Version 1.0

HYDRA SD MAX Storage Card

Programming and User Manual

Secure Digital + 128K EEPROM

Andre' LaMothe

Nurve Networks LLC

HYDRA™ SD MAX Storage Card User Manual v1.0 (SAMPLE)

Author
Andre' LaMothe

Editor/Technical Reviewer
The "Collective"

Printing
0001

ISBN
Pending

Licensing, Terms & Conditions

NURVE NETWORKS LLC, . END-USER LICENSE AGREEMENT FOR HYDRA HARDWARE, SOFTWARE , EBOOKS, AND USER MANUALS

YOU SHOULD CAREFULLY READ THE FOLLOWING TERMS AND CONDITIONS BEFORE USING THIS PRODUCT. IT CONTAINS SOFTWARE, THE USE OF WHICH IS LICENSED BY NURVE NETWORKS LLC, INC., TO ITS CUSTOMERS FOR THEIR USE ONLY AS SET FORTH BELOW. IF YOU DO NOT AGREE TO THE TERMS AND CONDITIONS OF THIS AGREEMENT, DO NOT USE THE SOFTWARE OR HARDWARE. USING ANY PART OF THE SOFTWARE OR HARDWARE INDICATES THAT YOU ACCEPT THESE TERMS.

GRANT OF LICENSE: NURVE NETWORKS LLC (the "Licensor") grants to you this personal, limited, non-exclusive, non-transferable, non-assignable license solely to use in a single copy of the Licensed Works on a single computer for use by a single concurrent user only, and solely provided that you adhere to all of the terms and conditions of this Agreement. The foregoing is an express limited use license and not an assignment, sale, or other transfer of the Licensed Works or any Intellectual Property Rights of Licensor.

ASSENT: By opening the files and or packaging containing this software and or hardware, you agree that this Agreement is a legally binding and valid contract, agree to abide by the intellectual property laws and all of the terms and conditions of this Agreement, and further agree to take all necessary steps to ensure that the terms and conditions of this Agreement are not violated by any person or entity under your control or in your service.

OWNERSHIP OF SOFTWARE AND HARDWARE: The Licensor and/or its affiliates or subsidiaries own certain rights that may exist from time to time in this or any other jurisdiction, whether foreign or domestic, under patent law, copyright law, publicity rights law, moral rights law, trade secret law, trademark law, unfair competition law or other similar protections, regardless of whether or not such rights or protections are registered or perfected (the "Intellectual Property Rights"), in the computer software and hardware, together with any related documentation (including design, systems and user) and other materials for use in connection with such computer software and hardware in this package (collectively, the "Licensed Works"). ALL INTELLECTUAL PROPERTY RIGHTS IN AND TO THE LICENSED WORKS ARE AND SHALL REMAIN IN LICENSOR.

RESTRICTIONS:
(a) You are expressly prohibited from copying, modifying, merging, selling, leasing, redistributing, assigning, or transferring in any matter, Licensed Works or any portion thereof.
(b) You may make a single copy of software materials within the package or otherwise related to Licensed Works only as required for backup purposes.
(c) You are also expressly prohibited from reverse engineering, decompiling, translating, disassembling, deciphering, decrypting, or otherwise attempting to discover the source code of the Licensed Works as the Licensed Works contain proprietary material of Licensor. You may not otherwise modify, alter, adapt, port, or merge the Licensed Works.
(d) You may not remove, alter, deface, overprint or otherwise obscure Licensor patent, trademark, service mark or copyright notices.
(e) You agree that the Licensed Works will not be shipped, transferred or exported into any other country, or used in any manner prohibited by any government agency or any export laws, restrictions or regulations.
(f) You may not publish or distribute in any form of electronic or printed communication the materials within or otherwise related to Licensed Works, including but not limited to the object code, documentation, help files, examples, and benchmarks.

TERM: This Agreement is effective until terminated. You may terminate this Agreement at any time by uninstalling the Licensed Works and destroying all copies of the Licensed Works both HARDWARE and SOFTWARE. Upon any termination, you agree to uninstall the Licensed Works and return or destroy all copies of the Licensed Works, any accompanying documentation, and all other associated materials.

WARRANTIES AND DISCLAIMER: EXCEPT AS EXPRESSLY PROVIDED OTHERWISE IN A WRITTEN AGREEMENT BETWEEN LICENSOR AND YOU, THE LICENSED WORKS ARE NOW PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE, OR THE WARRANTY OF NON-INFRINGEMENT. WITHOUT LIMITING THE FOREGOING, LICENSOR MAKES NO WARRANTY THAT (i) THE LICENSED WORKS WILL MEET YOUR REQUIREMENTS, (ii) THE USE OF THE LICENSED WORKS WILL BE UNINTERRUPTED, TIMELY, SECURE, OR ERROR-FREE, (iii) THE RESULTS THAT MAY BE OBTAINED FROM THE USE OF THE LICENSED WORKS WILL BE ACCURATE OR RELIABLE, (iv) THE QUALITY OF THE LICENSED WORKS WILL MEET YOUR EXPECTATIONS, (v) ANY ERRORS IN THE LICENSED WORKS WILL BE CORRECTED, AND/OR (vi) YOU MAY USE, PRACTICE, EXECUTE, OR ACCESS THE LICENSED WORKS WITHOUT VIOLATING THE INTELLECTUAL PROPERTY RIGHTS OF OTHERS. SOME STATES OR JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES OR LIMITATIONS ON HOW LONG AN IMPLIED WARRANTY MAY LAST, SO THE ABOVE LIMITATIONS MAY NOT APPLY TO YOU. IF CALIFORNIA LAW IS NOT HELD TO APPLY TO THIS AGREEMENT FOR ANY REASON, THEN IN JURISDICTIONS WHERE WARRANTIES, GUARANTEES, REPRESENTATIONS, AND/OR CONDITIONS OF ANY TYPE MAY NOT BE DISCLAIMED, ANY SUCH WARRANTY, GUARANTEE, REPRESENATION AND/OR WARRANTY IS: (1) HEREBY LIMITED TO THE PERIOD OF EITHER (A) Five (5) DAYS FROM THE DATE OF OPENING THE PACKAGE CONTAINING THE LICENSED WORKS OR (B) THE SHORTEST PERIOD ALLOWED BY LAW IN THE APPLICABLE JURISDICTION IF A FIVE (5) DAY LIMITATION WOULD BE UNENFORCEABLE; AND (2) LICENSOR'S SOLE LIABILITY FOR ANY BREACH OF ANY SUCH WARRANTY, GUARANTEE, REPRESENTATION, AND/OR CONDITION SHALL BE TO PROVIDE YOU WITH A NEW COPY OF THE LICENSED WORKS. IN NO EVENT SHALL LICENSOR OR ITS SUPPLIERS BE LIABLE TO YOU OR ANY THIRD PARTY FOR ANY SPECIAL, INCIDENTAL, INDIRECT OR CONSEQUENTIAL DAMAGES OF ANY KIND, OR ANY DAMAGES WHATSOEVER, INCLUDING, WITHOUT LIMITATION, THOSE RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER OR NOT LICENSOR HAD BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES, AND ON ANY THEORY OF LIABILITY, ARISING OUT OF OR IN CONNECTION WITH THE USE OF THE LICENSED WORKS. SOME JURISDICTIONS PROHIBIT THE EXCLUSION OR LIMITATION OF LIABILITY FOR CONSEQUENTIAL OR INCIDENTAL DAMAGES, SO THE ABOVE LIMITATIONS MAY NOT APPLY TO YOU. THESE LIMITATIONS SHALL APPLY NOTWITHSTANDING ANY FAILURE OF ESSENTIAL PURPOSE OF ANY LIMITED REMEDY.

SEVERABILITY: In the event any provision of this License Agreement is found to be invalid, illegal or unenforceable, the validity, legality and enforceability of any of the remaining provisions shall not in any way be affected or impaired and a valid, legal and enforceable provision of similar intent and economic impact shall be substituted therefore.

ENTIRE AGREEMENT: This License Agreement sets forth the entire understanding and agreement between you and NURVE NETWORKS LLC, supersedes all prior agreements, whether written or oral, with respect to the Software, and may be amended only in a writing signed by both parties.

NURVE NETWORKS LLC
12724 Rush Creek Lane
Austin, TX 78732

Version & Support/Web Site

This document is valid with the following hardware, software and firmware versions:

- HYDRA Game Console Revision A. or greater.
- Propeller Tool 1.0 or greater.

The information herein will usually apply to newer versions but may not apply to older versions. Please contact Nurve Networks LLC for any questions you may have.

Visit **www.xgamestation.com** for downloads, support and access to the XGameStation/HYDRA user community and more!

