An Easy SD Flash File System for the Propeller

Bob Belleville

Introduction and Scope

A Canadian company, Rogue Robotics, produces a tiny module which accepts SD or MMC flash memory and provides access to FAT16 and FAT32 files via serial interface. (FAT stands for File Allocation Table and goes back to the very early MS DOS days. FAT16/32 file systems are still fully integrated into MS Windows and can be directly handled by most operating systems.) The unit is called "uMMC Serial Data Module" (DS-MMC.) Detailed information is available at their web site http: //www.roguerobotics.com/products/electronics/ummc. It currently costs $50 without a memory card. I first found out about it at least a year ago from Hobby Engineering which is a Parallax dealer. (http://www. hobbyengineering.com/) At the time I needed it for a rather complex project which I expected to have to do using one or more SX processors. I was dreading trying to deal with a file system in such a small memory machine --- as much as I love it. Then came the Propeller. When I got my first machine, I hooked it up and I knew that all would be well, but not all that simple.

To get read/write speed of more than a few hundred bytes per second a careful assembly language implementation is needed. The module needs to be run at 115200 baud for good performance. With bulk data transfers even FullDuplexSerial isn't enough. Also the communication protocol with the ummc, while simple enough, is tricky because of error conditions and the need to prevent 'deadly embrace' --- that is both systems stuck waiting for the other.

Interfacing directly to an SD card will clearly give the best performance, but this approach is quick and easy. This approach also places the FAT part of the file system outside the Prop. Files can be written to the card using a flash reader on a PC. Cards can be formated and maintained on a PC. With this kit, a Prop can read at about 9.4K bytes/sec and write at just under 5K bytes/sec on an ordinary SD card (these are max binary rates and formatted transfers will be a good deal slower.) (At 115200 which is the highest rate the ummc supports, the theoretical max speed is 11.5K bytes per seconds so the read rate is quite good. There are reports on the Rogue site that write speeds of 7.5K have been achieved.)

The ummc's interface is well described in their documentation. For example 'o 1 r /file.txt' opens a file in the root directory named 'file.txt' for reading. 'r 1' reads 512 bytes from that file. This is great if you are a human sitting at a terminal communication with the SD card --- but why

would anybody do that for more that a few minutes except to check to see if the interface is working.

This kit contains objects to provide a more computer like interface.

version history:

development early March 2007 to
V1.0 - March 29, 2007

file list:

- serial_terminal
  - PC access much like SimpleDebug but call compatible with tv_terminal and adding a non-blocking input.
- FDS_sg.spin
  - A substantial variation of FullDuplexSerial which implements block transfer of data at high baud rates but which can still interface to spin programs. It is slightly smaller than FullDuplexSerial.
- format_memory.spin
  - Allows decimal, hex, and string data to be formated into a single memory string. Makes data records in comma separated file format (.csv) Supports the automatic generation of ummc commands.
- ummc_term.spin
  - Uses FullDuplexSerial and serial_terminal to check that Prop to ummc communication is actually working and builds confidence with the ummc commands.
- ummc.spin
  - The real meat. Uses FDS_sg, and format_memory to provide a comfortable interface from the Prop to FAT16/32 files on an SD card. Implements binary, text and comma separated file interfaces. Provides for 4 files to be open a a time. Provides random access reads. Passes error status in a consistent way and is unlikely to get stuck because it times out if the ummc module is unresponsive.
- ummc_demo.spin
  - Shows how to use the interface and provides confidence that things are actually working
- a.txt
  - A simple text file that needs to be written to the root directory of a card for ummc_demo to work. Any text file with that name will work.
- readme.pdf
  - This file in pdf format.
- readme.abw
  - The source of this file. AbiWord is a free wordprocessor.

Interface Object 'ummc.spin'

Ordinarily using an SD card with the Prop will have two main uses: logging data to a file for future processing on a PC and reading files generated on a PC to drive devices controlled by the Prop. This interface is designed to make these two tasks as easy as possible for application software both on the Prop and the PC. This is accomplished by providing routines to read and write so called 'comma separated files' these are often '.csv' and are readily created in any PC development language and can be read and written by Microsoft Excel and many others.

Another use will be to read and write 'binary files' for fastest speed and small size. This form of interface is also included. You will need a hex editor to see what is happening. Here is a good one (free and simple):

http://www.chmaas.handshake.de/delphi/freeware/xvi32/xvi32.htm

Documentation for ummc.spin is included at the start of the file. (Over 300 lines.)

Error Codes

The following are code generated directly by the ummc (from their documentation):

E02 Buffer Overrun: Too many bytes were sent in the command. All command can be a maximum of 256 bytes (including the path).
E03 No Free Files: This is a response from the ***Free File*** command. There are no more open handles. You must close an open file handle before a new one can be opened.
E04 Unrecognized command.
E06 Command formatting error: this occurs if parameters are missing or invalid.
E07 End of file
E08 Card not inserted
E09 MMC/SD Reset failure
E0A Card write protected
EE6 Read-only file: a Read-Only file (file attributes) is trying to be opened for write or append.
EE7 Not a file: an invalid path.
EE8 Write Failure: There could be many reasons for this (damaged card, card removed WHILE writing, etc…)
EEA No free space: There is no free space on the card.
EEB File not open: The file handle specified has not been opened with the ***Open*** command.

