THE DESIGN CYCLE

BY FRED EADY

ADVANCED TECHNIQUES FOR DESIGN ENGINEERS

GIVE YOUR DATA RADIO THE AX8052F100

Behind every good data radio, you will find a microcontroller filled with tricky radio driver firmware. In most cases, the microcontroller that supports the data radio moonlights as the LCD controller, the RS-232 driver, and the analog interface. These days, you can pick and choose from a variety of general-purpose and niche microcontrollers. There are microcontrollers that specialize in motor control and those that excel in number crunching. In this installment of Design Cycle, we are going to closely examine a microcontroller that was designed to drive RF ICs.

THE AX8052F100

As its name implies, the AX8052F100 is based on the tried and true 8052 microcontroller core. It comes wrapped in a tiny 28-pin QFN package. The typical data radio does not have the luxury of consuming power on a regular basis. So, to be able to eat when the radio eats, the microcontroller in charge of the radio station must have as low or lower power consumption figures than the radio itself.

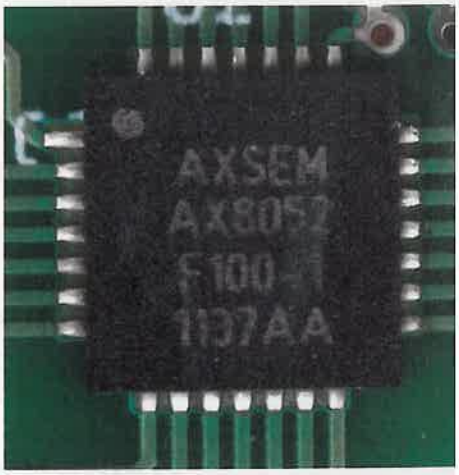
The AX8052F100 low power modes include standby, sleep, and deep sleep. The AX8052F100 sleeps at 850 nA, 1.5 µA, or 2.2 µA, depending on how much of its RAM you wish to protect while its eyes are closed. The RAM retention sleep numbers coincide with 256 bytes, 4 KB, and 8 KB of system RAM. When the AX8052F100 is actively computing, the current draw is based on the microcontroller clock speed. Typically, it burns 150 µA per MHz.

The AX8052F100's system clock can be generated externally or internally. You can hang a 32.768 kHz tuning fork crystal on the low power crystal oscillator pins or a 20 MHz crystal on the AX8052F100's XTAL pins. A 20 MHz clock can also originate from the AX8052F100's internal 20 MHz RC oscillator.

If your data radio application needs to be cheap and stingy, you can call upon the AX8052F100's 10 kHz/640 Hz super low power internal RC oscillator. To a data radio, timing is everything. So, the AX8052F100 allows you to calibrate the on-chip RC oscillators using a reference clock signal.

Don't let the AX8052F100's size fool you. The tiny QFN package packs 64 KB of wear-resistant program Flash which can withstand 100,000 erase cycles. There is 8.2 KB of RAM available for manipulating those incoming and outgoing data packets. The memory map of the AX8052F100 follows that of the legacy 8052 cores. So, everything you have learned about the 8052 over the years can be applied directly to the AX8052F100.

The AX8052F100 only has 28 pins to work with. To accommodate the UART, SPI, analog-to-digital



■ PHOTO 1. Don't fear the AX8052F100's QFN package! We've experienced laying them down on copper-clad fiberglass before (Design Cycle March 2009 issue).

converter (ADC), and analog comparator peripherals, its pins can be configured to act as peripheral I/O or general-purpose I/O pins. If you take a look at the AX8052F100 datasheet, you won't find a ground pin. To further conserve I/O pins, the AX8052F100's ground pin is actually the exposed pad on the underside of the QFN package.

