Simple USB Data Acquisition

Simple data acquisition is only a project away. Read on to learn how to build a simple data acquisition device around the LPC2138. The system features a simple GUI that allows you to view graphed data instead of the streaming serial data in a terminal emulator session.

Just ask any of my friends, and they'll tell you I'm definitely an embedded system nut. I love trying out the latest microcontrollers and chips that can breathe new life into my designs. A couple of my recent favorites are Philips ARM-based microcontrollers and USB-to-UART bridges. I've incorporated both of these types of devices into my last few designs. I've been extremely impressed with the results.

Another recent addition to my bag of tricks has been on the front end of my designs. Adding simple PC graphical user interfaces (GUI) that can communicate with my embedded designs has put the finishing touches on them. By adding a nice PC GUI that can communicate with the embedded system over a serial port, you can perform things like system setup, real-time diagnostics, and tests. Besides these benefits, your end user or customer will have a more professional, user-friendly interface to work with.

After thinking about ways to combine all of this in a single project, I decided to work on a simple USB data acquisition project that allows you to collect temperature data from an analog temperature sensor and graph it via a PC GUI. Everyone wants to collect data of some sort (temperature in my case). And what better way than over USB via an ARM-based microcontroller? Of course, taking the data and doing something with it is also an important part of the process, so I'll show you a PC GUI. By the end of the article, you'll be able to create a simple USB (embedded ARM-core-based) data acquisition device. Most importantly, though, you'll know how to develop with an ARM-based microcontroller, how to use a USB-to-UART bridge, and how to make a PC GUI to tie it all together.

SYSTEM OVERVIEW

I usually design my own boards, but for this project I used a couple of evaluation boards to implement my minimal USB data acquisition system. The boards are readily available, so a hardware design isn't required to get up and running.

The system is comprised of two boards, an analog temperature sensor, and a PC running the GUI (see Figure 1). The first board is the Keil MCB2130 evaluation board, which contains the new ARM-based LPC2138 microcontroller (see Photo 1). The MCU reads the temperature sensor's analog output voltage via its A/D converter and sends the reading via its UART. For this particular application, I used the board's serial port circuitry (RS-232 transceiver and connector), expansion connector (for hooking in the temperature sensor), and power input connection. One of the board's neater features is that it's powered off from an on-board USB connector. This means you don't need a clunky wall wart to power the system. You can just run another USB line to it for power. This is a definite advantage to using a USB, as long as your board doesn't draw more power than the USB connection can handle.

The Silicon Labs CP2101 evaluation board contains the CP2101

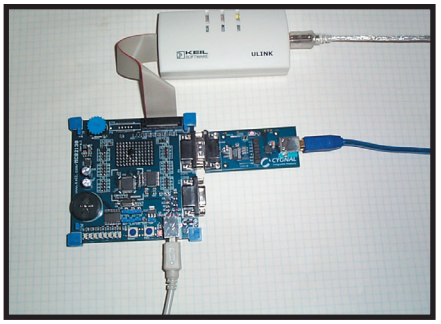

Photo 1—*I used a Keil ULINK JTAG debugger to in-circuit debug and program the LPC2138 microcontroller. I soldered the LM60 temperature sensor to the prototyping area of the MCB2130 board.*

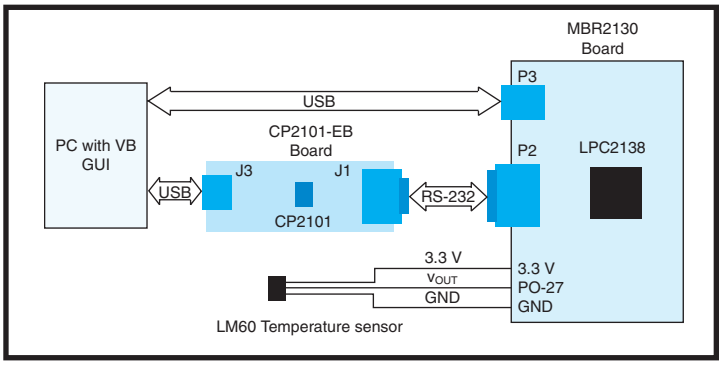

Figure 1—*Take a look at the USB connections in the system. Where does the power come from? The USB is used for more than just communicating with the PC; it's also used to power both boards, which enables you to remove those ugly black wall warts. The schematics are posted on the* Circuit Cellar *ftp site.*

