CIRCUIT CELLAR®
THE MAGAZINE FOR COMPUTER APPLICATIONS

FEATURE ARTICLE

by Brian Millier

BasicCards 101 (Part 1)

Program Your First Smartcard

As opposed to credit cards, which have magnetic strips that store data, smartcards contain microcontrollers that store data and run programs. In this series of articles, Brian introduces you to smartcard technology and explains how you can program your own card using the BASIC programming language.

Although I have as many credit cards in my wallet as the next guy, I haven't had much exposure to so-called "smartcards," except for the smartcard that fits into the front of my TV satellite dish receiver. Apart from knowing that this is what prevents my receiver from delivering any programming until I actually subscribe for it, I haven't paid too much attention to it. I expect that the satellite-programming provider is hoping that no one is getting too familiar with its smartcard!

Conventional credit cards contain a magnetic strip that can hold data for numerous purposes. This strip is reasonably robust and easy to read, but it isn't totally secure. The data contained therein can be read by any card reader and duplicated. On the other hand, a smartcard contains an internal microcontroller that runs a program as soon as it's inserted in a smartcard socket.

The firmware running in the smartcard is an interpreter with a passive nature, which means that instead of initiating its own actions, it merely responds in a predetermined way to commands sent in from the outside world. Furthermore, like most modern microcontrollers, its program memory can be locked so that it can be neither examined nor modified by any external means. Smartcards generally contain encryption routines built in their firmware. The combination of the three aforementioned characteristics makes these devices particularly well suited for applications requiring high security. In the case of my satellite dish receiver, this high level of security is used to selectively unlock reception of the various channels that I pay for with my monthly subscription.

Because smartcards originated in the high-security arena of the electronics world, their internal workings have been kept pretty secret, and programming them hasn't been something that the average *Circuit Cellar* reader could expect to do. I recently came across a company that produces smartcards that can be programmed by anyone who uses its inexpensive development kit and knows BASIC . As an added bonus, the necessary development software may be downloaded for free, and the smartcards themselves can be purchased for a few dollars apiece in small quantities. In this article, I'll introduce you to these devices and describe a simple application that you can implement with them.

Photo 1—*Check out the development kit. Don't bother unwrapping the CD-ROM. You should download the up-to-date software directly from ZeitControl's web site.*

ZeitControl SMARTCARD

ZeitControl Cardsystems is a German company that produces smartcards and related products. To make the adoption of these devices as simple as possible, ZeitControl markets the inexpensive development kit ($59) shown in Photo 1. The kit includes a card reader, several BasicCards, and all the necessary software. The software and manuals (PDFs) are available for free on the company's web site if you want to check things out before you make a purchase.

The smartcards are available in different models to suit various applications. Table 1 lists selected models recently available through ZeitControl's online store. Although it isn't listed in Table 1, the least expensive compact card, with its 1 KB of flash memory EEPROM, is sufficient to store a small user program and a limited amount of nonvolatile data storage. More complex applications can be handled by the Enhanced and Professional models, which sport up to 32 KB of EEPROM as well as more RAM.

The card's operating system is a BASIC interpreter. Like Java, it executes P-code, but the BASIC P-code is different from Java's P-code. The Enhanced card's interpreter is contained in 17-KB ROM memory. The Professional cards use various amounts of flash memory for the interpreter, so they can be upgraded and customized by Zeit more easily as a result. Because P-code consists of short "tokens" representing commands and functions, a substantial user program can fit within the 8-KB EEPROM available in the larger Enhanced cards, for example.

Apart from the low-end compact card model, all the other models contain a fairly comprehensive implementation of BASIC. For instance, along with byte, integer, and long integer numeric data types, these cards also support IEEE floating-point numbers. A DOS-like file system is available, using the EEPROM as the storage media. String conversion utilities are also available.

Because of their intended use for commerce, all BasicCards come with encryption capability built into the operating system. This is an important consideration because encryption routines are complex, and most users (myself included) do not have the knowledge to program these functions from scratch. The Professional cards, in particular, contain public key encryption algorithms such as RSA and elliptical curve, as well as AES and SHA-1.