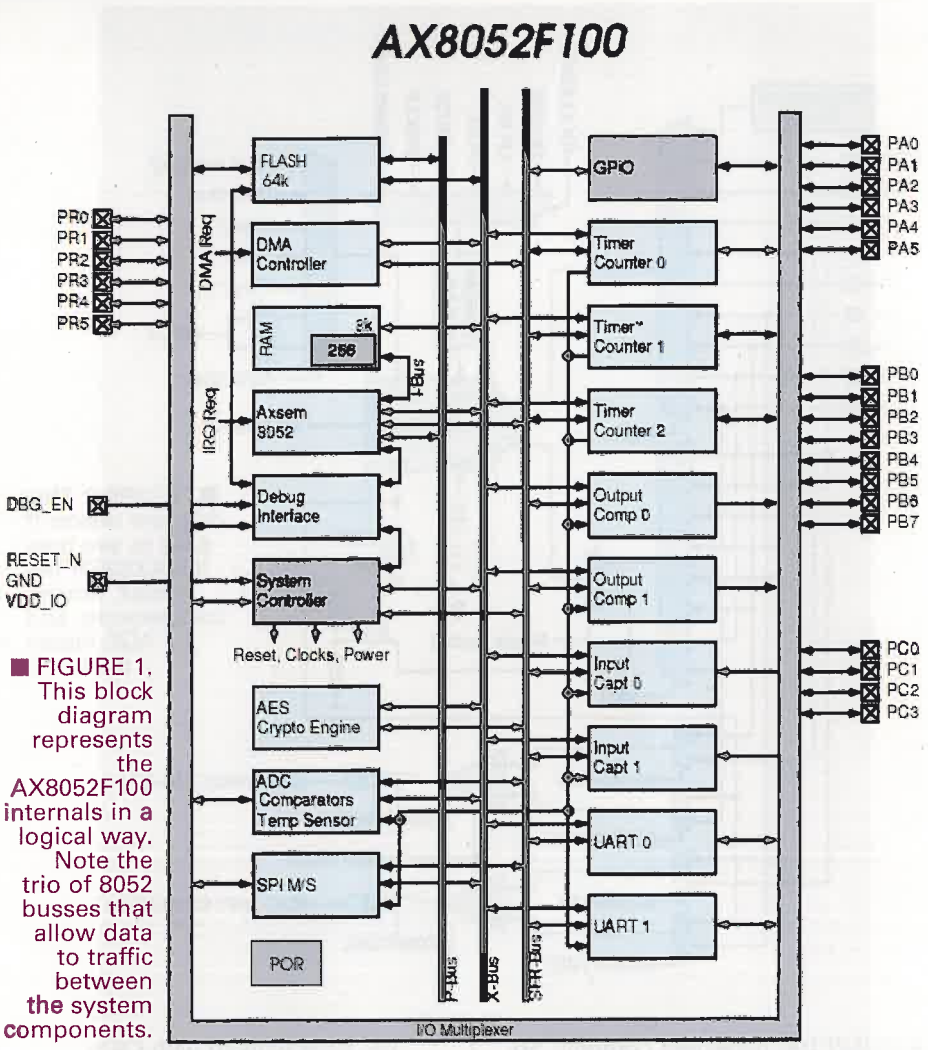
What would any microcontroller do without its internal timers? The AX8052F100 contains three 16-bit general-purpose timers. The timers can be used in conjunction with the pair of 16-bit output compare units to generate PWM signals. The timers can also be used to assist the pair of 16-bit input capture units in timing events based on internal and external signals.

Conserving power is great. However, if the AX8052F100 goes to sleep and can't wake up, you might as well not have it working for you at all. To assure wake-ups, the AX8052F100 is equipped with a pair of 16-bit wake-up timers.

In some microcontroller data radio applications, time is money. So, the AX8052F100 is equipped with an on-chip DMA engine to transport data between its internal logic blocks with minimal assistance from the CPU. If that data you're moving is super-secret and has to leave the confines of the AX8052F100, its dedicated AES crypto engine can be called upon to secure the data.

