CIRCUIT CELLAR ONLINE

FEATURE ARTICLE

Fred Eady

Fine Tuning an Embedded Idea

Part 3: Armed and Ready

As a result of torrential rain on his vacation, Fred got the "opportunity" to check out a TV documentary on Colt weapons. This triggered an idea for the final installment of articles on Rabbit Semiconductor and Z-World computing modules. This article is loaded with information, so sit back and imagine a beach bungalow on the Gulf shore....

This is the final installment in this series of articles describing Rabbit Semiconductor and Z-World computing modules. Instead of writing from the familiar confines of the Florida room, this installment is being brought to you from a Gulf shore beach bungalow in the Florida Panhandle. Right now, it's raining sideways and has been since I arrived. Consequently, I've done some TV time that was originally reserved for listening to the surf and watching the gulls and pelicans do some aerial beak fishing.

With rain pounding the ocean-side sliding glass doors, the only "surfing" I can do is with the television remote. I happened upon a History Channel episode featuring the guns of Sam Colt. My Dad is a retired Army Command Sergeant Major, and as a result, over the years I developed an interest in military history. If you're interested in the ways of the military, you're also intrigued with a major part of any military operation—guns.

I don't recall any Westerns where the shoot-out stalled for a major reload session. In fact, some cowboys seemed to have self-loading pistols, as they never seemed to run out of bullets. To my amazement, the TV documentary pointed out that all of Sam's early five-shooters (there were no six-shooters in the beginning) came with a set of specialized tools. The well-built early Colt revolvers had to be partially disassembled to load powder and ball, and the tools included with the guns were a necessary evil for the, at that time, much-sought-after multi-shot capability that the Colt pistols provided.

As a young man, Sam took a job as a common laborer on a cargo vessel. The idea of a multi-shot handgun came to Sam as he watched the helmsman turn and lock down the wheel. He applied the turn and lock method to bullets and barrels, and the rest is history.

As Sam's first series of revolvers found homes in the hands of lawmen and soldiers, the "tool set" disappeared as improvements suggested by the guns' users were rolled into later

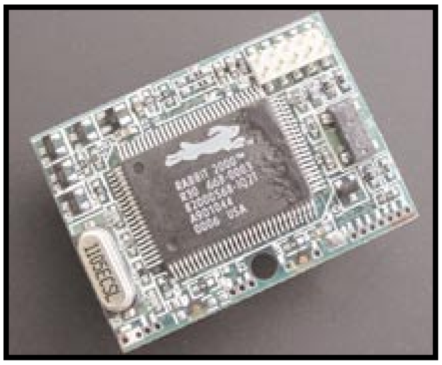

Photo 1—*This is the microprocessor business end of the RCM2300. The 11.0592-MHz microprocessor clock crystal is at the far left, and the black 32.768-KHz real time clock crystal can be seen on the far right. All of the MOSFETs and transistors used to switch various functions including the backup battery are located on this side of the RCM2300 board.*

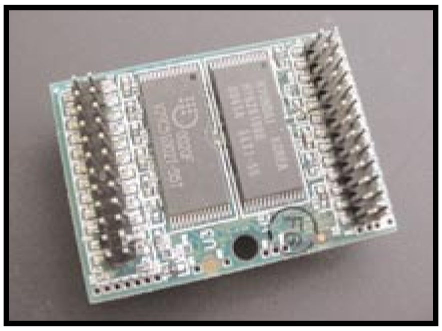
Photo 2—*This side of the RCM2300 holds the data processed by the topside. The physical interconnects are also here as 26-pin headers.*

production models. Ultimately, the 30-something parts of the first Colt-Patterson guns were reduced to seven. The most famous Colt six-shooter, the Peace Maker, is still in production today alongside the military M-16 automatic rifle, which is also a Colt product.

The success of Sam's revolvers came with the combination of a reduced parts count and prefabricated metal-jacketed ammunition. A minimum of moving parts coupled with a drastically shortened reload time made Sam Colt's guns must-have items for lawmen, native Americans, soldiers, and bandits.

This is hardly the forum for discussing weapons of war, but watching the piece on Sam's Colt revolvers put a relative thought in my mind about embedded computing. In the beginning, prior to the introduction of the 8047 and 8051 and even through the Z80 days, the smallest of embedded computing platforms consisted of the microprocessor, a clock generator IC with associated crystal, buffers and latches for the address and data busses, one or more ROMs, one or more RAM devices, and a UART IC. If the embedded device required interrupts or analog input, a few more specialty ICs could be added to the mix I just mentioned. Although you had to thoroughly understand the hardware and firmware aspects of such an embedded system to get anything out of it, this conglomeration of discrete ICs could be successfully brought together as a working embedded unit outside the lab by computer hackers of the day.

