CFFA – CompactFlash Interface for Apple II

Manual v2.11 for CFFA version 2.0 by Richard Dreher

Disclaimer of All Liability

Plain English Version:

Do not use this manual or the CFFA Interface board for any mission-critical applications, or for any purpose in which a bug or failure could cause you a financial or material loss. This product was designed to enhance your Apple II computing experience, but may contain design flaws that could inhibit its proper operation, or result in a loss of the data recorded on the storage devices attached to it. When using this product you assume all risks associated with operation or data loss. If these terms are not acceptable, you may return the product for a refund with 90 days of receiving it.

Legalese Version:

Richard Dreher, doing business as R&D Automation, makes no warranties either expressed or implied with respect to this manual or with respect to the software or firmware described in this manual, its quality, performance, or fitness for any particular purpose. All software and firmware is sold or licensed "as is". Any risk of incidental or consequential damages resulting from the use of the information in this manual or the software / firmware / hardware described herein, shall be assumed by the user or buyer or licensee. In no event will R&D Automation be liable for direct, indirect, incidental or consequential damages resulting from any defect in the software / firmware / hardware described in this manual.

R&D Automation reserves the right to make changes and improvements to the product described in this manual at any time and without notice.

Contents

Basic Information

Warranty and Return Information

You may return the CFFA Interface board for any reason within 90 days of receiving it. This should allow you enough time to evaluate the compatibility with your system. I guarantee your CFFA Interface board to be free of defects under normal usage for a period of one year from the date you receive the product. This means that if the board fails, and you have treated it properly, I will repair, replace, or refund your money at my discretion, to be determined by me on a case-by-case basis.

If you want to return the product under warranty, please contact me via E-mail at rich@dreher.net to discuss return arrangements. Include your name and the serial number from the sticker on the back of the board. It is your responsibility to get the product you are returning back to my door. I will not be responsible for lost shipments. Please choose shipping methods and insurance as you deem necessary.

Warnings

You should avoid electrostatic discharge to the CFFA Interface board. Like all electronics devices, static “shock” can destroy or shorten the life span of the CFFA Interface board. Avoid touching the CFFA Interface board after you have walked across the room, especially over carpet, and especially in dry weather.

You should safely discharge yourself before you handle the CFFA Interface board. This can be done by momentarily coming into contact with a grounded piece of metal.

In all cases, please exercise common sense and observe all electrical warnings provided by the manufacturers of the equipment you are using.

Quick Start Instructions

Most Apple II users will probably not need instructions to install the CFFA Interface board, but I have included Quick Start instructions below, as well as more detailed information in the next section. The information provided will give you insights into the behavior of the board, so you may better know what to expect.

1. Discharge yourself of excess static charge.

2. Open and remove the CFFA board from the anti-static bag.

3. Select the correct firmware for your computer, using the JP3 “Firmware Select (0/1)” jumper. Refer to Figure 1 and Table 1 for the location and description of various jumpers.

   - For 6502 machines: Apple II, II+, and IIe remove JP3.

   - For 65C02 or later machines: Apple *II*e Enh, *II*e Platimum, and *II*GS install JP3.

4. Insert a CF card into the CFFA board. Most CF cards have a ridge or lip that may catch on the CFFA board’s top edge during insertion. Do NOT force it –simply lift the CF card’s ridge over the top edge of the CFFA board and finish inserting the CF card.

5. Optionally, attach an IDE hard drive or IDE-to-CF converter board with a second CF card.
   ***NOTE**: Version 2.0 (and later) firmware supports the attachment of two storage devices on a single CFFA board. Some combinations of CF cards have been found to be problematic with the CFFA board. Example: Some Lexar and SanDisk cards together may malfunction. Generally two cards of the same brand work well.*

6. Turn off power to your Apple Computer.

7. Insert the CFFA board into any empty Apple slot. Avoid using slot 3 in *II*e enhanced and *II*GS machines.

8. Turn on your Apple computer. Assuming the attached storage device is not formatted yet, your computer will boot off another drive, for example, a floppy drive.

9. The factory default configuration of the CFFA board is set to support 4 drives in the CF socket and 0 drives attached to the IDE header. If you want to use a different configuration, configure your Apple to boot from the CFFA and press the “m” or “M” key just after power-on, to enter the CFFA board’s configuration menu. See **Configuration Menu Details** section for more information on using the CFFA menu.

10. The CFFA partition scheme is fixed so no user partitioning is necessary or possible. You simply need to format any or all drives on the device using any software that can format a ProDOS volume. I recommend Dave Lyons' Davex utilities. Remember that every drive after the second drive will show up in a different slot than the one the CFFA is actually using. Any utility that can format a ProDOS volume should work, for example, Copy II+, Apple System Utilities, etc.
    ***NOTE**: If you are planning to format more than two partitions on a device, see the section titled "Preparing the Storage Device" on page 22.*

11. If you want to be able to boot from the device, after formatting the drive(s), ProDOS users will have to copy PRODOS and a startup program like BASIC.SYSTEM to the card. I recommend using ProDOS version 2.0.3. GS/OS users will need to install Apple IIGS system software from floppies or another drive. I recommend GS/OS version 6.0.1.

12. Before using the board for storing information, spend some time testing the CFFA board in your particular environment. In any case, always back up important data onto multiple storage devices.

Note to GS/OS users:

If you are planning to use GS/OS and want an HFS partition, it is my understanding that there are critical patches needed for the HFS file system to be completely stable. I believe these are needed when the volume size exceeds 64MB, but I recommend consulting the oracle of all knowledge: Google.com, about this subject to find more information.

Figure 1: CFFA connections and jumpers PCB revB

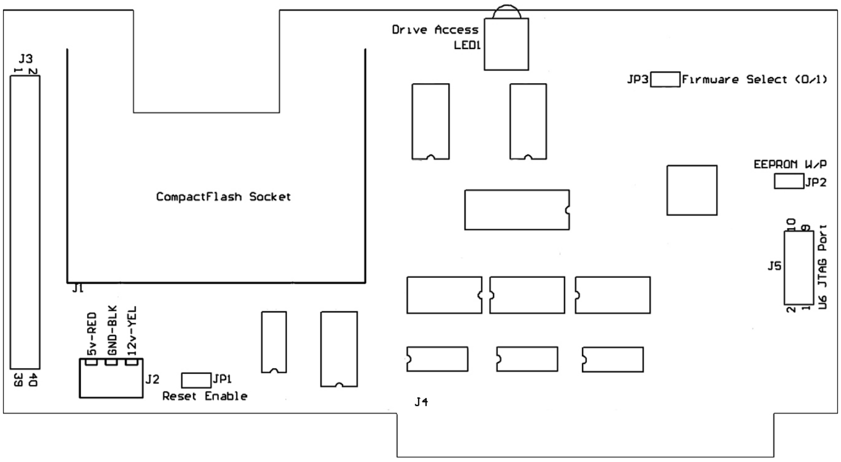


Table 1: CFFA connections and jumpers PCB revB

Name	Purpose
J1	CompactFlash Socket for CF card or Microdrive.
J2	Power Connector for IDE drive. Connector provides: +5v, +12v Note: Hard drives will likely exceed Apple's power supply current limits.
J3	IDE drive header for connection to IDE Hard Drive. Pin 1 is at the top of the board, near the J3 label.
J4	Apple II bus edge board connector. Plugs into any Apple II with expansion slots. Avoid slot 3 on Apples with 80-column hardware.
J5	JTAG programming header for CPLD U6. Note: power must be supplied to board during programming with this port. In other words, plug CFFA into an Apple II before using this port.
JP1	Reset Enable. Factory Default: jumper installed. Connects Apple bus reset signal to CF/IDE device reset. The only users who might not want this jumper on are those who are in the bad habit of pressing reset **all the time**. I recommend leaving this jumper in.
JP2	EEPROM Write Protect. When installed, this jumper prevents accidental and intentional writes to the EEPROM. Always leave this jumper installed until you want to change the EEPROM contents. Note: v2.1 CPLD logic introduced with Run 6 allows EEPROM locations $C800 to $C81F to be updated without removing this jumper. This override mechanism is controlled by two new soft switches. See Table 6 in the advanced section.
JP3	Firmware Select: The EEPROM is 8K and has room for two versions of CFFA firmware. See section "EEPROM Firmware Select Jumpers" on page 33 for more information.

Installation Details

The CFFA board comes in an anti-static bag. This bag is a good place to store the board when it is not installed in your computer. Before opening the zip-top bag, be sure to discharge yourself of any static charge.

The CFFA board can be installed in any Apple II slot, except slot 3 in Apples with integrated 80-column hardware. Depending on which Apple and what firmware or driver you are using, you may get varying degrees of functionality based on which slot you use.

V2.0 Firmware Features

CFFA boards from Run 6 ship with a new version of firmware: v2.0. This firmware requires CFFA hardware v2.0 revB or later. The major features added to the firmware include:

- support for two devices attached to a single CFFA board
- boot from any partition on either attached device
- user configurable partition counts on both devices via boot configuration menu
- approximate 25% speed increase for reading and writing over v1.2 firmware
- firmware-based configuration menu with parameters saved in EEPROM

Device Mapping

The CF socket on the CFFA board is permanently wired as Dev0 (master) and the device connected to the IDE header Dev1 must be setup as an IDE slave.

Partition Mapping

When multiple devices are attached to the CFFA board, the partition mapping is straight forward. Assuming you are running ProDOS 2.x or GS/OS, you can access up to 13 partitions from a single CFFA board. The partition mapping starts with all the partitions on the boot device first, followed by all the partitions on the other device. The number of partitions on each device is always the lesser of:

1) The number of actual partitions on the device, or

2) The number of partitions defined in the configuration menu.

Table 2 shows an example configuration. The actual Slot and Drive numbers you will see on your system may be different. This depends on which slots you are using for the CFFA and which slots have other storage interface cards (floppies, SCSI, etc) located in them. For this example let's assume that the CFFA is in slot 7, and you have configured Dev0 to support 4 partitions, and Dev1 to also support 4 partitions. Then you installed a 256MB CF card onboard and a 32MB CF card is attached to the IDE header via an adaptor. The 256MB card could hold up to 8 partitions, but only the first four are mapped because you have selected four partitions in the configuration menu. In the case of Dev1, the 32MB card, you have also selected four partitions, but since the CF card is only 32MB total, only get one partition is mapped.

Table 2: Example partition mapping with two devices attached

Partition	Slot, Drive	Device Accessed
1 *	7, 1	SanDisk 256MB CF card in CF socket on CFFA board.
2	7, 2	
3	1, 1	
4	1, 2	
5	4, 1	SanDisk 32MB CF card in CF-to-IDE adaptor.