Specialization comes into play on the AX8052F100's radio interface I/O pins which are located between pins 1 and 7. The AX8052F100's radio interface is actually a dedicated SPI portal which is designed to drive the ASXEM RF ICs. The AX8052F100's radio SPI portal can also be used to drive any other data radio that interfaces to a host microcontroller using SPI. If you don't wear a pointy hat adorned with quarter moons and stars, you can turn off the AX8052F100's radio interface and employ the pins as general-purpose I/O.

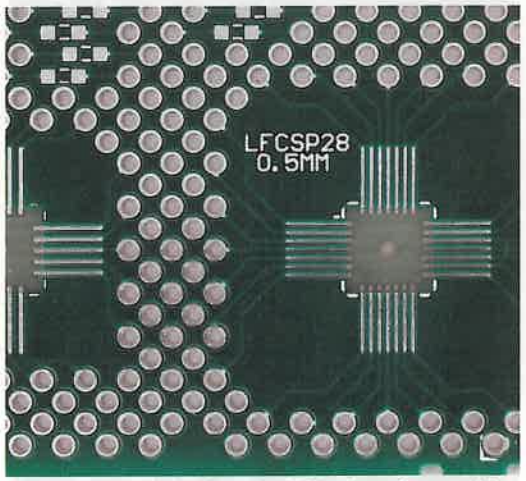
Need temperature data? The AX8052F100 can supply it via its on-chip temperature sensor. There is also a one volt internal ADC reference that allows you to measure the voltage at the VDDIO pin. Rather than go into a lengthy description of how the AX8052F100's peripherals, ADC, and logic blocks intertwine, I give you **Figure 1** and **Figure 2**. **Figure 1** is a satellite view of the AX8052F100's CPU, memory, and peripheral resources, while **Figure 2** shows how its analog peripherals are interconnected. A loose AX8052F100 is showing its teeth in **Photo 1**.



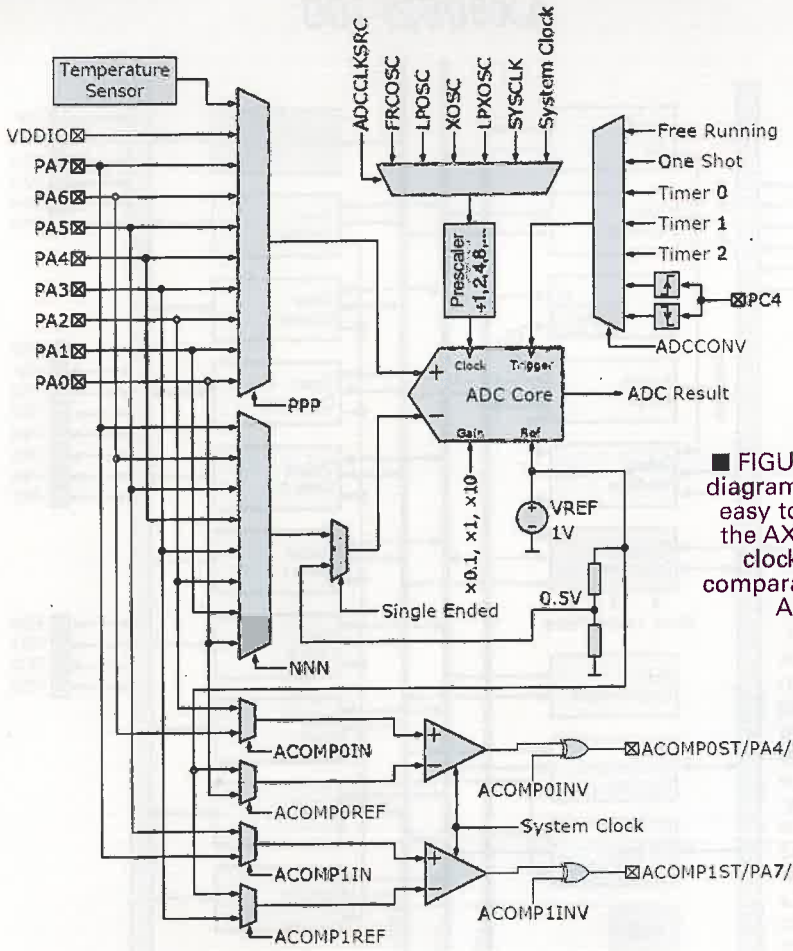
■ FIGURE 1. This block diagram represents the AX8052F100 internals in a logical way. Note the trio of 8052 busses that allow data to traffic between the system components.