My first throw at embedded computing was the breadboarding of an Intel 8088 machine, complete with 2716 EPROM, 2 KB of static RAM, a keypad with associated circuitry, and a 4-digit 7-segment LED display array with support circuitry. That was a mouthful, and the 8088 system I assembled was a mass of wire wrap wire crafted on a 0.100″-center perfboard and wire wrap sockets I purchased at Radio Shack. At the time, I didn't even own an EPROM programmer. I entered the assembled machine code into the 2716 EPROM by hand using a crude lash-up of latches and toggle switches.

It's still raining at the windows here, and I'm just as hard-code embedded now as I was then. I still like to assemble microprocessor- and microcontroller-based embedded systems piece by piece, but sometimes it's easier (and faster) to use an off-the-shelf embedded component. And, like Sam's Peace Maker, the less complicated the better.

RETRO RABBIT

Recently, I was given the "opportunity" to retrofit an existing microcontroller in a relay control module. The upgrade required that the new embedded computer fit within the existing area occupied by the current microcontroller. In addition, the new embedded microcontroller must include nonvolatile storage with unlimited read/write capability and a real time clock. With the addition of the real time clock and nonvolatile storage, a way was needed to retrieve or change the nonvolatile data and set and read the time if necessary.

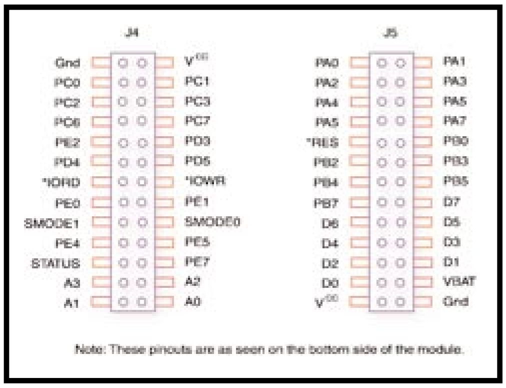

Figure 1—*If you're wondering where some of the other I/O pins are, they are implemented as connection points along the mounting hole edge of the RCM2300.*

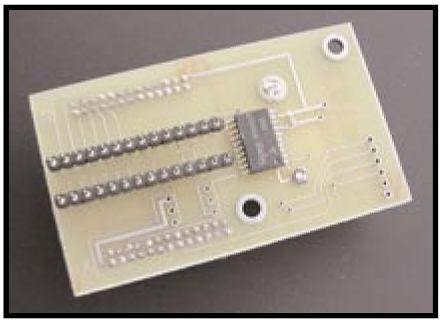
Photo 3—*This is the bottom side of the RCM2300-based upgrade module. Note the dual inline header pins. These pins allow the enhanced RCM2300-based embedded complex to assume the pinout of the microcontroller it is replacing.*

Serial communication between the relay module and a personal computer would be the simplest method to effect this. So, the new embedded computer would also need at least one UART capable of asynchronous operation. There was a possibility that the relay modules would need to be networked to a master controller, which uses RS-485, on the plant floor. To accommodate this anticipation, the newly installed microprocessor complex needed an additional serial port capable of RS-485 operation.

If you were paying attention, you already know this will not be an "8088 breadboard" solution. After carefully measuring the area around the relay module's current microcontroller, I conclude that I had about 4.5″^{3} of space to contain the upgraded embedded microcontroller. Using surface mount parts, I figured I could easily fit all of the necessary hardware in the space I had. Problem was, I'd be doing this more than once, as there were many units to be converted. Because I didn't farm out any of the assembly work, I wanted to make this as easy on myself as possible. So, on my part that means minimal printed circuit design with an absolute minimum of parts on the board.

Although this task could have been accomplished with most any microprocessor complex, the quickest way from point A to point B was the RCM2300. As of the time I wrote this article, the RCM2300 is the smallest RabbitCore module available. The RCM2300 is a small microprocessor complex (1.15″ × 1.60″) based on the Rabbit Semiconductor Rabbit 2000 microprocessor. The RCM2300 was designed to be "plugged in" and sits atop two 26-pin, 2-mm male headers. Most everything required to replace the old microcontroller and perform the newly required tasks was accomplished with the RCM2300 hardware.

Photos 1 and 2 are respective topside and bottom views of the RCM2300. The processing and switching components are located on the topside. As you can see in Photo 2, 256 KB of flash memory and 128 KB of SRAM are bolted to the bottom of the RCM2300 along with the male headers that provide the physical connections to the outside world. Figure 1 is a depiction of the layout of the male headers.

