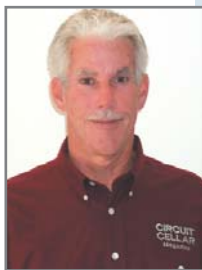


Circuit Cellar, the Magazine for Computer Applications. Reprinted by permission. For subscription information, call (860) 875-2199, or visit [www.circuitcellar.com](http://www.circuitcellar.com). Entire contents copyright ©2009 Circuit Cellar Inc. All rights reserved.



# Easy (E)mbed

## An Alternative Approach to Embedded Programming

Thanks to the march of silicon designing hardware these days is as simple as choosing which of the latest and greatest wonderchips to use. Ah, but software, well, that's another story. There's no Moore's Law for software and it isn't getting any easier. Maybe it's time for a change.

After all these years in the chip biz, one trend is clear to me. Ironically, it's the "hardware" that's gotten "easy," while writing "software" is "harder" than ever.

Actually, the irony goes deeper than that. After all, at a certain level—namely, chip design using Verilog or VHDL—creating hardware is little different from creating software. As I've said before, if you see an engineer scratching their head staring at a screen, the only way to tell whether they're designing software or hardware is to check and see if they're wearing shoes.

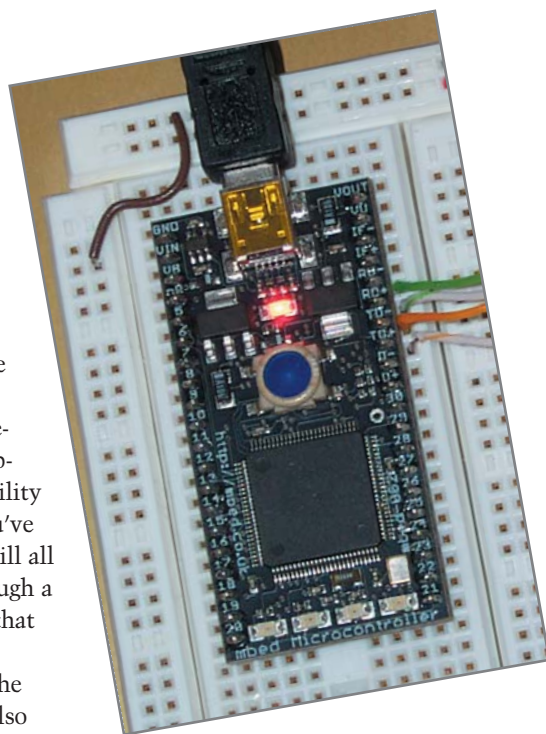
It's the users of the chips (i.e., the board- and box-level hardware designers) that have it easy. Thanks to Moore's Law, hardware design for the vast majority of embedded designs involves little more than dropping a few do-it-all chips onto a PCB. I was reminded of the fact when I recently fired up my first computer, a circa-1970s IMSAI. (Refer to my October 2008 article, "The Way We Were," which appeared in *Circuit Cellar News Notes*, Vol. 4.) With hundreds of chips and thousands of connections, that was "hardware" the hard (and expensive) way. These days, doing a similar hardware design would involve little more than choosing which 50-cent MCU to use.

Meanwhile, software stays hard. Yes, the tools get incrementally better and reuse is an easy way out. But the problem is that the aforementioned growth in hardware capability has simply fueled demand for more software features. You've heard the old saying that software, like a gas, expands to fill all available space. That old-timey, 8-bit micro chewing through a 1,000-line program has evolved into a fancy 32-bit MCU that hungers for a million lines of code. Feed me!

It's no surprise that in this month's column you'll see the usual miracle-working MCU. But this time, the story is also about the software tools as much as the silicon. Is it time for a change we can believe in?

### MACRO MICRO

The LPC236x lineup is just one horse in NXP Semiconductors' stable of ARM-based flash memory MCUs (see [Figure 1](#)). Keep my



**Photo 1**—The LPC236x-based mbed module uses a DIP form factor, a great option for easy prototyping. Note the "transformerless" Ethernet connection on the right side.

“hardware is easy, software is hard” premise in mind as you contemplate the long list of features packed onto this 100-pin hotrod.