* *Default boot device*

Access to Configuration Menu

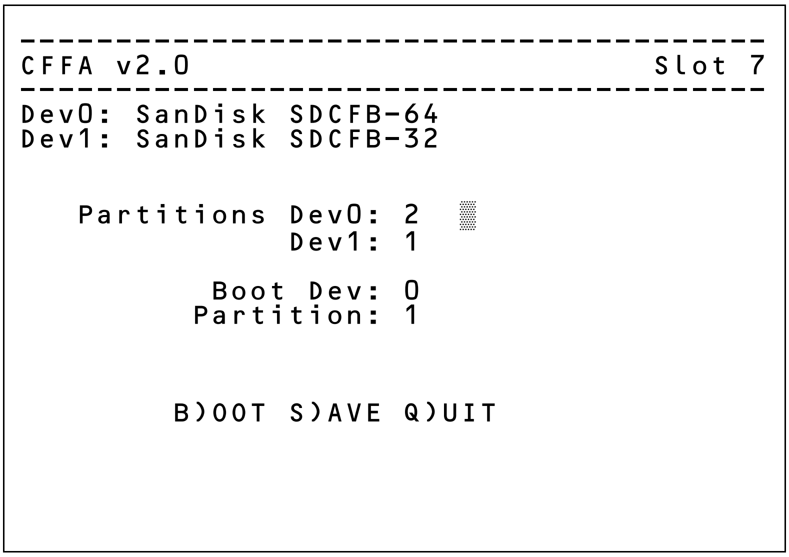
The configuration menu may be accessed at boot time by pressing the 'M' key just after power on. The default hot key may be upper or lower case. Assuming the CFFA board is the first boot device found during the boot slot scan, the CFFA configuration menu will appear. The Config menu hot key and the time delay waiting for you to press the hot key may be changed using a stand alone basic utility called CFFA.UTIL desribed below.

Configuration Menu Details

Figure 2 below shows an example of the configuration screen with two devices attached to the CFFA: a 64MB SanDisk CF card and a 32MB SanDisk CF card (attached via an IDE/CF converter board). Table 3 below explains the meaning of the different menu items. The menu items can be manipulated using the right/left arrows. Use the up/down arrows to select a different item. If your Apple does not have up/down arrows, use the <Enter> key to cycle through the configuration options. When you change the value of an item, it will change to inverse state to show you that the current value no longer matches the value saved in the EEPROM. Once the value is changed to the value you desire, press the ‘S’ key to save the new setting and the inverse colors should be removed.

Due to firmware space constraints, little input validation is done to **Boot Dev** and **Partition** parameters. Use care when setting these parameters to ensure they correspond with the defined partition counts for Dev0 and Dev1.

Figure 2: CFFA boot time configuration menu



```
---------------------------------------
CFFA v2.0                        Slot 7
---------------------------------------
Dev0: SanDisk SDCFB-64
Dev1: SanDisk SDCFB-32


   Partitions Dev0: 2  ▒
              Dev1: 1

          Boot Dev: 0
         Partition: 1



        B)OOT S)AVE Q)UIT
```

If you upgrade a Run 4 or 5 CFFA board to version 2.0 firmware or later, please be aware that without version 2.1 CPLD logic (new for Run 6), you will need to manually remove the Write Protect jumper (JP2) to save changes. If you have a Run 6 CFFA board with v2.1 CPLD logic, you do not need to remove the write protect jumper to save new configuration information. See Table 7 in the *Advanced*

Information section for detailed information about the configuration information that is stored in the EEPROM.

Table 3: CFFA configuration menu details

Configuration Setting	Purpose
`Dev0:`	Manufacturer’s name of device installed in CF socket. If this name does not appear, either a CF card is not installed, or the Dev0 partition count is set to 0, or the device is incompatible with the CFFA board. This name will also not appear when using 6502 version of firmare.
`Dev1:`	Manufacturer’s name of device connected to IDE header. If this name does not appear, either a CF card is not installed, or the Dev0 partition count is set to 0, or the device is incompatible with the CFFA board. This name will also not appear when using 6502 version of firmare.
`Partitions Dev0:`	The maximum number of partitions that will be reported to the operating system for the device in the CF socket. In other words, how many “drives” you want on this device. This sets the MAXIMUM number of drives that can be seen. You may set this value larger or smaller than the actual space on CF card being used. Valid range: 1 to 13 partitions. The default setting for this parameter is 4.
`Dev1:`	The maximum number of partitions that will be reported to the operating system for the device attached to the IDE header. In other words, how many “drives” you want on this device. This sets the MAXIMUM number of drives that will be seen. You may set this value larger or smaller than the actual space on CF card being used. Valid range: 1 to 13 partitions. The default setting for this parameter is 0.
`Boot Dev:`	The device the Apple II will see as the boot device. Valid values are 0 or 1. The default setting for this parameter is 0.
`Partition:`	The partition number on the Boot Device that the Apple II will try to boot from. Valid range: 1 to 13. The default setting for this parameter is 1.
`B)OOT`	Pressing B will cause the CFFA firmware to continue with the boot process using the current settings. For example, you might change the Boot Partition and then press ‘B’ without saving your changes if you want to boot from a different partition only once.
`S)AVE`	Pressing S will cause the CFFA firmware to save any changes made to the configuration parameters. If you have version 2.1 CPLD logic which shipped with Run 6 boards, you do not have to remove the EEPROM write protect jumper JP2 before hand. If your CFFA board has a previous version of CPLD logic you will have to remove the write protect jumper JP2 before pressing ‘S’. A beep will be heard if the save fails.
`Q)UIT`	Pressing Q will cause the configuration menu to exit to Applesoft leaving your Apple II unbooted.

CFFA.UTIL – Flash and Configuration Utility

The CFFA.UTIL utility was written in Applesoft BASIC to allow you to update the board's firmware and modify its configuration. This utility can modify all CFFA configuration parameters, including the boot configuration menu parameters. Table 7 shows the list of all config parameters. Items in gray may be changed using CFFA.UTIL. Select the **Settings** menu item to change these parameters.

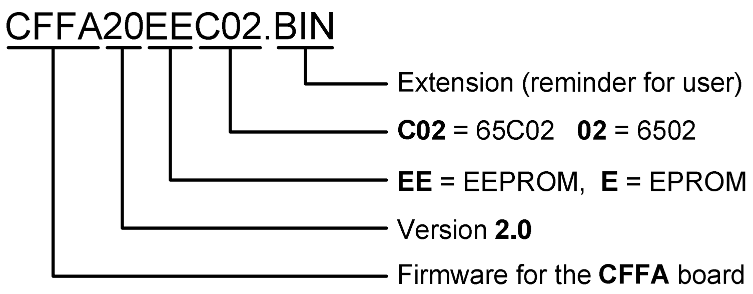
You may update the firmware located on EEPROM U5 using this utiltity. Select the **Update EEPROM** menu item and select the desired firmware from the list. Table 4 shows the firmware files that are currently available at the CFFA web site.

The list of firmware files shown under the **Update EEPROM** menu option is based on the files that are stored in a sub-directory called CFFA.FIRMWARE. A valid firmware file must be exactly 4096 bytes long and have a file type of $00 (none) or $06 (bin). The auxtype value doesn't matter.

Table 4: Currently available firmware versions

File Name	Description
`CFFA20EEC02.BIN`	Version 2.0 firmware for 65C02 and 665C816 machines: Apple *II*e Enhanced, *IIe* Platinum, and *II*GS.
`CFFA20EE02.BIN`	Version 2.0 firmware for 6502 machines: Apple II, II+, and *II*e.
`F02V10.BIN`	Old: Version 1.0 firmware for 6502 machines
`FC02V12D4.BIN`	Old: Version 1.2 (4 drives) for 65C02 and 65C816 machines
`FC02V12D8.BIN`	Old: Version 1.2 (8 drives) for 65C02 and 65C816 machines

The file naming convention used by CFFA beginning with version 2.0 can be decoded as show below. Although the file naming convention is not required by CFFA.UTIL, it does make it easier to decode the file's purpose.


CFFA20EEC02.BIN
Extension (reminder for user)
C02 = 65C02 **02** = 6502
EE = EEPROM, **E** = EPROM
Version **2.0**
Firmware for the **CFFA** board

Updating from Firmware v1.2 to v2.0

Customers who have Run 4 or 5 CFFA boards and want to update firmware in these boards can use the procedure below. This procedure will program the 65C02 firmware and the 6502 firmware into the two firmware banks selected via JP3.

You must have a bootable CF card with BASIC.SYSTEM and the CFFA.UTIL program on it, and a subdirectory called CFFA.FIRMWARE with the firmware files to which you want to upgrade. Example:

```
CFFA20EEC02.BIN
```

```
CFFA20EE02.BIN
```

The procedure described here is designed to let you boot and run the CFFA.UTIL off of the card you are updating. It is important you follow the steps in exact order or you may end up with a card that won't boot.

The CFFA card will have to be programmed in all the slots in which you want to use it. *Note: Apple IIe Enhanced, Platinum, and GS users should skip slot 3*. You will need to program just the boot ROM in each slot first, and then, for the last slot, program the boot ROM and the Aux ROM. If you mistakenly upgrade the Aux ROM first, you will no longer be able to boot from the CFFA card when you move it to the next slot because it will have an AuxROM and boot ROM that don't work together.

1. Turn off your Apple II.
2. Remove any peripherial cards from any slots in which you want to program the CFFA.
3. Remove the write protect jumper (JP2).
4. Install the firmware select jumper (JP3). The jumper installed will be for 65C02 firmware.

5. Install your CF card in the CFFA board. Insert the CFFA in slot 7 or whatever slot you want to start with.
6. *Apple IIGS users: Hold down Option key when booting Apple IIGS and change each slot to* ***Your Card*** *in which you want to program the CFFA.*
7. Turn on your Apple II. The CFFA board should boot into ProDOS using old v1.2 firmware. *GS/OS users: Go to P8 mode.*
8. Run BASIC program CFFA.UTIL.
9. Select Upgrade EEPROM menu.
10. Select **CFFA20EEC02.BIN** from the list. This is the 65C02 firmware.
11. Select Program Boot ROM. Do **NOT** program Aux ROM.
12. Exit back to main menu.
13. Select Upgrade EEPROM menu.
14. Select **CFFA20EE02.BIN** from the list. This is the 6502 firmware.
15. Remove the firmware select jumper JP3. This is to program the second bank of firmware.
16. Select Program Boot ROM. Do **NOT** program Aux ROM.
17. Exit back to main menu.
18. If there are more slots left in which to program the CFFA, turn off your computer and move the CFFA board to next slot. Install the firmware select jumper (JP3). Then continue at Step 7. If there are no more slots left, leave your computer on and proceed to step 19.

When all slots have been programmed for Boot ROM, program the Aux ROM for both banks:

19. Select Upgrade EEPROM menu.
20. Select **CFFA20EE02.BIN** from the list. This is the 6502 firmware. (JP2 should still be off at this point.)
21. Select Program Aux ROM.
22. Exit back to main menu.
23. Install firmware select jumper (JP3).
24. Select Upgrade EEPROM menu.

25. Select **CFFA20EEC02.BIN** from the list. This is the 65C02 firmware.
26. Select Program Aux ROM.
27. Turn off the computer.
28. Remove the CFFA board.
29. Install the write protect jumper and the firmware select jumper as desired.