PREFLIGHT CHECKLIST

Let's check off what the upgrade requirements were versus the RCM2300's capabilities. The first concern, size, was satisfied by the RCM2300's compact form factor. The relay controller module needed the ability to keep track of relay closures and durations. In addition, the relay controller module had to have a means of identifying itself to the master controller. If the relay controller has to be moved to another master controller network on the plant floor, the relay controller ID will need to be changed and remembered.

Because the specifications called for unlimited reads and writes to nonvolatile storage, I considered using RAMTRON's ferro-based nonvolatile serial memories for this, but as good as that sounds, it adds complexity and parts to the solution. The RCM2300 has all of the necessary circuitry onboard to support battery-backed SRAM operation. I had to add a battery to the solution to support the real-time clock anyway. So, with the battery being a necessary item, the RAMTRON part was one less part I needed to add.

By simply attaching a 950-mAh lithium cell between the RCM2300 VBAT pin and ground, unlimited read/write nonvolatile storage is effected and the real-time clock is implemented. The final requirement of serial communications was satisfied by the RCM2300's abundance of serial ports. In addition to the programming port, the RCM2300 comes standard with a serial port configuration capable of two 2-wire interfaces or one 5-wire interface. The upgrade uses the services of one of these serial ports in 2-wire mode for asynchronous communications between the RCM2300 complex and an external terminal or personal computer. If the need arises later, the remaining 2-wire port can be equipped with a standard RS-485 converter, and with the help of a single I/O for transmit/receive switching, the RCM2300 embedded complex can become a fully functional RS-485 network node.

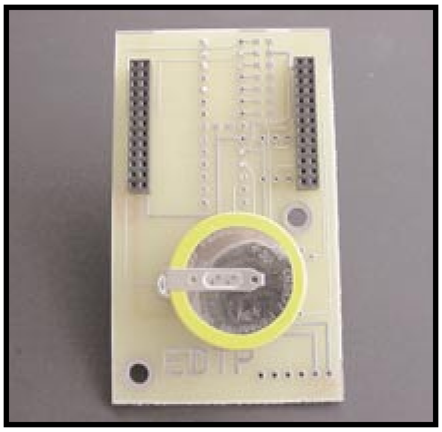
Photo 4—*This is a look under the RCM2300 module. As a precaution, I added pads for SMD pullup resistors on the input pins of the original microcontroller. I also made provisions for the installation of a couple through-hole MOSFETS to drive high current outputs. It's best to have more pads than you need than to need them and not have them.*

Basically, the hardware consists of a simple printed circuit board, a lithium backup battery, the RCM2300, and an RS-232 converter IC. If you're wondering why I included the printed circuit board as part of the hardware, you'll get your answer by studying Photo 3. What you see is a 28-pin dual inline arrangement that is designed to be plugged into the socket of the microcontroller the RCM2300 complex replaced. This arrangement allows the RCM2300 embedded computer to access the resources of the original relay controller via the pinout of the microcontroller it replaced.

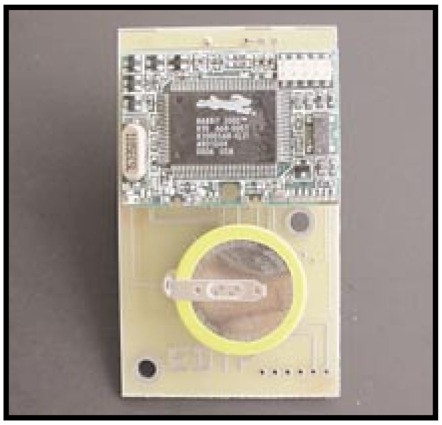
Photo 5—*I can assemble this upgrade module in about 15 min.*

For instance, if pin 1 was an output pin on the original microcontroller, All I had to do was assign and hardwire one of the RCM2300 I/O pins to the IC socket's pin 1 and code it for output in the code. The original equipment microcontroller had a single UART that was pinned for use with an external RS-232 converter. To mimic this functionality, I simply designed the RCM2300 motherboard to allow one of the RCM2300 serial ports to interface with the original UART pins. To go one better, I added the RS-232 converter IC to the RCM2300 motherboard in the form of the Sipex SP233. I chose the SP233 because it does not require the charge pump capacitors to operate in true RS-232 mode.

Moving to the topside of the RCM2300 motherboard (see Photo 4), you'll find the female headers that support the RCM2300 core module and the 950-mAh lithium backup battery. This view gives you an idea of what can take place under the RCM2300 module. As you can see, just in case I need them, I added some SMT resistor pads to accommodate pullup resistors on the old microcontroller input pins, which are mapped to I/O pins on the RCM2300 module. Photo 5 is a view of the complete RCM2300-based embedded upgrade module with the RCM2300 mounted on the motherboard.

