cellar/2003/158.

RESOURCES

8 × 12 Character set, www.sxlist.com/techref/datafile/charset/8x12.htm.

Epsom A0635-2 LCD Preliminary specification, www.supelec-rennes.fr/ren/fi/elec/ftp/lcd/a0635.pdf.

R. Fosler and R. Richey, *A FLASH Bootloader for PIC16 and PIC18 Devices*, AN851, Microchip Technology, Inc., 2002.

D. Jones, "Binary to Decimal Conversion in Limited Precision," The University of Iowa, www.cs.uiowa.edu/~jones/bcd/decimal.html, 1999.

Microchip Technology, Inc., *PIC18FXX2 Data Sheet: High Performance, Enhanced FLASH Microcontrollers with 10-Bit A/D*, DS39564B, 2002.

SOURCES

ECM-A0635-2 LCD
Epson Europe Electronics
+49 89 14005-0
www.epson-electronics.de

MCP1541 Voltage reference, MCP6022 analog amplifier, PIC18F252 microcontroller
Microchip Technology, Inc.
(480) 786-7200
www.microchip.com

LMC6462 Op-amp
National Semiconductor Corp.
(800) 272-9959
www.national.com