If you boot your Apple II and run the CFFA.UTIL program from a floppy drive or other storage device, then you don’t need to worry about the order you program the Boot ROM and Aux ROM in. But you may have to insert a CF card into the CFFA to get the system to boot, especially if you are upgrading from version 1.2 firmare. This was because of a bug in v1.2 firmware that would cause ProDOS to hang waiting for the CFFA card to become ready.

CompactFlash Memory Cards

This section provides detailed information specific to the use of CompactFlash (CF) memory cards with the CFFA board.

CompactFlash Socket

Connector J1 labeled “CompactFlash Socket” is a Type II socket. This allows you to connect either Type I flash cards or Type II devices, such as the Hitachi Microdrive. The CF socket is hardwired to use the attached device’s “True IDE” compatibility mode. It is also hardwired to address the device as an IDE master.

When inserting a CF card into the CFFA board, you should insert the card label side out. It takes very little force to insert a card. After the socket pins start to engage the card, a little extra force is needed to fully mate the two.

IMPORTANT: *Many CF cards have a ridge or lip that may catch on the CFFA board’s top edge during insertion. In this case it may seem to require a large force to completely insert the card. Do NOT force it—simply lift the CF card’s ridge over the top edge of the CFFA board and finish inserting the CF card.*

CF Advantages

CompactFlash cards have several advantages over traditional hard drives.

- CF cards are solid-state memory devices, which are completely silent and more reliable than IDE hard drives.
- CF cards use less power and generate less heat than IDE hard drives.
- CF cards have no seek delay times related to mechanical head movement. All data in a CF card is accessed at the same speed.

CF Disadvantages

CompactFlash cards do have a few disadvantages when compared to traditional hard drives.

- The cost per megabyte is higher for CF cards.
- Each sector on a CF card can only be written to a limited number of times. This is the write cycle endurance, and is a specification of the CF card itself, and not the CFFA board. You can typically find endurance specifications for CF cards on the manufacturer's web site. For example, SanDisk Corporation specifies that their SDCFB-XX line of CF cards has an endurance of greater than or equal to 300,000 write cycles per block. Because all CF cards today support auto wear-leveling, and auto bad block swapping in hardware, you should never run into this limitation, even after decades of use.

CF Removability

Although most CF cards are used in a "removable" sense, the CFFA board does not treat a CF card as a removable device. The board's firmware does **not** report to the OS that it supports removable devices. You should **not** treat the CF card like a removable device. If you want to remove the CF card from the CFFA board, shut down your computer first.

Although removing the card with the power on will not hurt the CF card itself, if you re-insert the CF card with the power on you will not be able to access the data until you do a complete power cycle of the computer. The reason the CF card is not "removable" is that it is being used in its *TrueIDE* mode and is functioning as a normal hard drive. The *TrueIDE* mode is only* tested when power is applied to the CF socket.

**Many newer CF card now also test for entry into TrueIDE mode when the CF card is reset too. Reset of the CF card will occur when the Apple performs a reset, as long as the CFFA's reset enable jumper JP1 is installed. So it may be possible to insert a CF card with power already on and just press the apple reset. This practice is not recommended and may end up with a corrupt filesystem on your storage device.*

IDE Drives

This section provides detailed information specific to the use of IDE hard drives with the CFFA board.

IDE Drives Compatible with the CFFA Board

Most IDE drives should be compatible with the CFFA board. It is necessary for the IDE drive to support LBA (Logical Block Addressing) mode in order to work with the CFFA board. All IDE drives larger than 528 MB today support this mode. Most old IDE drives smaller than 528 MB did not support LBA and therefore will not work with the CFFA board. If in doubt, try your drive to see if it works.

IDE Drive Connector

Connector J3 allows you to connect a single IDE hard drive or CF-IDE adaptor. Pin 1 of this connector is at the top of the board next to the label "J3". A drive connected to this connector should be set to a *Slave* drive if you are also using a CF card in J1. If no card is being use in J1, then a drive connected to this connector must be set to *Master*. Firmware version 2.0 (and later) now supports the use of two devices attached to the CFFA at the same time. The CF socket J1 is hardwired as the master on the IDE bus.

IDE Power Connector

Connector J2 on the CFFA board is provided to supply power to an IDE hard drive that you connect to J3. This connector provides access to the Apple's power supplies: +5v, +12v, and Ground. It is up to you to ensure that the devices attached to this connector do not consume more power than the Apple's power supply is capable of delivering. Remember to consider the load of all the other devices in your system.

J2 power connections are labeled as follows:

- +5v DC is labeled "5v-RED"
- +12v DC is labeled "12v-YEL"
- Ground is labeled "GND-BLK"

The labels RED, YEL, and BLK written on the CFFA PCB silkscreen show the standard wire colors used by most IDE drive connectors.

Because J2, the power connector, is a screw terminal type and requires you to connect wires, it is not fool proof. This connector can be miswired. Miswiring could cause the destruction of the IDE hard drive, the CFFA board, and possibly your computer. Use caution, and observe polarities when connecting power to an IDE drive.

IMPORTANT*: It is your responsibility to be sure that the device you attach to connector J2 is wired correctly, regardless of the wire colors involved.*

Preparing the Storage Device

Whether you use a CF card or an IDE drive you will need to prepare the storage device before you can use it to store your Apple II data.

Partition Scheme for Storage Devices

Here a “Partition Scheme” refers to the quantity and size of drive volumes available on a storage device. Most users are accustomed to deciding how the drive partitions are laid out. However, this is not possible with the CFFA board, which uses a static partition scheme. Because CF cards today are much larger than the ProDOS 32MB maximum partition size, there is no need to create variable partition sizes. The user simply needs to define how many partitions the CFFA’s firmware will report to the operating system. This is defined in the CFFA configuration menu as described above. The factory default partition count for the CF socket (Dev0) is four and the IDE header connected device (Dev1) is zero.

Formatting Storage Devices

After you attach a storage device to the CFFA board, if it is not already formatted, it must be formatted with the file system for the operating system you will be using. This would typically be ProDOS. It should be possible to use any software package that can format a ProDOS or HFS volume. Both Copy II+ and Apple System Utilities work fine for the first two partitions.

IMPORTANT:

When formatting the higher partitions, which appear as drives in other slots, Copy II+ and Apple System Utilities should **NOT** be used. They both have a bug that causes them to read the partition size incorrectly on all partitions after the second. To format the higher partitions I recommend Dave Lyons’ freeware utility Davex. After formatting is complete, any utility should work.

ProDOS versions

Different ProDOS versions have different capabilities related to the number of partitions you will be able to access on the CFFA board. Table 5 summarizes these capabilities based on the recommended OS version.

Table 5: ProDOS version recommendations

System Configuration	Recommended OS version	Max Partitions with recommended OS
Apple II+ with 48K	ProDOS 1.0.1	2 drives in all slots
Apple II+, //e with 64K	ProDOS 1.9	2 drives in all slots except 5 4 drives in slot 5* *NO auto-boot in slot 5 use PR#5.
Apple //e enhanced	ProDOS 2.0.3	1 to 13 partitions (user configured)
Apple //e platinum	ProDOS 2.0.3	1 to 13 partitions (user configured)
Apple //gs	ProDOS 2.0.3	1 to 13 partitions (user configured)
Apple //gs	GS/OS 6.0.1	1 to 13 partitions (user configured)

Formatting is a high level OS format, which does not perform any kind of media analysis or bad block checking. You may want to perform a disk verify using Apple System utilities or Copy II+ to see if your computer can read every block on the disk. This can be done before or after you format the volume. Note: A “disk verify” can take a very long time on a 32 MB partition. I recommend Copy II+ for this task.

Devices Compatible with the CFFA Board

The CFFA board was developed using SanDisk Compact Flash cards and 2.5” IBM brand hard drives. A wide variety of brands work with the board, but I can’t guarantee compatibility with all brands. To help determine which devices work with the CFFA board and which devices do not, I maintain a compatibility list on my web site, http://dreher.net/CFforAppleII/Compatibility.html

If you have information about the compatibility of a storage device with the CFFA board, feel free to post a message to the Discussion Forum or E-mail me about it. I update the compatibility list as information becomes available.

GS/OS Users

GS/OS will find and use your storage device without any additional drivers. GS/OS will use its internal “generated drivers” to access the CFFA board and make calls to the onboard firmware. However, you have the option of using a driver written specifically for the CFFA board by Dave Lyons (of Apple fame). This driver provides faster access to the CFFA board by replacing all the functionality of the onboard firmware, with driver code that runs out of system RAM that is typically much faster.

CFFA Firmware v2.0 (Run 6)

Version 2.0 (and later) firmware requires a new GS/OS driver. At the time of this writing the new driver was not completed and therefore the exact details of its use are not known. When the driver is ready it will be made available on the CFFA web site. This section of the manual will be updated and also made available on the web site.

***IMPORTANT**: The old GS/OS driver will not load when it detects version 2.0 or later firmware. This is expected behavior. Therefore you will not have access to any HFS partitions you may have created until the new driver is released.*

CFFA Firmware v1.2 (Runs 4 & 5)

The old GS/OS driver works with firmware v1.2 and provides support for one or two additional HFS partitions if your storage device is larger than 128 MB (four 32MB drives).

The driver can be downloaded via the Internet, from the CFFA web site in the “File Downloads” section or from Dave Lyons’ web site.

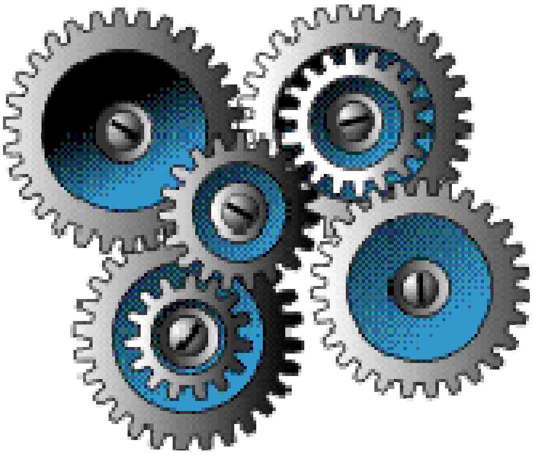
The driver is called: COMPACTFLASH. Once this driver is loaded it no longer uses the onboard firmware. It provides support for two additional drives, up to 1 GB each, and faster performance. Note: This driver should also work with most IDE hard drives. Getting the driver from the Internet to your CF card can be a bit tricky. There are utilities out there to help you with this. Windows users may want to check out a program called CiderPress. It makes the process very simple if you have CF card reader for your PC.

After downloading the driver to an Apple II readable media you should copy it into the DRIVERS folder inside your GS/OS startup disk's SYSTEM folder. With the driver installed, you may see one or two extra partitions on your CFFA board, rather than the usual four. This will depend on the size of your storage device. Obviously it needs to be larger than 128MB to see even one extra partition. You need to be sure that your CFFA has the “4 drive” firmware loaded into its EEPROM and you have selected that firmware with jumper JP3. Using a firmware that supports a different number of drives, for example 8 drives, will cause the GS/OS driver to not load.