All BasicCards connect to the outside world via a rectangular array of eight metallic pads on the card. When the card is inserted into the reader, tiny fingers contact the five pads necessary for operation. To allow the cards to work reliably over time, the pads are plated with a precious metal coating that looks like gold. Figure 1 shows the pinout of the device, which follows the ISO standard for smartcards. Photo 2 shows the card itself.

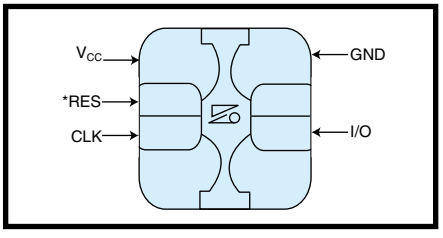

Figure 1—*Take a look at the connections to the BasicCard pads. Note that the larger ones are the power terminals.*

Model number	RAM	EEPROM	Encryption	FL. Point
ZC3.7 (Enhanced)	256 bytes	2 KB	DES	Y
ZC3.9 (Enhanced)	256 bytes	8 KB	DES	Y
ZC5.4 (Professional)	1 KB	16 KB	DES, AES	Y

Table 1—*Visit ZeitControl's on-line store to learn about these versions of the smartcard.*

In addition to the power and ground pads, which are a bit larger, there are three other signal lines: *Reset, I/O, and Clock. The *Reset line, when pulled low, resets the internal microcontroller. The Clock line provides a clock for the BasicCard's MCU. I don't know how they managed to fit such a powerful MCU into such a thin card, but they decided against fitting a quartz crystal in there, so you must provide the clock. Smartcards can operate with up to a 5-MHz clock. If you pick a 3.579-MHz clock, the BasicCard communicates at the standard communications rate of 9600 bps. The NTSC TV color burst frequency is 3.579 MHz. Because such crystals are extremely cheap, I suspect this influenced the BasicCard design.

The last line is the I/O line, which is used for bidirectional communication between the BasicCard and the host. Basically, it's an asynchronous 8-bit protocol with start and stop bits at TTL levels so you can interface it easily with the asynchronous port on your MCU.

The BasicCard is a slave device: it only transmits data when the host commands it to do so. The only exception to this is after the *Reset line is pulsed low. At that point, the BasicCard performs an answer-to-reset (ATR) procedure, which sends out a string of characters and identifies the device to the host. Therefore, however you choose to hook up the BasicCard to a host MCU, you must ensure that the MCU port connected to the I/O line is set up as an input immediately after a reset. Then, after reading the ATR string that's sent out by the BasicCard at reset, you can switch its direction to output and begin to issue commands to the card. Here, too, after a command is sent to the card, the port must be switched back to an input in preparation for receiving the response from the card. Although this can be done using the hardware UART found in most MCUs, it's probably easier to use a software UART routine and switch the data direction line of the MCU port pin from input to output as required.

All data transfers between the card and host are done using data packets, the exact format of which is specified by the ISO/IEC 7816-3 standard. This specification actually contains two standards: T=0 and T=1. The Enhanced cards use only the faster and more versatile T=1 block protocol. The Professional cards also support the T=0 protocol. The T=1 protocol includes longitudinal redundancy check (LRC) error checking. Chapter 7 of ZeitControl's BasicCard manual, which gives a detailed description of the ISO 7816-3 standard, contains enough information to allow you to write BasicCard drivers for the MCU of your choice. Next month, I'll describe the BCCARD library, which allows Atmel AVR MCUs running BASCOM compiler code to interface to BasicCards.

UNPACKING THE KIT

Now that you have an idea about what BasicCards are, let's take a closer look at ZeitControl's BasicCard development kit. Assuming you don't have any experience with smartcards, there are a few basic things you need to know to evaluate and program BasicCards.

First, you need a card reader for your computer. ZeitControl provides a Cybermouse PC/SC serial port reader, which works with both its software and any other software that expects a standard PC/SC reader.

Secondly, you need a few cards to experiment with. ZeitControl provides one ZC3.7 and two ZC3.9 Enhanced cards in addition to another card

Photo 2—*The BasicCard MCU is under the gold contact area.*

(unmarked as to its type), which contains a simple application already burnt in.

