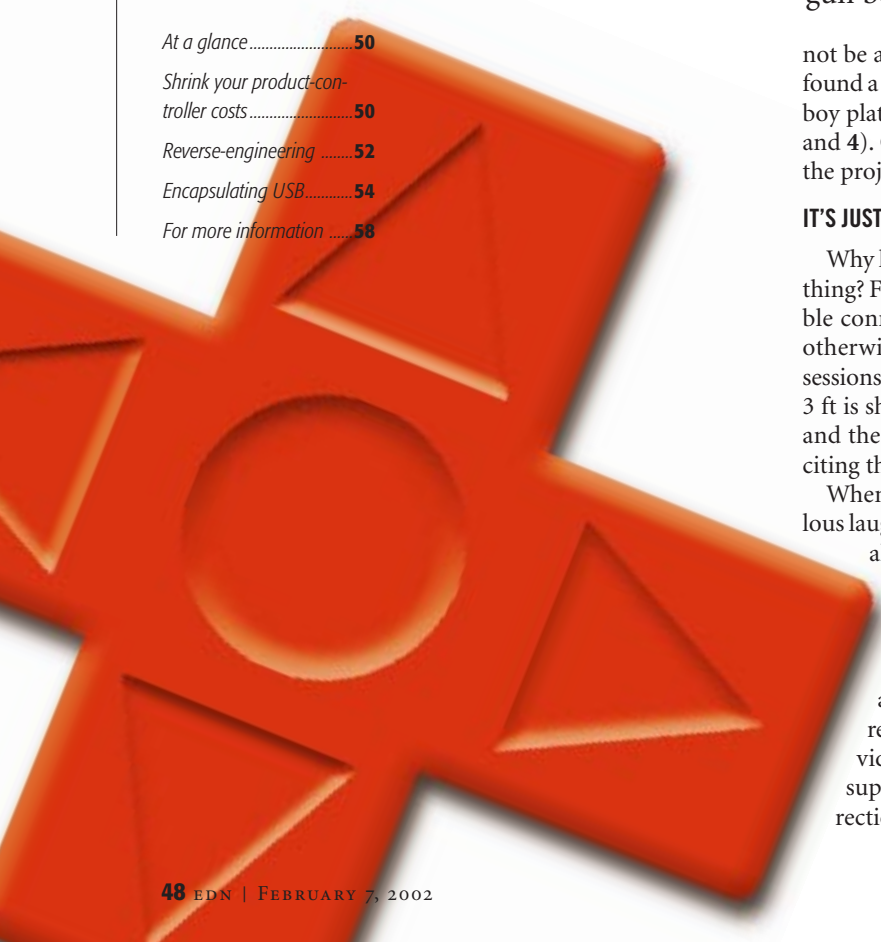
hands-on project

GAMING as *serious business*

SNICKER ABOUT PLAYING GAMES AT WORK, BUT THE LOW COST, APPROPRIATE FEATURES, AND AVAILABILITY OF GAMEBOY RESOURCES MIGHT CHANGE YOUR MIND ABOUT USING IT AS A NONGAMING, HANDHELD TERMINAL.

By Robert Cravotta, Technical Editor



It started with a "no thank you." I had asked for the technical specification for the Nintendo Gameboy game-link interface so that I could explore connecting it to USB and to an 802.11b wireless access station. This hands-on project looked finished before it had even begun because that interface specification would not be available. Not easily discouraged, I searched for and found a world of independent, public support for the Gameboy platform that I had no idea existed (**references 1**, **2**, **3**, and **4**). Official information was unnecessary; as quickly as the project had died, it was alive again.

IT'S JUST FOR GAMES...RIGHT?

Why link a Gameboy Color or Gameboy Advance to anything? From a pure gaming perspective, a 3-ft game-link cable connects Gameboys for multiplayer games, to unlock otherwise-unreachable features, and to trade virtual possessions among systems. In this day of online connectedness, 3 ft is short. Look at the Gameboy as a handheld terminal, and the reason for connectedness becomes even more exciting than mere games.

Whenever I told anyone about this project, an incredulous laugh and comment about playing video games at work always followed, but that situation changed as they considered this question: How many programmable, backward-compatible, handheld terminals can boast more than 100 million units shipped since 1989 and all that experience with ruggedness and ergonomic design issues that these devices represent? The older color and new advance versions provide a color LCD; sound; volume control; headphone support; multiple input buttons, including four in a directional configuration; serial and infrared I/O (color);

dc-power support; and a small cartridge to change programs. In addition, these versions can operate on AA batteries for 10 to 15 hours (20+ hours personal experience with the color version). A local retail store recently had the color version in stock for $70 and the advanced version for $90. The advanced version was not in stock because it was recently released, and holiday purchasing was at a peak. With a 32-bit ARM processor powering the advanced version and an 8-bit, z80-like processor powering the color version, these systems can be interesting for nongaming applications (see **sidebar** "Shrink your product-controller costs").

