BasicCard goes contactless

A discreet alternative

By Patrick Gueulle (France)

The only chip card that you can program in BASIC has now been on the market for more than ten years. It remains under continuous development: in a new twist, this well-known card with an open operating system is now available in an RFID version. As well as the new facilities for contactless operation, very powerful (and free!) development tools are now available to provide an easy way to get to grips with this fascinating technology.

The philosophy behind the product has not changed since the first 'Compact' BasicCard appeared in 1998: put into the hands of developers (which includes interested enthusiasts) the means to develop their own applications independently of the mass-market chip card manufacturers. The result is a complete range of asynchronous cards using Flash technology, with products available in small quantities and even individually. With the help of a (re-)programmable virtual machine, which in some versions was even capable of supporting several applications simultaneously, there is support for a high-level language simpler, but no less powerful, than Java: ZCBasic (**Figure 1**).

A complete development environment (compiler, simulator and double-debugger, along with a manual running to some 250 pages) is available for free download at [1].

With just a couple of lines of source code it is possible to make a BasicCard compatible with just about any terminal, including for example a mobile phone. This existing know-how can now be transferred to contactless applications using the ZC7.5 RFID version of the card. The transition is also simplified by the use of the ZC7.5 Combi card, which offers two interfaces: one over contacts (transport layer protocol T=0 or T=1) and one contactless (ISO 14443 type A T=CL). An ACR122 or Omnikey 5321 makes a suitable RFID reader for the contactless interface.

Card applications

Twenty or so lines of code (RFIDspy.BAS [2]) suffice to demonstrate the flexibility of the ZC7.5 Combi card. At under 400 bytes of P-code the program occupies just a tiny fraction of the 32 kbyte EEPROM space in the powerful IC. This short program is a T=CL version of the logger that we presented for the first T=0 BasicCard with contacts in the May 2002 issue of *Elektor* [3] (for card version ZC4.1). The program can store and subsequently dump the commands used by a reader as part of its dialogue with a card presented to it. Depending on the reader, this 'impostor' will either be rejected almost instantly or be accepted (for a while at least) by the reader as a genuine card intended for use with it.

The underlying idea, of course, is to use this code as a basis for incremental extension: as more and more of the commands emitted by the reading device are understood, code can be written to deal more precisely with them, thus better emulating a genuine card. There is a certain amount of detective work to be done in unmasking the security mechanisms used in the design and in determining their strengths and weaknesses. In summary, we have a very handy experimental tool.

Skipping over the first three lines of the source code, which are just preparatory declarations, we come to a series of '#Pragma' directives, of which the first two are specific to operation in con-

tactless mode. The first thing to know is that most contactless objects (cards and tags) have a unique number (or UID), several bytes long, which is written into ROM during manufacture. This is occasionally used as part of a defence against cloning, but its main use is during the anti-collision process used by the reader to communicate with a single specific card when more than one is within its range. The details of this process are relatively complex, but with luck card and reader will handle it all themselves: unless you really want to get his hands dirty, as an applications programmer you need not get involved. However, you can specify the number of bytes that will be used to form the UID in order to match the characteristics of another card as closely as possible. The ZC7.5 supports three standard variants: 'single' (four bytes, like MIFARE Classic), 'double' (seven bytes, like MIFARE Ultralight), and 'triple' (ten bytes). It is also possible to replace the fixed UID in the chip with a random value to help the owner of a card to avoid being tracked. In our example the UID is sent out as a group of four random bytes when a reader starts polling:

```
#Pragma UID(Random,Single)
```

At the beginning of communication begins between reader and card, the reader selects it and waits for its reply (ATS for 'answer to select'), which is comparable to ATR ('answer to reset') in the case of a card with contacts.

```
#Pragma ATS(TA1=0,FWI=7,TC1=0,HB="EMVA")
```

This second command allows the default communications parameters to be modified, either partially or completely, in order to optimise compatibility with a particular reader. In this example we choose compatibility with the 'EMV Contactless Specifications' (which are publicly available: [4]). These specifications ensure compatibility for electronic payments between chip cards and terminals that bear a special logo indicating that they comply with them (**Figure 2**).