Thirdly, you need the software to program and test the cards themselves. Although ZeitControl provides a CD-ROM containing all of the software, it recommends that you download the latest version from the web site rather than using the CD-ROM itself. I found that the software on the web site was indeed much newer than the CD-ROM version, so I followed the company's advice. I'll discuss this software in a later section.

Finally, you'll need a manual to explain how everything works. The development kit comes with a detailed 172-page manual. The PDF version is also on the CD-ROM and web site.

The kit came with a little bonus—a tiny card reader/LCD in the form of a key fob. This device is programmed to display specific information contained in a BasicCard. When you write your own BasicCard application, you can include a command that is invoked when the card is inserted in this little reader. As an example, if you were using the BasicCard as an electronic debit card, you could program it to send the card's current balance to this reader. ZeitControl actually calls this device a "Balance Reader" because that's likely its principal use. If you pop the preprogrammed BasicCard (included in the kit as a demo) into the Balance Reader, it displays the series of numbers printed on the card itself.

Although the BasicCard manual is extremely detailed, I found it a bit too technical for a beginner unfamiliar with the BasicCard system. Therefore, I'd like to describe a procedure any novice can follow to become familiar with the system.

FIRST STEPS

I'm assuming that you are going to follow ZeitControl's advice and download the latest version of the software from its web site. When the installation is complete, a new folder titled "BasicCardPro" will be created below the C: root. You should plug the CyberMouse reader into a free COM port on your PC. Keep track of which port this is because you may need this detail later.

The CyberMouse gets its operating power by tapping into either the PS/2 keyboard or PS/2 mouse ports on the PC. Although there is a green LED on the Cybermouse, it isn't a power light; it merely flashes when the PC is accessing the reader. To ensure that the card reader is talking to your computer, go to the Tools folder and run the DOS program Scanrdr.exe. The program should find your reader and report which COM port it is connected to.

If you run this program with a card in place and request that it test the device, the program will do so. More importantly, it will inform you about what you must do to ensure that the various DOS/Windows development programs know which port the reader is attached to. This is important: the development software supports virtual card readers, and you want to be sure that you are actually talking to your CyberMouse when you are trying to access a real card.

The first thing I wanted to do was program an empty BasicCard with one of ZeitControl's sample programs. Then, I wanted to see if I could read it using the Balance Reader key fob. In the Examples/Pocket folder there is a sample program that basically duplicates the operation of the preprogrammed demo card (the card with numbers and strings printed on it). Run the compile.bat program. Watch the results as this program runs in the DOS window: it should show no errors and finish off with an indication of how much of the available EEPROM in the card was used.

This is where things went off the rails for me. I kept getting errors, and I could not proceed. I'll save you some frustration and point you to the card.prm file in the Examples folder. Open this with a text editor and go through the list of the BasicCard variants. All but one of the variants are commented out. Make sure the active one corresponds to the card you are using (a ZC3.7 or 3.9), and then edit the file to match your card. Also, make sure that the compile.bat file contains the following text:

```
..\..\zcmbasic -OI @..\card.prm
  prcard -I..\..\Inc
```

The compile.bat file, as installed from ZeitControl's web site, was different and did not generate an image file. The significance of all of these command line switches is explained in the manual—but at this early stage, you read the entire manual, have you?

Assuming that you have managed to compile the program successfully, you should see the prcard.img card-image file in the Pocket folder. This is the binary image of the code that must be downloaded to the BasicCard. Run the download.bat command, which is also found in the Pocket folder, and watch the DOS window for messages as the programming proceeds. The final message should be "SET STATE TEST." Now that you have programmed your first card, place it in the little Balance Reader. It should display the same series of strings and numbers that are printed on the preprogrammed test card.

BasicCard/PC APPLICATION

The next logical step is to load a BasicCard with an application that operates in conjunction with your PC/CyberMouse card reader. ZeitControl provides a couple of samples, including a debit card application and a calculator application.

