CIRCUIT CELLAR ONLINE

FEATURE ARTICLE

Mark Samuels

PIC a CompactFlash Card

New tech toys are always exciting to any gadget junkie, and Mark's no exception. Talk about a cool little device, the CompactFlash card has countless possibilities. If you're looking to create a digital picture frame or an MP3 player, for example, the CF card can do it. With the wonder of removable memory, you can expect to see this format around for some time.

I am a gadget junkie. Anytime a new tech toy hits the market, I'm the first in line to buy one. However, when I bought a digital camera two years ago, it was not the camera itself that held my fascination, it was the cool little matchbook-sized card that stored all the pictures (see Photo 1). As more products appeared that used a CompactFlash card for removable storage, I was both delighted and despondent. Delighted because this seemed to end the apparent curse that my enthusiasm for a new format indicated its obsolescence (i.e., MiniDisc, CDi, etc.). And, despondent because every device that used a CF card seemed to have some serious processing horsepower, making it appear that using such an interface in lower-end designs would be difficult. When I discovered the Microchip PIC microcontroller, I knew it was my solution for integrating CompactFlash into my own projects. Now, I have the capability to add numerous megabytes of compact, removable, nonvolatile memory to virtually any system.

The CompactFlash Association was established in 1995 to specify the format of what was to become a subset of the PCMCIA or PC card specification. The result is a removable media standard that uses fewer interface signals (50 instead of 68) and is roughly one-third the size of a PCMCIA memory card. The internals of the memory card consist of a smart controller, buffer, and varying amounts of nonvolatile memory. The specification also defines different modes of access to the card, including Common Memory mode and True IDE mode. While in True IDE mode, the card can be directly connected to an IDE bus with no active circuitry, which makes it a great way to add removable storage to an embedded PC. However, because I was planning to use an 8-bit microcontroller, I chose to access the CompactFlash card in its Common Memory mode, taking advantage of an 8-bit wide data bus in this mode, instead of the 16-bit data bus required for an IDE interface.

A CHOICE PIC

As for the microcontroller, I chose a PIC16F877, one of Microchip's newer

Photo 1—*CompactFlash cards are available from a variety of manufacturers in several different capacities.*

parts. The "F" stands for flash memory, which makes development nicer than the old burn, test, and wait-for-half-an-hour-under-a-UV-light cycle. In addition, Microchip's integrated development environment (MPLAB) is available (free) from its web site, and an inexpensive in-circuit debugger is available for these flash memory parts.

The basics of the PIC are the same as they are for any other part in the PIC16*xxx* family, including a RISC design with 35 instructions and a plethora of peripherals. Although this project does not use any of the peripherals built into the device, the main reason for me picking this particular part was that those peripherals are still available for some other use. I had to keep in mind that this is not an isolated project but something that will be integrated into other projects. That is the reason I chose a 40-pin device that has 33 I/O pins when I am only using 17, and could therefore fit into a 28-pin device. In addition, this device has 8 KB of program memory available, yet I am using far less than that. All of these considerations were made with the knowledge that the PIC would be running more than the read and write routines and would be connected to more than just the CompactFlash card.