Under the hood, we find a venerable ARM7 core—not the newest micro on the block, but nevertheless running at up to 72 MHz, which is more than adequate for blue-collar embedded apps. Different models come with 128, 256, or a whopping 512 KB of flash memory, which is more than enough to keep the programmers busy. The smaller versions (i.e., with 128-KB flash memory) come with 8 KB of general-purpose RAM, while the larger ones boost that to 32 KB.

Different members of the LPC236x lineup offer a mix of high-end I/O capabilities including 10/100 Ethernet MAC and a full-speed (12 Mbps) USB device controller. These bring their own RAM to the party (16 KB for Ethernet and 8 KB for USB), which is also available for general-purpose use by the CPU. There’s a fancy DMA controller and dedicated bussing to keep the data flowing fast with minimal overhead. Other notable I/O options in the lineup include a CAN controller and multimedia card interface.

Along with the big-ticket peripherals, the LPC236x crams in virtually every flavor of serial I/O (UART, I<sup>2</sup>C, SPI, I<sup>2</sup>S,

and more), nearly a dozen timers including high-resolution (32-bit), PWM, a watchdog, and an RTC (which adds another 2 KB of battery-backable RAM into the memory mix). The analog bases are covered with a fast (2.4-μs) six-channel, 10-bit ADC and a 10-bit DAC.

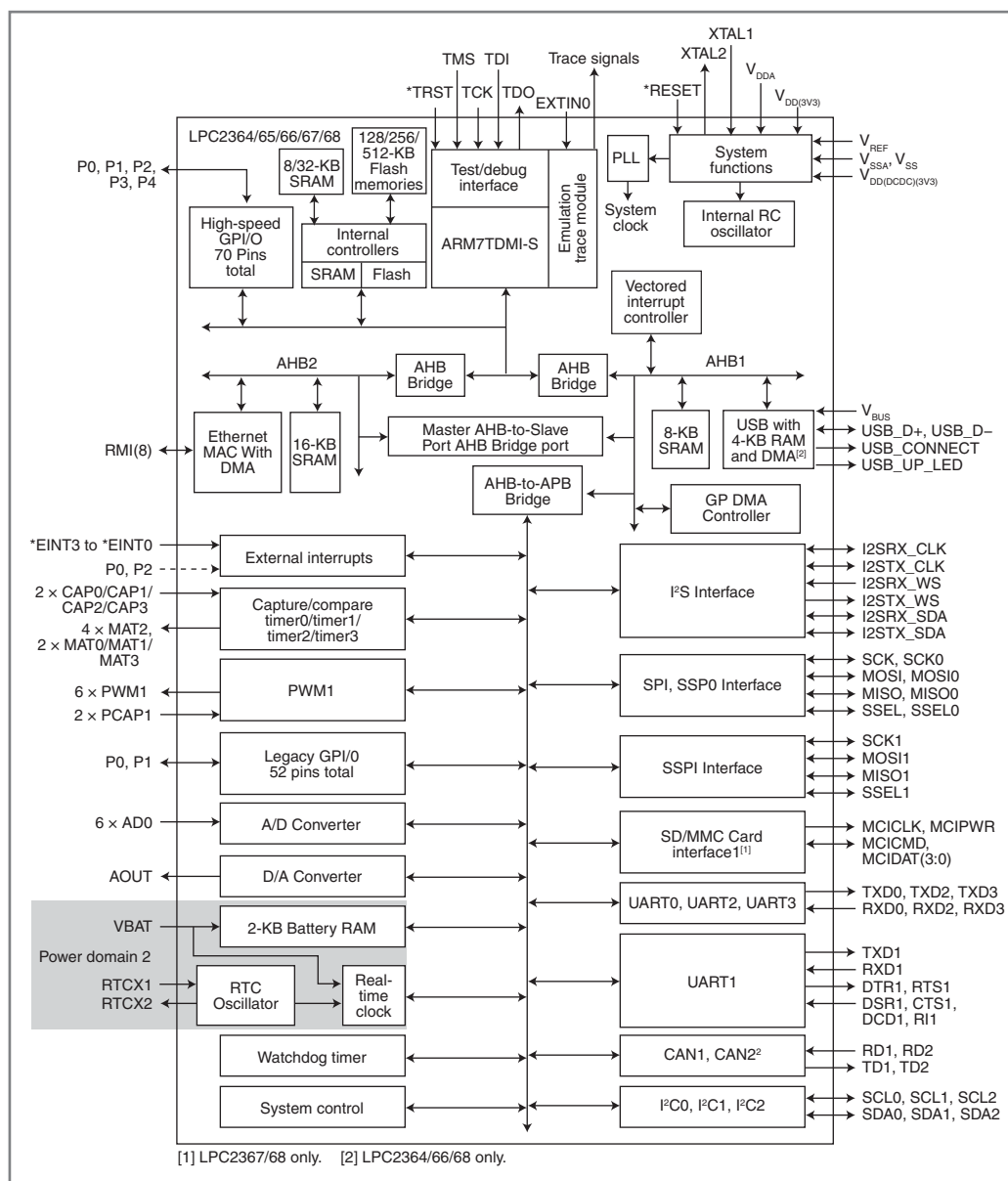
Going beyond the major bullets in the datasheet, I’m impressed with NXP’s attention to embedded details. For example, the memory interface features a sophisticated 128-bit prefetch mechanism to reduce the inevitable flash bottleneck. Bit-twiddling operations were never ARM7’s forte, so NXP includes dedicated bit set and reset registers for

