

BasicCards 101 (Part 2)

Use in a Liquid Nitrogen Monitor

Now that you're familiar with ZeitControl's BasicCard, it's time to take a closer look at the process of programming one and incorporating it in a design. Brian shows you how he integrated BasicCard technology in the design of a liquid nitrogen generator.

Last month, I introduced you to ZeitControl's BasicCard, which is a smartcard that is programmed in BASIC. In Part 2, I'd like to further explore the programming of these cards and discuss a project.

I built a device that allows you to use BasicCards like debit cards to control the dispensing of liquid from a liquid nitrogen generator/storage tank. The idea is to issue each liquid nitrogen user here at Dalhousie University with a BasicCard. Each card is personalized by entering the user's name, account number, and a zero balance. Card personalization is done on a PC using a Windows-based application. This user-friendly application interfaces to the BasicCard using the CyberMouse reader that comes with the development kit.

To access some liquid nitrogen (LN₂), insert your BasicCard into the custom controller connected to the LN₂ generator. The controller checks to ensure that the card is properly personalized for this use, and displays your name, account number, and the number of liters of LN₂ previously consumed during the current billing period. You then enter the amount of LN₂ desired on the controller's keypad. After updating the BasicCard EEPROM variable, which stores the accumulated LN₂ usage, the controller activates a relay that opens a valve and lets the

liquid nitrogen flow for a long enough period to dispense the necessary amount of LN₂.

Periodically, the BasicCards are collected and another PC application is run that zeroes out the accumulated LN₂ total. It transfers that figure to a table that is used for the actual billing process.

In the past, an honor system was used in which users entered their LN₂ usage on a sign-out sheet posted next to the generator. We lost some LN₂ billings here at the university because of dishonesty and low-ball estimates of the actual amounts taken. Also, there was a significant amount of clerical time spent tallying all of the entries on the daily sign-up sheets.

CIRCUIT DETAILS

Because the prototype would be the only unit I would need, I wanted to use a commercial development board and just add the few custom parts needed for the design. I chose the Futurlec 8535 development PCB because it's inexpensive, contains headers for commonly used peripherals, and has a large enough prototype area to mount the few extra components I needed to complete the design.

Figure 1 is a diagram of the circuit. Some of the circuitry on the Futurlec board that is not required for this project is not shown in the diagram. For example, the in-circuit serial programming port for the 8535's flash program memory is not shown. There is one small

change that I had to make to the Futurlec board itself. The board comes fitted with an 8-MHz MCU clock crystal. I changed that to 3.579 MHz so that I could use a common clock for both the MCU and BasicCard, which has a 5-MHz maximum clock rate and communicates at 9600 bps when clocked at 3.579 MHz.

I mounted an Amphenol C702 10M0008 2834 smartcard socket in the prototype area of the PCB. This socket contains an NC switch that opens up when a card is inserted, making it easy for the MCU to know when to establish communications with the card.



Photo 1—Check out the monitor before it's mounted in its cabinet. Note the BasicCard sticking out of the card socket on the right-hand side.

Although I connected this line to port D3 of the '8535 (INT1), I don't use this pin as an interrupt input, instead I poll the pin to see when a card is either inserted or removed. There are a few different phases in the program, and the removal of a card must be handled differently in each case, so using an interrupt to signal card-in-socket status was not ideal for this project.

The BasicCard interfaces to the '8535 using only two port lines. The *RESET input to the BasicCard is connected to port D2, and the I/O line connects to port D4. As I mentioned earlier, the BasicCard gets its clock signal directly from the '8535's XTAL1 pin, which is its oscillator buffer output. Although you have to be careful tapping off a clock signal from an MCU's oscillator output, the BasicCard presents, in this case, such a small load that the oscillator is unaffected by its presence.

The user interface is the standard keypad/LCD. I had a Grayhill series 88 keypad on hand, which contains the numbers in a telephone-like arrangement, as well as several extra keys labeled Enter, Clear, etc. that provide all the necessary input functions. Eight lines are needed to interface this Matrix keypad, and I used port A for that purpose. Doing so meant I could not make use of the eight-channel ADC contained in the '8535, but I didn't need an ADC for this project. The Futurlec board has a 10-pin header connected to this port, which made it easy to connect the keypad using a ribbon cable.

