Geoff's Compact Flash Access Guide

Introduction

So, you've built your adapter and now you want to see if it works. Well first an explanation of the how the card interface works.

The card adapter is hard wired to work in Memory Mapped mode with the 8 registers mapped onto ports &80 to &87. In fact when address bit A7 is set during any I/O port request the card will be active, this simple decoding also makes the card show up on port &90, &A0..etc. so be aware of this if you plan to use higher port numbers for other applications. The data in the table below provides an easy way to see how the ports relevant to the CF card relate to their offset from the base port address of &80. (More on registers later.)

A7...	A2	A1	A0	Port	READING	WRITING
1	0	0	0	&80	RD Data	WR Data
1	0	0	1	+1	Error	Features
1	0	1	0	+2	Sector Count	Sector Count
1	0	1	1	+3	Sector No.	Sector No.
1	1	0	0	+4	Cylinder Low	Cylinder Low
1	1	0	1	+5	Cylinder High	Cylinder High
1	1	1	0	+6	Select Head	Select Head
1	1	1	1	+7	Status	Command

So the BASIC command A=INP(&87) would read data from the Status register. Port &80 is used for data transfer, both reading and writing, and commands are sent out on register 7 (at port &87). One thing to be aware of is that the 8 bit data is transferred in two byte words with the low byte transferred first. The result of this is that when reading or writing to the card odd and even bytes need to be reversed in order (i.e. 2,1,4,3,6,5,8,7...etc.). This seems complicated but is very easy to deal with. It is a simple result of the card being read in 8 bit mode not 16 bit mode when the high 8 bits would appear on port &81 in parallel with low 8 bits on port &80.

Testing The Adapter

Here's a quick program to test the card to see if it's ready. It asks for input from port &87 (the status register) then compares the data with what is expected from a card that is ready for commands.

```
                    Program 1
10 A=INP(&87)
20 IF A=80 THEN PRINT "Card Ready!"
30 PRINT "Status bits - ";BIN$(A,8)
```

If the card is working you'll get the "Card Ready" message followed by a binary representation of the status bits that were sent by the card. If the card isn't ready you'll just get the binary status data.

Status Bits

7	6	5	4	3	2	1	0
Busy	Ready	Write Fault	Data Seek Complete	Data Request	Correctable Error	0	Error

As you can see reading port &87 and comparing the data to 80 in decimal is the same as checking for 01010000 in binary. This checks for bit 6, the ready bit, and bit 4, the data scan bit, to both be 1 and all other bits to be 0.

There is a special case for the Tatung Einstein; if the status register responds 01111000 or 01111001 then you've probably forgotten to plug the card into the adapter.

Accessing the card

The first thing to do now you know the card is ready is to try the "Identify" command. This is a nice easy command to start with. You simply have to send the command &EC to the command register.

```
                    Add to program 1
 40 OUT &87,&EC
```

...then read the data from the RD Data register..

```
                    Add to program 1
 50 FOR A=1TO512
 60 B=INP(&80)
 70 C=INP(&80)
 80 IF C>31 THEN PRINT CHR$(C);
 90 IF B>31 THEN PRINT CHR$(B);
100 NEXT
```

You'll notice that the program only does simple check to make sure it prints ASCII characters above 31, and reverses the order of Even/Odd bytes.

You should see a couple of lines of characters and if you look closely you'll see the name of the card manufacturer, the model number, serial number and other data. It's much easier to read all the data into an array, then pick the manufacturer's name, model number, default number of sectors, etc. from the array. (See the CF specifications, page 120, for a breakdown of all the data revealed by the Identify command.)

When reading data from (or writing data to) the card there are a couple of things to be aware of. Firstly unlike reading from the other registers the data port will detect a read (or write) and automatically place the next byte of data into the register for you to read (or, if writing, put the data into the buffer freeing up the register to be written to.). Secondly you will need to transfer the full 512 bytes of data (256 words) otherwise the card will still be waiting to send the next byte. This is more of a problem if you forget to send the full amount of data while writing to the card because if the buffer isn't filled the data won't be written to the card and will be lost (unless you remember to send the flush cache command &E7).

Reading From Any Sector

This isn't much harder than using the Identify command. The routine to read data is the same, it's just the initial command that is different, and setting some registers.