For technical support, sales, general questions, share feedback, please contact Nurve Networks LLC at:

support@nurve.net / **nurve_help@yahoo.com**

Table of Contents

HYDRA SD Max+128K Card User Manual

1.0 HYDRA SD Max Storage Card Manual Overview

Welcome to the user manual for the **HYDRA SD Max Storage Card**. The **HYDRA SD Max Storage Card** or **"SD Max"** for short, enhances the functionality of the HYDRA by adding the ability to read and write standard SD (Secure Digital) cards. The hardware interface is facilitated by a standard HYDRA expansion card with a SD card slot built in as well as a 128K EEPROM for program storage. The hardware interface exports lines for both **SPI** (serial peripheral interface) mode as we as proprietary **SD mode** (assuming you have a license).

Thus, you can use simple SPI routines to talk to the SD card or if you are a licensed SD card developer you can access the card at high speed. Also, you can always find "educational" notes on internet on how to access the high speed modes and then write code to do so, but if you every produce a consumer product with the high speed protocol you better have a license, or you might get a visit from the "SD Police". However, for the demos and drivers within we will stick with the free ***SPI modes*** of SD card operation.

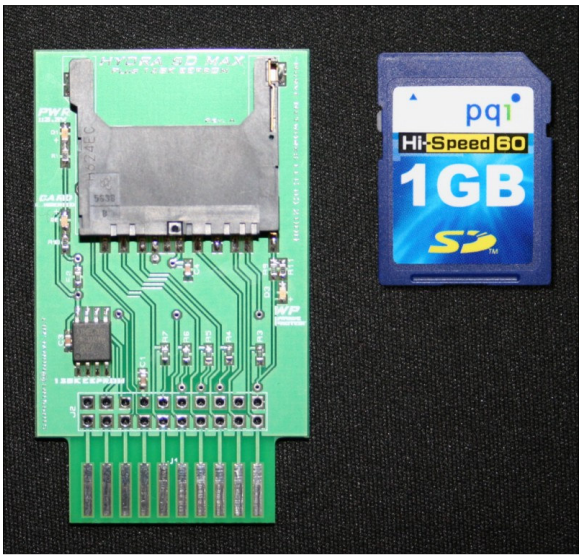
The HYDRA SD Max Storage Card not only has the SD card interface on it, but comes with a standard 128K serial EEPROM chip on-board, so that you can load your applications onto the 128K EEPROM and boot from the card as well. Thus, the card doubles as a 128K EEPROM memory expansion card when you're not interested in using the SD slot.

NOTE	The high speed modes of the SD card interface is ***not*** free to use. There are two modes of operation; **SPI mode** and **SD mode**. In SPI mode there are no royalties, but in SD modes you must pay a fee to license to use (there are two version **1-bit** and faster **4-bit** protocol). Read more about it here: http://en.wikipedia.org/wiki/Sd_card. However, 99% of hobby or educational products simply use the SPI mode which is more than adequate.

There is a lot you need to know to write software yourself to access SD cards. From the bottom up, first, there is the electrical interface to the SD card itself which is a number of signals and power, then the actual communications to the device is based on the **SPI** (serial peripheral interface) which is a **3-line serial protocol** (data in, data out, clock) in SPI mode of course. That's how you talk to the SD card. Then on top of that you need to send the SD card commands in **SD protocol format**. This gives you access to the card's features, but not the file system. You can read and write sectors, but they are in raw form. If you want to really take advantage of the SD card then you have to emulate the **DOS FAT16** (or FAT32) file system standard. Alas, you have to write a version of the FAT16 file system on the HYDRA to access the card!

This is a lot of work no doubt, but hopefully this manual will give you insight into how to do this yourself. And of course, I will provide pre-made drivers and an API written in **SPIN** for ease of understanding along with a little menu driven demo. Nevertheless, I highly recommend that you try and conquer everything from the electrical interface, the SPI protocol, the SD card protocol and FAT16, so you know every single bit (no pun intended) of operation of these amazing little storage devices. The ability to hold 4Gigs in something the size of a stamp is amazing when you think about it. Assuming a page of text is about 4000 characters and the average book 350 pages, that means that a 4Gig SD card can hold 4,000,000,000 / 350 * 4000 = 2857 books!!!! So you could easily store everything we know about math, physics, computers, electronics, chemistry, biology easily along with arts and a good helping of fiction if it was the end of the world and you had to leave the planet and could only take what you could fit in your pockets!

Figure 1.0 – The HYDRA SD Max Storage Card and Accessories.



2.0 Product Contents

The HYDRA SD Max Storage Card kit consists of the following items as shown in Figure 1.0:

- The HYDRA SD Max Storage Card itself with SD card slot and 128K EEPROM onboard.
- A 1GB SD card pre-formatted with a FAT16 file system (SD card may very from model shown).
- CD-ROM with source, tools, API, etc. (not shown).
- Printed Quick Start Sheet (not shown).

NOTE	The 1GB SD card that comes with the kit is pre-format FAT16 and has a file system on it with a number of binaries for the pre-programmed demo that comes on the SD cards. You can erase and re-format the SD card at any time.

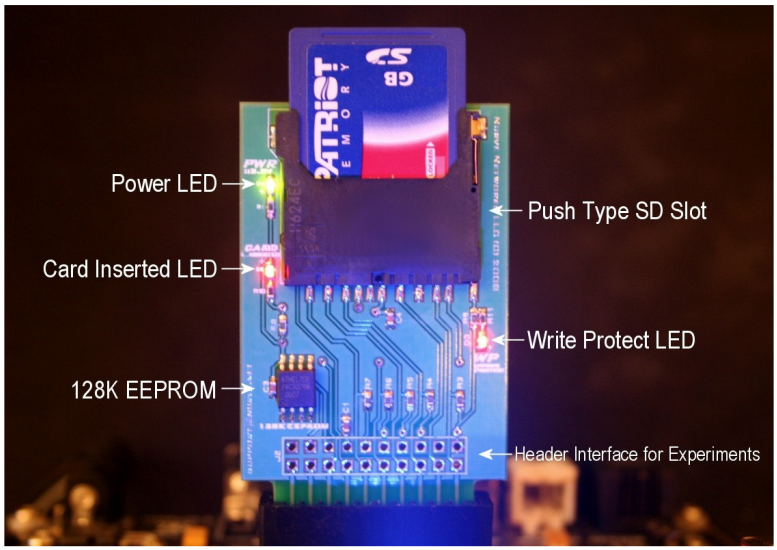
2.1 CD-ROM Contents

The CD-ROM contains the documentation, drivers, demos, and all source code for the SD Max. Additionally, there is a bonus sub-directory with the latest HYDRA demos and software that are user submitted. The CD-ROM layout is as follows:

```
CD_ROM:\        README.TXT
                AUTORUN.SYS
                LICENSE.TXT
                SOURCES\
                        \SD_MENU_IMAGE
                SCHEMATICS\
                DOCS\
                        \DATASHEETS
                        \FAT
                        \SD
                        \SPI
                TOOLS\
                GOODIES\
```

NOTE: “CD_ROM” is your CD-ROM drive letter; “D:”, “E:”, etc.

Figure 2.0 – Annotated close up of the HYDRA SD Max Storage Card.



3.0 Introduction and Quick Start

A close-up of the **HYDRA SD Max Storage Card** shown in Figure 2.0. The SD Max card has a simple friction type front loading standard SD card slot that accepts **"standard"** size SD cards (***mini*** and ***micro*** cards can be used with adapters). The SD Max card also has a 128K serial EEPROM on it, so SD card programs can be stored directly on the card and not need to be loaded separately into the HYDRA's base board 128K EEPROM.

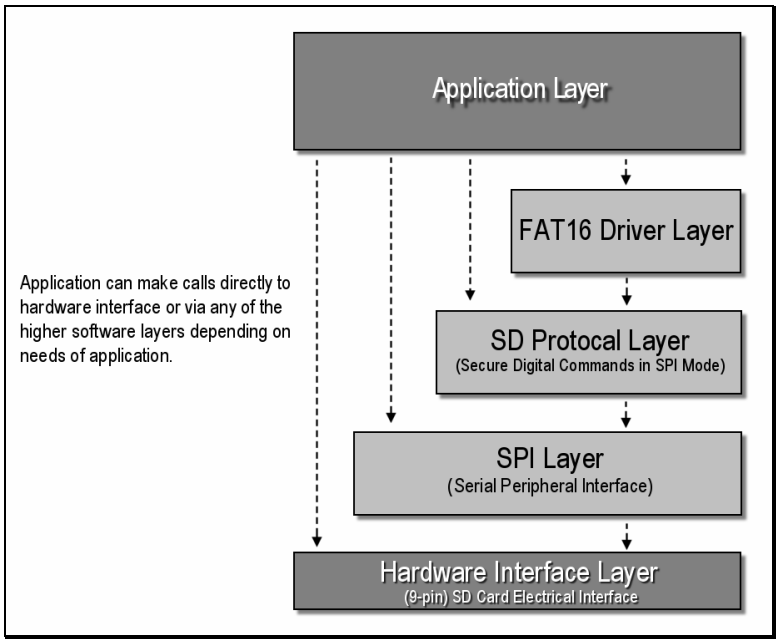
Additionally, the card has 3 LED indicators:

- Power.
- Write Protect.
- Card Inserted.