I didn't need a large LCD, because there isn't a lot of information to display. Therefore, I used a 2 x 16 display, and connected it to the '8535 in 4-bit mode using six lines of port C. The Futurlec board also provides a 14-pin header for the LCD, with all the signal and power wiring taken care of, as well as POT1 for contrast adjustment.

Apart from wiring a few lines to the smartcard socket, all the circuitry I needed to add was the driver and relay to

activate the dispensing valve. This was a solenoid valve that required 120 VAC to operate, so I decided to use a relay to keep any inductive kickback from its coil away from the MCU circuitry. Any relay with a 12-V coil and contacts able to switch 1-A AC will do for K1. If the coil needs more than a few hundred milliamps to operate, you'll need a larger transistor than a 2N3904 for Q1.

Diode D1 is placed across the relay coil to protect Q1 from the spike that occurs when the relay turns off.

The 5-V power supply regulator is contained on the Futurlec PCB, but the power transformer, bridge, and filter capacitor are mounted externally. Photo 1 shows the development board with the extra components needed for this project.

The development board comes with a MAX232 for RS-232 communications purposes. Although I did not use this feature, it would be easy to wire the controller to a remote PC, for example, if you want to log all the transactions in real time to a computer file.

Atmel's AT90S8535 is a versatile MCU. In some respects it is overkill for this application. With the program written in compiled BASIC, much of its 8-KB flash program memory is unused. It contains 512 bytes of EEPROM, which is unused apart from a few bytes used to store some valve time calibration parameters. The eight-channel ADC, UART, and SPI functions weren't needed either. However, the assembled Futurlec development board, including the '8535 MCU and programmer cable, cost less than \$30, so I thought it made a lot of sense to do things this way.

INTERFACE TO THE AVR MCU

Before I ordered the BasicCard development kit, I wanted to be sure I could use the cards with my MCU of choice, the AVR series from Atmel. The BasicCard development kit contains software to program the cards themselves, as well as a few different programs to develop PC applications. However, the kit doesn't provide any