I/O ports. The clock generation and power distribution scheme is fully integrated so you can tweak all the dials to minimize power. For example, clocking can be driven on-chip (built-in RC oscillator) and off-chip (main and RTC oscillators), and each peripheral’s clock can be individually controlled (i.e., disabled or divided).

## CHIPS & DIP

Although the LPC236x is impressive, the chip itself isn’t really the focus of this month’s column. The fact is, it isn’t even the newest arrow in NXP’s quiver, and it’s no secret all the major MCU suppliers offer parts with similar capabilities. The most amazing thing about the LPC236x is that it really isn’t all that amazing in this day and age. Referring back to my opening premise, rather than the chip itself, I’d like to focus on a unique approach to software development that that comes with it.

The “mbed” module shown in [Photo 1](#) highlights how hardware has become “easyware.” Just plop down the MCU and a few LEDs, and voila, instant EV board. Actually, there are a couple of extra housekeeping chips



**Figure 1**—In the old days, the hardware design for a 32-bit micro with almost half a meg of memory, Ethernet, USB, and a bunch of I/O was “hard.” Now, thanks to flash MCUs like the NXP LPC236x, it’s easy. Software? That’s another story.

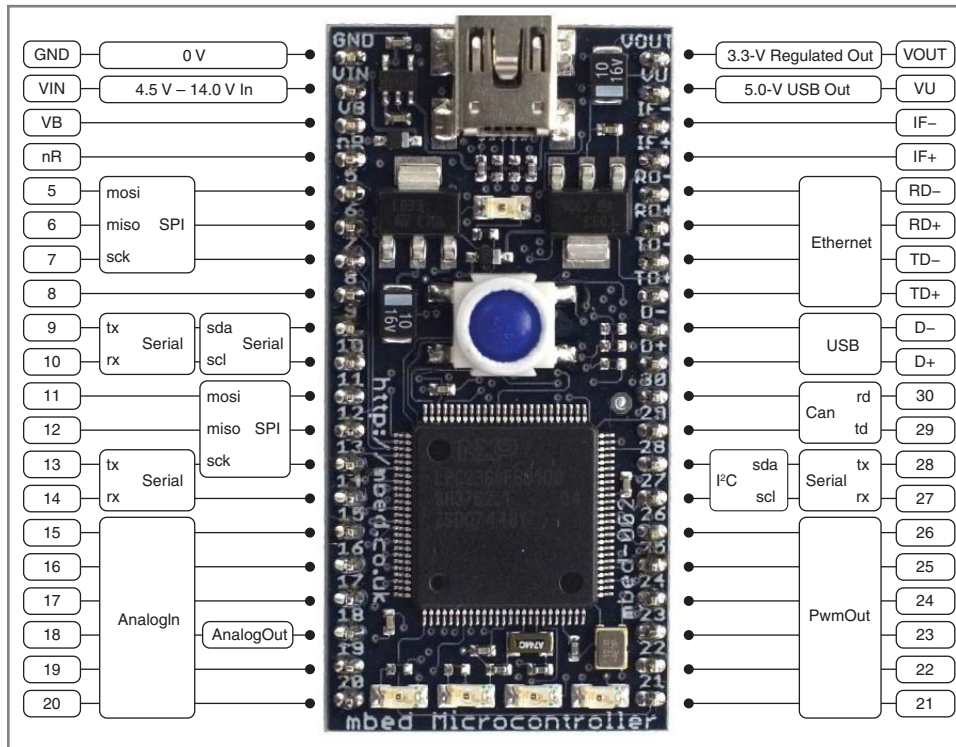


Figure 2—The mbed module DIP form factor makes hardware experimenting and prototyping, well, easy.

on the underside of the board starting with a separate LPC MCU devoted to handling the USB interface with the host PC. The extra MCU frees up the main LPC USB interface for your application and minimizes the intrusiveness of host PC communication. There's also an Ethernet transceiver (aka "PHY") from National Semiconductor (DP83848). Getting on the 'Net can be as easy as adding an RJ-45 connector, especially since the National part has a "transformerless" option that uses capacitors, instead of a transformer, for coupling.<sup>[1]</sup>