Both the signals ***write protect*** and ***card inserted*** are actually interfaced to the I/O pins of the HYDRA's expansion slot as well to help determine these if the inserted card is write protected, or that a card is inserted at all. The SD card can be queried as well with ***software techniques*** to determine if its inserted and if so write protected, but having the extra signals makes life a little easier.

The main idea of the SD card is augment the HYDRA with a complete file system for loading and storing large amounts of data. The SD card format is perfect since the hardware interface is very modest (only 3 lines needed in SPI mode). And the speed is quite respectable at **25MHz** even in **license free SPI mode**. The circuit design (discussed in the next section) supports high speed 4-bit modes as well for users that want to experiment with the other SD card modes for non-commercial applications.

Figure 3.0 – A simplified illustration of the SD Max software model.

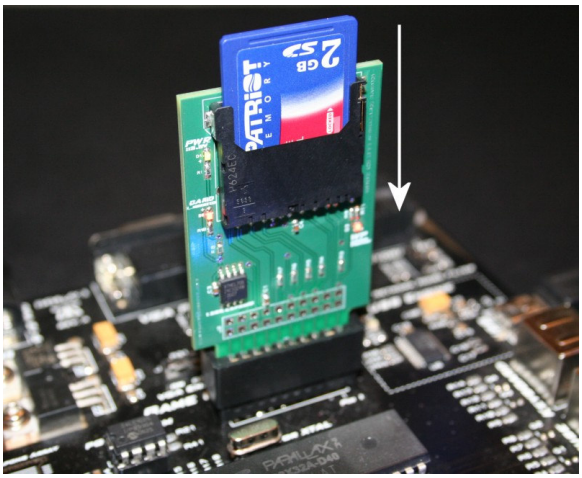


The software interface to the SD card is a layered model as shown in Figure 3.0. At the lowest level there is a **SPI driver** which does nothing except send and receive bytes to and from the SD card. On top of this protocol is the **SD SPI mode protocol** which can be used to completely control the SD card as well as read/write **"sectors"**. But, to interface the cards with the PC and be able to transparently swap the card from the PC to the HYDRA, a **FAT16** file system is needed since that's what the PC uses to format and access the SD card. Thus, we need yet one more layer on top of the SD card protocol that implements FAT16 which is no easy feat.

NOTE	FAT16 is a file system used on DOS and Windows PCs, originally introduced in 1987 as a successor to the FAT12 system for floppy drives. FAT16 is simply a method of organizing a hard drive or floppy disk into sectors, clusters, and a directory system that allows a file system to be efficiently developed to access files, read and write. More on the FAT16 system later in the manual. A good overview can be found here: http://en.wikipedia.org/wiki/File_Allocation_Table.

The SD card can be used for all kinds of applications from simple file storage to virtual memory and advanced media applications. Later in the summary some ideas about cool things to do with the SD card will be discussed.

Figure 4.0 – Inserting the HYDRA SD Max card into the HYDRA.



3.1 Quick Start Guide

The HYDRA SD Max card's 128K EEPROM is pre-loaded with a little demo program that lists a number of demo, programs, games, and applications that you can scroll thru and select and load into the HYDRA and run as shown in Figure 5.0 below. If you are an old Atari, Commodore 64, or Apple programmer this will look very familiar to the old loaders that used to boot and give you a selection of games to play!

This functionality is accomplished by an SD card reader driver coupled with software that allows Propeller 32K EEPROM ***"images"*** with extension **.EEPROM** to be loaded directly into SRAM and then the Propeller to be re-booted. A binary image is simply a 32K byte image that is the result of compiling a standard SPIN/ASM program.

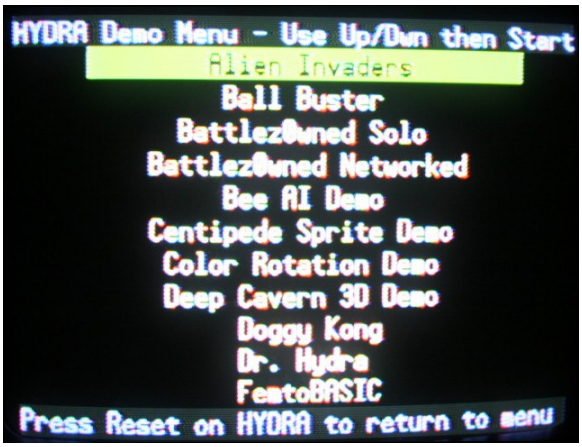
The demo binary images that the demo uses are pre-loaded onto the SD card's root directory, so all you have to do is insert the SD card itself into the HYDRA SD Max card slot and then insert the HYDRA SD Max card into the HYDRA itself. This is shown in Figure 4.0. Of course, you can hot slot the SD card itself (as well as the HYDRA SD Max card), thus, first you can insert the HYDRA SD Max card into the HYDRA and then insert the SD card into the SD card acceptor on the HYDRA SD Max card. Here are the steps to play with the pre-loaded demo and games on:

- ✓ **Step 1.** With the HYDRA hooked up to power, TV and the PC, simply insert the SD Max card into the expansion slot facing the front of HYDRA as shown in Figure 4.0, be careful not to force it. The HYDRA can be on or off. Make sure you have the game controller in the ***left*** controller port of the HYDRA.

- ✓ **Step 2.** Turn the HYDRA on and/or reset it and the card should reset and boot up the test suite. You will see the power LED on the top-left of the card light up as shown in Figure 4.0. If the SD Max card doesn't boot, leave the power on and simply remove and re-insert the card to get a better insertion connection and try hitting reset.

- ✓ **Step 3.** You should see the demo menu on screen as shown in Figure 5.0, use the up/down directional arrows on the game controller to scroll thru the demos, once you find something that sounds interesting press the Start button, the demo will load. When you want to try another demo, simply reset the HYDRA with the Reset button.

Figure 5.0 – The pre-loaded SD loader menu.



If you don't see the menu or the menu shows no programs available to load then either the pre-programmed demo program on the 128K EEPROM needs re-programming, or the SD card needs re-flashing. Refer to the sections below for steps to restore your product.

3.1.1 Re-programming the HYDRA SD Max Card with the Menu Demo Program

With the HYDRA SD Max card inserted into the HYDRA, simply load the program on the CD named **MENU_001.SPIN** located here:

CD_ROM:\sources\menu_001.spin

and write it to the 128K EEPROM on the HYDRA SD Max card with **F11** from the Propeller IDE. This should refresh the damaged program.

3.1.2 Re-formatting the SD Card with the Demo Binary Images

If the demo programs on the SD card get damaged somehow, you simply need to restore the SD card, which is trivial. You will need an SD card reader plugged into your PC. Then simply plug the SD card into the reader, open the card up and then inspect the files. You should see a long list of files with various names (each with the **.EEPROM** file extension) and a file named DIR.TXT that is used as an internal **"directory"** for the menu program. If you don't then chances are the files got fragged, so you need to re-fresh them. Here are the steps:

Step 1: Format the SD card FAT16. Perform a full format, ***not*** a quick format.