You can verify that the COMPACTFLASH driver is being used successfully from the device name of any partition on the CFFA. In the Finder, select a partition and type Apple-I (Icon Info). Then click the "Where" tab and note the device name. If GS/OS is using a "generated driver" that calls the CFFA's firmware, you will see a name like ".DEV3". If the COMPACTFLASH driver is being used you will see something like ".CFFA.7A.SANDISK_SDCFB-16", where 7 is the slot number and the following letter indicates the partition (A through F).

The COMPACTFLASH driver can control one or two CFFA boards in your system. Normally, each slot containing a CFFA board will be set to “Your Card” in the Apple IIgs Control Panel. However, the COMPACTFLASH driver will find and use the CFFA board even if the slot is NOT set to “Your Card”. In this case, you can use the CFFA board while in GS/OS, but it will be invisible to ProDOS 8 making it impossible to boot from the CFFA board. If you need to boot from the CFFA board, be sure the slot is set to “Your card”.

Advanced Information



Hardware

This section gives detailed information about the hardware used on the CFFA board.

Altera CPLD

The chip U6 is an Atmel ATF1502AS-10AC44 part, which is compatible with an Altera EPM7032STC44-10 CPLD (Complex Programmable Logic Device). It is flash based and can be reprogrammed via the JTAG Port J5. For this you will need Altera compatible programming cable and Atmel's ATMISP program for Windows. When programming the CFFA board's CPLD via the JTAG header, the CFFA board must be plugged into an Apple II bus to supply power to the board. I do not recommend programming the CPLD with the storage device installed.

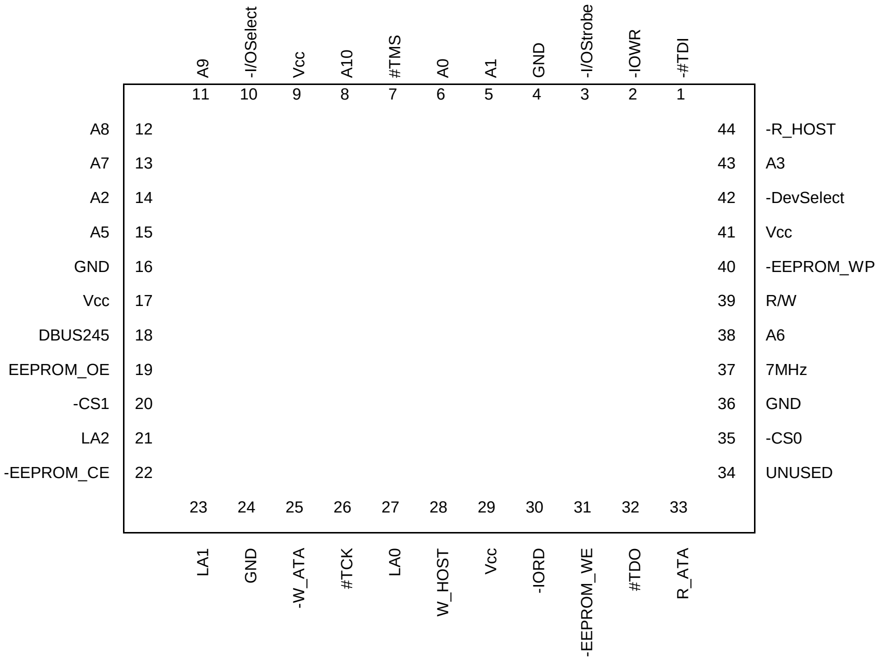
Programming cables are available from many places, including Ebay, for as little as US$15. Search for "ByteBlaster Cable". The older 5 volt cables will work fine. The more expensive low voltage cable is not needed.

Atmel CPLD Pinout

Figure 4 provides the signal names for version 2.0 of the CFFA CPLD firmware (here “firmware” refers to the AHDL logic files used to program the CPLD). Different firmware versions could have a slightly different pinout.

Figure 4: Altera’s EPM7032STC44-10 or Atmel’s ATF1502AS-10AC44

NOTE: Signal names are specific to CFFA logic



CPLD Logic Files

Listing 1 is the ADHL source file used to create the programmer-ready .pof file needed to program U6. Comments in the file start and end with the % character. This file was compiled using Altera's MAX+Plus II Baseline 10.1 development software. This software is free and can be downloaded from Altera's web site.

Listing 1: CPLD Logic - AHDL Source File Version 2.1

```
%------------------------------------------------------------------------
 CompactFlash/IDE Interface for the Apple II computer
 Project Home: http://dreher.net/CFforAppleII/
 Project Version 2.1   March 09, 2008

 Version 2.1  -  Inhibit writes to $CFFF from creating a write cycle on the EPROM
              -  Add pair of soft switches to enable ($C0n3) / disable ($C0n4)
                 writing to the EEPROM at locations $C800 to $C81F. This 32 byte
                 range will be used for configuration information in v2.x firmware.

 Version 2.0  -  Added support for AT28C64 EEPROM and WP jumper

 Version 1.4  -  Fix timing problem introduced in V1.3: /IORD was being
                 being de-asserted before the falling edge of 6502's PH2
              -  Latch A3-A0 to insure that address is stable through entire
                 ATA cycle.
              -  Add outputs for latched A2 thru A0 signals. This supports a
                 hardware change that allows many more CF cards to work
                 with the CFFA card.
              -  Remove use of nandltch and replace with SRFF. This is now
                 synchronous logic.
              -  Adjust W_HOST timing slightly
              -  Adjust /W_ATA timing so it only asserts when needed
              -  Adjust /R_ATA timing so it only asserts when needed
              -  Remove one delay FF on /DSEL, it was not needed

 Version 1.3  -  Greatly improved timing margins on ATA signals:
                 /IORD, /IOWR, /CS0, /CS1, by adding two additional
                 D-FF and inverting 7M Clk.
              -  Now works with Apple ][+ bus timing.

 Version 1.2  -  Two or more cards would not work in system at same time
                 because the DBUS245 select logic did not take into account
                 expansion ROM enable flipflop state.
              -  Moved all logic into appleidelogic.tdf file.

 Version 1.1  -  Add 7M clk & DFF to fix IDE drives /IORD & /IOWR timing

 Version 1.0  -  Initial Release

Note: Remember that the Apple II Bus does not have Phase 2!
--------------------------------------------------------------------------------%
FUNCTION DFF (D, CLK, CLRN, PRN) RETURNS (Q);
FUNCTION DFFE (D, CLK, CLRN, PRN, ENA) RETURNS (Q);
FUNCTION SRFF (S, R, CLK, CLRN, PRN) RETURNS (Q);

SUBDESIGN AppleIDELogic
(
   A0, A1, A2, A3, A5                     : INPUT;                            %
Note: A4 is missing, it didn't fit in device %
   A6, A7, A8, A9, A10                    : INPUT;
   /RW, /DSEL, /IO_STRB, /IO_SEL, 7Mclk    : INPUT;
   /EEPROM_WP                             : INPUT;

   /R_HOST, R_ATA, W_HOST, /W_ATA          : OUTPUT;
   /IOWR, /IORD, /CS0, /CS1               : OUTPUT;
   /DBUS245, /EEPROM_CE                   : OUTPUT;
   LA0, LA1, LA2                          : OUTPUT;
   /EEPROM_OE, /EEPROM_WE                 : OUTPUT;
   /C800_FF                               : OUTPUT;
)

VARIABLE
```

```
RESET_MASK, SET_MASK, /CFFF                : NODE;
DelayDSEL1, DelayDSEL2, NOT_7Mclk, LA3     : NODE;
/EEAddr, CS_MASK                           : NODE;
ENABLE_CONFIG, DISABLE_CONFIG              : NODE;
CONFIG_ALLOW_WRITE, /CONFIG_DATA_RANGE     : NODE;
/CONFIG_DATA_WE                            : NODE;

BEGIN
   DEFAULTS
       CS_MASK  = GND;
       /C800_FF = VCC;
       CONFIG_ALLOW_WRITE = GND;
   END DEFAULTS;

% Create an inverted version of the 7Mhz clock                                 %
  NOT_7Mclk = !7Mclk;

  LA0 = DFFE(A0, 7Mclk, VCC, VCC, /DSEL);
  LA1 = DFFE(A1, 7Mclk, VCC, VCC, /DSEL);
  LA2 = DFFE(A2, 7Mclk, VCC, VCC, /DSEL);
  LA3 = DFFE(A3, 7Mclk, VCC, VCC, /DSEL);

% Expansion Slot ROM enable Flip-flop. Active low signal                       %
 /C800_FF = SRFF(!/CFFF, !/IO_SEL, 7Mclk, VCC, VCC);

% EEPROM Address. Active low signal %
 /EEAddr = (/C800_FF # /IO_STRB # !/CFFF) !$ /IO_SEL;

% EEPROM Chip Enable %
 /EEPROM_CE  = /EEAddr # (!/RW & /CONFIG_DATA_WE & !/EEPROM_WP);

% EEPROM Output Enable %
 /EEPROM_OE = !/RW # /EEAddr;

% EEPROM Write Enable %
 /EEPROM_WE  = /RW # /EEAddr # (!/EEPROM_WP & /CONFIG_DATA_WE);
% /EEPROM_WE  = /RW # !/EEPROM_WP # /EEAddr; %

%------------------------------------------------------------------------------%
% Support for CFFA firmware version 2.0 feature to allow modification of the   %
% 32 bytes of EEPROM at address $C800 even if the user has not removed the WP  %
% jumper. This will allow CFFA firmware to "soft" enable writing those 32 bytes%
%------------------------------------------------------------------------------%
  ENABLE_CONFIG  = /DSEL # (LA3 # LA2 # !LA1 # !LA0);
  DISABLE_CONFIG = /DSEL # (LA3 # !LA2 # LA1 # LA0);
  CONFIG_ALLOW_WRITE = SRFF(!ENABLE_CONFIG, !DISABLE_CONFIG, 7Mclk, VCC, VCC);

% Memory range $C800 to $C81F (32 bytes) can be written under control of soft- %
% soft switches                                                                %
  /CONFIG_DATA_RANGE = (A5 # A6 # A7 # A8 # A9 # A10 # /IO_STRB);
  /CONFIG_DATA_WE    = /CONFIG_DATA_RANGE # !CONFIG_ALLOW_WRITE;

%------------------------------------------------------------------------------%
% Fix for SanDisk Family of CompactFlash drives. True IDEmode is not quite     %
% True! The idea here is to mask the read cycle that proceeds all write cycles %
% because the read cycle was confusing the Sandisk                             %
%------------------------------------------------------------------------------%
  SET_MASK = /DSEL # (LA3 # LA2 # LA1 # !LA0);
  RESET_MASK = /DSEL # (LA3 # LA2 # !LA1 # LA0);
  CS_MASK  = SRFF(!SET_MASK, !RESET_MASK, 7Mclk, VCC, VCC);

%------------------------------------------------------------------------------%
% Device select is clocked through two D-FF to create a late falling edge      %
% and a early rising edge for /IORD and /IOWR. The address lines A0-A3 are used%
% are used to inhibit /IORD and /IOWR signal generation on reads to onboard    %
% registers that don't map to the CF card, e.g. Latches                        %
% the /IOWR signal is delayed one 7M clock longer that /IORD to allow for worst%
% case data setup time of 200ns on 65C02 + 60ns setup time before IOWR is      %
% asserted on CF card                                                          %
%------------------------------------------------------------------------------%
 DelayDSEL1 = DFF(/DSEL, NOT_7Mclk, VCC, VCC);
 DelayDSEL2 = DFF(DelayDSEL1, NOT_7Mclk, VCC, VCC);
 /IOWR  = /DSEL # DelayDSEL2 #  /RW # !(LA3 # LA2 # LA1 # LA0);
 /IORD  = /DSEL # DelayDSEL1 # !/RW # !(LA3 # LA2 # LA1 # LA0);

% Decode address range $CFFF and $CFEF for deselecting the onboard EPROM       %
 /CFFF   = !(A0 & A1 & A2 & A3 & A5 & A6 & A7 & A8 & A9 & A10 & !/IO_STRB);

% Output Enable when reading High Byte Latch by host: $C0n0                    %
```

```
 /R_HOST  = /DSEL # LA3 # LA2 # LA1 # LA0 # !/RW;

% Latch High byte of ATA device when reading $C0n8 %
  R_ATA   = !/DSEL & !DelayDSEL1 &  LA3 & !LA2 & !LA1 & !LA0 & /RW;

% Latch data written to C0n0 by host(eg. Slot 7: $C0F0). This will become the  %
% high byte of the ATA data word when a write to C0n8 occures.                 %
  W_HOST  = !/DSEL & !DelayDSEL1 & !(LA3 # LA2 # LA1 # LA0) & !/RW;

% Output enable logic to drive the high of the ATA bus from U3 when a write    %
%  to C0n8 occures                                                             %
 /W_ATA   = DelayDSEL1 # !LA3 # LA2 # LA1 # LA0 # /RW;

% Device chip select logic                                                     %
 /CS0   = (/DSEL & DelayDSEL1) # !LA3 # (CS_MASK & /RW);
 /CS1   = (/DSEL & DelayDSEL1) # (LA3 # !(LA1 & LA2)) # (CS_MASK & /RW);
 /DBUS245 = /DSEL & /EEPROM_CE & /IO_SEL;

END;
```