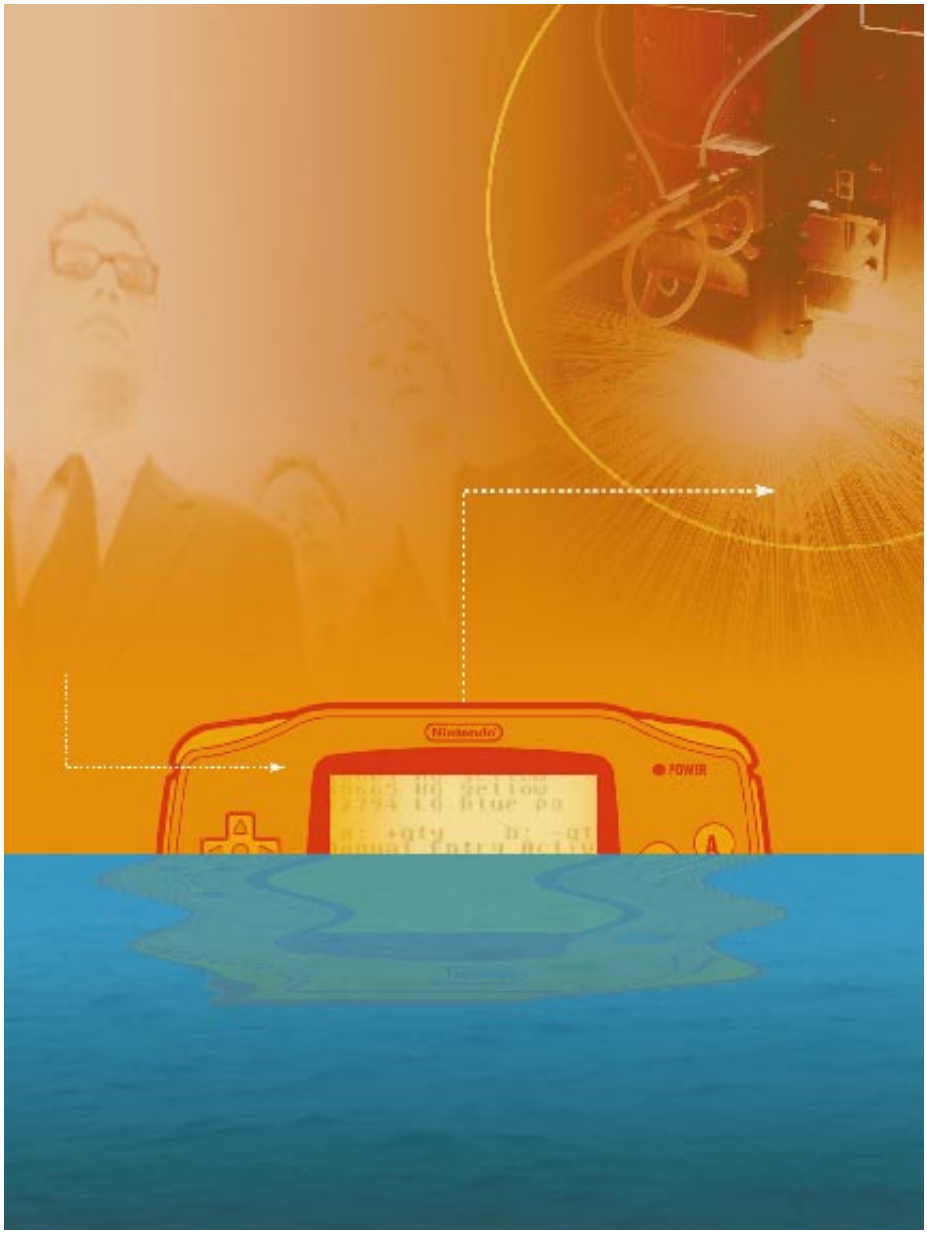


Image courtesy of Mike O'Leary

WHICH INTERFACE?