Step 2: Locate the folder on the CD in the **\Sources** directory named **\SD_Menu_Images** located here:

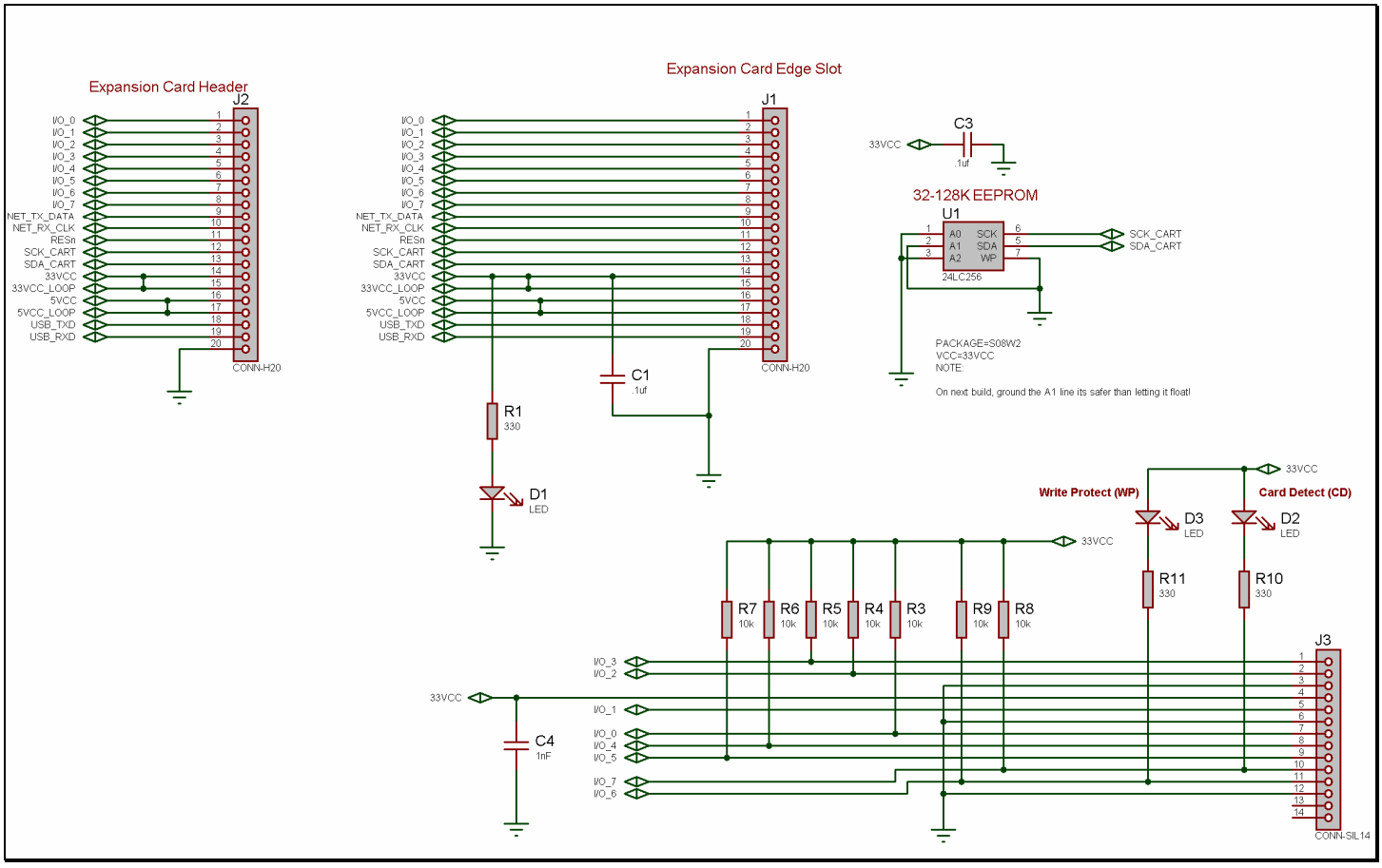
CD_ROM:\Sources\SD_Menu_Images

Enter the directory and ***select all*** the files and copy them into the SD card. This should re-fresh the card and you are back in business. Note: You will need to do this anytime you want to play with the SD card menu demo. Since the demo assumes the card is freshly formatted and has a root copy of all the files on it.

The menu demo is just a taste of what's possible with the HYDRA SD Max card. You can load potentially thousands of demos and languages into the HYDRA with a single SD card and never have to remove the card from the HYDRA!

NOTE	The menu program and underlying drivers are based on the hard work of **Mike Green's** Femto BASIC and drivers developed by **Tomas Rokicki** of Radical Eye Software, they are free for use and sources available on the CD as well as the Propeller/HYDRA forums and object exchange at www.parallax.com. The formal HYDRA SD Max driver API written by yours truly is completely in SPIN for educational reasons, thus once you start really hacking away at the SD card you might want to use the faster lower level drivers by Mike Green and Radical Eye Software or write your own.

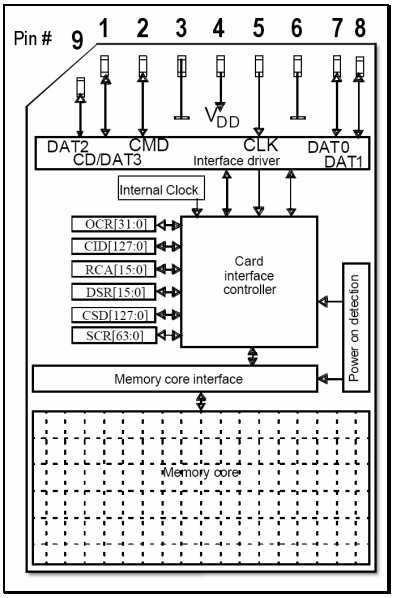
Figure 6.0 – The HYDRA SD Max Storage Card's electrical design.



4.0 Circuit Design, Electrical and Mechanical Interface

The HYDRA SD Max card is a very simple design since all of the work is inside the SD card itself. The SD card has a complete microcontroller, memory, and of course the flash storage all tucked away inside. All we really need to do is interface to the SPI bus of the SD card via a mechanical SD card holder and that's about it. If you look at the design in Figure 6.0, you will see that the HYDRA's expansion bus interface is used to interface to the SD card's interface. To make things simple the I/O lines are used for the interface, but in a pinch the networking and I2C lines connected to the HYDRA's EEPROM chips could be used, but then it would be hard to implement a complete SD interface. If you recall in the introductory paragraphs on the SD card there are a few different modes of operation; ***SPI***, ***1-bit SD***, and ***4-bit SD*** modes. The various modes use different signals on the SD card's edge connector, thus, to support future expansion and more advanced modes the HYDRA's IO expansion bus is used to implement a complete interface. But, if all you needed was SPI mode then 3-lines is all you need to do it.

Figure 7.0 – Standard SD card pin labeling (contact side).



4.1 Electrical Interface Design

Referring to Figure 6.0, every single line **I/O_0** to **I/O_7** is used by the design. However, in SPI mode only a subset of these lines is necessary. However, to support future expansion into the SD modes of operation, the design connects all important lines to the HYDRA's expansion interface. Take a look at Figure 7.0, it depicts the actual SD card physical interface (if you were looking at the gold fingers). Notice the numbering is left to right, with the last pin, number 9 on the 2nd row, bottom left. You will also notice on Figure 6.0 that the I/O lines all have pull-ups on them. This isn't 100% required, but in 1 and 4-bit SD modes its definitely suggested by the SD specification, so we have added them in the design.

Also, the additional ***Write Protect*** and ***Card Detection*** signals (I/O_6 and I/O_7) are pulled up as well as ran thru LEDs for visual indication. The SD card connector will short these lines to ground when the card is inserted or the write protect is unlocked (see Figure 8.0(b)), thus the Propeller can read these event on the I/O lines and HIGH or LOW signals respectively and figure out if a card is in the slot and if its write protected. It's important to note that write protect and card detection are NOT part of the SD card's electrical interface, but simply added to each SD card mechanical slot by the manufacturer. And therefore, each has their own electrical interface and operation. Thus, to clarify:

Card Inserted Logic

- I/O_6 = HIGH when card ***not*** inserted, LED off.
- I/O_6 = LOW when card ***is*** inserted, LED on.

Write Protect Logic

- I/O_7 = HIGH when card's write protect ***is*** locked (read only), LED off.
- I/O_7 = LOW when card's write protect is ***not*** locked (read/write), LED on.

That's about the extent of the electrical design, very simple as you can see. Now, let's take a look at the actual signals themselves and see what's what. Table 1.0 lists the complete electrical interface for all modes of operation of the SD card.

Table 1.0 – SD card interface lines used.

Pin	SD 4-bit mode		SD 1-bit mode		SPI mode	
1	CD/DAT[3]	Data line 3	N/C	Not Used	CS	Card Select
2	CMD	Command line	CMD	Command line	DI	Data input
3	VSS1	Ground	VSS1	Ground	VSS1	Ground
4	VDD	Supply voltage	VDD	Supply voltage	VDD	Supply voltage
5	CLK	Clock	CLK	Clock	SCLK	Clock
6	VSS2	Ground	VSS2	Ground	VSS2	Ground
7	DAT[0]	Data line 0	DATA	Data line	DO	Data output
8	DAT[1]	Data line 1 or Interrupt (optional)	IRQ	Interrupt	IRQ	Interrupt
9	DAT[2]	Data line 2 or Read Wait (optional)	RW	Read Wait (optional)	NC	Not Used

Referring to Table 1.0, depending on the mode of operation the electrical interface of the SD card can change meaning. For example in SD 4-bit mode pin 8 is data line 1, but in SD 1-bit mode its an interrupt line. Therefore, by exporting all the lines to the HYDRA's expansion interface we can write drivers to support the 1-bit and 4-bit modes if we wish. However, we are more interested in the SPI mode which is shown on the right hand side of the table. In this mode, the interface is a straight SPI interface consisting of pin 1,2,3,4,5,6, and 7. Pin 8 and 9 are not needed. Also, note that pins 3 and 6 are ground and 4 is power, so in reality there are only 4 signal lines needed for SPI mode (CS, DI, DO, SCLK).