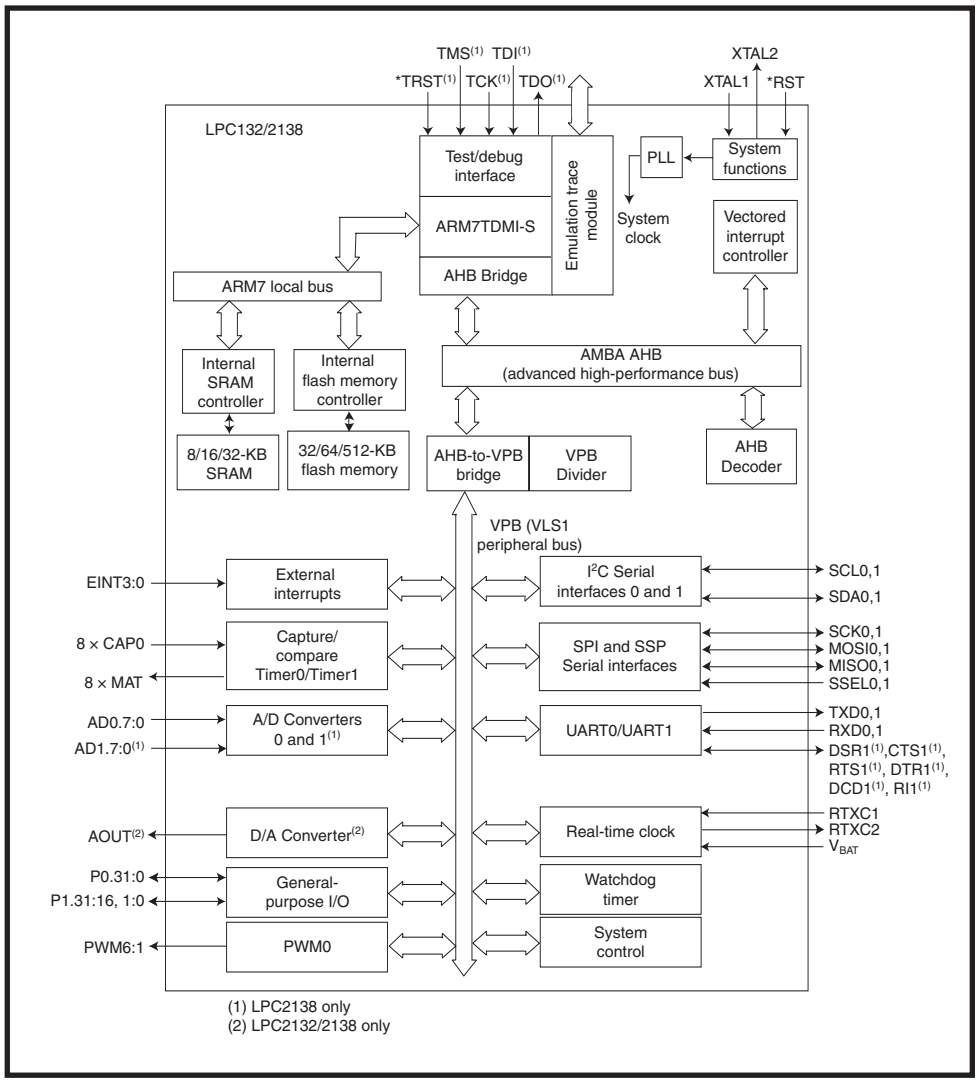

Figure 2—*A 32-bit ARM7 core lies at the heart of the LPC2138. Given the chip's high-performance core and vast number of peripherals, it can cover a vast number of applications.*

USB-to-UART bridge chip and an RS-232 transceiver. This allows you to plug in an RS-232 communicating device on one side and a USB communicating device on the other. The board and its virtual COM port software drivers form the link or bridge between the MCB2130 board's RS-232 port and the PC's USB port.