AX8052F100 101

We can talk about the AX8052F100 and its features all day long. However, that's not why you are reading this. You are most likely here because you are interested in applying the



■ PHOTO 2. A QFN pit is formed by eliminating a section of PCB coating that is the size of the QFN package. The resultant pit helps to perfectly align the QFN device pins to the SchmartBoard's PCB pads.



■ FIGURE 2. This diagram makes it easy to see how the AX8052F100 clocks, timers, comparators, and ADC inputs interplay.

AX8052F100 datasheet contents. So, I decided to put together a minimal AX8052F100 embedded system.

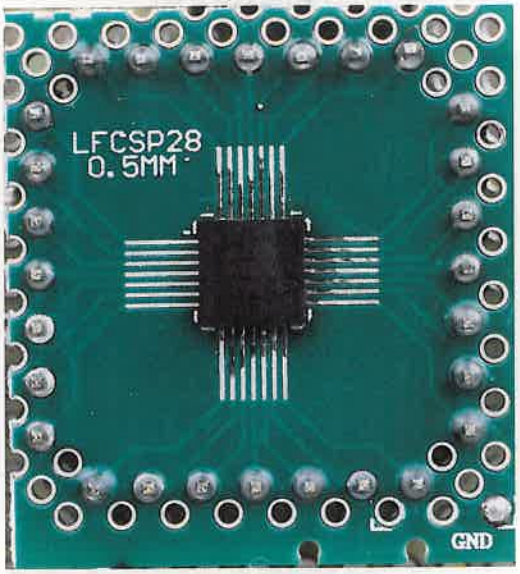
We have worked with QFN packages before. Reference the March 2009 Design Cycle column in which we designed and scratch-built a QFN-based USB interface. This go-round, we're going to work "Schmarter" not harder.

The folks at SchmartBoard offer a very clever way of mounting various QFN packages. Take a look at **Photo 2**. Notice the absence of the printed circuit board (PCB) coating in each QFN body area. The size of the removed PCB coating matches that of the package size of the associated QFN part we wish to mount. Note also that the QFN interface leads are formed to allow it to be cradled in the bare pit.

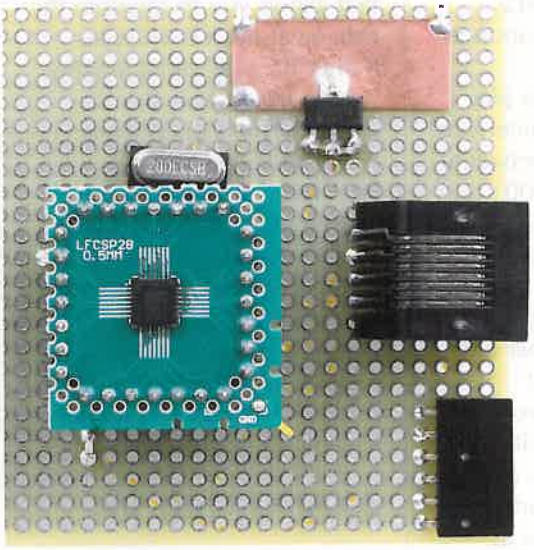
When placed in the pit, the QFN device's pins are aligned perfectly with the SchmartBoard's QFN PCB traces. How cool is that? **Photo 3** is a photographic capture of an AX8052F100 in the pit.