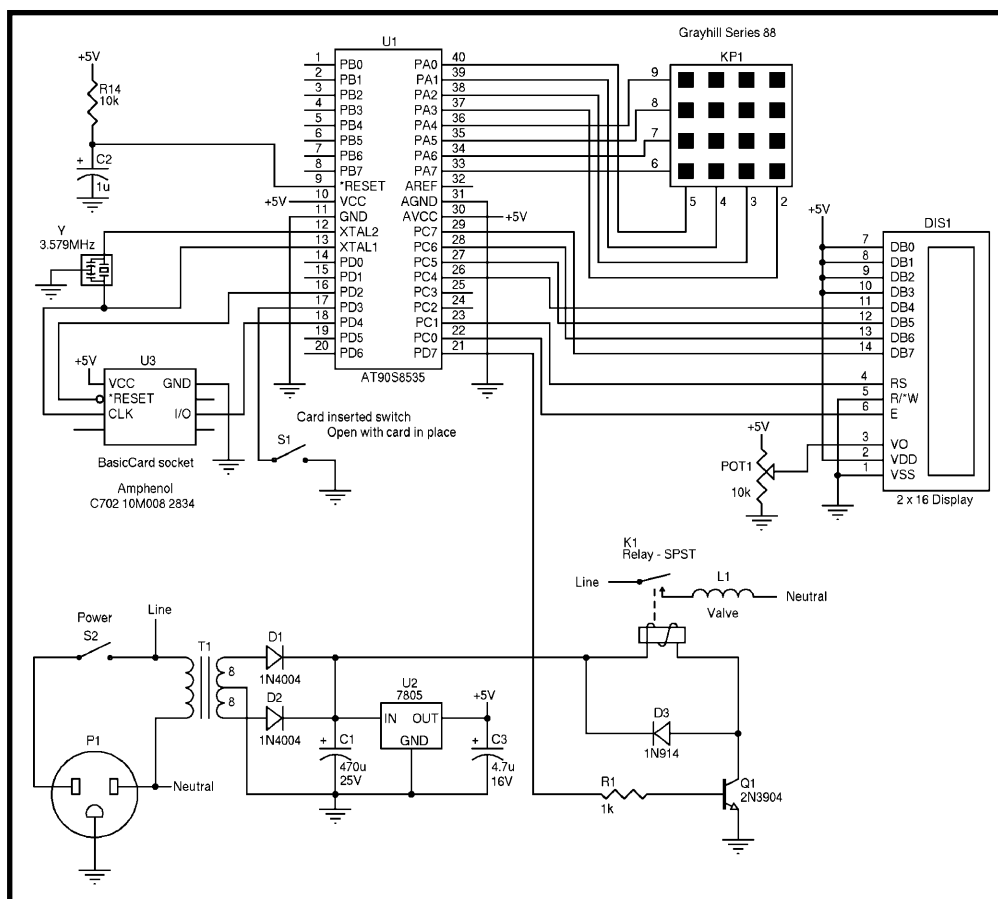


Figure 1—The liquid nitrogen dispensing monitor was built on a Futurlec 8535 development board. Apart from the card socket, keypad/LCD, and relay driver, most of the circuitry already exists on the development board.

Command Name	Description	Used by
PersonalizeCard	Initial card with name, account number, and balance	PC
IncreaseAmount	Increase balance by amount dispensed	AVR
GetCardData	Read name, account number, and balance	AVR and PC
CancelLastTransaction	Self-explanatory	AVR
ReadLastTransaction	Needed by previous command	AVR
PRDisplay	Display information on key fob balance reader	Balance reader

Table 1—These six commands are implemented in the BasicCard for this application. As you can see, some commands are used by both the LN₂ dispensing controller and the PC application that personalizes and reads the cards.

support for interfacing them to MCUs, apart from providing a technical reference document describing the ISO/IEC 7816-3 standard that the cards use for communication. From what I could see, it would have been quite time-consuming to personally write a suitable driver to allow AVR MCUs to talk to BasicCards. As luck would have it, the developer of the BASCOM AVR compiler (the language I use exclusively for all of my AVR-based projects) had a beta version of a driver library available. This was one of those good news/bad news situations. I was able to get the library free from the developer, but I was, at that time, the first and only beta tester! I ended up figuring out on my own some of the patches needed to make it work.

To interface a BasicCard to an AVR MCU using the BASCOM compiler, you must run BASCOM version 1.11.6.8 or newer. You must also load the BasicCard library (available from MCS) into BASCOM's LIB folder.

Next, within your BASCOM program, you must do the following four things. First, use the CONFIG BCCARD command to tell the driver which port pins you have connected your BasicCard socket to. This is simple because you only have to tell it which port you are using and to which pins the *RESET and I/O lines are connected. This is clearly described in the documentation that comes with the library.

Second, you must declare each of your BasicCard commands to the BASCOM program using the BCDEF command. Because you have already written the BASIC code that runs in the BasicCard, you will have already defined the commands that the BasicCard recognizes. These command names, followed by their parameter lists, are used with the BCDEF command.

Next, instruct the compiler to use

the proper data rate. Use the \$BAUD = 9600 and the \$Crystal = 3579000 directives. Finally, issue a BCRESET command to initialize the BasicCard as soon as you see that a card has been inserted into the card socket.

After this preparation/initialization, all you have to do to access a particular command in the BasicCard is use the BCCALL procedure. This is similar to any BASIC procedure, except that it includes some BasicCard-specific parameters. The following is the syntax for this command:

```
BCCALL name( nad , cla , ins , p1 ,
            p2 [param1 , paramn])
```

where name is the name of the procedure in the BasicCard to call. It must be defined first with BCDEF. The name used with BCDEF and BCCALL does not need to be the same as the procedure in the BasicCard, because the CLA and INS bytes actually tell the BasicCard which command to execute. However, it makes sense to use the same names for consistency.

nad is the node address byte. BasicCards respond to all node address values; zero is used here as a default. cla is the class byte. It is the first of a 2-byte CLA-INS command. It must match the value you used for the command in the BasicCard program itself.

ins is the instruction byte. It is the second of the 2-byte CLA-INS command. The same consideration applies as for the CLA byte. p1 is parameter 1 of the CLA-INS header. (Use a zero for your purposes.) p2 is parameter 2 of CLA-INS header. (Again, use a zero for your purposes.) param1 through paramn are the parameters you want to pass to the BasicCard (as required by the command).