Firmware

This section gives detailed information about the EEPROM based code (firmware) shipped with the CFFA board. CFFA firmware version 2.0 introduces the idea of user configurable firmware. It is no longer necessary to ship different versions of firmware to support 4 or 8 partitions. The user can now change the firmware configuration via the onboard firmware or the CFFA flash utility called CFFA.UTIL. A limited subset of features can be changed from the on-board menu, while the rest may be changed from the flash utility CFFA.UTIL.

Static Partitions

The partition scheme is static because the CFFA board does not have onboard RAM, so there is no temporary place to store partition information. In order to implement a user-configurable partitioning scheme, the firmware/driver would have to read the partition table from the device on every read or write access, affecting performance in a negative way. Also today’s CF cards are much larger than the natural 32MB partition supported by ProDOS.

Firmware Updates

All CFFA boards v2.0 revB and later have an EEPROM which may be flashed by the user. When new versions of firmware become available they may be downloaded from the CFFA web site. A flash utility, CFFA.UTIL is available on the CFFA web site along with instructions on the flash procedure. The EEPROM write protect jumper JP2 should be installed, at all times, until you are ready to update your flash.

Manually Changing EEPROM contents

Changing data on the EEPROM is as simple as removing the write protect jumper, going into monitor, and entering a new value at the desired location. Ex: C800: 08. This would set the max number of partitions for device zero to 8. (Assuming the CFFA’s expansion ROM was already selected.) If you plan to use the monitor to change several bytes, be sure to enter one byte at a time. Example: To change three consecutive bytes in EEPROM, you might be inclined to type:

```
C800: 04 04 00 <enter>
```

This will not work! Only the first byte will get changed correctly. You need to type the following instead:

C800:04 <enter>

C801:04 <enter>

C802:00 <enter>

This is necessary because there is a 200µs delay after writing to any EEPROM memory location, and monitor is too fast for the EEPROM. This also means that you can't use the Apple monitor's "memory move" command to write to the EEPROM because it is too fast; many bytes will not get updated properly.

EEPROM Firmware Select Jumpers

There is a single jumper JP3 that allows you to choose between two versions of firmware. When the jumper is removed, Address line A12 on the 8K EEPROM is pulled up to logic 1 level. The CFFA ships with two versions of firmware. Table 6 shows the factory default.

Table 6: Firmware Select - Standard firmware arrangement – JP3 Jumper Settings

JP3	EEPROM Offset	Firmware Selected
IN	$0000	Apple IIe Enhanced, IIe Platinum, IIGS ROM1, ROM3 SmartPort / ProDOS 8 firmware – Supports 1 to 13 32MB drives and one or two IDE devices. Requires 65C02 or later.
OUT	$1000	Apple II, II+, IIe ProDOS 8 firmware supporting the 6502 (Apple][+ and IIe). Supports 2 drives in all slots except 5. Slot 5 supports 4 drives, but does not auto-boot. Use PR#5 to boot from slot 5.

OUT = Jumper block is not installed
IN = Jumper block is installed

CFFA Firmware Configuration Area

Table 7 shows the layout of the configuration area in the EEPROM. The user is encouraged **not to change** these locations manually, but instead use the CFFA.UTIL program. If you do experiment with the CFFA settings, please back up your data first. Changes to these settings could result in corruption of your CF card's file system and data loss.

Table 7: CFFA firmware version 2.0: Configuration Locations

Location	Configuration setting	Size	Purpose
$C800	Max32MBPartitionsDev0	1	Max partitions on device 0. Range 0 to 13. Default: 4
$C801	Max32MBPartitionsDev1	1	Max partitions on device 1. Range 0 to 13. Default: 0
$C802	DefaultBootDevice	1	Boot device. Range 0 to 1. Default: 0
$C803	DefaultBootPartition	1	Boot Partition. Range 1 to 13. Default: 1
$C804	Reserved	4	Reserved
$C808	WriteProtectBits	1	Reserved for write protect bits $80 = protect Dev1. $40 = protect Dev0.
$C809	MenuSnagMask	1	Allows "m" or "M" to activate the boot-time menu. Default: $DF
$C80A	MenuSnagKey	1	Configuration menu activation key + $80. Default: $CD
$C80B	BootTimeDelayTenths	1	Time delay in tenths of a second to delay before checking keyboard at boot. Default: 5.
$C80C	BusResetSeconds	1	Maximum number of seconds to wait for IDE bus to become ready. Default: 31
$C80D	CheckDeviceTenths	1	Time delay, in tenths of a second, to delay waiting for a device to become ready. Default:100
$C80E	ConfigOptionBits	1	Set high bit ($80) to skip bus reset. Default: $00
$C80F	BlockOffsetDev0	3	Range 0 to 2^24 blocks (8 GB). Default: $00. *Warning: changing this may destroy the file systems on the installed device!*
$C812	BlockOffsetDev1	3	Range 0 to 2^24 blocks (8 GB). Default: $00. *Warning: changing this may destroy the file systems on the installed device!*
$C815	Unused	10	Reserved

Items shown in gray may be modified using the CFFA.UTIL program. Other items can only be modified from Apple monitor.

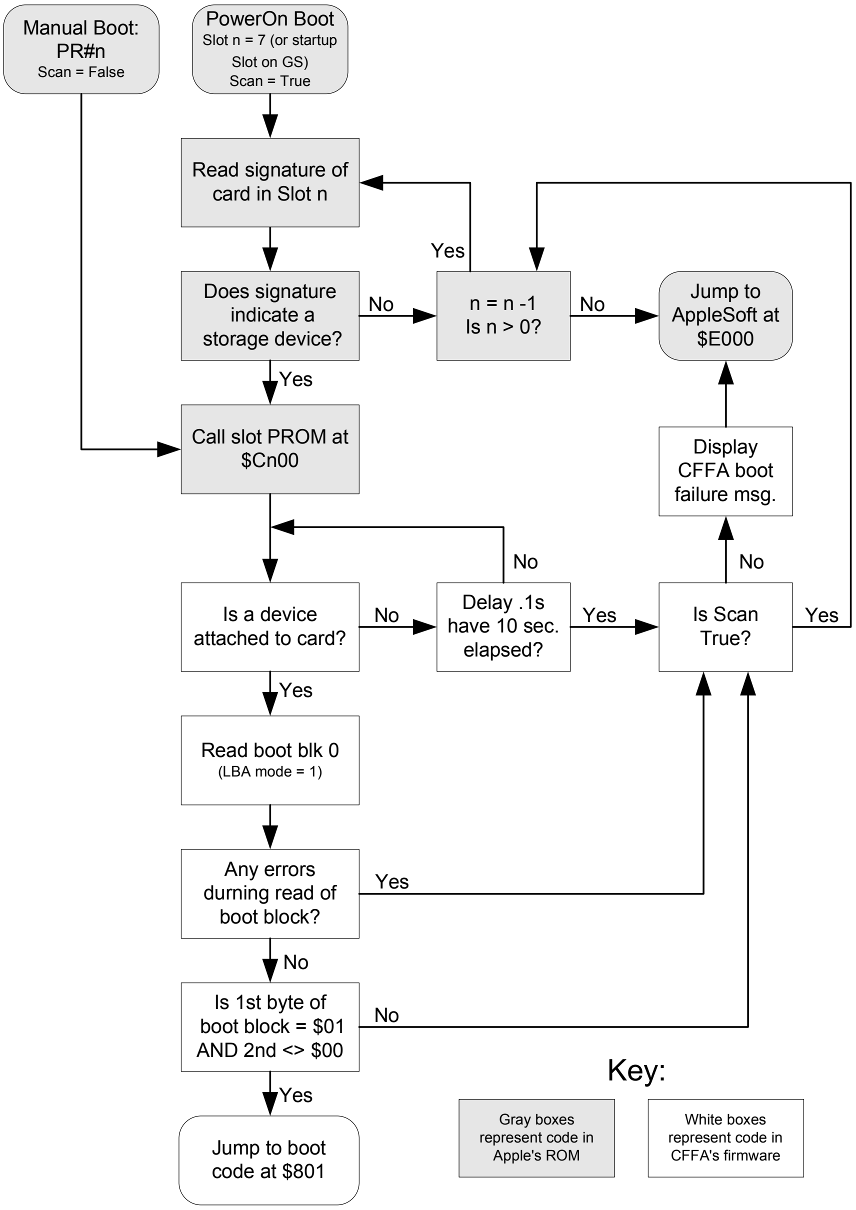
Apple II Boot Procedure with CFFA Installed

When the Apple II boots, it begins a slot scan process, starting with slot 7, or Startup Slot on a IIgs, looking for a bootable device. If no bootable device is found in that slot it then proceeds to the next lower slot. For example, if you install your CFFA board into slot 7 with an unformatted device connected to it (as you might after having just received your board), the CFFA firmware will be called by the Apple's ROM because the CFFA board will be recognized as a smartport compatible storage device.

