An Easy SD Flash File System for the Propeller

Bob Belleville

Introduction and Scope

Parallax now has a serial to flash memory product. The Memory Stick Datalogger #27937 allows a memory stick, a USB flash card reader or in fact a USB hard disk to be interfaced to the Propeller. My previous 'object exchange' kit interfaced with SD cards using a different hardware board. (ummc file system) This kit continues that work but using the #27937 which is usually just called memStick in this document.

Neither interface method is a clear winner. Both have advantages and disadvantages. Both can achieve data rates of one half or more of the interface baud rate. ummc allows 4 files open at once but isn't quite as fast as the memStick. memStick only allows one file open at once but has more directory management functions. ummc needs only 2 i/o pins but memStick needs 4. memStick write is quite slow unless special code is used to speed things up. (See manageWB.spin and notes below.)

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 memStick, 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/sticks can be formated and maintained on a PC. With this kit, a Prop can read at about 5.9K bytes/sec and write at just above 5K bytes/sec on an ordinary SD card (these are for formatted transfers, raw binary rates can be near 9.3K bytes/sec.) At 115200 baud, the theoretical max speed is 11.5K bytes per seconds so the raw rate is quite good.

This kit contains objects to provide computer like interface to files on the flash memory.

version history:
    V1.0 - November 2007

file list:

serial_terminal
: PC access much like SimpleDebug but call compatible with tv_terminal and adding a non-blocking input. Also operates at 115200 baud.

FDS_sgf.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 memStick commands.

memStick_check.spin
: Uses FullDuplexSerial and serial_terminal to check that Prop to memStick communication is actually working and builds confidence with the memStick commands.

memStick_dev.spin
: Uses FDS_sgf, and format_memory to develop and test the methods which will make up memStick.spin. Provides additional functionality and test.

memStick.spin
: A basic interface object for the memStick.

memStick_demo.spin
: Shows how to use the interface and provides confidence that things are actually working

mangageWB.spin
: Code to copy to your application to speed writing.

readme.pdf
: This file in pdf format.

readme.abw
: The source of this file. AbiWord is a free wordprocessor.

Interface Object 'memStick.spin'

Ordinarily using an SD card/stick 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

Routines Include

The following routines provide the file system interface:

PUB start(rxpin, txpin, rtspin, ctspin, rate, syncMaxSeconds)
PUB stop
PUB getErrorString
PUB sync(maxSecToWait)
PUB open(fn, mode) (fn is a 8.3 file/directory name zero terminated)
PUB close
PUB read(data, n, posf)
PUB write(data, n)
PUB gbyte g* routines provide bytewise or fieldwise i/o.
PUB asciihex2bin(c)
PUB gdec(along)
PUB ghex(along)
PUB gstr(p)
PUB gline(p)
PUB gbin(addr, n)
PUB seek(n)
PUB getDirectory(dbuf, dmax)
PUB getFileLength(fn)
PUB getFreeSpace
PUB sleep(wake)
PUB cd(name)
PUB renameFile(old, new)
PUB deleteFile(name)
PUB makeDirectory(name)
PUB deleteDirectory(name)

Method 'sync' returns 0 if there is something seriously wrong, 1 if all is well and some kind of flash memory is available. It returns 2 if there is no device attached. The way the 'monitor' is designed it is tricky to find out this information. My 1G stick takes 17 seconds to become ready after power up. The argument to 'sync' tells how long to wait before giving up and returning 0.

Error Codes

Generally methods in this interface return false (0) if all is well. This allows spin code to put a call in an IF statement and only execute the block below on error. Errors are indicated by negative number:

-1 eof (end of file) (not really an error)
-2 memStick error of some sort, message can be read by a separate call.
-3 time out, more than 2 seconds between consecutive received bytes
-4 time out, memStick not responding at all
-5 no file open for the operation requested
-6 empty field in g* .csv format reader functions (not really an error)

If the error is -2 getErrorString returns the address of the message sent from the memStick.

Some routines, getFileLength(name) for example, give a 0 or positive value as you would expect. See the code.

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 stored into a string in memory. See the object for all the documentation.

Getting started.

Obtain a #27937. Connect it up through 4.7K ohm resistors to the Prop. Remember that Receiver of one goes to Transmitter of the other. RTS and CTS are also crossed. (See the object.) 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.

Obtain a memory stick or flash memory card and a USB card reader. Format these using your PC. I tried a 1G stick, 2 SD cards and a Compact Flash card and all worked fine although the write speed varied dramatically. Use FAT16 or FAT32 formats.

Use memStick_check.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. Also you will need to set the local echo and cr/lf handling to see what is happening (also on the right click).

Power this all up. You can type commands in native memStick format and see the result. You are communciating the stick at 9600 baud. You will find this painfully slow.

Re-Flash the Memstick

If 9600 baud solves all your problems then you are done and can use the other objects in this kit by setting _rate1 to 9600. Have fun.

The memStick can be upgraded. The Vinculum web site (www.vinculum.com) has upgrade files which can be down loaded, renamed FTRFB.FTD, and placed in the root directory of a stick. When the memStick is started an automatic upgrade is done. My memStick came with version 3.57 and I upgraded to 3.60. This worked fine.

Also at Vinculum there is more comprehensive documentation which you may wish to have.

To move to higher baud rates you will need a small program from the same site that allows you to change default parameters. It is called VncFwMod.exe. Basically it modifies an upgrade file. Get it and its documentation. I ONLY changed the baud rate to 115200 and upgraded to that firmware version. Go slow, read carefully and good luck, you are on your own. My suggestion is this: use memStick_check.spin at 9600 baud to make sure all is working; get the latest .FTD upgrade; run VncFwMod on it and select 115200 baud; save to FTRFB.FTD; copy to root of a stick; boot it; turn off the power when it says "rebooting"; remove FTRFB.FTD from the root; change to using memStick_dev.spin. This should do it.

Please be really careful with this. You could be forced to rig up a serial interface to the memStick and use another of Vinculum's tools to re-flash the chip.

memStick_dev.spin

This is a big module that allows development of memStick.spin. It has many features and tests. Read the code and experiment. If you want a custom version of memStick.spin this is the place to start.

memStick_demo.spin and memStick.spin

This runs the same demo which is part of _dev with all the debug removed. If this works than you can build you own application.

manageWB.spin

Write speed seems to be more related to the number of calls to WRF than to the number of bytes written. manageWB allows you to buffer up write bytes in a easy way and write larger blocks for higher speed. In my case, 5 times faster. Just cut and paste this code into your app. Because it has to call methods in memStick.spin it can't be a separate object.

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!