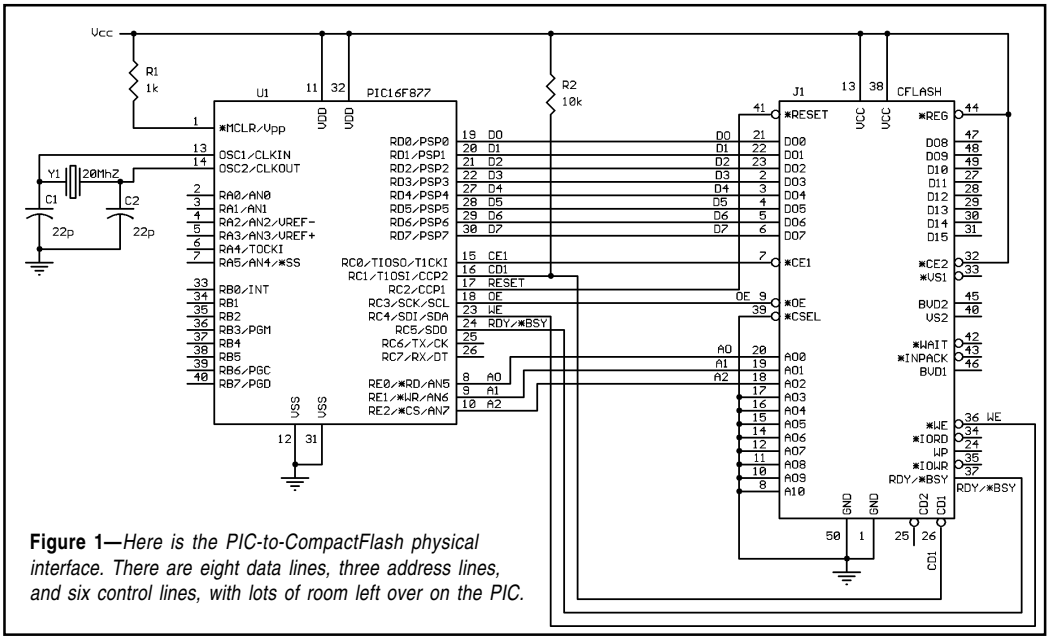

Figure 1—*Here is the PIC-to-CompactFlash physical interface. There are eight data lines, three address lines, and six control lines, with lots of room left over on the PIC.*

GETTING CONNECTED

The theory of operation in interfacing to the CompactFlash card in its Common Memory mode is similar to an IDE interface. In a nutshell, eight registers are accessible by the host and loaded with various data. After the first seven registers are loaded with the appropriate information, the eighth (command) register is loaded with a command, and that command is executed. To accomplish this, I have a 17-pin connection from the PIC to the card, including three address lines (E2:E0), eight data lines (D07-D00), and six control lines. For an absolute minimum interface to a CF card, a few of these control signals are not necessary.

According to the CompactFlash specification, signals CD2 and CD1 are grounded internally to the card to indicate that a card has been inserted. [1] I connect the PIC to CD1 (with a 10-kilohm pull-up), and if the pin is low, a card is present. Another connection that can be removed from the interface is the RDY/–BSY signal. In my example code, I poll this line prior to read and write operations to ensure that the card can accept a command, but a fixed-time delay can also be used if free I/O pins are getting tight.

A third signal that is not absolutely necessary but nice to have is a reset line. Strobing the reset line low will, as the name suggests, reset the controller in the CF card. The other three control lines that I use are CE1, which will connect the card data bus (useful if sharing an 8-bit data bus), the active low OE (output enable), and WE (write enable) strobe signals.

As you can see in Figure 1, the remaining interface lines are fairly straightforward. The three address lines to the card are on Port E, which is conveniently a 3-bit wide port. The eight data lines from the card are connected to Port D, and the control lines are on Port C. So much for the hardware interface.

ADDRESS DECODING

The functional procedure to access the CF card is also easy, after the 116-page CompactFlash Association Specification is dissected. The most important bit of information in the specification is the address decoding of the eight control registers within the controller portion of the card (see Table 1). Access to the full range of necessary registers can be achieved using only three address lines, so –REG and A10–A3 are hardwired to V_{CC} and GND, respectively.

To load one of these registers, the data sent is placed on the data lines to the card, the 3-bit address of the desired register is placed on the address lines, and –WE is strobed low. It's that easy. By loading the various registers with sector addresses (either in logical block addressing or cylinder/head/sector addressing) and a command in the command register, commands can be sent to perform a read, write, or any number of other operations. The results of these operations most often go through the internal buffer, and sequential accesses to the data register incrementally access each byte.

For example, if a card has a sector size of 512 bytes, a buffer size of 512 bytes, and the appropriate addressing registers have been loaded followed by a read sector command, the buffer is then loaded with the contents of that sector. The first access to the data

register (accomplished by putting the address of the data register on the address bus and strobing –OE) will read the first byte in the buffer. The next strobe of –OE will read the next byte in the buffer and so on.