Over the years, I've seen a lot of "evaluation boards," usually one of two varieties, the first being the older-school larger boards and more recently all manner of cute USB gadgets. To my mind, the mbed module, with its DIP plug-in approach, is a smart way to go (see Figure 2). Although small and inexpensive like other USB plug-ins, it is breadboard-friendly for prototyping and experimentation, and it could even make sense for low volume production.

Usually, it's about now that I'm taking a coffee break while gigs of the typical "C" IDE get installed on

my PC. And it's here where mbed thinks outside the box, literally. To make a long story short, there's zero software to install because the tools are web-based.

Here's how it works. All you do is connect the mbed module to a USB port and it is recognized by your PC as a generic mass-storage device just like a typical thumbdrive. Open the mbed

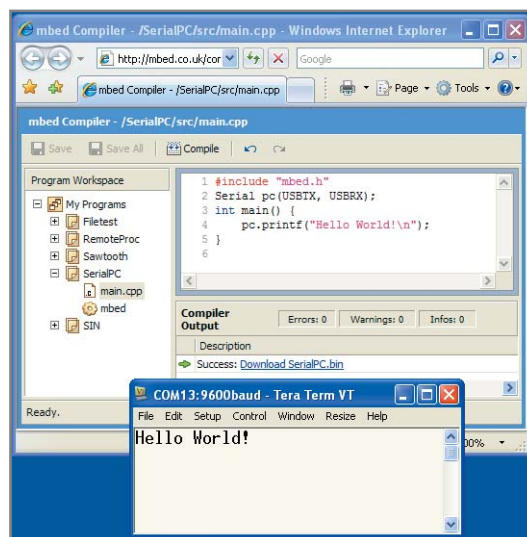


Photo 2—Absent a debugger, you'll need to fall back on PRINTF to test your program. Fortunately, there are built-in functions that implement a virtual serial port via USB so you can use a terminal emulator on the PC to see what's going on.

"drive" and look in the directory to find the "mbed.html" file. Double-click it and it will open with your browser just like any other web page. Voila, you're on the air and ready to start programming with nary a byte of software installed on your PC.

Invoking the "Compiler" link takes you to a web page that acts as an extremely simplified GUI. I guess you could call it a "WUI" (i.e., "Web-based User Interface") that pretty much boils down to basic file operations (create, open, save) and a button that says "Compile." Click that and a moment later a link appears that allows you to download the compiled code to your mbed "drive." Then push the Reset button on the mbed module and the LPC236x flashes itself with the new code and you're off to the races.

Needless to say, this is a different way of doing things that has rather profound implications worthy of further contemplation. But for now, let's hold those thoughts while we take a closer look at a web-based vision of what might be called "Embedded Programming 2.0."