The National Semiconductor LM60 is a simple three-pin analog Celsius temperature sensor. It's wired into the expansion connector on the MCB2130 board, which connects to the LPC2138's A/D converter to read the analog voltage from the sensor. Its output is linearly proportional to Celsius (6.25 mV/°C), and it has a DC offset of 424 mV to accommodate negative temperatures. This makes it a fairly easy sensor to deal with in software after it's read via the LPC2138's A/D converter.

The PC contains the Visual Basic GUI that I used for this application. It simply reads the raw temperature data sampled by the A/D converter over the virtual USB COM port, converts it to temperature, and displays and charts the results over time. The GUI puts the finishing touches on the design, making it a more user-friendly and professional-looking system. Just imagine how much easier it will be to look at graphed data instead of the streaming serial data in a terminal emulator session.

LPC2138 MCU

The LPC2138 is one of Philips's newest ARM based microcontrollers. Having previously designed with the LPC2106, the LPC2138 piqued my interest given its vast assortment of added peripherals. The addition of A/D converters, D/A converters, an external memory controller, and edge-sensitive interrupts made it the perfect migration part for my LPC2106 designs (see Figure 2).

The small LPC2138 contains everything but the kitchen sink. In addition to a ton of peripherals and general-purpose I/O, it's loaded with 512 KB of flash memory (128 bits wide for high speed) and 32 KB of RAM—definitely not the typical memory sizes I'm used to seeing in plain-vanilla 8-bit microcontrollers. Another remarkable feature is the chip's size. The 64-pin QFP part measures in at 10 mm × 10 mm, making it perfect for tightly spaced applications.

And then, of course, there's the one thing that makes this microcontroller shine: an ARM 32-bit ARM7TDMI-S core. This 32-bit ARM core yields 54 MIPS when running at 60 MHz, which is easily achieved by utilizing the LPC2138's on-board PLL. So, not only do you get a vast number of peripherals and tons of memory, you get all the benefits of using an ARM core! What are the benefits, you ask? An obvious one is its high performance and low-power consumption combination. Others are its vast software tool support, real-time debugging, and code density options (Thumb) for high-volume applications with memory restrictions.

The ARM core definitely has found its way into numerous applications via microprocessors, ASICs, SoCs, and FPGAs. And now, with its growing use in cost-effective microcontrollers, I may think twice before choosing a performance-limited 8-bit microcontroller for my next application. Either way, if you're into embedded design, there's no doubt that having ARM experience under your belt would be beneficial to your career. ARM is an interesting and detailed topic in and of itself, so refer to the Resources section

of this article for more information.

I hope I've piqued your interest in the LPC2138. Now let's switch gears and examine how the LPC2138 fits into the USB ARM data acquisition (DAQ) design.

DAQ VIA ARM

In this simple USB ARM DAQ application, the LPC2138 must read analog temperature sensor voltage at a timed interval. The data must be formatted and then sent to the MCB2130 board's serial port. Therefore, you must use the LPC2138's timer peripheral for the interval timer, its A/D peripheral for reading the analog temperature sensor voltage, and its UART peripheral for serial communication.

The LPC2138's timer is a 32-bit timer/counter with a programmable 32-bit prescaler. It's an extremely flexible timer given its capture channels, match registers, external outputs, and interrupt capabilities. I preloaded a timer match register for this application. The LPC2138 is used generate a timer interrupt when the timer counter matches this value. Its A/D converter is a 10-bit successive approximation A/D converter. A 10-bit reading of the analog temperature sensor provides more than enough resolution for the USB ARM DAQ example.