The goal was to connect a Gameboy with one of its proprietary interfaces to a standard interface, such as would be available on a desktop computer. I started with my child's Gameboy Color that supports connection through a game-link port, an infrared port, or the cartridge slot (**Figure 1**). My equipment lacks infrared ports but has wired and wireless interfaces. Also, the Gameboy Advanced dropped infrared support, so I didn't consider using the infrared port. The initial project plan was to implement both a wired and a wireless connection that would not drain too much of the Gameboy's batteries.

A wireless connection complements the handheld, untethered feel of the Gameboy, and 802.11b was the obvious choice for the wireless connection because my system already had the capability. One reservation I had about using a wireless connection was that it would cost more than the Gameboy itself, but realizing that the system would still cost less than a contemporary handheld computing device with the same connection offset that drawback. For a wireless connection, external power to the 802.11b interface device would have been an unreasonable luxury; it would too quickly drain the Gameboy's batteries and could risk damaging the Gameboy. When I discovered the Gatesboy cartridge on the Web, implementing the wireless connection through the Gameboy-cartridge slot became an obvious choice.

The Gatesboy acts like a color-version game cartridge to a Gameboy Color or Gameboy Advanced, but you wouldn't confuse the Gatesboy for an official cartridge because it is about the size of a Gameboy and lacks a connector to accommodate an official cartridge, further emphasizing the nongame intention of this unit (see **sidebar** "Reverse-engineering"). The fully encased unit extends into the Gameboy-cartridge slot; supports program downloads and five bidirectional lines through a DB25 connecter; accommodates two daughterboards for custom hardware; has a slot suitable for a SmartMedia card or ribbon cable; and uses an STMicroelectronics' PSD813F2 with 128 kbytes of flash memory, 2 kbytes of RAM, and 3000 gates of programmable logic. It has space in the casing that can accommodate batteries. Even though I used the Gatesboy to load and run the custom programs, I had to drop the wireless connection portion of the project due to time constraints.

For a wired connection, the goal was to use a smart cable that would bridge the game-link interface to a desktop computer and receive power externally to minimize the eventual controller bulge in the cable. The game link can provide power through one of the pins, so the smart cable can, in a worst-case scenario, obtain

power through it, but power from the desktop connection is better.

A few years ago, I owned nothing that supported USB. Now, it seems that everything new supports it, and I recently connected a new printer via USB instead of the traditional parallel port. USB offers hot-swapping and plug-and-play simplicity so that any user can connect his or her USB enabled Gameboy to the computer. I have less confidence that any user can properly use a serial or a parallel port connection. Because USB defines that the host will provide a minimum current, the smart cable would not have to drain power from the Gameboy batteries. A search for a USB/Gameboy adapter revealed that none was available but that interest for one exists.

As a future-looking consideration, the developments in the USB OTG (On-The-Go) specification allow USB devices to appear peer-to-peer (**Reference 5**). It defines dual-role devices with limited host capabilities that support host negotiation that transfers the host function between two devices without switching the cable. This development suggests interesting direct-connection possibilities between enabled devices in the future. Because the USB host must provide a current, the USB/game-link adapter can never be the A device in an USB OTG connection unless modified to provide current on the Vbus.

AT A GLANCE

- Thinking out of the box can help you find new strengths for narrowly defined systems.
- Successful and extensible architectures encourage reverse-engineering and can spawn unintended uses.
- Unintended uses can sometimes hurt a product's profitability or open new markets.

PARTS AND TOOLS

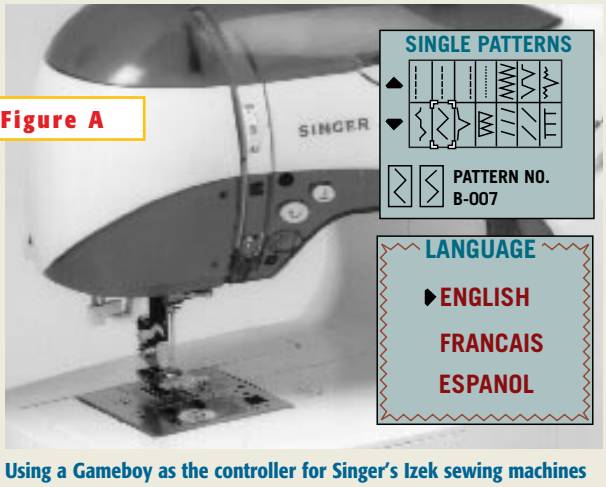
Trying to avoid building the adapter from scratch, I found a number of USB interface and development boards at DevaSys and a reference to an interface board that would be using a Cypress EZ-USB FX2 USB microcontroller, a candidate device for this project. When I inquired about the board and explained what I was trying to do, DevaSys owner Michael DeVault offered to modify the existing USB I²C/I/O interface board that uses a Cypress AN2131QC, also an EZ-USB microcontroller, to work with the game-link interface. The board comes with a programmer API that simplifies adding USB to your application (see **sidebar** "Encapsulating USB"), programming examples, and a debugging utility to monitor what API transactions the board firmware is executing.