## IT'S ALL "C" TO ME

Now that you've compiled and downloaded your code, it's time to get up to speed with the debugger. That doesn't take long because there isn't one. Time to resurrect the old ways and use PRINTF for debugging. To that end, the mbed library includes a feature that routes serial traffic across the USB connection to a terminal emulator running on your PC (see Photo 2).

Other handy built-in routines are shown in Table 1. For example, AnalogOut allows you to set the level on the LPC236x DAC output pin (see Photo 3). For convenience, two versions of the AnalogOut function are provided. The one used here expresses the rail-to-rail output as a fraction between 0 and 1, while another version uses unsigned 16-bit integers between 0x0000h and 0xFFFFh.

The LocalFileSystem function

enables you use standard “C” file operations (fopen, fprintf, fclose, etc.) on the mbed flash drive. This is noteworthy because it allows straightforward interchange of data between your program and a PC via the FAT file system (see Photo 4). For instance, it would be easy to use an mbed module as a data logger, creating a log file on its own flash memory that could subsequently be opened with a PC application (e.g., word processor, spreadsheet, etc.). Similarly, a PC could be used to “configure” an mbed module by writing a data file to the flash drive which the mbed application could query after being disconnected from the PC.

Going beyond the built-in functions, the mbed webpage has a “Cookbook” link that takes you to a bunch of add-on projects and examples. Many of these show how to connect the mbed module to popular I/O devices (e.g., LCDs, sensors, motors, etc.) and there are some handy utilities (e.g., a routine to set the real-time clock).

I found some interesting dishes in the “Cookbook,” starting with a simple web server that exercises the mbed Ethernet interface (see Photo 5). Taking advantage of the National transceiver’s “transformerless” feature mentioned earlier, I was able to jack-in simply by cutting one end off an Ethernet cable and connecting the transmit and receive wires directly to the mbed module.

Another demo takes advantage of the mbed “Remote Procedure Call” (RPC) capability. In essence, RPC is a mechanism that provides hooks to the functions shown in Table 1. The example in Photo 6 shows how to assign a service name (“pot”) to an mbed function (AnalogIn). Once the logical connection is established, the mbed functions are exposed to the outside world and you can use the terminal to interact with them.

## NEW IDEA

Now that you’ve got a glimpse under the hood, let’s contemplate the big picture and examine the various factors

Routines	Uses
PwmOut	Set a PWM output period, frequency, duty cycle
SPI	SPI master configuration (frequency, format) and data transfer
AnalogOut	Set analog output voltage level
Wait	Wait specified number of milliseconds/microseconds
Timer	General-purpose timer (seconds/milliseconds/microseconds)
DigitalIn	Read the state of an input pin
LocalFileSystem	Use C file functions to access mbed disk drive
Debug	Output error message to PC and debug LEDs
Ticker	Call a function at a recurring interval
DigitalOut	Set the state of an output pin
BusOut	Set the state of a collection of output pins
AnalogIn	Read analog input voltage level
Serial	UART configuration (bitrate, format) and data transfer
RTC	Set and read the real-time clock
Timeout	Call a function at a future time
BusIn	Read the state of a collection of input pins
SPI3	3-wire SPI master configuration (frequency, format) and data transfer
Stackheap	Functions return code, data, stack and heap addresses
I2C	I2C master configuration (frequency) and data transfer

Table 1—The mbed library includes built-in functions that make programming easier. The more software someone else writes, the less you have to.

comprising the altogether unique mbed experience. “Keep it simple” is advice that’s clearly taken as gospel by the mbed folks. I must say I found it refreshing that I didn’t have to deal with the clutter and complexity of the usual big-ticket IDE. There’s no doubt it lacks the features and options that a professional programmer is used to. But on the other hand, the mbed approach makes things easy for anyone who just wants to get a simple app working without a lot of fuss and bother. Recalling my opening “hardware easy, software hard” premise, it seems to me that a straightforward solution to the “software crisis” is simpler tools that enable more folks to write their own code.

Simplicity is nice, but it isn’t really what makes mbed stand out from the crowd. Rather, it’s the fact the entire experience is web-based that is totally unique and would seem to have rather significant implications.