First you need to decide on a sector to read from. So let's go back to the identify command for a minute.

```
                    Program 2
 10 DIM(512)
 20 OUT &87,&EC
 30 FOR A=0TO255
 40 D(A*2+1)=INP(&80)
 50 D(A*2)=INP(&80)
 60 NEXT
 70 PRINT "Default"
 80 PRINT "Cylinders","Heads","Sectors"
 90 C$="":FOR L=2TO3:C$=C$+HEX$(D(L),2):NEXT
100 H$="":FOR L=6TO7:H$=H$+HEX$(D(L),2):NEXT
110 S$="":FOR L=12TO13:S$=S$+HEX$(D(L),2):NEXT
120 PRINT C$,H$,S$
130 PRINT
140 PRINT "Actual"
150 PRINT "Cylinders","Heads","Sectors"
160 C$="":FOR L=108TO109:C$=C$+HEX$(D(L),2):NEXT
170 H$="":FOR L=110TO111:H$=H$+HEX$(D(L),2):NEXT
180 S$="":FOR L=112TO113:S$=S$+HEX$(D(L),2):NEXT
190 PRINT C$,H$,S$
```

You'll notice I've read two sets of data (and hopefully displayed them properly), the first is the default number of cylinders, heads and sectors defined by the compact flash card. The second are the actual figures used to format the card. These two sets of figures may not be the same. If you are going to format a card you're better off using the manufacturers default settings, but if you are using a previously used card you'll need to know how it was formatted. (you could, of course, choose to format the card with different numbers of cylinders, heads and sectors to better match up with existing operating systems on the Tatung Einstein and make life easier while making a patched CF DOS)

Anyway pick a cylinder, head, and sector in range and write them to the appropriate registers, along with a sector count of 1, then issue the Read Sector command (&20)

You'll need a nibble (4 bits) for the head, 2 bytes for the cylinder, and one byte for the sector.

The numbers picked here for Head, Cylinder and Sector were just randomly selected and may be empty on your card.

```
                    Program 3
 10 H=&3:CH=&00:CL=&33:S=&02
 20 H=&A0+(VAL(HEX$(H,1)))
 30 OUT &86,H
 40 OUT &85,CH
 50 OUT &84,CL
 60 OUT &83,S
 70 OUT &82,1
 80 OUT &87,&20
 90 FOR A=0TO255
100 B=INP(&80)
110 C=INP(&80)
120 IF C>31 THEN PRINT CHR$(C);
130 IF B>31 THEN PRINT CHR$(B);
150 NEXT
```

By now you should, hopefully, know what's going on here. All except line 20. That needs some explanation. Register 6, Select Head, needs to have bits 7 and 5 set to 1, bit 6 set to 0 (otherwise the addressing will be in LBA mode, more on that later) and bit 4 set to 0 for drive 0 (we only have one card in the interface as it can only take one). In other words the head register needs to be set so;

Bit7 - always 1
Bit6 - LBA 1=yes, 0=no
Bit5 - always 1
Bit4 - Drive 1 or 0
Bit3 - Head bit 3
Bit2 - Head bit 2
Bit1 - Head bit 1
Bit0 - Head bit 0

To set bits that way (1010xxxx is equivalent to &A0) we need to make sure that H is only 4 bits long and we need the lowest 4 bits. HEX$(H,1) takes just the least significant nibble of H, VAL turns it back into a number, and adding &A0 adds the high nibble of the byte.

Now the output from the program might not might not make sense, it could be a graphic file, or anything else, but you have read it and displayed it nevertheless. If you're lucky it's a raw text file and you're very impressed.

If you had read this data into an array, like the one used in the program to Identify the number of cylinders and sectors earlier, you would now be able to examine it, edit it and write it back (using the command &30)

WARNING – writing data back may make your card unusable (until you format it again). This is like having raw hard drive access. Use a blank card or one that you don't mind losing the data on.

LBA or Logical Block Address Mode

Having got to grips with cylinders, heads and sectors. We'll introduce another method of accessing the Compact Flash card and that's Logical Block Address Mode. The card it just treated like a whole unit with a sectors in sequential order. It uses the same registers as the hard ATA addressing

LBA bits 7-0	Sector Number Register
LBA bits 15-8	Cylinder Low Register
LBA bits 23-16	Cylinder High Register
LBA bits 27-24	Head Register

The Head register (&86) is still just a nibble for the bits 27-24 and is also the used to activate LBA mode. You'll remember from the previous program that we needed to set the most significant nibble of the &86 register to 1010 in order to activate drive 0 (there is only one) and ensure that LBA mode was off.

To activate LBA mode we need to set the highest nibble of the Head register to 1110 and then put the LBA bits 27-24 on the lowest nibble. In practice this means that the Head register must always contain the bits 1110xxxx. We can obtain this result as we did above, by stripping any stray bits from the the address (just in case) and adding &E0 instead of &A0 in the program above.

Now before implementing an LBA addressing program it's worth checking the card for the maximum number of sectors that are accessible in LBA mode. If you go back to program 2 and add the following lines;

Add to program 2

```
200 PRINT "LBA sectors"
210 L$="":FOR L=120TO123:C$=C$+HEX$(D(L),2):NEXT
220 PRINT L$
```

If you now run program 2 you get the same information about Cylinders, Heads and Sectors followed by the information you need about the number of accessible LBA sectors.

LBA mode addressing may be easier to implement but otherwise has little effect on the contents of the card. You don't need to worry about counting the number of sectors/heads per cylinder, and incrementing them at the right time. You just need to have one counter to be aware of.