The EZ-USB has "re-enumeration" that can release the on-chip 8051 from reset after downloading custom code from the host and appear as a new device that has just entered the bus to the host. This technique allows the firmware updates in the interface board by changing the code loaded during start-up from the host. This approach meant for the project that firmware updates were as simple as loading a file into a Windows system directory. The most complicated part of updating the firmware was handling filtered files because the e-mail server kept replacing executables with text files. This situation was true even for .txt files in a compressed file, which is baffling, be-

SHRINK YOUR PRODUCT-CONTROLLER COSTS

What would it mean to your product's bill of material, assembly, and maintenance costs if you could reduce the control interface to a single connector supporting an electrical interface to your control processor? Using a programmable, handheld terminal, such as a Gameboy, that presents a nonthreatening form factor to interface and transfer the operator's commands to the product controller, you can amortize the cost for that handheld terminal across many products.

Figure A



Using a Gameboy as the controller for Singer's Izek sewing machines reduces not only the cost and mechanical complexity of the stitch-programming interface, but also the time a user must spend at the sewing machine when programming stitches (top). It also allows for on-the-spot localization of control, help, and tutorial material (bottom).

The Singer Sewing Co achieved this goal with its Izek sewing machine (Figure A). The company first distributed these sewing machines in Japan and assumed that buyers owned Gameboys. In contrast, when Singer marketed the machine in the United States, the company bundled a Gameboy with it. You can see a controller demonstration at www.meetizek.com. Using a terminal allows localized control, help, and tutorial information to reside in the terminal, available when the user needs it most without referring to the owner's manual. You can program new stitches and manage and share your stitch patterns with others nearly anywhere and at any time using the Gameboy handheld system.

Unfortunately, Singer has stopped further work on the second generation of the Izek sewing machines. Even though the interface is easier to use than the traditional mechanical one, sales were growing too slowly. Does this mean that a handheld controller is a bad idea or that users are reluctant to consider a game toy for controlling a serious piece of equipment?

Convergence may happen in consumer-electronic products. However, nearly ubiquitous, programmable, handheld terminals, such as cell phones, might be avenues of convergence for control interfaces, provided that the various control devices can operate the same software. Amortizing the control interface across many products represents significant cost savings to manufacturers and consumers.

cause you cannot execute a .txt file. Chalk it up to overcaution about virus attacks. The work-around (proceed immediately to the next paragraph if you work for IT) was to rename the files to an uncensored type.

There were two other targets for software development: the Gameboy and the Windows host computer. Because the demonstration application was not going to be a performance-stressing application, I opted for the Gameboy Developer's Kit and Microsoft Visual Studio so that I could program both targets in C. The last set of tools I used were some DOS-based utilities to convert the Gameboy binaries and load them onto the Gatesboy (**Reference 6**). Although several Gameboy emulators are available, I did not use any of them and instead debugged directly on the target. There were no parts to remove or insert, and I quickly automated the compilation and loading process so that it took only a few moments between iterations. And, because the Gameboy did everything in response to a button push or a message from the host, plenty of mechanisms were available to simulate breakpoints and examine the system state.

THE GAME-LINK PORT

After configuring the tools and hardware connections (**Figure 2**), it was time to bridge the game-link interface. The specification of the game-link port is a sensitive issue, so I will describe it in general terms. The game-link port on a Gameboy Color is not a UART but a memory-mapped shift register, and, when you link it by a serial cable to another Gameboy, it can simultaneously feed and be fed by the shift register on the other system. In a normal configuration, either Gameboy generates the shared clock pulses that swap the shift-register contents.

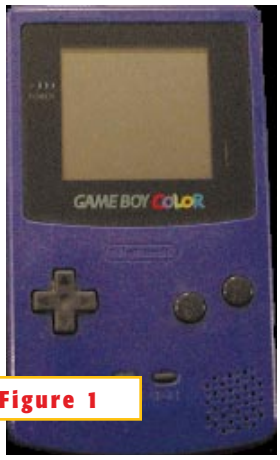


Figure 1

The Gameboy Color system I used in this project can do more than play games.