The debit card application demonstrates, among other things, the encryption routines. The calculator application demonstrates the floating-point math routines in the BasicCard. In both cases, these applications split up the actual task between the PC host and the BasicCard. The calculator program is strictly a demo, and it doesn't serve a useful purpose in the real world. However, the debit card program actually performs substantially the same function as a real debit card would (i.e., the PC runs a program that initializes the BasicCard for a particular name/PIN number and then preloads it with the desired amount of credit). The BasicCard itself stores this information in its nonvolatile EEPROM memory, which is only readable and alterable when connected to a reader implementing the proper encryption routine/key value. Then, another PC (or POS terminal), which is running a "dealer" program, has the ability to subtract amounts from the remaining balance in the card (i.e., debit it) until the card is used up, so to speak.

To try this application, you should follow the same procedure outlined for the Balance Reader sample. However, you must open the Debit folder and run the compile.bat and download.bat programs.

The same two warnings I mentioned before also apply here, except that the change involving the card.prm file applies globally to all sample programs contained in the Examples folder. Chances are, you already have looked after this concern.

With the debit program downloaded on the card, you now have to run the part of the application that runs on the PC. ZeitControl calls all programs that run on the host "terminal" programs. I expect that comes from the concept that the BasicCard is running the guts of the program, and the PC is acting as a "terminal"—from the old days of computing, when the operator used a terminal and the mainframe computer was elsewhere.

As part of the compilation, Dealer.exe and Issuer.exe files are generated. You can run the DOS issuer program to initialize the card—or to personalize it, as ZeitControl calls the process. Assuming you have the newly programmed BasicCard still in the reader, you should be able to complete this procedure successfully. You can then run the Dealer.exe program to debit amounts from the card after you have correctly entered the PIN number.

Using either ZeitControl's zcpde.exe program, which is basically a Windows-based text editor, or any other text editor, you can examine or modify Dealer.bas or Issuer.bas to see exactly how ZeitControl's Basic language works. You'll note that these listings depend heavily on Include files for many definitions. I found it useful to print these common Include files because it made understanding the code somewhat easier.

You will notice that the interaction between the terminal program and the BasicCard program is done using commands. These command definitions contain the actual token that is sent by the terminal program and parsed by BasicCard's interpreter for any given command. Following the ISO 7816 standard, the tokens actually consist of two bytes: a class (CLA) byte and an instruction (INS) byte. Some tokens are reserved for a few predefined functions that exist in all BasicCards and shouldn't be used for your own functions.

In addition to specifying these tokens, the command definition also includes the exact format of any parameters that are passed between the terminal program and the BasicCard program. If you are familiar with the different ways of passing parameters to functions and subroutines (by reference, by value) you should note that these conventions are somewhat different in this unique programming environment. This is well documented in the manual.

As I alluded to before, the program running in the BasicCard is basically a group of functions that are triggered by the receipt of command tokens from the device to which it is connected. The BasicCard program likely also contains initialization code that runs when the card is inserted into a reader; however, by and large, it is a slave device. A BasicCard program therefore looks a bit strange in that it doesn't contain what you would normally call the "mainline" code. That part of the application resides in the terminal program running in the external device.

WRITE TERMINAL PROGRAMS

If your application is simple and intended to run on a PC, you might get by with writing the terminal part of your application using ZeitControl's ZCMBASIC compiler. This compiler is a DOS program with a command line interface. The BASIC source code can be created and modified using any text editor, or you can use ZeitControl's Windows-based ZCPDE professional development environment application. Whether or not you use ZCMBASIC to write the terminal portion of your program, you still have to use it to compile the code that is loaded into the BasicCard itself.

The ZCMBASIC compiler is simple, quick, and doesn't contain a lot of fancy features. The programs it produces are text-based; there is no provision for graphics. It supports the various encryption schemes used in the BasicCard itself, and the applications it produces consist of one small EXE file, which is easy to distribute.

Although I used ZCMBASIC to learn how to write a terminal program, it did not have enough features for my target application. Before getting too involved with the concepts of the BasicCard, I investigated the two other pieces in the

puzzle that were necessary for a useful BasicCard application: a Windows API to a high-level language and driver routines to interface the BasicCard to the AVR family of microcontrollers that I routinely use. The next two sections describe my experience with these two pieces of software.