The LPC2138's UART is your typical UART with data rate generation, but it also includes 16-byte receive and transmit FIFOs for added flexibility. Given that temperature data is sent out every few minutes, the data rate is set to 9,600 bps. Now that you're familiar with the LPC2138's peripherals, let's move on to the embedded software.

EMBEDDED SOFTWARE

Before writing actual LPC2138 application code, the device needs to be set up after it's powered on. Fortunately, most IDEs will either set this up for you or provide some kind of boot assembly code to handle the task. For this particular project, I used the evaluation version (16-KB code size limitation) of the Keil µVision3 environment. I was pleasantly surprised with its boot-up implementation. The graphical configuration wizard allows you to modify the proper setup registers for your application. This made the boot and start-up process transparent and allowed me to focus on the application itself.

Let's look at the LPC2138's PLL setup as an example. In order to change the PLL multiplier value on the LPC2138, you must perform a few extra steps after writing the new multiplier and control values to the PLLCFG and PLLCON registers. You must first write 0xAA and then 0x55 to the PLL feed register (PLLFEED). This action loads the PLL control and configuration information from the PLLCON and PLLCFG registers into the shadow registers that actually affect PLL operation. It's basically a good way to prevent accidental changing of the PLL value. This code implementation is taken care of with the provided boot code in µVision3. Punching in the desired multiplier in the GUI automatically updates the boot code. I learned this the hard way in a different IDE when designing with the LPC2106. The point is that using the graphical configuration tool is an easy and fast way to set up the microcontroller and start working on your application.

Now that the boot up code is taken care of, let's concentrate on the main application. I chose C language over the native ARM assembly language to write the driver and application code. So, the next step involved writing a C code driver for a timer interrupt, an A/D scan, and the UART. Fortunately, the example C code that came with the µVision3 IDE had examples for all the peripherals. I modified and used them.

The code for each peripheral was extremely straightforward and easy to understand and integrate. Creating the application code, including the C code for each peripheral, resulted in the code shown in Listing 1. In this code a timer match interrupt occurs from the interval timer, and then the AIN-1 A/D channel is read to sample the analog output voltage from the

Listing 1—*The C function from the LPC2138 takes care of reading the LM60 temperature sensor via its on-board A/D converter. The data is then sent out the serial port to the PC running the Visual Basic application.*

```
void tc0 (void) __irq        //Timer counter 0 interrupt executes
                             //every second at 60-MHz CPU clock
                             //for testing purposes
{
static char LedFlag = 0;
unsigned int AtoDValue;

if (LedFlag == 0)  //Simply toggle the on-board LED for debug
     {
     IOSET1 =   0x00010000;
     LedFlag = 1;
     }
else
     {
     IOCLR1 =   0x00010000;
     LedFlag = 0;
     }
AD0CR |= 0x01200000;          //Start an A/D conversion
do
     {
    AtoDValue = AD0DR;        //Read A/D data register
     }
while ((AtoDValue & 0x80000000) == 0); //Wait for end of A/D
                                       //conversion

AD0CR &= ~0x01000000;         //Stop A/D conversion
AtoDValue = (AtoDValue >> 6) & 0x03FF; //Extract AIN1 10-bit A/D value
printf ("A%d\n", AtoDValue);  //Output A/D conversion result to
                              //serial port

T0IR        = 1;              //Clear interrupt flag
VICVectAddr = 0;              //Acknowledge ARM interrupt
```

LM60 temperature sensor. The analog temperature data is then masked because only 10 bits are valid because of the 10-bit A/D resolution. Now the reading is ready to be sent out the serial port via the UART through the `printf` statement. Listing 1 is all the code you need to read the LM60 temperature sensor every few minutes and send the raw ASCII-converted A/D result out of the serial port.