My minimal design is graphically depicted in **Schematic 1**. All of the radio-related SPI portal pins are aligned on the left side. The debug/programming interface is situated on the right quarter of the AX8052F100. The debugging interface is composed of the DBG_EN, DBG_CLK, and DBG_DATA signals. The PB3 I/O pin is used by the debugger to awaken the AX8052F100 from deep sleep. The crystals and associated capacitors are optional equipment. Just for grins, I decided to include a 20 MHz crystal in our design.

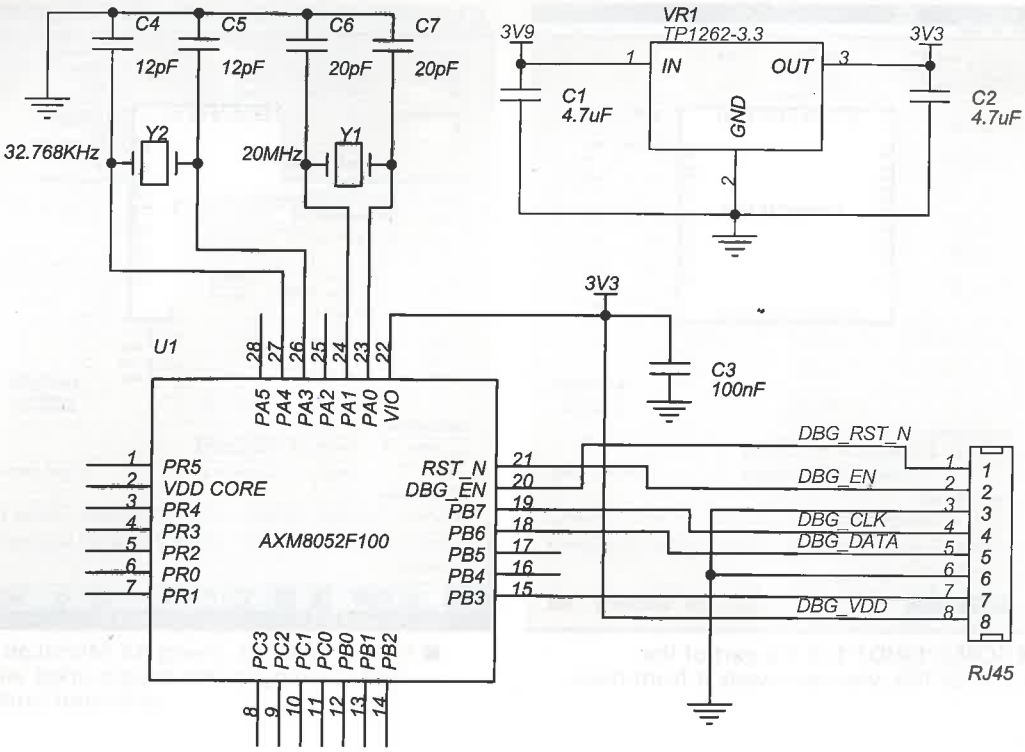
The hardware that is presented graphically in **Schematic 1** is physically represented in **Photo 4**. I trimmed the SchmartBoard before mounting the AX8052F100 in the pit. The SchmartBoard pad that electrically contacts the AX8052F100's exposed pad is isolated from the PCB's common ground plane. So, I scratched off some of the coating that surrounds the isolated pad and



■ PHOTO 3. This shot shows the AX8052F100 mounted in the SchmartBoard QFN pit.



■ PHOTO 4. A little bit of slicing and dicing plus some solder, wirewrap wire, and sockets make up our minimal AX8052F100 hardware design.



■ SCHEMATIC 1. This is as basic as it can get. The RJ45 connector is used to interface with the AXSEM AXDBG debugger.

made an electrical connection between it and the ground plane. I also placed a dab of solder into the hole at the center of the QFN pit to make sure the AX8052F100's exposed pad had an electrical path to ground via the isolated pad, which is now a grounded pad.

In this design, the AXSEM AXSDB debugger supplies power to the AX8052F100 by way of pin 8 of the RJ45 connector. The incoming 3.9 volts is regulated by the Microchip 3.3 volt voltage regulator (VR1).