PICK YOUR API

ZeitControl provides a number of application programming interfaces (API) to the commonly used high-level languages. They can be found on the development kit's CD-ROM (in the API folder), but I chose to download them from ZeitControl's web site, figuring I'd get the newest versions. There are libraries for Visual Basic, C, and Delphi. Although I've heard good things about Delphi, I must admit that I haven't tried it, nor am I proficient in C. I've used Visual Basic for years, so that's the BasicCard API that I picked for my application. Incidentally, the API routines need Visual Basic version 6 to operate.

In addition to the API and documentation, there are sample programs that allow you to become familiar with the API itself. Basically, the same examples that are provided using the ZCMBASIC compiler, which produces DOS programs, are duplicated in Visual Basic. If you are producing a commercial application, it's helpful to know that the necessary runtime files from ZeitControl's API library can be distributed with your application without any licensing fees. This includes the basic encryption routines. I'm not sure if the more esoteric encryption routines supported by the Professional BasicCards are included in this API.

I got off to a bad start with this aspect of the software. For some unknown reason, I decided to try out a different application than the ones I already had tested out using ZCMBASIC running in DOS. Basically, that meant compiling and downloading a new program into the BasicCard itself. Within the API folder, for each Visual Basic sample application, in addition to the Visual Basic files, there are compile.bat, download.bat, and source code files intended to produce the proper image file for the BasicCard. I already mentioned the tweaking I had to do to make those batch files work with the BasicCards provided in the development kit.

In this case, however, these batch files were completely wrong in that they refer to the compiler by the name it had in an earlier version. I did not expect to have outdated files because I had passed over my development kit's CD-ROM and downloaded the API files directly from the web site. To get around this problem, I ignored the API folder's files pertaining to the code meant to be downloaded to the BasicCard. This code, written for ZCMBASIC, is also contained in the BasicCardPro/Examples folder. But, in the latter case, the batch files are the correct version to match the current development software.

After that shaky start, getting the terminal part of the application to run under Visual Basic was not difficult. Although my earlier BasicCard sample programs ran easily from DOS with the CyberMouse card reader connected to COM1, it was a bit different in Visual Basic. Because Windows can support

many different card readers on different types of ports, you must first run an application called "API Select Card Reader" in order to inform Windows which port your reader is connected to. This application can be found on the Start Programs taskbar within the BasicCardPro group. To ZeitControl's credit, its example code contains a routine that checks to see what the default reader is, and it generates a user-friendly error message, informing you to run this utility program if no default reader is found.

It's a good thing that several sample Visual Basic applications are included. I have used Visual Basic for many years, but I'm not as familiar with the newer OCX Class libraries as I am with the earlier VBX controls. I don't mean to go off on a rant here, but a BasicCard sample program in ZCMBASIC, under DOS, is pretty easy to follow. In contrast, a functionally similar program written in Visual Basic would be extremely difficult for a newcomer to write from scratch without first studying some of the Visual Basic sample code. Of course, the same could be said for all modern Windows-based software.

So far, I haven't mentioned any of the debugging facilities contained in the development software suite. ZeitControl provides various Windows simulator programs for both the Terminal and matching BasicCard parts of an application. You can have a window open running a terminal program as well as another window that is simulating a virtual card in a virtual card reader. I have to admit that I did not make use of these simulators because I prefer to work with real devices running in real time. My application was not complex; but, if it were, the simulators would have probably come in handy.

BasicCard PROJECT

In the second part of this series, I'll describe a small project that I designed using an Enhanced BasicCard. The project consists of a board containing a BasicCard socket, an Atmel AVR MCU, and a keypad/LCD user interface. It controls the dispensing of liquid nitrogen (LN_2) from a liquid nitrogen generator at Dalhousie University, where I work.

Brian Millier is an instrumentation engineer in the Chemistry Department at Dalhousie University in Halifax, Canada. He also runs Computer Interface Consultants. You may reach him at brian.millier@dal.ca.

SOURCES

AVR Assembler and Simulator
Atmel Corp.
(714) 282-8080
www.atmel.com

90S8535 Development board
Futurlec
www.futurlec.com

BASCOM-AVR Compiler/programmer, BasicCard driver library
MCS Electronics (Holland)
+31 75 6148799
www.mcselec.com

BasicCard
ZeitControl Cardsystems
+49 0 571-50522-0
www.zeitcontrol.de