In a similar way we can modify the 'ATR' parameters of the card, which affect its communications over the contact interface when it is connected to a suitable reader:

```
#Pragma ATR(Direct,T=1,HB="RFIDspy")
```

In this case we specify use of the T=1 protocol and select 'direct convention' for communications; we could equally well have used the T=0 protocol and/or 'inverse convention'. We will now look at what happens in the card when it is selected by the reader. The BasicCard has an internal file system, similar to MS-DOS. Opening a file called, for example, 'Card.Log', can be done as follows:

```
Open "Card.Log" For Append As #1
```

To allow this file to be deleted using an external instruction, it is convenient to define a special command for the purpose. Here we

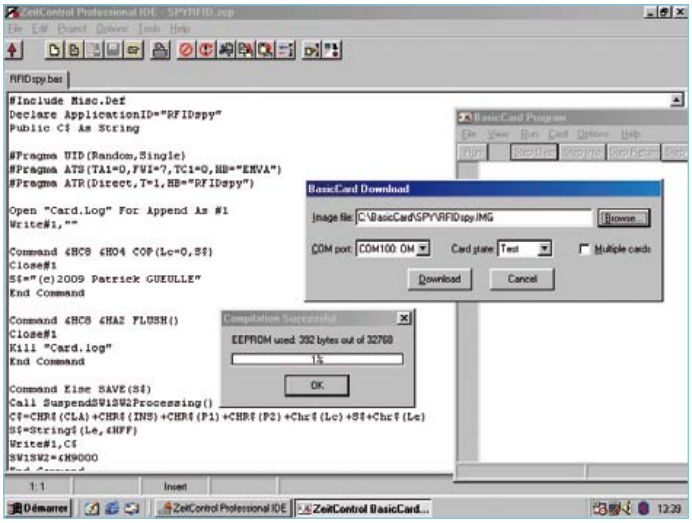

Figure 1. The ZeitControl BasicCard development environment in action.

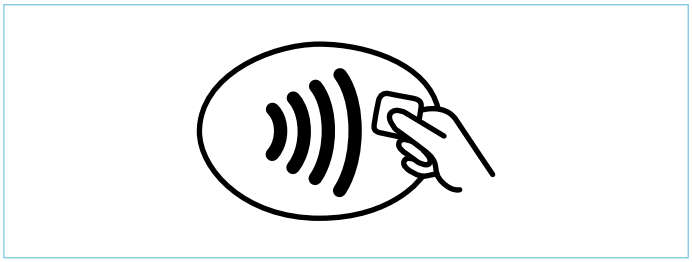

Figure 2. The *EMV contactless* logo is found on an increasing number of point-of-sale terminals.

have called the command 'FLUSH':

```
Command &HC8 &HA2 FLUSH()
Close
Kill "Card.Log"
End Command
```

Now, if we send the byte sequence `C8 A2 00 00 00` to the card it will delete the file. The main part of the program is found in the next seven lines:

```
Command Else SAVE(S$)
Call SuspendSW1SW2Processing()
C$=CHR$(CLA)+CHR$(INS)+CHR$(P1)+CHR$(P2)+Chr$(Lc)+Chr$(Le)+S$
S$=String$(Le,&HFF)
Write#1,C$
SW1SW2=&H9000
End Command
```

That is all that is needed to trap any unrecognised command (hence the 'Else') received by the card and store it in the file Card.Log along with the parameters CLA, INS, P1, P2, Lc, Le and any data received from the terminal ('incoming' commands).

For the outgoing message the card delivers by default a number of FFh bytes equal to the value of Le (the expected data length). A different reply can be constructed if required by changing the contents of S$ as required. The status bytes SW1 and SW2 can also be changed from their default values of 90 00 depending on the desired effect on the reader.

Terminal application

Having collected some information in the file Card.Log we will want to read it from the card for further analysis. The file is normally left open so that data from several consecutive sessions can be logged, and so the first thing the program RFIDutil.BAS has to do is send the command C8 04 00 00 00, which closes the file as follows:

```
Command &HC8 &H04 COP(Lc=0,S$)
Close#1
S$="(c)2009 Patrick GUEULLE"
End Command
```

Just two corresponding lines are required in the source code for the terminal:

```
Declare Command &HC8 &H04 COP(S$,Le=&H17)
Call COP(S$)
```

Recovering the contents of the file is equally straightforward. In the terminal code we add the prefix '@:' to the filename, and read the file as normal. The operating system generates all the necessary commands automatically:

```
Open"@:card.log" For Input As #1
```

The following instruction is then used to extract one by one the commands for which the file contains the reply information:

```
Input#1, Z$
```

The rest of the terminal code is concerned with converting the contents of the data file into readable text, storing it on the hard disk and displaying it on the screen.

A practical example

Once you have had a look at the manual you can decide whether you prefer to use the development environment, which is well suited to organising projects, or to drive the ZCMBasic compiler from the command line. The result of compilation is a file RFIDutil.EXE, which can be run directly from the Windows command line, and a file RFIDspy.IMG (or RFIDspy.DBG), which has to be loaded into the memory on the card. With the card thus prepared, all you need to do is bring it within range of the terminal whose characteristics you are investigating.