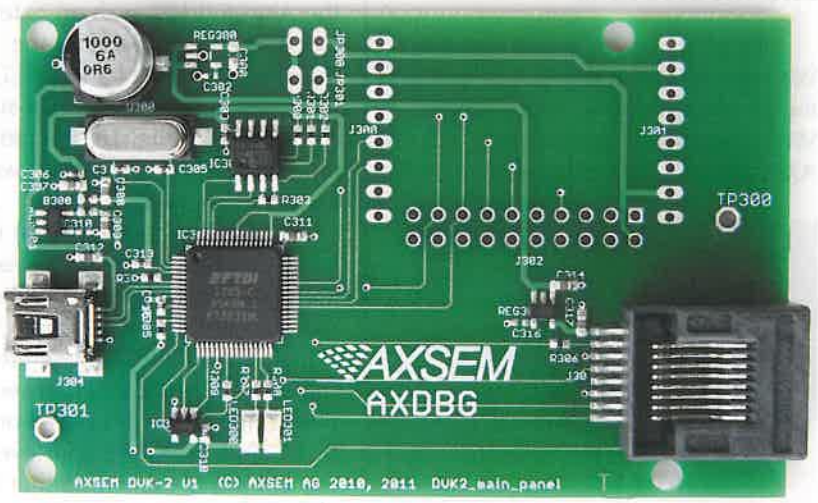
THE AXSEM AXSDB DEBUGGER

Our minimal AX8052F100 system will take its orders from the AXSEM AXSDB debugger until it is able to walk on its own. The AXSDB debugger you see in **Photo 5** is based on an FTDI FT2232HL dual UART/FIFO converter. The +5.0 volts supplied by the USB host is initially converted to 3.3 volts by the first voltage regulator. The 3.3 volt power rail also supplies power to the Microchip 93C46 EEPROM which is driven by the FT2232HL. In our case, the USB-supplied +5.0 volts is also used to drive the debugger target. To be sure that a stable 3.3 volts results at the AX8052F100's power pin, a 3.9 volt voltage regulator is used to drive the target's incoming voltage regulator.

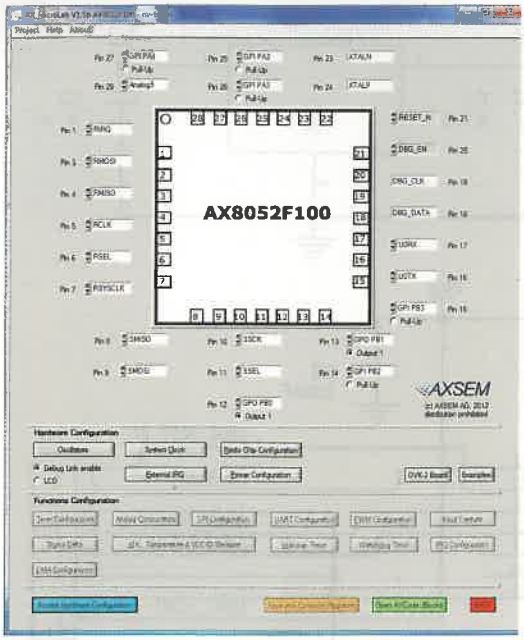
Judging from the debugger schematic – which can be obtained from the AXSEM website – the debugger data line (DBG_DATA) is bidirectional. So, to allow the FT2232HL to read and write the DBG_DATA pin, a tri-statable buffer is used to drive outgoing data on the DBG_DATA pin. Tri-stating the buffer allows incoming data to reach the FT2232HL uninhibited.

The debugger operates under the control of AXCode::Blocks which drives the FT2232HL using the FTDI D2XX direct driver. The debugger comes preloaded with an FTDI template that configures the FT2232HL's I/O subsystem.