After the CFFA's firmware is called, it will test to see if the IDE bus is ready and wait up to 31 seconds for it to become ready. Once ready, it will then check for two devices being attached to the IDE bus. It will wait up to 10 seconds for each device to respond. If one or more devices are found, the firmware will attempt to read the boot block off of the current configured boot partition into memory at location $800. If no devices are found the firmware will return control to the Apple's boot scan ROM routine or to Applesoft if it was called fusing PR#n.

If a boot block was read into memory, the firmware then tests to see if the first byte of the boot block code is equal to $01 and the second byte is **not** equal to $00. If both tests pass, the firmware will jump to the boot block code and the boot sequence will proceed under the boot code's control. If both conditions are not met, the firmware will return control to the Apple's boot scan ROM routine. See Figure 5 for a general flow chart diagram of the boot sequence logic.

Figure 5: Apple II boot sequence logic with CFFA board.


Manual Boot: PR#n
Scan = False
PowerOn Boot
Slot n = 7 (or startup Slot on GS)
Scan = True
Read signature of card in Slot n
Does signature indicate a storage device?
No
n = n -1
Is n > 0?
Yes
No
Jump to AppleSoft at $E000
Yes
Call slot PROM at $Cn00
Display CFFA boot failure msg.
No
No
Is a device attached to card?
No
Delay .1s have 10 sec. elapsed?
Yes
Is Scan True?
Yes
Yes
Read boot blk 0
(LBA mode = 1)
Any errors durning read of boot block?
Yes
No
Is 1st byte of boot block = $01 AND 2nd <> $00
No
Yes
Jump to boot code at $801
Key:
Gray boxes represent code in Apple's ROM
White boxes represent code in CFFA's firmware

EEPROM Layout

U5 located in the middle of the board is an Atmel AT28C64E-15SI 64Kbit EEPROM. The E stands for high write endurance (100K writes). There are two 4K sections of this 8KB EEPROM. Which section gets mapped into the Apple's address space starting at $C000 is determined by jumper JP3. Because the EEPROM is mapped over the **entire** I/O space, the EEPROM has a unique image for **each** slot ROM. The firmware supports this setup by repeating substantially the same code for each $CnXX slot page: $C100, $C200, to $C700.

Note that even though the base address for the EEPROM is $C000, the EEPROM is not enabled for addresses in the range of $C000 to $C0FF or for locations $CFFF and $CFEF. Table 8 shows the relationship between an address on the Apple's bus and the EEPROM response. The address $CFEF, if accessed, will deselect the EEPROM just like access to $CFFF would. This is because address line A4 is not connected to the CPLD (U6) so the address $CFFF could not be uniquely decoded.

Table 8: EEPROM Offsets shown when jumper JP3 is installed

Address on Apple's Bus	EEPROM Offset	EEPROM's Response
$C000 to $C0FF	$0 to $FF	Not Used. EEPROM never enabled in this range.
$C100 to $C1FF	$100 to $1FF	Slot 1 ROM Space. Enabled when board is in slot 1.
$C200 to $C2FF	$200 to $2FF	Slot 2 ROM Space. Enabled when board is in slot 2.
$C300 to $C3FF	$300 to $3FF	Slot 3 ROM Space. Enabled when board is in slot 3.
$C400 to $C4FF	$400 to $4FF	Slot 4 ROM Space. Enabled when board is in slot 4.
$C500 to $C5FF	$500 to $5FF	Slot 5 ROM Space. Enabled when board is in slot 5.
$C600 to $C6FF	$600 to $6FF	Slot 6 ROM Space. Enabled when board is in slot 6.
$C700 to $C7FF	$700 to $7FF	Slot 7 ROM Space. Enabled when board is in slot 7.
$C800 to $CFFF*	$800 to $FFF	Expansion ROM space. Must be previously enabled by access to $CnXX. (n = slot)
* NOTE: locations $CFEF $CFFF		EEPROM never enabled for these two locations. Any access to either disables Expansion ROM space. But always use read of $CFFF to disable.

CFFA Hardware Memory Map

Table 9 shows all of the slot-specific I/O addresses decoded by the CFFA board. These addresses are used to interface a storage device's task register file to the Apple's bus. There is an extra register to allow the translation from the 16-bit ATA device to the Apple's 8-bit bus. Also, due to a bug in some CF card implementations of "TrueIDE" mode, there are two special soft switches used to inhibit CPU read cycles from confusing CF cards during block write routines.

Table 9: Slot specific I/O used by the CFFA Board

Apple Address (for Slot 7)	**Name Used in Source Code**	**Read / Write**	**Description**
$C080+$n0 ($C0F0)	ATADataHigh	R/W	This register is used in combination with the ATADataLow register $C080+$n8. See $C080+$n8 description.
$C080+$n1 ($C0F1)	SetCSMask	R/W	Special soft switch to **disable** CS0 & CS1 signaling to attached device during 65C02 read cycles that always precede write cycles
$C080+$n2 ($C0F2)	ClearCSMask	R/W	Special soft switch to **enable** CS0 & CS1 signaling to attached device during 65C02 read cycles that always precede write cycles
$C080+$n3 ($C0F3)	WriteEEPROM	R/W	Special soft switch to **enable** override of the write protect jumper (JP2) for locations $C800 to $C81F.
$C080+$n4 ($C0F4)	NoWriteEEPROM	R/W	Special soft switch to **disable** override of the write protect jumper (JP2) for locations $C800 to $C81F.
$C080+$n5 ($C0F5)			Unused
$C080+$n6 ($C0F6)	ATADevCtrl	W	This register is used to control the device's interrupt request line and to issue an ATA soft reset to the device.
$C080+$n6 ($C0F6)	ATAAltStatus	R	This register returns the device status when read by the host. Reading the ATAAltStatus register does NOT clear a pending interrupt.
$C080+$n7 ($C0F7)			Unused

Apple Address (for Slot 7)	**Name Used in Source Code**	**Read / Write**	**Description**
$C080+$n8 ($C0F8)	ATADataLow	R/W	This register is used to transfer data between the host and the attached device. It is used in combination with the ATADataHigh register to form a 16 bit data word. When reading words from the attached device, this register **must** be read first, and then the High byte can be read from the ATADataHigh register. When writing to the attached device, the ATADataHigh register **must** be written first, and then the ATADataLow register can be written.
$C080+$n9 ($C0F9)	ATAError	R	This register contains additional information about the source of an error when an error is indicated in bit 0 of the ATAStatus register.
$C080+$nA ($C0FA)	ATASectorCnt	R/W	This register contains the number of blocks of data requested to be transferred on a read or write operation between the host and the device. If the value in this register is zero, a count of 256 sectors is specified.
$C080+$nB ($C0FB)	ATASector	R/W	This register contains the starting sector number or bits 7-0 of the Logical Block Address (LBA) for any device access for the subsequent command.
$C080+$nC ($C0FC)	ATACylinder	R/W	This register contains the low order 8 bits of the starting cylinder address or bits 15-8 of the Logical Block Address.
$C080+$nD ($C0FD)	ATACylinderH	R/W	This register contains the high order bits of the starting cylinder address or bits 23-16 of the Logical Block Address.
$C080+$nE ($C0FE)	ATAHead	R/W	This register is used to select LBA addressing mode or cylinder/head/sector addressing mode. Also bits 27-24 of the Logical Block Address.
$C080+$nF ($C0FF)	ATACommand	W	A write to this register will issue an ATA command to the device.
$C080+$nF ($C0FF)	ATAStatus	R	This register returns the device status when read by the host.

n = the slot number in which the CFFA board is installed.

Marketing Megabytes

At some point when storage device sizes became sufficiently large, some marketing genius* decided that their products would appear more impressive if they used the standard SI definition of the prefix Mega, where 1 MB = 1,000,000 Bytes instead of the widely used computer-centric definition where 1 MB = 1024 x 1024 = 1,048,576 Bytes. It appears that after one company did this the rest were forced to follow suit.

Therefore, you may notice that the last drive on your storage device has a few less blocks than you may have expected. For example, one might expect that a SanDisk 64 MB CF card would provide space for two full 32 MB drives. However, this is not the case. It provides only 29.25 MB (30.67 MB marketing) for the second drive.

Given that the SanDisk 64 MB card that has 125440 ($1EA00) blocks instead of the expected 131072 ($20000) blocks, and that the first drive consumes the first 65536 ($10000) blocks (ProDOS wastes 1 block), that leaves the second drive with only 59904 ($EA00) blocks. Which is a drive size of 29.25 MB (30.67 MB marketing), but not 32 MB.

Note: *Block size is always 512 ($200) bytes.*

** Sarcasm alert*

Contact Information

The following is a list of information resources for the CFFA board project. If you have questions, comments, or problems to report, please contact me using one of the methods listed below.

CFFA Web Site

The CFFA web site is located at: http://dreher.net/CFforAppleII/. There you will find any new firmware revisions, project revisions, and general project status information.

Internet E-Mail

I can be reached via E-mail at: rich@dreher.net.

If you are reporting a problem, please use “CFFA problem“ or something similar as your subject line. In your E-mail you should include the firmware version number, which can be determined by entering the configuration menu. The version is display in the banner at the top. Also, if possible, describe the conditions necessary to cause the problem.

CFFA Message Web Forum

To post a message in the forum, simply browse to: http://dreher.net/phpBB/. The CFFA message forum is a good place to post technical problems and solutions. Other users may have had similar problems and know of a possible solution. You will have to register before you can post to the forum the first time. From time to time I close the registration for the forum because of spam bots. If registration is closed just email me with a user name and password and I will create the account manually for you.

Acknowledgements

I would like to thank the following people for providing help on this project:

Sherry Dreher

Chris Schumann

Dave Lyons

Josh King

Jeff Pagel

Vince Briel

Rob Greene

Dave Schmenk

Appendix 1: Firmware Listing

The following is a listing of the Firmware #0 located on the EEPROM at offset $0000, typically accessed when jumper JP3 is installed. The code accessed when JP3 is removed is not shown in this listing. Please check for future firmware listings on the CFFA web site. The list file is included in each firmware archive. A very special thanks to Dave Lyons for developing firmware v2.0 and implementing the configurable firmware concept.