Even though the USB I²C interface board was overkill for the USB/GL adapter, it reduced the effort for new hardware and permitted the adapter effort to focus exclusively on the game-link interface. As a result, I accepted a few compromises. The adapter failed to offer exactly the same functions as a real game link: It generated the serial clock for all reads and writes, whereas the Gameboy generated none. The physical link was half-duplex. When the adapter was transmitting to the game link, it received no data, and, when receiving data from the game link, an unspecified value was in the game-link register. These compromises were acceptable because the adapter was not intended for use with standard game communications. This situation permitted the effort to focus on relevant data communications and circumvented the irrelevant specifics regarding message delays to bridge two standard game instances through the interface.

I didn't need to test the USB connection because the interface board already supported it. To initially test the game-link connection, the adapter passed through single-byte messages from a modified sample program running on the host to the game-link connection. The software on the Gameboy echoed the byte received to the display. The demonstration software used custom routines to replace the low-level serial-I/O functions in the development kit. Port initialization became important because the first serial character received after powering the Gameboy on could be undefined.

While debugging the link interface, I discovered how mixing Windows- and DOS-based tools can lead to unintended results. Windows supports long file names, and I named the project and the

REVERSE-ENGINEERING

One of the challenges of success is that other people will attempt to reverse-engineer your design for their own ends, despite your efforts to the contrary. Although an emulator is a powerful development tool and the ability to make safeguard copies of programs you have purchased is legitimate, the availability of these two capabilities have meant potential loss of revenue for Nintendo because people can avoid buying a Gameboy and cartridges to play games. Nintendo has successfully shut down some companies that sold products that have little apparent value except to permit someone to access, copy, and play games without purchasing a proper license. The Gameboy Advanced has the ability to copy a program from one device to another without using a cartridge in the other Gameboy Advanced, and that fact may explain the reluctance to share the game-link-interface specifications. The Gameboy Color, which this project used, does not support this capability, but, as projects on the Web show, a little information can result in many unintended uses of a narrowly defined product (**references A**, **B**, **C**, and **D**).

Not all reverse-engineering efforts have negative results for a company. Lego *Mindstorms*, which incorporates microprocessors, sensors, and motors into Lego bricks may have actually benefited from reverse-engineering activities (**Reference E**). Engineers, researchers, and hobbyists reverse-engineered the firmware, created new programming tools, and developed unintended ways of connecting the systems to the world. Rather than pursuing legal action to squelch this activity, the company has made available internal documentation for its firmware, sponsored conferences discussing theory and applications, and includes links to independent development tools on its own Web pages. Don't expect this kind of support if you try to produce blocks that interlock with Lego's.

REFERENCES

A. Gameboy and Gameboy Advanced Development Web Rings, http://d.webring.com/hub?ring=gameboydev and http://c.webring.com/hub?ring=thegameboyadvanc.

B. *Gatesboy*, www.gatesboy.com.

C. Frohwein, Jeff, *Gameboy and Gameboy Advanced*, www.devrs.com.

D. Ziegler, Reiner, *Gameboy and Gameboy Advanced*, www.reinerziegler.de.

E. Wallich, Paul "Mindstorms: not just a kid's toy," *IEEE Spectrum*, September 2001, Volume 38, No. 9, www.spectrum.ieee.org/pubs/spectrum/0901/mind.html.

file "terminal.c." I added a version letter at the end of the name to differentiate between iterations and usually remembered to change the version identifier in a display message. Imagine my surprise when I finally figured out that I had been loading the terminal.c version of the code instead of the terminalx.c version because DOS truncated the file name at eight characters. I might have noticed sooner if the terminal.c version had more than just the subtle initialization issue. I ultimately confirmed this fact by checking the display version identifier. Consequently, I shortened the project name to termx.c.

Figure 2

The development system includes a Gameboy Color with a Gatesboy cartridge inserted (right). The parallel cable (top right) allowed program downloading to the flash memory. A modified game-link cable connects the Gameboy to the DevaSys USB I²C/I/O interface board (left), and the USB cable (top left) completes the datapath to the desktop computer.

After the interface link was working properly with single bytes, the development added multiple-byte-message support. When testing focused on receiving multiple-byte messages from the game link, the USB I²C debugging utility helped identify that clock generation stopped after receiving the first message byte. This was an artifact from the single-byte message testing and was fixed by removing processing for the purely test-case single-byte message. The last development step added the communication-link protocol that the adapter processed for each side of the connection. The game-link link side comprised a length byte followed by a message of that length. The USB side added a control for intercharacter delay and return status that required nothing from the game link. The final deliverable for the adapter was a Windows driver file, an 8051 executable that the Windows driver automatically loaded onto the adapter when you plugged it in, and a Windows DLL that included the new Gameboy-specific API functions.