Nevertheless, we want to support all these modes for experimentation, so we need a connection to every single pin. If you count up all the signals there are a total of 6 signal lines and 3 power lines. Now, since the HYDRA's expansion bus has 8 available lines, we still have 2 more signals we can send thru the interface which is perfect for the ***Card Detect*** signal and the ***Write Protect*** signal which are implemented on the card slot itself by means of little switches implemented on the connector itself. That is, when you insert the SD card a short is made which is senses as well as the state of the write protect tab. But, these signals are not part of the SD card interface and are solely at the discretion of the manufacturer of the particular SD card slot mechanical you choose for your design. But, we will get to that in a moment.

Right now, let's take a look at the interface between the SD card signals and the HYDRA expansion port itself. Basically, we have 6 data signals from the SD card plus 2 extra signals from the mechanical interface itself (indicating card insertion and write protect tab state) along with the power and grounds. Thus, we have to decide what signal to connect where on the HYDRA's I/O_0 to I/O_7 signal lines. This is mostly arbitrary, but I matched them to the most common connections that people have been using when interfacing SD cards to the HYDRA themselves. Table 2.0 below shows the final electrical interface from the SD card to the HYDRA expansion interface.

Table 2.0 – SD card interface to HYDRA expansion interface mapping.

SD Card Pin	Function	Name (EXP/PROP/LOG)
SD 1	(**CS**/DAT3 - chip select/DAT3)	I/O 3 (4/24/P19)
SD 2	(**DI**/CMD - data in/command)	I/O 2 (3/23/P18)
SD 3	GND	GND (20)
SD 4	3.3V	3.3V (14)
SD 5	(**CLK**/SCK – clock/serial clock)	I/O 1 (2/22/P17)
SD 6	GND	GND (20)
SD 7	(**DO**/DAT0 - data out/DAT0)	I/O 0 (1/21/P16)
SD 8	(DAT1/IRQ)	I/O 4 (5/25/P20)
SD 9	(DAT2/NC)	I/O 5 (6/26/P21)
Extra Mechanical SD card holder signals not part of the SD interface		
SD 10 [(1)]	(CD/Card Detect)	I/O 7 (8/28/P23)
SD 11 [(1)]	(WP/Write Protect)	I/O 6 (7/27/P22)

Note 1: SD 10 and 11 are not part of the SD interface, but simply pins that happen to be on the particular SD card mechanical chosen for the HYDRA SD Max (AVX Inc. makes them). Most SD card slots have similar signals.

The table is a little tricky to read especially in the right most column, so let's try and example. Let's trace SD card pin 7 which is the **D0** or **Data Out** line. Starting from the left most column, SD 7's function is **D0** or **Data Out**. Meaning, in 4-bit mode its **D0** of the 4-bit data, or in 1-bit mode it's the single **"Data Out"** line. Then moving to the right column, each entry is in the format **"signal name"** followed the three numbers in the order (**EXP**ansion port pin, **PROP**eller pin, **LOG**ical Propeller signal name).

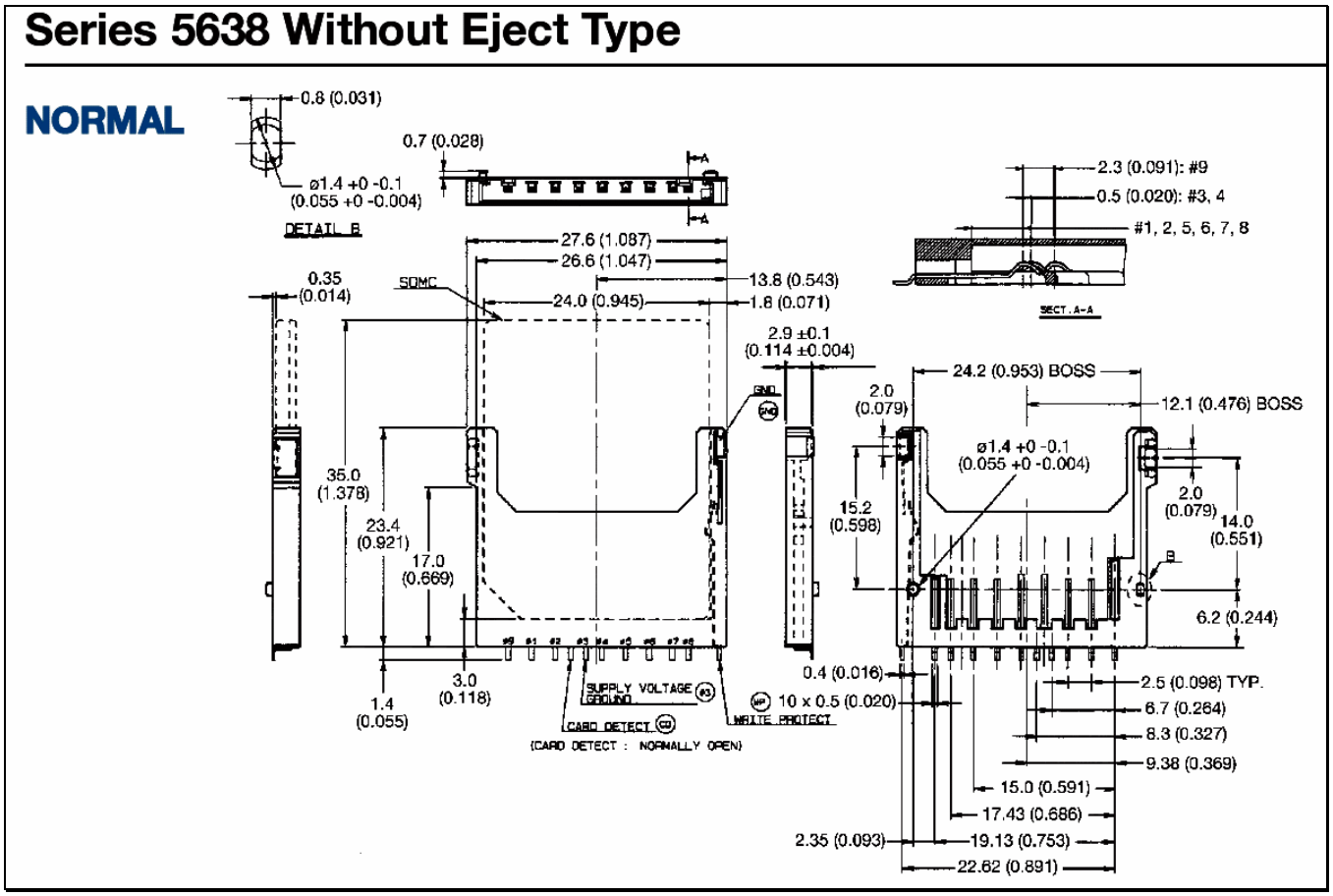
Thus, the SD card's pin 7 is connected thru the HYDRA expansion port's pin 1 which in turn is connected to the Propeller chip at **pin 21** which is named **P16** on the Propeller chips I/O interface. A little confusing, but that's what happens when you go from interface to interface to the chip to the final internal Propeller signal name. At the end of the day, as a programmer all you need to know are the Propeller's I/O pin assignments which are as follows:

```
' SPI (serial peripheral interface) interface pins to SD card
spiDO  = 16 ' SD data out
spiCLK = 17 ' SD clock
spiDI  = 18 ' SD data in
spiCS  = 19 ' SD card select

' these two signals aren't really part of the SPI interface, but are defined in this block of code
sdWP   = 22 ' write protect line
sdCD   = 23 ' card detect line
```

Next, let's take a look at the actual mechanical SD card holder used in this project. I selected one from AVX Corporation based on price, interface, and availability.

Figure 8.0(a) – The HYDRA SD Max SD card connector from AVX Corporation.



4.2 Mechanical Interface

All SD cards must plug into a physical connector that is mounted to the PCB itself. There are hundreds of manufactures that make these connectors and selecting one is more subjective than objective in many cases. However, some things that you should look out for if you decide to design an SD card reader are:

1. Is the connector a ***normal*** or ***reverse*** type?
2. Does the connector have a spring loaded release? Nice, but expensive.
3. Is the connector plastic or metal or have adequate shielding?
4. Cost, cost, cost! And did I mention cost?