The same would be true of a write sector operation. After the sector location has been loaded into the appropriate registers and the write sector command has been loaded into the command register, the first byte to be written is placed on the data bus. When –WE is strobed, that byte is written to the first location in the buffer. The next byte to be written is then placed on the data bus, –WE is strobed again, and so on. Because these operations go through the buffer, which is internally read from and written to a full sector at a time even if the desire is only to write one byte to the card, a full 512 bytes must be written to the buffer. The data in the buffer does not transfer to the card memory until the buffer is full.

–REG	A10	A9–A4	A3	A2	A1	A0	–OE = 0	–WE = 0
1	0	X	0	0	0	0	Even RD data	Even WR data
1	0	X	0	0	0	1	Error	Features
1	0	X	0	0	1	0	Sector count	Sector count
1	0	X	0	0	1	1	Sector no.	Sector no.
1	0	X	0	1	0	0	Cylinder low	Cylinder low
1	0	X	0	1	0	1	Cylinder high	Cylinder high
1	0	X	0	1	1	0	Select card/head	Select card/head
1	0	X	0	1	1	1	Status	Command
1	0	X	1	0	0	0	Dup. even RD data	Dup. even WR data
1	0	X	1	0	0	1	Dup. odd RD data	Dup. odd WR data
1	0	X	1	1	0	1	Dup. error	Dup. features
1	0	X	1	1	1	0	Alt status	Device Ctl
1	0	X	1	1	1	1	Drive address	Reserved
1	1	X	X	X	X	0	Even RD data	Even WR data
1	1	X	X	X	X	1	Odd RD data	Odd WR data

Table 1—*Here you can see the register address decoding. Note that all data can be accessed using only three address lines.*

THE IDENTIFY DRIVE COMMAND

Almost all CompactFlash cards on the market today have both sector and buffer sizes of 512 bytes. To ensure that's true for a particular card, use the identify drive command, which can be sent to the card without loading any of the other registers. When the value 0xEC (identify drive) is strobed into the command register, the buffer is immediately filled with 512 bytes of useful information, including sector size, buffer size, model numbers, serial numbers, number of heads, cylinders, tracks, and too many other goodies to be listed here. However, they are well documented in the CompactFlash Association Specification.

The sample source code I have provided for downloading shows a simple application of reading from and writing to a CompactFlash card. The program essentially uses two bytes on a card to store a counter, which keeps track of how many times the card has been inserted into the socket. This is accomplished by waiting for the card detect signal (CD1) to go low, indicating that a card is present.

An identify drive command is executed to get the sector size (word 5) and buffer size (word 21) of the particular card that has been inserted. The identify drive command doesn't require any parameters, so none of the other registers have to be loaded with any data. Simply load the command register and strobe –WE.

The CompactFlash Association Specification states details about each command and the results, including the breakdown of the 512 bytes of information dispensed by this particular command. To get to a specific piece of information, simply strobe –OE enough times to get to that data in the sequence. For example, the number of bytes per sector is stored in the sixth word (word 5), so strobing –OE eleven times will make the first byte of that particular data field appear on the data bus.

Most of the information on the card is stored as 16 bits, so the subroutine in the code (*CF_READ*) actually strobes –OE twice, storing the first byte read in *DATA_LO* and the second byte read in *DATA_HI*. The program then uses these values to know how many times to write to the buffer before that information is actually transferred from the buffer to the card memory. Next, the program reads a sector at a particular location but only reads the first two bytes from the buffer. These two bytes are the counter, which is incremented and written back to that same location on the card.

One note of caution, running this program on a CF card that has been formatted and has data already stored on it may corrupt that data and make the card unreadable until it is reformatted, because this program will overwrite a specific sector on the card. That read and written location is clearly annotated in the source code, so feel free to play around with the target location and amount of data read or written.

There is a useful shareware application that I used to read the raw hex data from the card to verify the actions of the PIC. The program is called WinHex and is available from CNET and various other shareware archives on the Internet. As long as you have the means to access a CompactFlash card as some sort of disk on your PC, you can use WinHex to view each individual byte stored on the card.