You must download code to the board and begin debugging at this point. Use the Keil ULINK JTAG debugger, which integrates nicely with the Keil µVision3 IDE. The debugger connects to the MCB2130 debug connector and communicates directly with the ARM7 core inside the LPC2138 via its EmbeddedICE logic (see Photo 1).

The typical debug options are available in µVision3. Single stepping, watch windows, break points, and memory snooping are all possible with the LPC2138. An interesting item in the µVision3 IDE is the ability to interact with LPC2138 peripherals while the program is idle. A separate GUI can be opened for the various LPC2138 peripherals that allows for interaction and control of them. Things like manually scanning the LPC2138's A/D converter and flipping of one of its GPIO bits are possible. A couple of the windows are shown in Photo 2. This is a good way to get to know some of the peripherals and their associated registers on the LPC2138. Given all these features, I quickly downloaded code to the board, ran it, and debugged it.

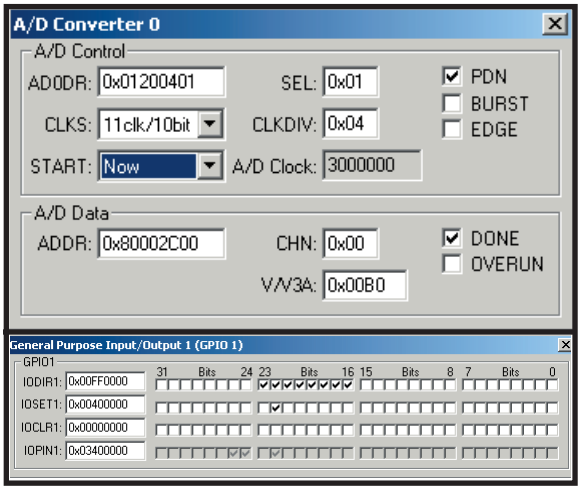

Photo 2—*The debug peripheral windows in the Keil debugger were useful when I was experimenting with the A/D converter. They enabled me to alter the state of the GPIO bits. It's an easy way to scan the A/D for the analog temperature value prior to connecting up to the GUI. This eliminated the need for writing special test code.*

CROSSING THE BRIDGE

After the code is running on the LPC2138, the raw temperature data exits the MCB2130 board's serial port and meets the USB-to-UART bridge board, which contains the CP2101 USB-to-UART bridge controller and an RS-232 transceiver. The board's power also comes from the USB port, once again eliminating the need for an ugly wall wart power supply. I used the board to convert RS-232 serial data from the MCB2130 board to compatible USB data for the PC.

The CP2101 is highly integrated and requires no components other than a USB connector. It includes a USB 2.0 full-speed function controller, USB transceiver, oscillator, EEPROM, and asynchronous serial data bus (UART) with full modem control signals. The device's packaging is an unbelievably compact: 5 mm × 5 mm MLP-28. I've soldered many surface-mount components under the microscope on prototype boards, but this device was by far the trickiest, especially because it doesn't have external leads! (If you plan on trying to solder this part on by hand, make your PCB footprint pads a bit longer to allow for better solder flow with a fine-tipped iron.)

Looking toward the software end of things, the nice part about using this device is that special software isn't needed for the RS-232-to-USB conversion. This allows USB communication to become totally transparent for the LPC2138 and its UART. Just connect the LPC2138 UART pins to the CP2101, and it will take care of the rest.

You've probably guessed that there must be some software intervention for CP2101 data to get to the PC over USB. Yes, there is. It's via a virtual COM port driver installed on the PC side. These drivers make your USB port seem like another COM port on your PC's operating system (thus the virtual COM port name). Silicon Labs provides the virtual COM port drivers with its development board for Windows, MAC, and Linux.

The really interesting thing about these drivers is that existing PC applications, like terminal emulators, will work with them. You can have a terminal session over USB or use existing applications that use COM ports to talk over USB. This is what the Visual Basic PC application does.