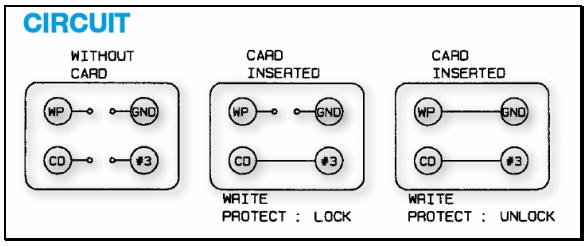
TIP	Normal or reverse connector indicates which way the SD card goes into the connector. ***Normal*** means label facing the ***front*** and the contacts facing the PCB. Reverse means the opposite. This is important depending on which way the PCB is oriented in the final product. In our case, the PCB of the SD Max card plugs face outward on the HYDRA, so when you plug in the SD card it faces outward as well.

In the case of the SD Max card we don't need all the bells and whistles, so we decided on a standard normal connector, plastic, without a spring eject mechanism (which is probably a good idea since its just one more thing that can break). The connector itself is manufactured by AVX Corporation and you can find the data sheet on the CD here:

CD_ROM:\docs\datasheets\avx_sd_mech_5638.pdf

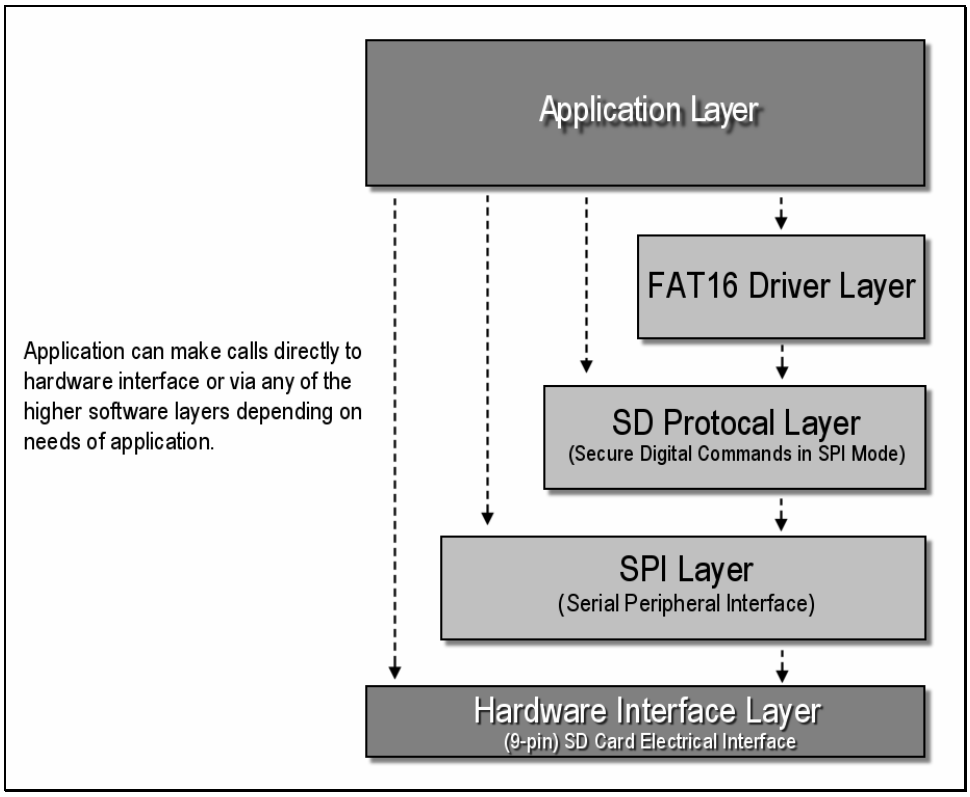
Figure 8.0(a) above shows and excerpt from the datasheet illustrating the mechanical specifications of the connector. Notice that there are a lot of specs and this is one of the biggest challenges when making an SD card reader; designing the PCB to accept the connector is no easy task due to the small tolerances. Also, notice the additional annotation relating to the Write Protect and Card Inserted circuits as shown in Figure 8.0(b).

Figure 8.0(b) – Card connector from AVX Corporation showing Write Protect and Card Inserted logic.



These extra signals are available on the SD connector, but not part of the SD specification, thus they aren't in any specific location manufacturer to manufacturer. Therefore, when designing SD card PCBs you will find that if you change the SD card connector, you will have to redesign the PCB footprint quite a bit.

Figure 9.0 – HYDRA SD Max software / hardware model.



5.0 Interfacing to the HYDRA SD Max Card

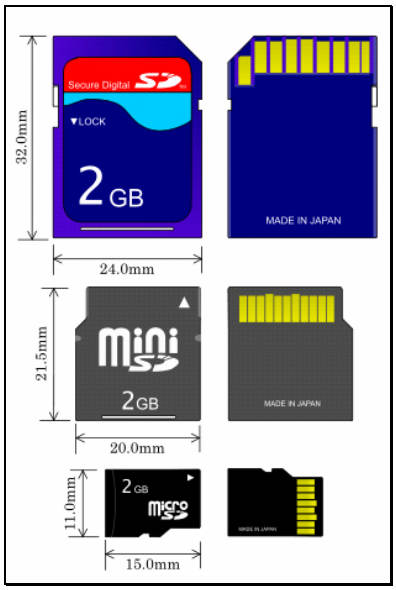
In this section we'll discuss interfacing to the SD card at all levels; electrical to FAT16. Referring to Figure 9.0 above you can see that the SD card software / hardware model looks a little like the 7-layer OSI model for networking. There is the **application layer** at the top. Then under it is the **FAT16 file system** which facilitates interfacing the SD card with the PC (optional). Next down is the **SD card protocol layer** which is what the SD card speaks and gives access to the internal controller in the SD card as well as raw sector reads and writes. This is followed by the actual communication layer to the SD card which is a **SPI interface** consisting of 4-lines; data out, data in, clock, and chip select. Then finally where the rubber metal the concrete are the **electrical connections** to the Propeller chip itself that we can toggle via port I/O commands. Thus, we have to implement every single one of these layers, so we have our work cut out for us!

Our goal is to become familiar with writing a SPI driver all the way up to a DOS/Windows compliant FAT16 file system. There is a lot of material to cover on the these subjects and each warrants a lot of information, so I suggest referring to the related links and documents on the CD for more in depth coverage. Take a look in these sub-directories for a number of relevant documents:

CD_ROM:\DOCS
\FAT - Documents about the FAT file system.
\SD - Documents about the SD card protocol and design in general.
\SPI - Documents about the SPI interface and protocol.

But, before we begin let's take an aside briefly and talk about some of the history and design of the SD card protocol.

Figure 10.0 – SD card physical packaging examples.



5.1 SD Card Overview & History

The SD card was originally invented in 1999 by a collaboration between **Panasonic**, **Toshiba**, and **SanDisk** (lead developer) to compete with **Sony's *Memory Stick*** technology which was proprietary and a closed standard. The secure digital aspect of SD card simply has to do with the added functionality supporting encryption that was missing in the **MMC** cards beforehand (**M**ulti **M**edia **C**ards) that the SD card was supposed to replace. Encryption was requested since the main purpose of the SD card was to hold media data and media providers wanted encryption and some kinds of digital rights management.

In any event, the SD has become the most popular solid state mass storage device along with Compact Flash storage devices. Due to this success and the continually shrinking size of consumer electronics new, smaller size, and faster SD cards are now available. Take a look at Figure 10.0 to see the ***mini*** and ***micro*** SD card footprints. The mini size is cute, but the micro is ridiculously small, similar to a cellular phone **SIM** (**S**ubscriber **I**dentity **M**odule) card. The size of the cards doesn't effect the storage capacity. Also, each smaller footprint is designed with the exact same electrical and signal interface, just smaller and adapters are available to plug each smaller size into each larger size, so even if you have a normal SD card reader, you can use a micro SD card with a little adapter.