END-TO-END DEMONSTRATION

It was now time to develop an end-to-end demonstration of the USB/GL adapter in a nongame application. The choices were limited only by how much time it would take to build something the Gameboy could interact with or control. I decided to implement a handheld inventory terminal. When I was working in retail years ago, we frequently performed a physical inventory to synchronize the computer inventory with reality. If the project had included extending the capability of the Gameboy beyond a USB connection, the Gatesboy could add a bar-code-reader interface to make this application more robust and functional. However, the retail application lacked the time for that operation, so the operator would manually update the inventory by pushing buttons on the front panel.

The first challenge I faced while working with the Gameboy is its small display: 20 characters wide and 18 lines high. (The advanced version extends those specs to 30 characters wide and 20 lines high.) This amount of space is not enough to display on one line a stock number, an item description, a quantity, and an indicator that detects when a selected inventory item still needs updating. One choice would be to use more than one display line per item. Another approach would segregate the display into limited-list and detailed-item sections. I could also have used one line per item but supported scrolling the item description. With the two-line approach, item descriptions could be longer than 20 characters, so I would still need a scroll routine. During an inventory exercise, the item description is a sanity check before updating a stock-item entry, so seeing only a portion of the description is fine. The four cross-hair-configured buttons provide a convenient way to move a selected line up or down and scroll the item description left or right. I opted for the one-line-per-item approach, although if I were to repeat the project, I would seriously consider doing a segregated display (**Figure 3, top**).

Programming in graphics mode is the

ENCAPSULATING USB

The USB I²C/I/O API encapsulates the USB specifics on the Windows host computer using static or dynamic linked DLLs in your C or Visual Basic program so that you can concentrate where it matters most: your application. **Listing A** shows a subset of the available API functions, including the two application-specific calls for the game-link interface.

The USBGB_TRANS structure includes, in addition to the message, pretransfer control to the USB I²C/I/O board, such as intercharacter delay, and posttransfer status to the host, such as distinguishing between no message was ready and a disconnected or unpowered Gameboy.

LISTING A—SUBSET OF API FUNCTIONS

```
BOOL _stdcall DAPI_DetectDevice(HANDLE hDevInstance);
HANDLE _stdcall DAPI_OpenDeviceInstance(LPSTR lpsDevName, BYTE
byDevInstance);
BOOL _stdcall DAPI_CloseDeviceInstance(HANDLE hDevInstance);
LONG _stdcall DAPI_ReadGb(HANDLE hDevInstance, USBGB_TRANS *
UsbGbTrans);
LONG _stdcall DAPI_WriteGb(HANDLE hDevInstance, USBGB_TRANS *
UsbGbTrans);
```

only way to produce colored text. I worked in text mode to minimize the complexity of the project, so I created a blinking routine to easily identify which line was selected. Also, the Gameboy LCD has no backlighting so as to prolong battery life. Viewing the display requires a good ambient-light source—usually not a problem in most business environments.

I developed this inventory application on a system with 128 kbytes of program flash and 2 kbytes of RAM. The flash memory is not internally programmable, so I designed the memory to emulate nonvolatile memory and limited the database to fit within the available RAM. As a practical matter, though, 64-Mbyte flash cards are available for the color version, and 256-Mbyte cards are available for the advanced version. The color version has 16-bit-based addressing and uses bank switching to extend the available address space to 64 Mbytes.

The Gameboy lacks a touchscreen, so the eight buttons and their arrangement require your display interface to accommodate mouselike or joysticklike controls. The Gameboy also lacks an extended keypad or keyboard. This constraint was surmountable because many applications run well with only mouse controls. I defined the cross-hair buttons to support navigation and description scrolling and the A and B buttons to increment and decrement the quantity of the selected item. You could use the remaining two buttons, which remained undefined, for mode-switching and menu navigation to set display filters, searching in a long list, handling nonstandard items, or doing anything else that might enhance accuracy during a manual inventory.