For instance, the fact the GUI is a web-browser feels quite liberating. Notably, it sweeps away all religious issues related to your platform of choice. PC, Mac, Linux—have it your way. Heck, I suppose you could figure out a way to program mbed with your iPhone or any other web-capable gadget.

Another thing that felt very liberating was being totally relieved of “IT” responsibilities. Keeping up with revising, upgrading, and migrating (e.g., XP to Vista) a pro toolchain can be a chore, one I’m glad to let the mbed folks take care of. It’s all the more liberating, for your wallet, to realize you’re getting access to some pretty fancy software (ARM’s own Realview compiler) for free.

On the other hand, I found my browser (Explorer) had some trouble with GUI basics. Keyboard and screen updates are a bit sluggish and the interface can be quirky or lacking compared to a real editor

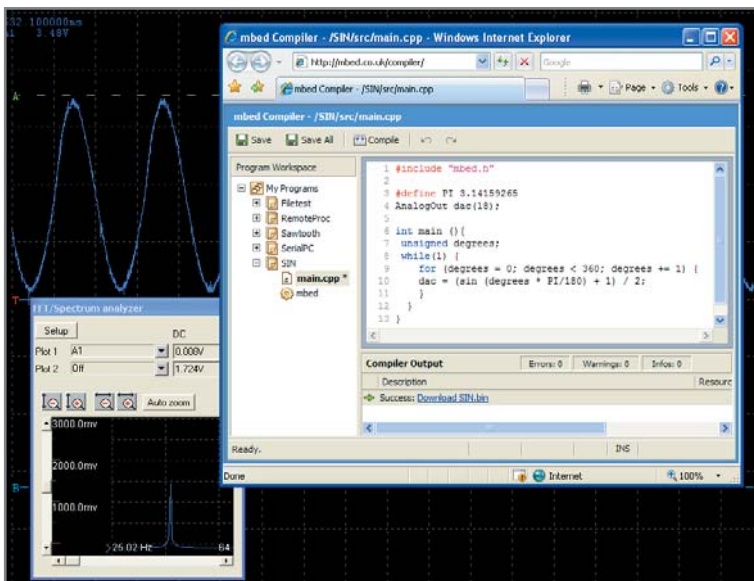
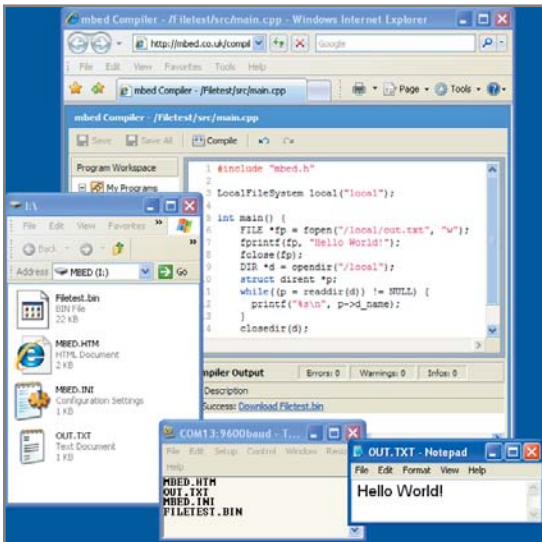
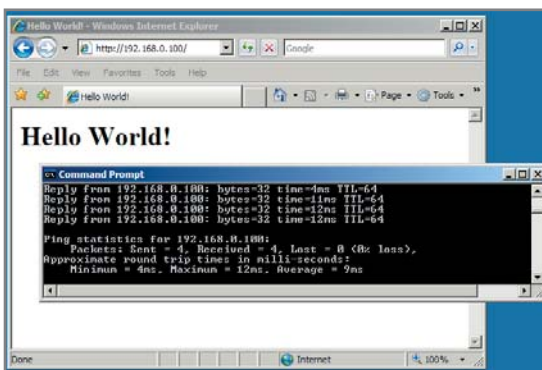