The BasicCard operating system

employs a comprehensive error-reporting scheme. This is necessary in any device used for commerce, particularly in a device that you can pull out of its socket in the middle of program execution! Describing the error handling is beyond the scope of this article, but I will mention that the BCCARD library supports it to some extent. In essence, if the BasicCard returns an error, the library will set the BASIC variable ERR and two status bytes—SW1 and SW2—will contain error codes. Many of these codes are predefined in the BasicCard protocol, but you can also set the value of these variables yourself within your BasicCard program should your code encounter an error condition.

In most designs, the card-in-socket switch will notify the program if the card is removed during use; but, if an actual data transaction is in progress when this happens, the BasicCard driver within BASCOM may hang the program until a time-out interval passes. Depending on a number of variables, this might take up to a minute, so beware of this during debugging.

Before trying your own Basic-Card program, I recommend that you load the sample Bccard.bas program into your AVR MCU, program a BasicCard with the Calc demo program provided with the development kit, and try that combination. If everything is satisfactory, you will see the results of the communication between the two devices. You can then enter some numbers into your AVR MCU, and let the BasicCard act as a calculator.

CARD FIRMWARE

Table 1 shows a list of the commands that I've defined in the BasicCard for this application. The last column in this table indicates where the command is used: The LN₂ Dispenser is labeled "AVR." The PC application personalizes the card and later reads the accumulated total usage (labeled "PC"). The balance reader is the little key fob device that comes with the development kit.

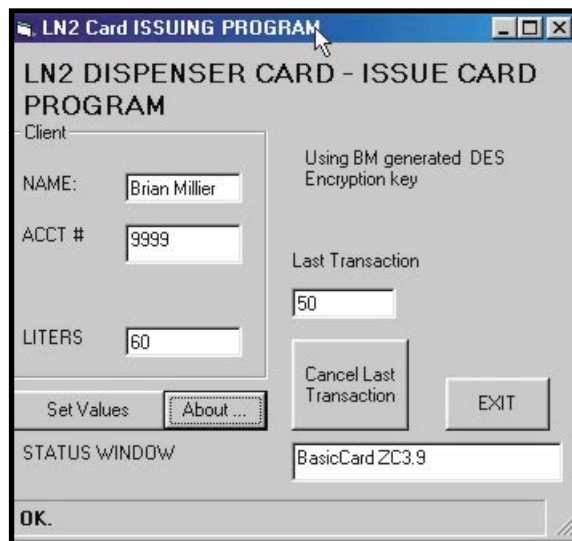


Photo 2—This is a screen shot of the PC application that initially personalizes the BasicCard and reads out its balance periodically.

The PersonalizeCard command, which the PC application uses to initialize the card with user information and zero out the balance, is the only command that requires encryption in order to work. This guards against the possibility of fraudulent card production. ZeitControl's Visual Basic API, which is part of the development kit software, includes support for encryption. This command is only used by the PC program that personalizes the cards, and that is written in Visual Basic. Although not a concern in this application, the fact that the BASCOM AVR BasicCard library does not currently support encryption could be a disadvantage in other applications.

For this application I used the BasicCard ZC3.9 enhanced card, because it was included with the development kit. However, the lesser models of the card would work as well because this is a simple application. In the first part of this series I gave you some hints on how to program the BasicCards by supplementing the instructions in the user manual.