There's one more piece of hardware I want to mention before I get into the development of the final firmware. Although this hardware doesn't have any direct bearing on the operation of the RCM2300-based upgrade module, it does have a big impact on the design of the upgrade module's printed circuit board. The piece of hardware I'm talking about is the RCM2300 Development Kit. I used the RCM2300 Development Kit to baseline the upgrade module's design.

As you can see from Photo 6, I added the optional RS-232 converter IC to my RCM2300 Development Kit to allow testing of my firmware with a known good serial configuration. To help in firmware development, I wired the upgrade module's serial interface just like it is wired on the RCM2300 Development Kit. My RCM2300 Development Kit came with Dynamic C SE, a RCM2300 core module, prototype board, programming cable, power supply, a "Getting Started" manual, and a Rabbit 2000 Easy Reference poster.

FIRMWARE DEVELOPMENT

Using the RCM2300 core module made the hardware portion of this upgrade project simple. Fortunately, the Dynamic C software development platform does the same for the working code. I made sure that I wired the upgrade module in such a way to allow for quick removal of the RCM2300 core module from the upgrade motherboard and the ability to plug it into the RCM2300 Development Kit board to verify proper operation of my code and the hardware.

An added plus in selecting the RCM2300 as the upgrade engine is the ease of coding and debugging the upgrade firmware. In Photo 5, you can see the male white-based 2-mm programming header. The programming cable that comes with the RCM2300 Development Kit has an integral RS-232 converter inline with the programming cable itself. This arrangement allows the development platform to be a simple laptop or desktop. No external EPROM or microcontroller programmers or emulators are required to write, debug, and load RCM2300 code. All you need is a serial port and a programming cable.

Because I used the schematic to wire my upgrade module serial ports, I was able to take advantage of the sample code that comes with Dynamic C SE to quickly verify my RS-232 connectivity with the RCM2300 attached to the upgrade module. In fact, I developed 99% of the code with the RCM2300 core module mounted on the upgrade motherboard. I really didn't miss the coding, erasing, programming, and then testing it loops. I also didn't miss having to expend funds for external debugging and emulation hardware, either.

I'm used to writing all of my code from scratch. Using Dynamic C SE was a pleasant experience, in that it is standard C with extensions for the RCM2300. If I had problems with syntax or logic, I simply loaded in one of the sample programs and studied the code. The built-in debugger also came in handy, as I could just stop the code, set my watches, and single step through the problem areas. The Easy Reference poster that comes with the RCM2300 Development Kit also came in handy a few times, as I had to determine how to control the secondary serial port and how to manipulate one of the parallel ports. I was able to figure it out using the reference poster, which saved me the pain of digging through a mass of documentation.

Another advantage to using the RCM2300 and Dynamic C SE is that Dynamic C provides cooperative, time slice, or preemptive multitasking. In addition to controlling the activation of relays, the relay controller also monitors a number of control panel switches. These switches must be monitored at all times and switch activations cannot be missed while the controller is servicing the relays.

I used costatements to solve the switch scanning problem. Each costatement can be equated with a task. For instance, if there were four switches that must be monitored in addition to four relays that must be controlled, the simplest method to provide coverage for all of the devices would be to assign each task to a costatement code segment. Each switch would be interrogated and serviced within its own costatement segment, as would each relay. The costatements have internal structures that keep up with the progress of the task while yielding to other events

Photo 6—*I added this Sipex 232 RS-232 converter IC to initially test the upgrade module's serial port design. The RCM2300 Development Kit has provisions for this addition. All you have to do is add the parts.*

Listing 1—*This is really cool stuff! By simply using the costatement feature and some simple logic, I can write cooperative multitasking code with simple C statements.*

```
while(1)
{
              costate
              {
                      waitfor switch_closed();
                      waitfor activate_relay1();
                      waitfor (DelaySec(60));
              }
              costate
              {
                      waitfor (DelaySec(1));
                      waitfor  flash_LED();
              }
      }
```

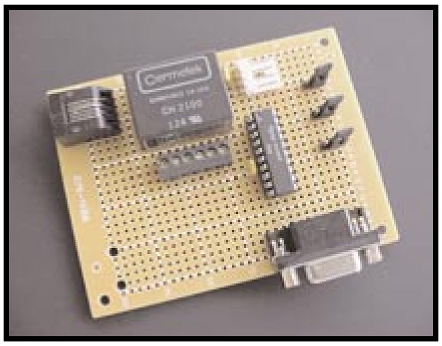
Photo 7—*The 276 part number on the perfboard says Radio Shack, but the dominant component is a full-blown Cermetek 2400-bps modem.*

during nonproductive time. For instance, take a look at Listing 1.