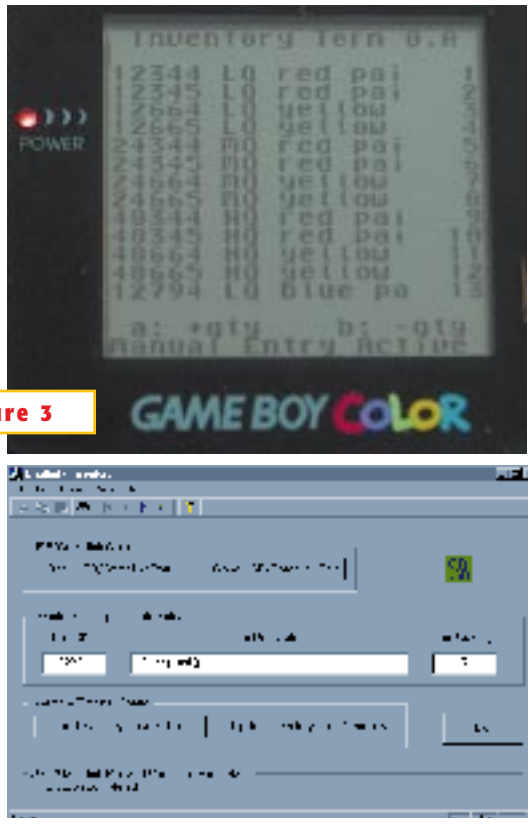


Figure 3

An inventory terminal program runs on the Gameboy (top), which communicates through the USB/game-link interface with the Windows master inventory program (bottom).

WIZARDS CANNOT SLAY ALL MONSTERS

To complete the application, the master inventory system would reside on a Windows computer and communicate with the Gameboy module (**Figure 3, bottom**). My first inclination was to write a console/DOS-like program for the master inventory system. Working with the sample programs for the USB adapter, I decided to try my hand at Windows programming using the Microsoft Visual Studio. Although the code wizard was moderately useful to get the initial database structure running with a Windows interface, the code did not help me figure out how to manipulate the data from the code rather than the display. The Microsoft's online references and technical FAQs were invaluable for understanding how to manipulate the Windows data structures (**Reference 7**).

I lacked the time to figure out how to display the database as a 2-D table, but I could read and modify it with Access 2000. The working program can display, examine, and modify all of the inventory items in the database. I used Access 2000 to expand and shrink the inventory rather than spend time implementing those functions within the main application.

Beware when programming for multiple targets in the same development environment: It can cause some confusion if you're not careful. I extended the Microsoft Visual Studio to support the DOS-based Gameboy cross-compiler and linker so that I could do all of my programming in one environment. The development environment looked identical for both programs, but the two compilers did not implement all functions, such as type casting, in the same way.

Even with program wizards to help with Windows programming and an USB/game-link API to encapsulate the interface, the task of connecting the Gameboy and desktop computer required another communication layer. Sending messages from the Windows program involved calling an API function followed by another call to retrieve a response from the Gameboy. The inventory terminal program constantly stuffed an empty message into the transmit buffer in case the host requested a response before one was ready. The host program might have to repeatedly query the terminal program to allow time for a response. The low-level serial routines on the Gameboy disabled external clocking when reading or writing to the register port. Sometimes, a serial glitch occurred or the Gameboy was powered up; either of these situations could throw the two programs out of step. When the terminal program received an invalid message, it flushed the input buffer and load the null message into the transmit register while waiting for a new valid message. The host resent the message if it received no valid responses. This series of retries and resends defined a time-out error condition and provided a method for resynchronizing the two programs.

After I completed the integration and the programs were all playing nicely with one another, I could load the master copy of the inventory database from the Windows host to the Gameboy. An operator using the system could now take the Gameboy to the stockroom and modify the count accordingly. If the operator needed to see more of an item description, he or she could scroll the description left and right to verify the description with the stock number. After completing the manual inventory, the operator could synchronize the master database. The system could recover from a broken communication. The project successfully demonstrated a realistic,

nongaming application on a Gameboy that used the connection of the proprietary game-link interface to a standard USB interface.

ISN'T IT OBSOLETE?

With the introduction of the Gameboy Advanced, why would anyone develop Gameboy Color applications? New Gameboy Color systems will not likely be available soon. The Gameboy Color system and development tools were more readily available than the advanced version when I started the project. Since then, the available resources for Gameboy Advanced development have significantly matured. Because the Gameboy Advanced executes legacy Gameboy software, little risk exists that the effort of this project would become obsolete. The game link of the two systems, however, is not identical. Advanced mode defines the reserved game-link pin, but it is unused and, thus, is transparent to this project.