DAQ GUI

The user interface is the final piece of the USB ARM DAQ system. It's the finishing touch that gives the system a professional-looking way to view the temperature data. The data transmits through the USB port from the CP2101 bridge board. Note that the PC application thinks the data is coming over a standard COM port. In reality, however, it gets data from the USB port via the Silicon Labs virtual COM port driver. This gives you the benefit of using the USB port for communication without all the complexity because it looks like just another serial COM port.

The GUI was developed in Visual Basic. If you're familiar with BASIC programming languages like Qbasic, you'll probably find the migration to Visual Basic to be fairly straightforward. Writing code for various actions or events, like receiving serial characters or a button click, is extremely simple. For this application, the goal was to take serial data from the virtual COM port, convert it from a raw temperature A/D value to a real temperature (Celsius), and then graph and display it. Visual Basic provided the control components for the graphing and the serial data communications. I wrote the temperature conversion code.

The serial data is received over the COM port. It's provided by a control component called MSComm Control in Visual Basic. By simply adding this component to the project, you can set

Listing 2—*The Visual Basic code takes the serial string from the LPC2138, converts it, and then graphs it to the chart.*

```
Case A_COMMAND                  //Temperature sensor A reading
 CommandInProcess = False       //End of current command
 If (CurrentStringSize >= 1) Then //Get any characters?
  ArrayCount = 1                //Start of data

  For i = ArrayCount To CurrentStringSize
   TemperatureString = TemperatureString & TemperatureSerialData(i)
                                //Make the string
  Next i
  Led1.LED_Colour = vbRed       //Blink the virtual LED
  Timer1.Enabled = True

  TemperatureDecNum = Val(TemperatureString) //Convert string to decimal
  DebugText.Text = "Raw A to D value = " & TemperatureDecNum //Display
                                //raw A/D value as read from LM60
                                //sensor via the LPC2138's ADC

  TemperatureDecNum = TemperatureDecNum * (3.3 / 1024) //Convert
                                //A/D reading, 3.3-V ref/10 bit A/D
  TemperatureDecNum = TemperatureDecNum - 0.424 //Remove 424-mV offset
  TemperatureDecNum = TemperatureDecNum / 0.00625 //6.25 mV per
                                //degree C
  Temperature = TemperatureDecNum //Convert the converted value to
                                //an integer

  TemperatureText.Text = Temperature & "°c" //Output the string

  With MSChart1                 //Now lets graph it
    .Data = TemperatureDecNum //Plot data to current location
                                //(autoincrement is on)
  End With

 End If
End Select                      //Command finished
```

up and open and close serial ports (data rate, etc.). It also allows you to respond to a variety of events, like receiving characters.

You must open and close the COM port. The speed is preset to 9,600 bps to match the speed from the MCB2130 board. The receive Comm event will provide a receive character event and allow you to view and react to incoming serial characters from the virtual COM port. After the raw temperature A/D data has been received, it can be converted to real temperature data for graphing. This conversion involves multiplying the raw temperature data, which is the LPC2138's analog-to-digital-sampled voltage, by the A/D reference (3.3 V) divided by 10 bits (2^{10} = 1,024). Following this, the LM60 sensor's 424-mV offset must be subtracted. The value left must be divided by 6.25 (6.25 mV/°C) to get the actual temperature in degrees Celsius, and then converted to an integer.

After the temperature conversion is finished, MSChart Control provides the graphing. By adding this component to the project, a variety of charts and display options are provided for graphing data. Little set-up code is needed because the chart is set for you beforehand via the chart properties windows. This is definitely an advantage. You can modify the component controls via a categorized list of options without having to write code to do it.

Combing all the control component interface and application code results in the code in Listing 2, which is all you need to accept a serial string from the virtual COM port via USB, convert it to degrees Celsius, and graph and display it. Graphing the data takes only one real line of code!