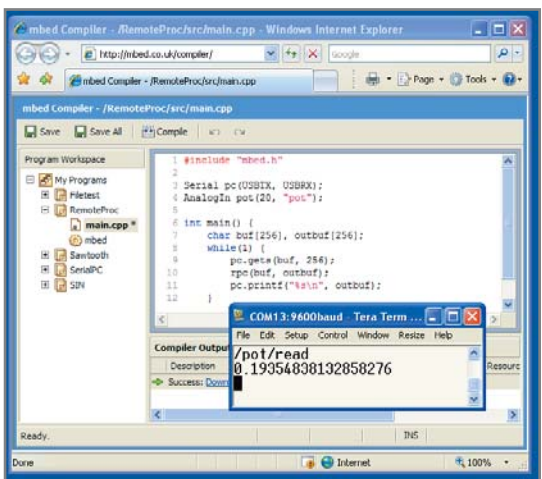
Photo 3—Another built-in function, AnalogOut, makes it easy to drive the MCU’s on-chip DAC.



**Photo 4**—From the PC’s perspective, the mbed module appears as a typical USB thumbdrive. The LocalFileSystem function takes advantage of that by making it easy for an application program to exchange file data with the PC.



**Photo 5**—One of the projects in the mbed “Cookbook” is a mini-web server that made it easy to check out the “transformerless” Ethernet feature.



**Photo 6**—Remote Procedure Call (RPC) is an interesting feature that provides hooks to access the mbed built-in functions. This example shows the AnalogIn function being invoked via the console.

when it comes to things like cut and paste, find and replace, etc. And though intuitive, having to click through Windows hoops (e.g., “Replace existing file?”) every time you compile gets old really fast.

The web angle puts an interesting spin on issues like community, collaboration, commerce, reuse, and support. In principle, a web-based solution like mbed could evolve to become a comprehensive programming “portal.”

Of course, there are issues like security and intellectual property that can’t be overlooked. It’s easy to forget everything you’re doing is getting sent across the Internet. In fact, as best as I can tell, the programs you enter and save aren’t even stored on your own PC. If Big Brother is watching me hack away, so be it. Maybe he’ll even help me get my program working.

## FORWARD TO THE PAST?

It’s kind of ironic how similar mbed’s “leave the driving to us” web-based approach is to the way things worked back in the days of mainframe

computers, albeit now on a much grander I-Way scale. After the mainframes came the “Personal Computer,” which itself felt liberating, at least at the time. But when it comes to wrestling with complicated toolchains and IT chores, I’m not sure today’s PC experience feels that liberating anymore.

Of course, the issue of web-hosted software goes beyond the niche of embedded programming to encompass all types of applications. Indeed, when you consider the likes of Google or iTunes, it’s apparent a transition to web-based functionality is already underway.

Keep in mind I’ve been working with a beta pre-release version of the mbed hardware and software. Things may change by the time you read this, so check the NXP web site (see the Sources below) for the latest information. Sure there are lots of questions and no doubt some bumps in the road ahead. But my gut feeling is that the mbed folks are onto something here. The web-based approach won’t totally replace the old way of doing old things (i.e., the traditional toolchain running on the PC), but it doesn’t need to. Instead, consider the mbed solution an alternative approach to embedded programming that may not be industrial strength, but is easy and accessible for all. ☑

*Tom Cantrell has been working on chip, board, and systems design and marketing for several years. You may reach him by e-mail at tom.cantrell@circuitcellar.com.*

## REFERENCE

[1] D. Miller, D. Seely, and T. Ngo, “DP83848 PHYTER Transformerless Ethernet Operation,” Application Note 1519, 2006, [www.national.com/an/AN/AN-1519.pdf](http://www.national.com/an/AN/AN-1519.pdf).

## RESOURCE

T. Cantrell, “The Way We Were,” *Circuit Cellar News Notes*, Vol. 4, 2008, [www.circuitcellar.com/newsletter/1008.html#1](http://www.circuitcellar.com/newsletter/1008.html#1).

## SOURCE

### DP83848 Transceiver

National Semiconductor Corp. | [www.national.com](http://www.national.com)

### NXP LPC236x “mbed” module and web-based C IDE

NXP Semiconductors | [www.nxp.com/microcontrollers](http://www.nxp.com/microcontrollers)