AVR FIRMWARE

Next, I'll describe the program that runs on the '8535, which controls the LN₂ dispenser. After some initialization of the ports, the program basically enters a loop. The program prompts you to insert the card and then waits for this to happen. The ReadCardData command is then issued to the

BasicCard, and the user information contained in the card is displayed on the LCD. You are then prompted to enter the amount of LN₂ needed. This amount is first added to the BasicCard's balance variable, and then the program waits until you indicate you're ready for the dispensing to start. It then activates the relay for the amount of time necessary to dispense the necessary LN₂.

This is where calibration is needed. When dispensing LN₂, it takes a little while for the transfer tube to become chilled enough to pass the liquid. Thereafter, the liquid flows steadily, with the amount dis-

persed being proportional to the time the valve remains open. After this amount of time, the program waits until the card is removed, and then goes back to the start of the loop. Calibration involves determining the initial chill time and the time/liter parameters empirically, and storing them as calibration parameters.

It's important that this calibration can be done in a way that's both convenient for the operator of the generator and secure from tampering. To that end, I designed the program so that when it sees a card with an account number equal to zero (the operator account), it goes into a calibration routine and prompts the operator for an initial (transfer tube chilling) time, as well as the number of seconds required to transfer 1 liter of LN₂. These parameters are subsequently saved to EEPROM in the '8535 and used in all future liquid transfers, unless the calibration is changed.

If you remove the card before entering the desired number of liters to dispense, the LCD indicates that the card has been removed prematurely and returns to the start of the loop.

CARD ISSUER PROGRAM

Photo 2 shows a screen capture of the Visual Basic Card Issuer application. This program is run on a secure PC because it is used to personalize the BasicCard with the username and account number, and to set the initial

balance variable to zero. Obviously, anyone with access to this program can insert their card and wipe out its balance easily.

After start-up, the program waits until a card is inserted. It checks that the card has the proper program in it for this application. It does this by reading the `applicationID` variable contained in the card's EEPROM. If it matches, the program displays the user information and the current balance. If it's a new card that's been programmed with the LN₂ application but not yet personalized, it will show blank fields and allow you to enter the applicable name/account information and also a balance amount (typically zero for a new account).

There is also a provision to cancel the last transaction. The amount of the last transaction is shown, and there is a button to subtract this amount from the accumulated total. The purpose of this function is to allow the operator of the machine to cancel a transaction if the machine fails to deliver LN₂ as requested (i.e., the tank is empty or a valve sticks).


I also included a box that displays the type of card that has been inserted into the reader. This card-specific information is returned by all smart-cards as part of the answer-to-reset (ATR) routine, which is invoked at card insertion.

GIVE IT A TRY

I must admit that I was intrigued by the fact that these tiny BasicCards contain more resources (except for I/O) than the typical MCUs I routinely use. Furthermore, all of this capability is sandwiched into the small area under the gold contact pad. The rest of the card is just plastic filler to make the card easy to handle.

I am definitely in the compiler camp, and generally avoid MCUs like BASIC Stamps, which run as interpreters. However, for this purpose, I have to admit that the BasicCard interpreter design is quite useful.

I suspect that many readers, like myself, are not quite up to the task of designing their own device drivers for high-level PC languages like Visual Basic running on Windows. It was a

different story in the past, when you could easily design an ISA card, plunk it into the PC, and access it using direct in and out instructions. Now, it's important that the device manufacturer supplies APIs that allow these compilers to easily interface to the device. ZeitControl certainly has taken care of this aspect, and it provides an extremely inexpensive development kit. For BASCOM AVR users, the necessary BasicCard drivers exist as well. So, there's really no reason not to give this a try if you have an application in need of a secure smartcard. 

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.

PROJECT FILES

To download the code, go to ftp.circuitcellar.com/pub/Circuit_Cellar/2004/165.

SOURCES

C702 10M008 2834 Acceptor
Amphenol-Tuchel
(734) 451-6400
www.amphenol-tuchel.com

AT90S8535 Microcontroller, AVR assembler and simulator
Atmel Corp.
(714) 282-8080
www.atmel.com

90S8535 Development board
Futurlec
www.futurlec.com

Series 88 keypad
Grayhill, Inc.
(408) 354-1040
www.grayhill.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