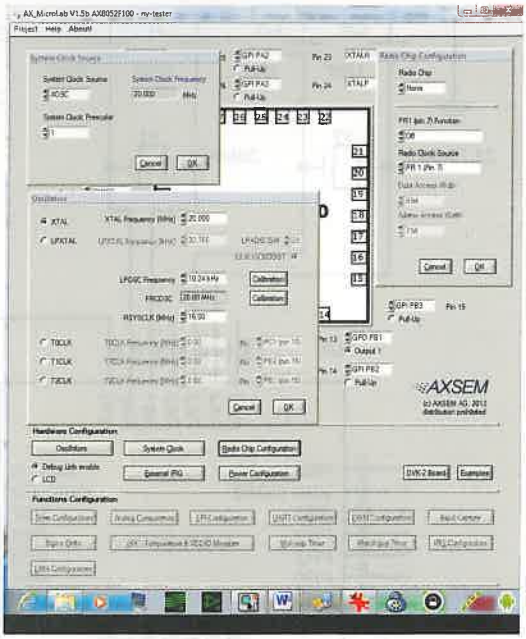
The debugging interface is enabled when the DBG_EN is driven logically high. Once it's enabled, I/O pins PB6 and PB7 relinquish their jobs and become dedicated debug interface pins. If the design uses PB6 and PB7 as I/O pins, the debugger software allows PB6 and PB7 to be set via an emulation feature. The emulation feature also allows the debugger software to read the pins



■ PHOTO 5. The AXSEM AXSDB debugger consists of an FTDI FT2232HL, a Microchip 93C46 EEPROM, a 12 MHz crystal, a couple of voltage regulators, and a single buffer IC.



■ SCREENSHOT 1. If it's part of the AX8052F100, you can tweak it from here.



■ SCREENSHOT 2. Using AX-MicroLab is akin to going to the supermarket with unlimited funds.

and set their I/O direction (input or output).

A form of code protection comes in the guise of a 64-bit key that can be selected by the AX8052F100 designer. Without the 64-bit key, the AX8052F100 firmware cannot be accessed via the debugger portal.

The AX8052F100 can be pushed into factory state using the secure erase feature. The 64-bit key is not needed to initiate a secure erase. Secure erase completely erases the AX8052F100's program Flash before erasing the 64-bit key.

AX-MICROLAB

It's all starting to come together. We have AXCode::Blocks to assist in the firmware development process. We've assembled a minimal AX8052F100 embedded system and our design can be programmed and debugged.

AXSEM provides yet another tool to take some more of the pain out of the AX8052F100 development process. AX-MicroLab is a software tool that runs on a PC and it allows the user to configure all of the AX8052F100's available features.

Take a look at **Screenshot 1**. I've created a configuration project called nv-tester which is based on our minimal AX8052F100 design. You can get lots of information about our design by simply examining the I/O pin configuration. For instance, pins 23 and 24 tell you that I'll be driving the AX8052F100 with an external clock source. Notice that the dedicated RESET_N and DBG_EN pins are nailed shut. If I had some sort of radio attached, the radio interface pins (1 through 7) would be locked in, as well.

In that we have no radio in our initial design, the radio interface pins are up for grabs; I didn't bother to set them up as general-purpose I/O. Rotating counterclockwise around the AX8052F100, you can see that I've configured a SPI portal on pins 8, 9, 10, and 11. While your eyes are in the SPI portal area, note that I've configured PC3, PB0, and PB1 to output a logical high on startup.

Moving to the far right quadrant of the AX8052F100, there are clues as to a potential serial port coming to this design. It's also pretty obvious that I have enabled the debugger interface. Finally, PA5 has been assigned analog input duty.

It's not enough just to perform the AX8052F100 pin assignments. If a pin is selected to do something other than general-purpose I/O, you may need to give AX-MicroLab a bit more information. For instance, in **Screenshot 2** I've specified that the external oscillator runs at 20 MHz and is the system clock source. Even though I didn't assign alternate duties to the radio SPI portal pins, I did take the time to release them by not defining a particular radio IC.