As you can see, this is powerful design tool for GUIs. The end GUI result is shown in Photo 3. The temperature data is graphed nicely (as shown by the red line), and the current temperature updates every time a serial string is received over the virtual COM port. This puts the finishing touch on the USB ARM DAQ system and makes for a professional-looking demo.

SWITCHING TO ARM?

I hope you now have a better understanding of ARM-based microcontrollers, USB-to-UART bridges, and the process of implementing simple GUIs. You can combine all three to make a simple data acquisition device.

The new ARM microcontrollers like the LPC2138 are opening new doors for designers, many of whom are now questioning the use of the venerable 8-bit microcontroller for some applications. When you account for the ARM7 core's processing power, its low-power consumption, the vast number of peripherals, the memory size, the tool/debug support, and the incredibly small physical footprint of the devices, switching to a 32-bit ARM microcontroller may be a reasonable choice.

The USB-to-UART bridges like the CP2101 make it simple to update UART peripherals on microcontrollers (or legacy RS-232 devices) and enable USB connectivity. Embedded code isn't required to make this transition, so the update process is fairly seamless. The virtual COM port drivers provided by companies like Silicon

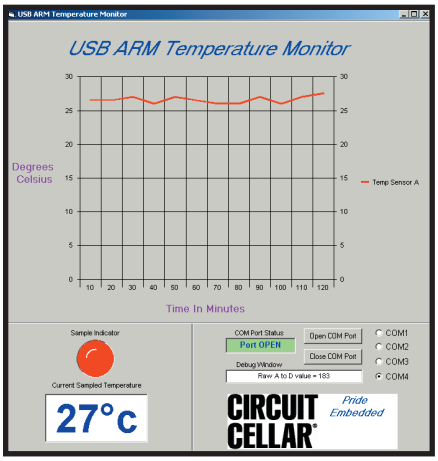

Photo 3—*The final Visual Basic application provided a professional and visually appealing front end for the system. Watching live graphics updates proved a lot more interesting than watching raw datastreams in a terminal emulator.*

Labs also allow PC applications, such as the Visual Basic GUI I created, to send and receive USB data without the additional code overhead.

The Visual Basic-based GUI allows for a professional-looking GUI in an easy-to-use design environment. The built-in component controls put much of the complex pieces in a simple format that you can easily integrate into your application. The language should be familiar to anyone with BASIC language experience. You should explore these topics in greater detail before you begin your next project. Good

Bruce M. Pride holds an A.S. and B.S. in electronics and currently works as a senior electrical engineer. He enjoys all aspects of embedded system hardware and software design, particularly designs with 32-bit microprocessors, FPGAs, and various microcontrollers. In his spare time, Bruce enjoys spending time with his family, playing acoustic guitar, competing in Circuit Cellar *design contests, and consulting via his side business, Pride Embedded. You may contact him at bruce pride@prideembedded.com.*

PROJECT FILES

To download the code and schematics, go to ftp.circuitcellar.com/pub/Circuit_Cellar/2005/177.

RESOURCES

S. Furber, *ARM System-on-Chip Architecture*, Addison-Wesley, Boston, MA, 2000.

R. Martin, "ARMs to ARMs," *Circuit Cellar*, 148–150, November 2002–January 2003.

Philips Semiconductors, "LPC2131/2132/2138," Preliminary Datasheet, rev. 0.1, November 2004.

Silicon Laboratories, "CP2101: Single-Chip USB to UART Bridge," rev. 1.6. 2005.

SOURCES

MCB2130 Evaluation board and ULINK JTAG
Keil Software, Inc.
www.keil.com

LM60 Temperature sensor
National Semiconductor
www.national.com

LPC2138 Microcontroller
Philips Semiconductors
www.semiconductors.philips.com

CP2101 USB-to-RS-232 bridge
Silicon Labs
www.silabs.com