The author tested the card at the point-of-sale terminal at the checkout in a French supermarket. The supermarket accepts payments of up to 20 Euro using contactless EMV cards such as MasterCard PayPass or Visa payWave. The BasicCard was brought within range of the terminal immediately before the real payment was carried out using a conventional bank card. Subsequent analysis of the file Card.Log revealed a sequence of select commands resembling the following:

```
00 A4 04 00 0E 32 50 41 59 2E 53 59 53 2E 44 44 46 30 31
00 A4 04 00 07 A0 00 00 00 04 30 60
00 A4 04 00 07 A0 00 00 00 04 10 10
00 A4 04 00 07 A0 00 00 00 04 99 99
00 A4 04 00 07 A0 00 00 00 03 20 10
00 A4 04 00 07 A0 00 00 00 03 10 10
00 A4 04 00 07 A0 00 00 00 43 10 10
00 A4 04 00 07 A0 00 00 00 42 10 10
```

The initial 00 stands for the ISO class (CLA) of the command, and A4 for the opcode (INS). Then 04 00 give the parameters P1P2, followed by a length indicator byte (Lc) and the application identifier (AID). We have discarded the null byte at the end of each line as it is not of any interest for analysis and only serves to indicate that the command does not expect a reply (Le = 0).

The first line represents an attempt to select the 'PPSE' (Proximity Payment System Environment) with the identifier, transmitted in ASCII, 2PAY.SYS.DDF01. This is exactly analogous to PSE in the case of EMV cards with contacts, which use the identifier 1PAY.SYS.DDF01 [5]. Our card replies with invalid data and so the terminal deduces that the PPSE, which at this point would normally supply a list of applications supported by the card, is not available. The terminal then proceeds to attempt to select in turn all the applications which it supports in the hope of finding one which also recognised by the card.

The next two lines show the terminal attempting to select two MasterCard applications, with priority given to Maestro (A0 00 00 00 04 30 60), a card which requires systematic (online) authorisation.

Before the terminal goes on to attempt to select the Visa Electron (A0 00 00 00 03 20 10) and Visa credit/debit (A0 00 00 00 03 10 10) applications, there is an attempt to select the mysterious application A0 00 00 00 04 99 99. This might correspond to a supermarket loyalty card: the contactless reader is apparently capable of dealing with these as well as with bank cards. The final selection attempt is for the French Carte Bleue credit card (A0 00 00 00 42 10 10).

What's in the kit?

The BasicCard is developed by ZeitControl, a small business based in Germany that evolved from being a vendor of time tracking systems into a specialist in chip cards. The first BasicCard was produced in 1996.

Compared to earlier BasicCard kits the 'dual interface' version features the addition of an RFID prototyping board carrying the TagTracer 14443 with a USB connection, buzzer, LED indicators and a printed antenna. These make developing applications much easier. In the author's opinion this development kit is a distinctive product, and is considerably more flexible than other similar units on the market.

- Omnikey 5321 USB – dual interface PC/SC smart card reader/writer
- Pocket card reader (balance checker)
- Development PCB for contactless ISO 14443 USB reader/writer
- Software development kit (SDK) for Windows
- Documentation on CD-ROM
- Printed technical manual (250 pages)
- 4 off BasicCard ZC7.5 Combi (32 kbyte EEPROM)

Further information is available at www.basiccard.com

On the other hand, a terminal that only accepts cards with contacts (such as a public telephone or petrol pump) might attempt to select applications in the following sequence:

```
00 A4 04 00 07 A0 00 00 00 42 10 10
00 A4 04 00 07 A0 00 00 00 42 20 10
00 A4 04 00 0E 31 50 41 59 2E 53 59 53 2E 44 44 46 30 31
00 A4 04 00 07 A0 00 00 00 03 10 10
00 A4 04 00 07 A0 00 00 00 03 20 10
00 A4 04 00 07 A0 00 00 00 04 10 10
00 A4 04 00 07 A0 00 00 00 04 30 60
```

Here we see that the terminal attempts to select French bank cards before the PSE. Only after that does it attempt to select international applications. In both cases the terminal's strategy is designed to make the transaction as quick as possible, which is especially critical in contactless applications.
One will sometimes encounter an attempt to select the Moneo application (`00 A4 04 00 06 A0 00 00 00 69 00`): this is an electronic wallet that is available in versions with and without contacts. Identifiers more than ten bytes long betray the existence of a card 'co-branded' with one or more commercial suppliers.
The author's next step is to experiment with contactless bank cards outside his home country. Contactless payment systems are being rolled out in many European countries including the UK, and it is expected that the system will be in widespread use for small payments within the next few years.

(090378)

Internet Links

[1] www.basiccard.com
[2] www.elektor.com/090378
[3] www.elektor.com/010138
[4] www.emvco.com