Listing 2: CFFA Firmware v2.0 for 65C02 (JP3 IN)

```
ca65 V2.11.0 - (C) Copyright 1998-2005 Ullrich von Bassewitz
Main file   : CFFAv2.0.s
Current file: CFFAv2.0.s

000000r 1               ;-----------------------------------------------------------------------------
000000r 1               ; ProDOS/SmartPort driver for CompactFlash/IDE Interface for Apple II computers
000000r 1               ; CFFA.s  Version 2.0 - 27-Apr-2008
000000r 1               ;
000000r 1               ; Firmware Contributors:        Email:
000000r 1               ;  Chris Schumann                cschumann@twp-llc.com
000000r 1               ;  Rich Dreher                   rich@dreher.net
000000r 1               ;  Dave Lyons                    dlyons@lyons42.com
000000r 1               ;
000000r 1               ; This code can be built to require a 65C02 or 65816, or to work on a 6502.
000000r 1               ;
000000r 1               ; Tools used to build this driver: CA65: 6502 Cross Assembler
000000r 1               ;     http://www.cc65.org/
000000r 1               ;
000000r 1               ; Here is the copyright from that tool using --version option
000000r 1               ;    ca65 V2.11.0 - (C) Copyright 1998-2005 Ullrich von Bassewitz
000000r 1               ;
000000r 1               ; Example build instructions on an MSDOS based machine:
000000r 1               ;------------------------------------------------------
000000r 1               ; Assumes you have installed the CC65 package and set your path, etc.
000000r 1               ;
000000r 1               ; 1) c:\firmware> ca65 -t apple2 --cpu 65C02 -l CFFA.s
000000r 1               ; 2) c:\firmware> ld65 -t apple2 CFFA.o -o CFFA.bin
000000r 1               ;
000000r 1               ; 3) If you have a newer, EEPROM-based CFFA card:
000000r 1               ;
000000r 1               ;    Use CFFA.UTIL to copy the firmware onto the card.  The Firmware Select
000000r 1               ;    jumper allows two versions of the firmware on the card (for example, a
000000r 1               ;    6502 version and a 65C02 version).
000000r 1               ;
000000r 1               ;    If you have made any changes to the "slot ROM" portion ($Cnxx), you
000000r 1               ;    must update the slot ROM repeatedly, once for each Apple II slot where
000000r 1               ;    you will ever use the CFFA card.  After updating the last slot, then
000000r 1               ;    you can update the Aux ROM, just once.
000000r 1               ;
000000r 1               ; -- or --
000000r 1               ;
000000r 1               ; 3) If you have an older, EPROM-based CFFA:
000000r 1               ;
000000r 1               ;    Because the EPROM can hold up to 8 user selectable versions of firmware
000000r 1               ;    you must load CFFA.bin into your EPROM programmer with one of the
000000r 1               ;    following offsets:
000000r 1               ;    (Note: this offset has nothing to do with the card slot offsets)
000000r 1               ;
000000r 1               ;  for driver #0: use offset $0000.  Selected with: A14= IN, A13= IN, A12= IN
000000r 1               ;  for driver #1: use offset $1000.  Selected with: A14= IN, A13= IN, A12=OUT
000000r 1               ;  for driver #2: use offset $2000.  Selected with: A14= IN, A13=OUT, A12= IN
000000r 1               ;  for driver #3: use offset $3000.  Selected with: A14= IN, A13=OUT, A12=OUT
000000r 1               ;  for driver #4: use offset $4000.  Selected with: A14=OUT, A13= IN, A12= IN
000000r 1               ;  for driver #5: use offset $5000.  Selected with: A14=OUT, A13= IN, A12=OUT
000000r 1               ;  for driver #6: use offset $6000.  Selected with: A14=OUT, A13=OUT, A12= IN
000000r 1               ;  for driver #7: use offset $7000.  Selected with: A14=OUT, A13=OUT, A12=OUT
000000r 1               ;
000000r 1               ;  where IN = jumper shorted, and OUT = jumper open
000000r 1               ;  Driver #0 through #7 correspond to the user selectable
000000r 1               ;  jumpers: J6 (A14, A13, A12) on the interface card.
000000r 1               ;
000000r 1               ; 4) Load as many firmware versions, up to 8, into the EPROM programmer as you
000000r 1               ;    want. Remember that most programmers will, by default, clear all unused
000000r 1               ;    memory to $FF when you load a binary file. You will need to disable that
000000r 1               ;    feature after loading the first file.
```

```
000000r 1               ;
000000r 1               ; 5) Now you have an EPROM ready image to be programmed.
000000r 1               ;    Using a standard 27C256 EPROM or similar, program your EPROM.
000000r 1               ;
000000r 1               ; Firmware Version History
000000r 1               ;-------------------------
000000r 1               ; Version 2.0 (February to April 2008, DAL)
000000r 1               ;   - Started with version 1.20 source and merged in the 6502-specific changes.
000000r 1               ;   - Merged in flow control changes from Dave Schmenk and made a further speed-
up
000000r 1               ;     in the Write loop.
000000r 1               ;   - Added support for Dev0 and Dev1 (2 devices on the ATA bus)
000000r 1               ;   - Added an interactive boot-time menu (press "M" during boot)
000000r 1               ;   - Allows booting from any partition (on either device).  On an EEPROM-based
000000r 1               ;     CFFA, you can save the setting into EEPROM, or just use it temporarily.
000000r 1               ;   - $C800 space contains user-configurable parameters: number of partitions,
000000r 1               ;     default startup partition, 3-byte block offset allowing the partitions
000000r 1               ;     to reside anywhere in the first 8GB of the device.
000000r 1               ;
000000r 1               ; 6502-only Version 1.0:  (6502 version located at EPROM offset $4000, selection
#4)
000000r 1               ;   - Start of 6502 specific driver. Based on Version 1.2 spdrv.s driver
000000r 1               ;   - Now works with 6502 CPU. Removed all 65C02 specific instruction from the
ver 1.2 firmware
000000r 1               ;   - NOTE: To work in 6502 based machine, logic CPLD version 1.3 or later is
required.
000000r 1               ;   - Changed error messages to upper case for Apple ]['s that don't support
lower case.
000000r 1               ;   - Set up slot specific behavior: Slot 5 has smartport enabled but will not
autoboot.
000000r 1               ;       All other slots have smartport disabled, but will autoboot in an Apple
][+ or ][e.
000000r 1               ;   - Made 4th boot signature byte a parameter to boot code macro to ease slot
specific setup.
000000r 1               ;   - Changed GSOS_DRIVER signature to $10. This will prevent Dave Lyons' GS/OS
driver from
000000r 1               ;       loading should someone inadvertently use this firmware in a IIgs.
000000r 1               ;   - Changed offsets for code loading with addition of .RES command
000000r 1               ;       bin files can now be loaded in EPROM at normal 4K offsets:$0, $1000,
$2000, etc,
000000r 1               ;       instead of offsets $100, $1100, $2100, etc.
000000r 1               ;
000000r 1               ; Version 1.2
000000r 1               ;   - Start of SmartPort driver. Based on Version 1.1 ProDOS driver
000000r 1               ;
000000r 1               ; Version 1.1
000000r 1               ;   - dynamically calculate drive sizes in GetStatus function
000000r 1               ;   - turn off interrupts in case boot code is called manually
000000r 1               ;   - add cardID/firmware revision bytes: CFFA$xx
000000r 1               ;   - added continuation of boot scan if device not present
000000r 1               ;   - added continuation of boot scan if boot block code looks invalid
000000r 1               ;   - reformatted this source file, removed tabs and added function headers
000000r 1               ;
000000r 1               ;
000000r 1               ; PLD Logic Firmware Information
000000r 1               ; -------------------------------
000000r 1               ; This version of firmware assumes you are using PLD logic of at
000000r 1               ; least CPLD logic version 1.3.  The source files for U6, the Altera PLD are:
000000r 1               ;       Appleideinterface.gdf
000000r 1               ;       Appleidelogic.tdf
000000r 1               ;
000000r 1               ; The programmer ready output file for the PLD logic is:
000000r 1               ;       Appleideinterface.pof
000000r 1               ;
000000r 1               ; These files are not included with this code.
000000r 1               ;
000000r 1               ; Acknowledgements
000000r 1               ; -----------------
000000r 1               ; Thanks to:
000000r 1               ;    Chris Schuman - for his extensive initial development work
000000r 1               ;    David Lyons   - for technical information, many improvement ideas and
000000r 1               ;                    SmartPort code development, and v2.0 work
000000r 1               ;    Dave Schmenk  - for read/write speed-ups
000000r 1               ;--------------------------------------------------------------------------------
000000r 1
000000r 1               .segment "EXEHDR"            ; just to keep the linker happy
000000r 1               .segment "STARTUP"           ; just to keep the linker happy
000000r 1
000000r 1               .linecont +                  ; Allow line continuations
000000r 1
000000r 1               .ifpc02
000000r 1                  USE_65C02 = 1
000000r 1               .else
000000r 1                  USE_65C02 = 0
000000r 1               .endif
000000r 1
000000r 1               REQUIRES_EEPROM           = 1
000000r 1               IN_DEVELOPMENT            = 0          ; show a non-"v" version letter
000000r 1
000000r 1               FULL_MENU                 = REQUIRES_EEPROM
000000r 1
000000r 1               MENU_IDENTIFY_DEVS = (USE_65C02 & FULL_MENU)  ; 87 bytes (as of 15-Mar-2008)
000000r 1               MENU_DIGIT_INPUT          = 0          ; 12 bytes
000000r 1               SMARTPORT_RAW_BLOCK_MAGIC = 0          ; 30 bytes
000000r 1               SMARTPORT_STATUS_D0_D1    = 0          ; 14 bytes
```

```
000000r 1               MENU_STAR_MONITOR         = 0          ; 10 bytes
000000r 1               WRITE_PROTECT             = 0          ; 18 bytes (incomplete, don't use)
000000r 1
000000r 1               .if WRITE_PROTECT
000000r 1                  .error "WRITE_PROTECT isn't finished.  It needs to affect status calls."
000000r 1               .endif
000000r 1
000000r 1               ;------------------------------------------------------------------------------
000000r 1               ; Firmware Version Information
000000r 1               ;------------------------------------------------------------------------------
000000r 1               FIRMWARE_VER    = $20        ; Version 2.0 (Version of this code)
000000r 1               SPDRIVERVERSION = $2000      ; Version of our SmartPort implementation (2.0)
000000r 1               ;
000000r 1               ; The GS/OS driver checks the GSOS_DRIVER byte to see if it is compatible with
000000r 1               ; this firmware.
000000r 1               ;
000000r 1               ; Increment by one, when something changes that would require a change
000000r 1               ; in the GS/OS driver.
000000r 1               ;    $01 = ProDOS Driver supporting 2 drives.
000000r 1               ;    $02 = SmartPort Driver supporting 4 drives
000000r 1               ;    $03 = SmartPort Driver supporting 8 drives. NOTE: GS/OS driver won't load
000000r 1               ;    $10 = 6502 version 1.0: ProDOS Driver supporting 2 drives. NOTE: GS/OS
driver won't load
000000r 1               ;    $11 = for CFFA firmware 2.0 (a future version of the GS/OS driver may load)
000000r 1               ;
000000r 1               GSOS_DRIVER     = $11
000000r 1