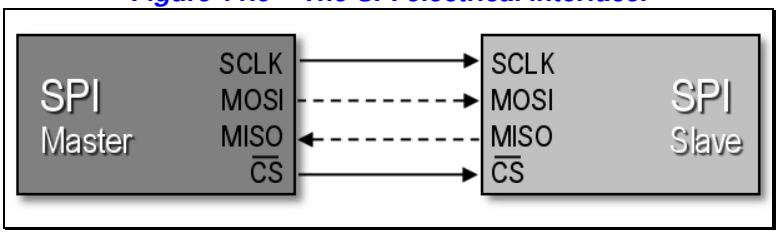
Architecturally, each SD card has a microcontroller inside along with the actual flash memory storage area. Therefore, when communicating with an SD card you are more or less communicating with another computer. Thus, SD cards are rather complex pieces of technology that abstract the memory interface and the I/O interface. For example, before SD cards there were all kinds of memories; RAMs, ROMs, FLASH, EEPROM, etc. but the problem was that if you wanted to use one of these memories then you had to build a physical interface to them. SD cards removed this problem by separating the interface from the electronics/memory, so that a user could use a standardized set of communication lines to talk to a mass storage device. This is the genius of the SD card design. No matter what's inside the SD card; NAND flash, NOR flash, holographic memory or little elves, its always the same from the electrical point of view and the programmers interface. That is, 9 signals that always mean the same thing.

From a programmers point of view the SD card acts like a number of registers and a memory. Communication to the SD card is done thru a serial/parallel interface by issuing commands and sending and receiving data. The memory in an SD card is arranged as a number of sectors (512 bytes each usually) and currently SD cards max out at about 4GB of memory, but 8GB are available. The communication speeds of SD cards depend on the version of the card; 1.0, 1.01, 1.1, or 2.0. Most manufacturers stick with **version 1.1** specs which run at about 10-20 megabytes per second. The speed measuring of SD cards is based on CD ROM standard. Where **1X** CD speed is about **150 kB** (kilobytes) per second. Therefore, an SD card that runs at 66X speed means it runs at 66 x (150 kB) = 9.9MB per second. Currently, 66X and 133X are commonly available with even higher speeds if you're will to pay the money. However, these speeds are only attainable in licensed true SD protocol modes, not the free SPI mode (which we are using).
Finally, SD cards are nothing more than a collection of sectors, but to use with PCs, SD cards are setup or "formatted" to mimic the target file system of the PC they are connected to. Typically, this is FAT16 or FAT32 both Microsoft standards. Thus, when using an SD on the HYDRA if you want to plug it into the PC you must format in one of these standards.

Preferably **FAT16** since it's the only format that the drivers support. Of course, you can code FAT32 drivers if you like yourself.

Summing up, SD cards are memory ***plus*** microcontrollers. They are interfaced via a ***9-pin*** electrical interface which is always the same. Internally, they are organized as a collection of ***sectors*** that the programmer can access and read and write to via issuing commands to the internal microcontroller. If desired, one can layer a file system on top of the raw sectors like FAT16 or whatever is desired, but you the programmer must do this, SD cards have ***no built in*** internal support for a file system. The actual electrical communication to SD cards is performed via a 1-bit or 4-bit protocol. However, there is also a standard SPI protocol that can be used which is straightforward to implement. Therefore, the best place to start talking with SD cards is to build the SPI software and work our way up. In the following sections you will see SPIN code snippets excerpted from the API library, don't worry if they don't make complete sense since we will cover the API library in detail later in the manual, but for now if you see a global, constant that isn't defined, trust that it is and you will learn about it later.

Figure 11.0 – The SPI electrical interface.



5.2 SPI Bus Basics

SPI stands for **Serial Peripheral Interface** originally developed by **Motorola**. Its one of two very popular modern serial standards including **I²C** which stands for **Inter Integrated Circuit** by **Phillips**. SPI unlike I²C (which has no separate clock) is a clocked synchronous serial protocol that supports full duplex communication. However I²C only takes **2 wires** and a ground. Where SPI needs **3 wires**, ground, and potentially chip select lines to enable the slave devices. But, SPI is much faster, so in many cases speed wins over and the extra clock line is warranted. The advantage of I2C is that you can potentially hook hundreds of I²C devices on the same 2-bus lines since I²C devices have addresses that they respond to. SPI bus protocol on the other hand requires that every SPI slave has a chip select line.

Figure 11.0 shows a simple diagram between a **master** (left) and a **slave** (right) SPI device and the signals between them which are:

- **SCLK** - Serial Clock (output from master).
- **MOSI/SIMO** - Master Output, Slave Input (output from master).
- **MISO/SOMI** - Master Input, Slave Output (output from slave).
- **SS** - Slave Select (active low; output from master).

Note: You might find some devices with slightly different naming conventions, but the idea is there is data out and data in, a clock, and some kind of chip select.

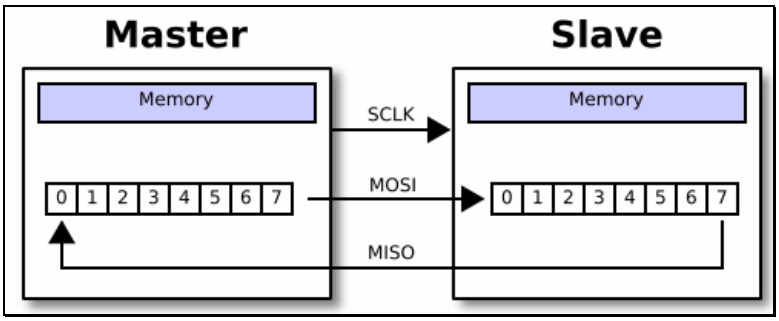
If you refer back to the SD Max interface signals then you will see that the SD card signals have slightly different names. Assuming the HYDRA is the Master and the SD card is the slave, the mapping of SPI signals are shown in Table 3.0 below:

Table 3.0 – Mapping of SD card signals to official SPI signal names.

SD Card Pin	SD Name	SPI Signal Name	Function
1	CS	SS	Chip select active low.
2	DI	MOSI	Master out, slave in.
5	CLK	SCLK	Clock signal.
7	DO	MISO	Master in, slave out.
Note: *Power and ground signals of course need to be connected as well.*			

SPI is very fast since not only is it clocked, but it's a simultaneous ***full duplex*** protocol which means that at you clock data out of the master into the slave, data is clocked from the slave into the master. This is facilitated by a transmit and receive bit buffer that constantly re-circulates as shown in Figure 12.0.

Figure 12.0 – Circular SPI buffers.



The use of the circular buffers means that you can send and receive a byte in only 8 clocks rather than clocking out 8-bits to send, then clocking in 8-bits to receive. Of course, in some cases the data clocked out or in is "dummy" data, meaning when you write data and you are ***not*** expecting a result the data you clock in is garbage and you can throw it away. Likewise when you do a SPI read, typically you would put a $00 or $FF in the transmit buffer as dummy data since something has to be sent and it might as well be predictable.

Sending bytes with SPI is similar to the serial RS-232 protocol, you place a bit of information on the transmit line, then strobe the clock line (of course RS-232 has no clock). As you do this, you also need to read the receive line since data is being transmitted in both directions. This is simple enough, but SPI protocol has some very specific details attached to it about ***when*** signals should be read and written that is, on the rising or falling edge of the clock as well as the polarity of the clock signal. This way there is no confusion about edge, level, or phase of the signals. These various modes of operation are logical called the SPI mode and are listed in Table 4.0 below:

Table 4.0 – SPI clocking modes.

Mode #	CPOL (Clock Polarity)	CPHA (Clock Phase)
0	0	0
1	0	1
2	1	0
3	1	1

Mode Descriptions

Mode 0 – The clock is **active** when **HIGH**. Data is **read** on the **rising edge** of the clock. Data is **written** on the **falling edge** of the clock (default mode for most SPI applications).

Mode 1 – The clock is **active** when **HIGH**. Data is **read** on the **falling edge** of the clock. Data is **written** on the **rising edge** of the clock.

Mode 2 – The clock is **active** when **LOW**. Data is **read** on the **rising edge** of the clock. Data is **written** on the **falling edge** of the clock.

Mode 3 – The clock is **active** when **LOW**. Data is **read** on the **falling edge** of the clock. Data is **written** on the **rising edge** of the clock.

Note: Most SPI slaves **default to mode 0**, so typically this mode is what is used to initiate communications with a SPI device.

Figure 13.0(a) – SPI timing diagrams for clock phase polarity (CPHA=0).