Clicking on the Accept Hardware Configuration button gives us access to the Functions Configuration buttons. In **Screenshot 3**, I completed the SPI portal setup by enabling the portal as a master running under the system clock at 1.25 MHz. The UART configuration has been altered to run with an eight-bit word and one stop bit at 9600 baud. Recall that the AX8052F100 can measure temperature and its incoming supply voltage. I've set that up in the ADC Settings window along with assigning the ADC type and gain. Note also that the ADC is free running and is set up to take 38.15 samples every second.

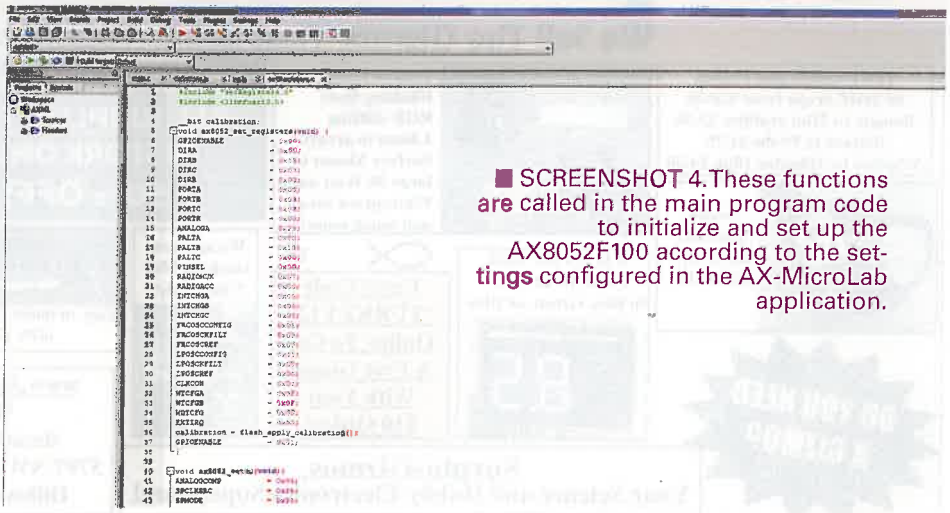
We've done the easy portion of the AX8052F100 setup. Clicking on the Save and Compute Registers button creates the code behind our selections. AX-MicroLab creates a function called *ax8052_set_registers* and another called *ax8052_setup*. I've captured a portion of each function

Lemos International
AX8052F100
AXSDB Debugger
www.lemosint.com

AXSEM
AXCode::Blocks
AX-MicroLab
www.axsem.com

in **Screenshot 4**. As you can see, the *ax8052_set_registers* is concerned with things like setting up the direction of the AX8052F100's I/O ports. The *ax8052_setup* function takes care of the AX8052F100's functional blocks. For instance, the UART is initialized inside of the *ax8052_setup* function. If you turned on the debug interface, that too is initialized and started from within *ax8052_setup*.

In addition to building the *ax8052_set_registers* and *ax8052_setup* functions, AX-MicroLab generates nine additional example files. The idea behind the example files is to show you how to write the C source code to manipulate a number of the AX8052F100's resources. In that the AX8052F100 startup code is also part of the example code, you can use the examples to seed your home-grown applications.



■ SCREENSHOT 4.These functions are called in the main program code to initialize and set up the AX8052F100 according to the settings configured in the AX-MicroLab application.

WHAT'S NEXT

Although the SchmartBoard was there for us this time, we will have to eventually build up our AX8052F100 on a professionally manufactured PCB. So, if that's the case, I'll think about what other goodies we can add to our board.

I also want to further examine the example firmware and the AX8052F100 firmware library. After all, we've got a SPI portal, a UART, and some ADC circuitry that needs to be exercised.

Once we get our AX8052F100 legs under us, we'll be one more step closer to scratch-building our integrated AXSEM radio and earning the privilege of wearing that pointy hat. **NV**

■ SCREENSHOT 3. This beats the **heck** out of sifting through the AX8052F100 datasheet and programming manual for bit settings.