LAYING THE FOUNDATION

In order to make my development as easy as possible, I took the PIC-to-CF interface shown in Figure 1, added an IDE connector so I could plug the card right onto the IDE bus of my PC, added in a full set of test points for each signal on the CF card (along with in-circuit serial programming and in-circuit debugger headers for the PIC), and had some boards made. The end result was a nice little development kit

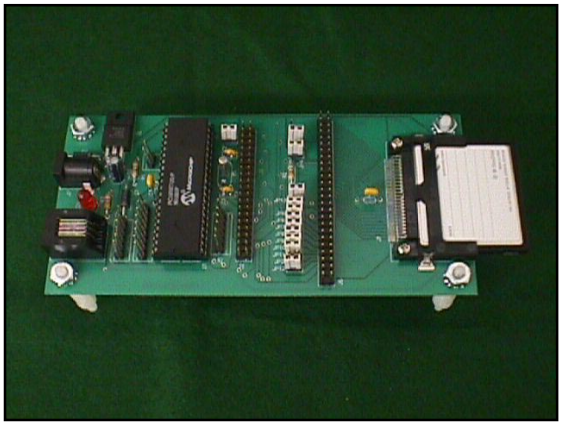

Photo 2—*The development board consists of a CompactFlash card connector, a 40-pin DIP socket for the PIC, a test point for each individual CF card pin, a 40-pin IDE connector, a Microchip in-circuit debugger connector, and jumpers galore for various configurations.*

(see Photo 2).

This simple example program is the backbone of the lowest level of interface to the CompactFlash card. I want to stress that this is only the bare minimum for an extremely simple interface. Additional layers of firmware would be needed to read and store information that spans several sectors, and external memory may be required if the application is not one that caters to streaming data.

This foundation, combined with some deep reading of Microsoft's FAT specification [2], has allowed me to use a PIC to extract information stored in specific files on a Windows-formatted CF card. Again, there is a lot more overhead that has to go in the PIC program to read various data from the file structure system just to find the location of the file on the card and be able to follow the data through various sectors. However, the actual data on the card is the same, with no regard to the mode of operation of the card (IDE or common memory), so it's only a matter of understanding any other file specification to be able to conform to that specification within the PIC. Being able to access information on a card that has been formatted to a specific file system is a nice feature, allowing you to simply copy a certain file from a PC onto a CF card. And, the PIC can access the information in that same format, without having to restructure the data into some proprietary configuration.

NO LIMITS

Being the gadget junkie that I am, it was exciting just being able to read and write a little 2-byte counter on a cool little device. But, using those same basic subroutines, the possibilities are endless. Want to store a year's worth of a regularly sampled analog signal? Put a PIC and a CompactFlash card on it. Want a digital picture frame? Store bitmap files on a CF card and hook the PIC to an LCD. Want an MP3 player? Keep the music on a CF card and tie the PIC to one of the now readily available MP3 decoder parts.

The point is that CompactFlash cards are a great medium for removable memory, and the format is not going away any time soon. I doubt IBM would have chosen the CompactFlash form factor for its amazing 1-GB Microdrive if support for CF was not going to be around for years to come. And yes, the Microdrive conforms to the CompactFlash card specification, so anything you do to interface to a CF card will also work with the Microdrive (but with a higher current demand). One gigabyte of removable storage for any project you can think of! I love gadgets!

Mark Samuels works at ARMA Design, a custom design house in San Diego, CA, where he does firmware and hardware design for a wide variety of embedded projects, most often using PIC microcontrollers. He can be reached at mark@ARMAnet.com or visit ARMA's web site at www.ARMAnet.com.

FIRMWARE

You can download the source code from the *Circuit Cellar* web site or from http://www.ARMAnet.com/CompactFlash/.

SOURCES

Printed circuit board
ARMA Design
(858) 549-2531
www.ARMAnet.com

PIC16F877
Microchip Technology, Inc.
(888) 628-6247
(480) 786-7200
Fax: (480) 899-9210
www.microchip.com

REFERENCES

[1] CompactFlash Association Specification, http://www.compactflash.org/specdl1.htm.

[2] Microsoft Corp., "FAT: General Overview of On-Disk Format," http://209.67.75.168/hardware/fatgen.htm.

RESOURCE

Microchip Technology, Inc., "PIC16F877," http://www.microchip.com/download/lit/pline/picmicro/families/16f87x/datasheet/30292b.pdf.