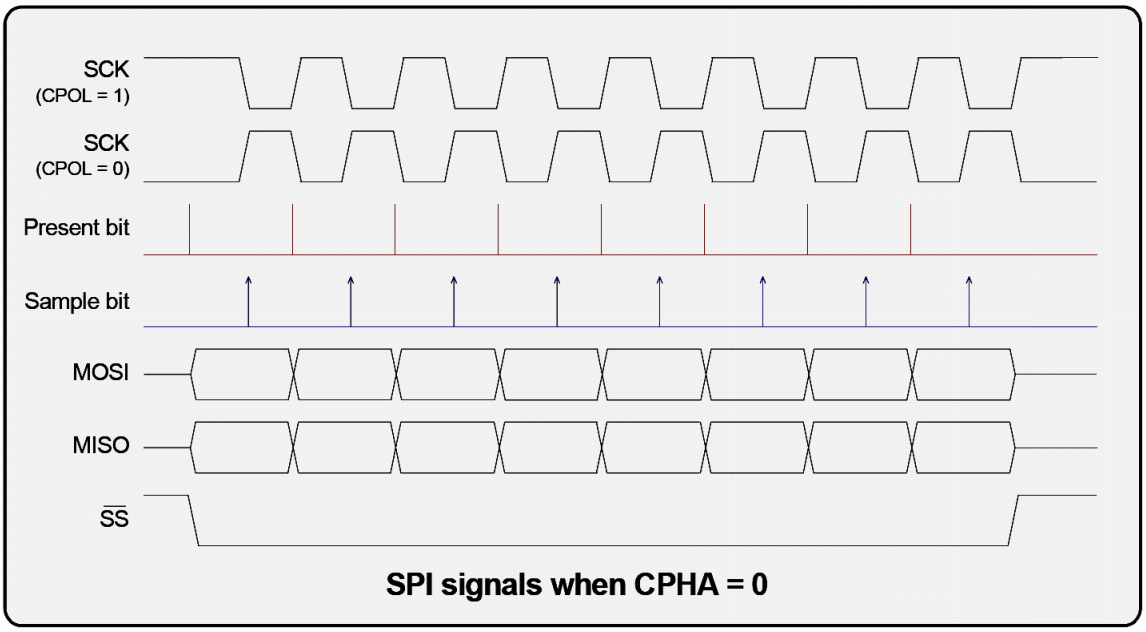
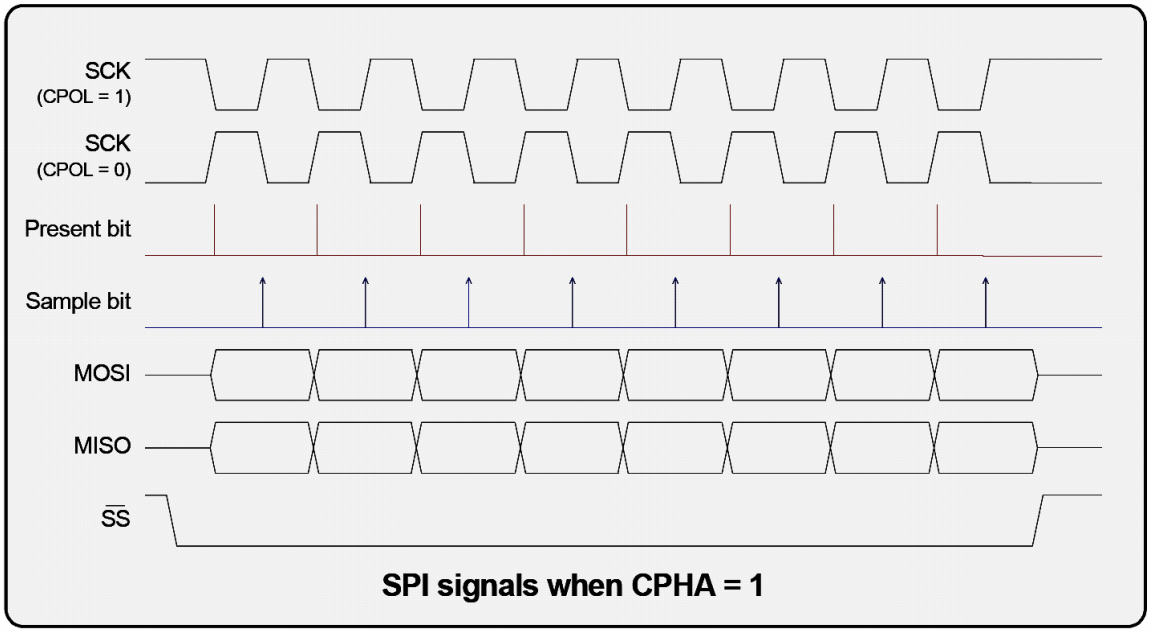


Figure 13.0(b) – SPI timing diagrams for clock phase polarity (CPHA=1).



5.2.1 Basic SPI Communications Steps

Figures 13.0(a) and (b) show the complete timing diagrams for all variants of **clock polarity** (CPOL) and **clock phase** (CPHA). You must adhere to these timing constraints during communications. In most cases, you will use **mode 0** since it's the default that most SPI devices boot with. On the HYDRA there is no support for SPI buses, so we must bit bang the protocol ourselves by toggling the I/O lines connected to the SPI interface of the SD card. Once again these signals and Propeller I/O bits are:

```
spiDO  = 16 ' SD data out from SD (input to HYDRA)
spiCLK = 17 ' SD clock (output from HYDRA)
spiDI  = 18 ' SD data in (output from HYDRA)
spiCS  = 19 ' SD card select (output from HYDRA)
```

Where the signals are from the perspective of the SD card (the slave in SPI terminology). For example spiDO is data out from the SD card, but data in to the HYDRA. Moving on, the first step to communicating with the SD SPI interface is to setup the I/O directions. In SPIN, the code looks like (excerpt from SD library):

```
PUB SPI_Init(mode)

' DESCRIPTION: this function initializes the hardware interface to the SPI bus. on the HYDRA SD MAX card
' (as with most SPI buses) there are only 4 signals; clk, data in, data out, and chip select
'
' INPUTS: mode - controls the phase of the clock and the logic levels
' mode 0 - default support
' mode 1 - not supported
' mode 2 - not supported
' mode 3 - not supported
'
' OUTPUTS: none
' ------------------------------------------------------------------------------

  ' set directions of SPI interface, be careful not to clock the SPI interface accidentally
  ' also, data in and data out are referenced from TARGET, so DI means the input to the SPI
  ‘ device which would be
  ' an OUTPUT from the propeller
   OUTA [ spiDI ]  := 0 ' set output LOW
   DIRA [ spiDI ]  := 1 ' set "data in" to output

   DIRA [ spiDO  ] := 0 ' set "data out" to input

   OUTA [ spiCLK ] := 0 ' set output LOW
   DIRA [ spiCLK ] := 1 ' set "clock" to output

   OUTA [ spiCS ]  := 1 ' set output HIGH (de-select the device)
   DIRA [ spiCS ]  := 1 ' set "chip select" to output

' end SPI_Init
```

Once the I/O interface is initialized then you can clock in/out data to the SPI bus. Here’s a sample function to transmit and receive (excerpt from SD library):

```
PUB SPI_Send_Recv_Byte(spi_data8) | spi_result8, index

' DESCRIPTION: this functions sends spi_data8 and receives spi_result8, remember all spi operations are ‘
‘ circular, so when a byte is sent a byte is received at the same time, you can ignore the sent or received
‘ byte as you wish. For example, to do a read just send the dummy value $FF
' NOTE: assumes caller is controlling the chip select line
'
' INPUTS: spi_data8 - 8 bit data that is going to be sent
' OUTPUTS: returns the 8-bit data received at the same time
' ------------------------------------------------------------------------------
  ' reset result
  spi_result8 := 0

  ' shift 8-bits of data out while shifting in 8-bits of data
  ' data is sent and receiving MSB to LSB, so bit 7,6,5,4,3,2,1,0
  ' assume SPI interface is in mode 0/1, later make more flexible
  repeat index from 0 to 7
    ' place next bit on data in pin to device
    OUTA [ spiDI ] := ((spi_data8 & $80) >> 7)

    spi_data8 <<= 1 ' shift data to left and prepare next bit for transmission

    ' pulse the clock line
    OUTA [ spiCLK ] := 1

    ' data is ready now on output line from device
    spi_result8 <<= 1            ' shift result to left
    spi_result8 |= INA [ spiDO ] ' read data bit and OR into result

    ' finish clock pulse
    OUTA [ spiCLK ] := 0

 ' return result
  return (spi_result8)

' end SPI_Send_Receive_Byte
```

Notice that there is no timing related logic in the function. Since SPI protocol is totally **synchronous** the master in charge of the clock can run the clock as fast (to maximum speed) or slow (static if desired). Also, notice the data is sent out most significant bit (msb) to least significant bit (lsb).

These above two functions are the entire SPI software necessary to communicate with a SPI device! The only detail is that the ***chip select*** (CS) line must be toggled by the application as desired during operations. Later when we cover the API, we will see a couple more functions layered on top that read and write SPI data, but these are nothing but dummy functions that send dummy data or ignore the results for read and write operations respectively. The important point is that byte is always sent and received at the same time with SPI protocol.

END SAMPLE PAGES