The first costate waits for a switch to close. Assume that all of the switch detect and debounce code is included within the switch_closed function. If the switch_closed function in the first costate segment is not satisfied, the first statement in the second costate segment is executed. The code won't hang at the DelaySec(1) statement. Instead, it will jump back to the first unfinished statement within the first costate segment, switch_closed.

Let's assume the switch never closes in the first costate segment. One second will eventually pass and the next waitfor statement in the second costate segment will execute. The LED will be flashed according to the coding within the flash_LED statement, and the next unfinished statement in the first costate segment (switch_closed) will again be examined. Again, assuming the switch never closes, the first statement in the second costate segment will again begin to run for 1 s.

As you would logically conclude, if the switch closed, the activate_relay1 function would execute, and upon completion, the first costate segment would enter a 60-s delay. The second costate segment would have the opportunity to execute the flash_LED function once per second while the top costate segment 1-min. delay was being observed.

My costate example is simplistic. Depending on what you wanted to accomplish, the costate functionality can be expanded in many other ways. Costates can among other things be named, paused, and aborted. Cofunctions can be intermixed with costatements for additional functionality. Using costatements, I was able to provide multitasking functionality to the upgrade module without the need to do it all from scratch.

LEFT TURN, CLYDE

As I was finishing up the upgrade module firmware, the customer threw in an additional requirement. The upgraded module should be able to operate standalone and dial up a remote plant floor master. The modem hardware was not to be included with every module, however, each module should have the capability to be attached to the modem hardware. Fortunately, the specification stated that the RS-485 and modem functions be mutually exclusive. That meant that I could use the second serial port for the modem interface as well as an RS-485 port.

As of this writing, I did not have the professional modem boards back to show you. However, I followed the same baseline technique I used with the RCM2300 Development Kit and the serial circuits on the upgrade module. I built the development prototype in Photo 7, which allowed me to connect the Cermetek CH2100 modem directly to a personal computer serial port or the upgrade module by simply changing the three jumpers.

The Sipex SP233 on the modem module is there to interface the personal computer serial port to the CH2100. The RCM2300-based upgrade module and the CH2100 don't need an RS-232 converter between them. Instead, the RCM2300 and CH2100 use standard TTL levels to communicate with each other. The CH2100 is capable of 2400 bps and uses the standard Hayes AT command set. The command set responses replace the hardware modem signals and, thus, reduce the CH2100 pin count to eight. This allows the CH2100 to be directly connected to microprocessor UARTs with no additional supporting parts.

Two wires, power, and a phone jack are all that's required to place a data call with the CH2100. The interface for the modem or RS-485 lines are supplied by a 5-pin header of solder connection directly to the right of the EDTP symbol on the upgrade motherboard. Screw terminals on the modem module allow for quick installation in the field.

SUCCESS

When the dust cleared, what I ended up with was a compact, highly functional, embedded computing system capable of keeping time, processing and retaining statistical data, controlling external devices on a timed basis, and communicating with other devices by various means. Because the RCM2300 can be programmed using a laptop, servicing the upgrade module firmware in the field becomes a no-brainer. The customer did not choose to employ the Internet communications and programming options offered by the RCM2300, but if things change, I'm ready to perform with the platform I've designed.

This particular application of the RabbitCore module showcases the advantages of employing the RabbitCore modules in places where you would normally put a microcontroller or multi-part microprocessor complex. By employing an off-the-shelf RCM2300 and Dynamic C, I saved hours of design and debug time and was able to bring a reliable solution to my customer in much less time than if I had done this in the traditional way.

Well, it has stopped raining now, and the sun is trying to poke out of the clouds. I think I hear a gull calling my name. Until next time…

Fred Eady has more than 20 years of experience as a systems engineer. He has worked with computers and communication systems large and small, simple and complex. His forte is embedded-systems design and communications. Fred may be reached at fred@edtp.com.

SOURCES

CH2100 Modem
Cermetek
(408) 752-5000
Fax: (408) 752-5004
www.cermetek.com

RabbitCore RCM2300, Rabbit 2000 microprocessor
Rabbit Semiconductor
(530) 757-8400
Fax: (530) 757-8402
www.rabbitsemiconductor.com

SP233 Transceiver
Sipex Corp.
(978) 667-8700
Fax: (978) 670-9001
www.sipex.com