If I began the project now, the Gameboy Advanced seems a clear choice of platform. It offers not only more mature tools, but also a 50% larger LCD, greater color depth, two extra buttons, 50% longer battery life, 50% smaller game cartridges, and a new UART interface in a thinner system that is only 77 mm^2 larger (**Reference 8**). The Gameboy Advanced includes a z80-like processor for color mode and an ARM/Thumb processor for advanced mode. The processors are mutually exclusive, and the presence or absence of a physical notch on the game cartridge determines their selection. For software-only projects, a proprietary MultiBoot cable allows you to test programs using the 256 kbytes of writable RAM that every Gameboy Advanced has without needing a program-cartridge programmer. You cannot test z80-based programs in this way.

Using the cartridge interface and custom hardware capability of the Gatesboy still works with a Gameboy Advanced because the system runs in the z80 mode, so a project using it to extend the hardware capabilities to include 802.11b or Bluetooth is still viable. To extend that capability using advanced-mode programs, though, you must change not only the pc board inside the Gatesboy, but also the casing with a physical notch to select the advanced mode in the Gameboy.

After working with this platform, I can think of many exciting ways to use handheld terminals to enhance the interface and control for many applications. The dual-processor architecture of the latest system is evidence of the commitment to support legacy resources spanning the last 12 years, and it provides better capabilities. The Gameboy, at the height of competition for handheld gaming systems, did not provide the highest performance or the largest color support. Instead, it provided just the right amount of performance, features, pricing, software selection, and battery life that best met the customer's needs. Interestingly, the new Gameboy Advanced adopted an ARM processor, not unlike many other handheld devices. A ubiquitous handheld architecture that many applications can share, that can maintain that kind of legacy support, and that can garner widespread adoption as an interface/control device will do much to decrease the cost and increase the reliability of those devices that consumers and businesses use every day.□

Acknowledgments

Special thanks to Michael DeVault of DevaSys for working closely with me and modifying the DevaSys USB I²C/I/O interface board to interact with the game-link interface. Also special thanks to David Nathan of Gatesboy for making the Gatesboy cartridge available.

References

1. Gameboy Developer's Kit (GBDK) homepage, http://gbdk.sourceforge.net.
2. Gameboy and Gameboy Advanced Development Web Rings, http://d.webring.com/hub?ring=gameboydev, http://c.webring.com/hub?ring=thegameboyadvanc.
3. Frohwein, Jeff, *Gameboy* and *Gameboy Advanced*, www.devrs.com.
4. Ziegler, Reiner, *Gameboy* and *Gameboy Advanced*, www.reinerziegler.de.
5. On-The-Go Supplement to the USB 2.0 Specification, www.usb.org.
6. *Gatesboy*, www.gatesboy.com.
7. Microsoft Visual Studio Technical FAQs, http://msdn.microsoft.com.
8. Nintendo, www.nintendo.com.
9. Wallich, Paul "Mindstorms: not just a kid's toy," *IEEE Spectrum*, September 2001, Volume 38, No. 9, www.spectrum.ieee.org/pubs/spectrum/0901/mind.html.
10. DevaSys drivers and application notes, www.devasys.com.

Author's biography

Technical Editor Robert Cravotta first cut his programming teeth on a z80 microprocessor similar to the one in the Gameboy Color. Some of the projects he completed with that processor were an assembler, a database, and a method for dynamically relocating code that accommodated ROM calls in a nonbanked architecture. You can reach him at 1-661-296-5096, fax 1-661-296-1087, e-mail rcravotta@cahners.com.

FOR MORE INFORMATION...

For more information on products such as those discussed in this article, go to www.ednmag.com and click on the Reader Service link under the Tools & Services section. When you contact any of the following manufacturers directly, please let them know you read about their products in *EDN*.

ARM
1-408-579-2200
www.arm.com
Enter No. 301

Cypress Semiconductor
1-408-943-2600
www.cypress.com
Enter No. 302

DevaSys Embedded Systems
1-716-377-9428
www.devasys.com
Enter No. 303

Gatesboy
+41-22-366-81-11
www.gatesboy.com
Enter No. 304

Lego
+45-79-50-60-70
www.lego.com
Enter No. 305

Microsoft
1-425-882-8080
www.microsoft.com
Enter No. 306

Nintendo
1-800-255-3700
www.nintendo.com
Enter No. 307

Singer Sewing Co
1-615-213-0880
www.singerco.com
Enter No. 308

STMicroelectronics
1-718-861-2650
www.st.com
Enter No. 309

Zilog
1-877-945-6427
www.zilog.com
Enter No. 310

SUPER INFO NUMBER
For more information on the products available from all of the vendors listed in this box, go to www.ednmag.com, click on the Reader Service link, and enter no. 311