EEC Improper mode: A ***Read*** command was attempted while the file has been opened for writing, or vice-versa.
EED Invalid ***Open*** mode: only 'R', 'W', and 'A' are acceptable open modes.
EF1 Handle in use: The specified handle is already being used.
EF2 File does not exist: The file in the path specified does not exist.
EF4 File already exists: A ***Write*** command was issued, and the file in the path already exists.
EF5 Path invalid: The path specified does not exist. Ensure that all directory names in the path exist.
EF6 Invalid handle: The handle specified is not valid.
EFB Bad FSINFO Sector (FAT32 only)
EFC Unsupported FAT version: Ensure the card is inserted correctly and that the card has been formatted to FAT16 or FAT32.
EFD Unsupported Partition type
EFE Bad Partition information
EFF Unknown Error

ummc.spin also catches errors. In fact some of the above error codes will not be seen. These errors come directly from ummc.spin itself:

EB1 Invalid Handle - (0..3) only
EB2 Handle already in use
EB3 Handle not open - make sure a successful call to took place
EB4 Open mode can only be "aAwWrR'
EB5 ummc module timed out - could be powered off, etc.
EB6 Invalid number of bytes requested - 512 bytes max
EB7 Write request on handle open for read - ummc can only do read or write on a single handle, have to close and reopen to change modes.
EB8 Read request on handle open for write - note above
EB9 Sync failed - communication with the ummc somehow failed

Interface Object 'format_memory.spin'

This object is like the various forms of dec and hex in many objects. In this case the value isn't sent to a device like the tv_terminal but store into a string in memory. See the object for all the documentation.

Getting started.

Obtain a ummc. Connect it up through 4.7K ohm resistors to the Prop. Remember that Receiver of one goes to Transmitter of the other. The resistors make sure that even if two outputs get tied together by accident

that neither device is damaged. Also power the module with 5VDC (but it actually works at 3.3VDC.) Obtain a SD card (you are unlikely to actually find a MMC card but you might have one about.) Format it using a PC card reader and put some file called a.txt in the root directory --- that is some file called /a.txt (the SD card doesn't have a drive number in the ummc but it does on the PC.)

Use ummc_term.spin (make it the top level object and load it into your Prop. Get some terminal emulator (http://www.hw-group.com/products/hercules/index_en.html and http://realterm.sourceforge.net/) and start the emulator at 115200 baud on the same COM port the PropTool uses. Beware that the PropTool and a terminal emulator don't play nice together. You have to disable the emulator before loading the Prop and then re-enable. This is easy with the two emulators shown. Also note that the Hercules 'setup' terminal (it is used to 'setup' their hardware devices) does have file capture on a right click in the serial tab. Their documentation is online. I find the Hercules to be most handy for this work because there is a single open/close button for the COM port.

First time only! You have to set the ummc module's baud rate to 115200. Recompile ummc_term.spin with the _rate con set to 9600. If v works type s 0 4 (zero four) and enter. This will select 115200. Recompile with _rate of 115200. (In both cases serial_terminal's baud rate to the PC is still 115200. Baud rate is stored in the ummc and stays fixed even through power cycles. It only has 5 rates so if you get lost get try each one.

Hit a key. Type v and enter to get the ummc version number. Upgrade if necessary -- see Rogue's site. (Upgrade will require a different hardware link to the module --- be careful and read their instructions. This kit uses 101.56 SN:UMM1-OEM --- this what you see with the v command.) Try some things:

o 1 r /a.txt (all command end with enter - cr 13)
- this opens the file
i 1 - shows the current file position and total length
r 1 12 - reads 12 bytes from the file
c 1 - closes it

Special note: ummc module uses file handles 1..4 and this kit maps those to 0..3.

Now load ummc_demo and run it as top level object. Output should look something like this (this is a capture file with comments added):

```
any key to begin
any key to begin
ummc_demo
```

```
sync TRUE: communication ok

used: 704K bytes of total: 123767K bytes

just open and read a file using gline
/a.txt
this is a test
of the early warning
system.

writing data.csv

now read back using gline --- no parsing
/data.csv
15161,69157402,1234FCA0,,text bit
45749,694D4CD2,1234FCA0,,text bit
26115,696D2DE2,1234FCA0,,text bit
26037,698CEF22,1234FCA0,,text bit
25909,69AC8FE2,1234FCA0,,text bit

now read back parse fields and show g* return
values
/data.csv
15161 0002C 69157402 0002C 1234FCA0 0002C
          00 4002C text bit 0000A
(lines deleted - note 4002C shows that this
 field is actually empty (default 0) and that
 the field ends with a comma)

delete this .csv file

if necessary create a file with 256 bytes 0..255
attempt to open: /postest.dat
error code: EF4
EF4 means file already exists
/postest.dat

now open the file and jump to various bytes
file position is: 0
file length is: 256
value at position 88: 88
file position now is: 88
value at position 157: 157
file position now is: 157
value at position 300: 00020000
file position now is: 256

append to a binary file 100k bytes - hit y key
                                     else skip
```

```
.............. ( 250 dots ) ............
binary read using gbin 100k bytes - hit y key
                                    else skip
.............. ( 250 dots ) ............
binary read using read 100k bytes - hit y key
                                    else skip
.............. ( 250 dots ) ............
these show the space and > on the ends of bintxt
20000000 0000003E
any key to repeat demo
```

End Note

At release this all worked on my demo board, and Dell Latitude 600 laptop but no doubt there are errors in this code. See the forum for a thread announcing this kit to post problems. I am an erratic visitor to the forum so months may pass before I get back to this work as it is part of a very large project. I have no relationship with any of the companies or products mentioned except as a user.

There are an infinite number of ways to build this kind of interface. Hopefully this will either be of direct use to you or serve as model for your own version. Good luck!