000000r 1               ;------------------------------------------------------------------------------
000000r 1               ; ProDOS request Constants
000000r 1               PRODOS_STATUS   = $00
000000r 1               PRODOS_READ     = $01
000000r 1               PRODOS_WRITE    = $02
000000r 1               PRODOS_FORMAT   = $03
000000r 1
000000r 1               ; ProDOS Return Codes
000000r 1               PRODOS_NO_ERROR      = $00    ; No error
000000r 1               PRODOS_BADCMD        = $01    ; Bad Command
000000r 1               PRODOS_IO_ERROR      = $27    ; I/O error
000000r 1               PRODOS_NO_DEVICE     = $28    ; No Device Connected
000000r 1               PRODOS_WRITE_PROTECT = $2B    ; Write Protected
000000r 1               PRODOS_BADBLOCK      = $2D    ; Invalid block number requested
000000r 1               PRODOS_OFFLINE       = $2F    ; Device off-line
000000r 1
000000r 1               ; SmartPort return codes
000000r 1               BAD_PARM_COUNT       = $04
000000r 1               BAD_UNIT_NUMBER      = $11
000000r 1               INTERRUPT_NOT_SUPT   = $1F
000000r 1               BAD_STATUS_CODE      = $21
000000r 1
000000r 1               ; ATA Commands Codes
000000r 1               ATACRead        = $20
000000r 1               ATACWrite       = $30
000000r 1               ATAIdentify     = $EC
000000r 1
000000r 1               ;------------------------------------------------------------------------------
000000r 1               ; Constants for Wait / ROMWait
000000r 1               ;   Constant = (Delay[in uS]/2.5 + 2.09)^.5 - 2.7
000000r 1               ;------------------------------------------------------------------------------
000000r 1               WAIT_100ms      = 197
000000r 1               WAIT_40ms       = 124
000000r 1               WAIT_100us      = 4
000000r 1
000000r 1               ;------------------------------------------------------------------------------
000000r 1               ; ASCII characters
000000r 1               ;------------------------------------------------------------------------------
000000r 1               BELL            = $07
000000r 1               kLeftArrow      = $08
000000r 1               kDownArrow      = $0A
000000r 1               kUpArrow        = $0B
000000r 1               CR              = $0D
000000r 1               kRightArrow     = $15
000000r 1               kEscape         = $1B
000000r 1
000000r 1
000000r 1               ;------------------------------------------------------------------------------
000000r 1               ; Apple II I/O locations
000000r 1               ;------------------------------------------------------------------------------
000000r 1               KEYBOARD        = $C000
000000r 1               KBDSTROBE       = $C010
000000r 1
000000r 1               ;------------------------------------------------------------------------------
000000r 1               ; Slot I/O definitions
000000r 1               ;------------------------------------------------------------------------------
000000r 1               MSLOT           = $7F8         ; Apple-defined -- $Cn for slot owning Aux ROM
000000r 1
000000r 1               IOBase          = $C080        ; indexed with X = $n0
000000r 1               ATADataHigh     = IOBase+0
000000r 1
000000r 1               ; SetCSMask / ClearCSMask - these strobe locations set and clear the MASK bit on
000000r 1               ; the CPLD that disables the CS0 line to the CompactFlash during the CPU read
000000r 1               ; cycles that occur before every CPU write cycle.  (These "false reads" would
000000r 1               ; cause a device to double increment during a write.)
000000r 1               SetCSMask       = IOBase+1
000000r 1               ClearCSMask     = IOBase+2
000000r 1               WriteEEPROM     = IOBase+3     ; with CPLD v2.1 - force-allow write $C800..C81F
```

```
000000r 1               NoWriteEEPROM   = IOBase+4      ; with CPLD v2.1 - respect W/P jumper for
$C800..C81F
000000r 1
000000r 1               ATADevCtrl      = IOBase+6      ; when writing
000000r 1               ATAAltStatus    = IOBase+6      ; when reading
000000r 1               ATADataLow      = IOBase+8
000000r 1               ATAError        = IOBase+9
000000r 1               ATASectorCnt    = IOBase+$0A
000000r 1               ATASector       = IOBase+$0B
000000r 1               ATACylinder     = IOBase+$0C
000000r 1               ATACylinderH    = IOBase+$0D
000000r 1               ATAHead         = IOBase+$0E
000000r 1               ATACommand      = IOBase+$0F    ; when writing
000000r 1               ATAStatus       = IOBase+$0F    ; when reading
000000r 1
000000r 1               ;
000000r 1               ; "Screen hole" Scratchpad RAM base addresses.
000000r 1               ; Access using the Y register containg the slot number (lda DriveNumber,Y)
000000r 1               ;
000000r 1               DriveResetDone  = $478          ; remember the device has been software reset
000000r 1               DriveNumber     = $4F8          ; 0 for the first 32MB partition, 1 for the
second, ...
000000r 1               BootDevice      = $578          ; 0 = boot from Dev0, 1 = boot from Dev1
000000r 1               TempScreenHole  = $5F8
000000r 1               PartitionsDev0  = $678          ; number of 32MB partitions on Dev0
000000r 1               PartitionsDev1  = $6F8          ; number of 32MB partitions on Dev1
000000r 1               DrvMiscFlags    = $778          ; see kMiscXXX constants below
000000r 1               BootPartition   = $7F8          ; physical index of boot partition (0 for no
remapping)
000000r 1
000000r 1               kMiscRaw        = $80           ; set this bit to read/write blocks on an entire
device
000000r 1               kMiscDev1       = $40           ; set this bit in DrvMiscFlags,y to access
Device 1
000000r 1
000000r 1               ;-----------------------------------------------------------------------------
000000r 1               ; Zero-page locations
000000r 1               ;-----------------------------------------------------------------------------
000000r 1               CH              = $24
000000r 1               INVFLG          = $32
000000r 1                 kNormalText   = $FF
000000r 1                 kInverseText  = $3F
000000r 1                 kFlashingText = $7F
000000r 1
000000r 1               zpt1            = $EF           ; data at this location is saved/restored
000000r 1               bootError       = $FF
000000r 1
000000r 1               ; Used only during the boot-time menu
000000r 1               menuSlot        = spSlot
000000r 1               menuSlot16      = spSlotX16
000000r 1               menuRow         = $82
000000r 1               menuMustSave    = $83
000000r 1               menuJumpVectorH = $84
000000r 1               menuValues      = $85           ; kMenuInteractiveRows (4) bytes
000000r 1               WriteOneByteToEEPROM = $90      ; 16 bytes
000000r 1
000000r 1               ; Saved and restored, used during Status calls to hold block counts
000000r 1               blockCount      = $06           ; $06 to $08 (3 bytes)
000000r 1
000000r 1               ; ProDOS block interface locations
000000r 1               pdCommandCode   = $42
000000r 1               pdUnitNumber    = $43
000000r 1               pdIOBuffer      = $44
000000r 1               pdIOBufferH     = $45
000000r 1               pdBlockNumber   = $46
000000r 1               pdBlockNumberH  = $47
000000r 1
000000r 1               ; Arbitrary locations for Smartport data,
000000r 1               ;  these locations are saved/restored before exit.
000000r 1               spCommandCode   = pdCommandCode
000000r 1
000000r 1               spSlot          = $40           ; menuSlot = spSlot
000000r 1               spSlotX16       = $41           ; menuSlot16 = spSlotX16
000000r 1
000000r 1               spParamList     = $48           ; 2 bytes
000000r 1               spCmdList       = $4A           ; 2 bytes
000000r 1               spCSCode        = $4C
000000r 1               spLastZP        = spCSCode
000000r 1
000000r 1               spZeroPgArea    = spSlot        ; $40
000000r 1               spZeroPgSize    = spLastZP-spZeroPgArea+1
000000r 1
000000r 1               StackBase       = $100
000000r 1
000000r 1               ResetVector     = $3F2
000000r 1
000000r 1               ;-----------------------------------------------------------------------------
000000r 1               ; SmartPort constants
000000r 1               ;-----------------------------------------------------------------------------
000000r 1               ; Use these in SmartPort ReadBlock/WriteBlock calls for raw LBA blocks.
000000r 1               .if SMARTPORT_RAW_BLOCK_MAGIC
000000r 1               MagicRawBlocksDev0Unit = 127
000000r 1               MagicRawBlocksDev1Unit = 126
000000r 1               .endif
000000r 1
```

```
000000r 1               SMARTPORT_STATUS     = 0
000000r 1               SMARTPORT_READ       = 1
000000r 1               SMARTPORT_WRITE      = 2
000000r 1               SMARTPORT_FORMAT     = 3
000000r 1               SMARTPORT_CONTROL    = 4
000000r 1               SMARTPORT_INIT       = 5
000000r 1               SMARTPORT_OPEN       = 6
000000r 1               SMARTPORT_CLOSE      = 7
000000r 1
000000r 1               SP_STATUS_IDENT_DEV0 = $49     ; 'I' for Identify
000000r 1               SP_STATUS_IDENT_DEV1 = $48     ; identify the other device
000000r 1
000000r 1               ;------------------------------------------------------------------------------
000000r 1               ; Apple II ROM entry points
000000r 1               ;
000000r 1               ; We can use these at boot time, but not while handling a call through
000000r 1               ; our ProDOS or SmartPort entry points, as the ROM may not be available.
000000r 1               ;------------------------------------------------------------------------------
000000r 1               AppleSoft       = $E000
000000r 1               ResumeBootScan  = $FABA     ; SLOOP
000000r 1               TABV            = $FB5B
000000r 1               SETPWRC         = $FB6F     ; fix up Reset vector
000000r 1               BELL1           = $FBDD     ; delay, then beep
000000r 1               KEYIN           = $FD0C
000000r 1               CROUT           = $FD8E
000000r 1               PRBYTE          = $FDDA
000000r 1               COUT            = $FDED
000000r 1               SetKBD          = $FE89
000000r 1               SetVID          = $FE93
000000r 1               INIT            = $FB2F
000000r 1               HOME            = $FC58
000000r 1               ROMWAIT         = $FCA8
000000r 1               SETNORM         = $FE84
000000r 1               MONITOR         = $FF69
000000r 1
000000r 1               ;------------------------------------------------------------------------------
000000r 1               ; Macros
000000r 1               ;------------------------------------------------------------------------------
000000r 1               .macro BRA_OR_JMP Destination
000000r 1                  .IF USE_65C02
000000r 1                      bra Destination
000000r 1                  .ELSE
000000r 1                      jmp Destination
000000r 1                  .ENDIF
000000r 1               .endmacro
000000r 1
000000r 1               .macro INC_A_OR_ADC1
000000r 1                  .IF USE_65C02
000000r 1                     inc a
000000r 1                  .ELSE
000000r 1                     clc
000000r 1                     adc #1
000000r 1                  .ENDIF
000000r 1               .endmacro
000000r 1
000000r 1               .macro DEC_A_OR_SBC1
000000r 1                  .IF USE_65C02
000000r 1                     dec a
000000r 1                  .ELSE
000000r 1                     sec
000000r 1                     sbc #1
000000r 1                  .ENDIF
000000r 1               .endmacro
000000r 1
000000r 1               .macro STZ_OR_LDA_STA mem
000000r 1                  .IF USE_65C02
000000r 1                     stz mem
000000r 1                  .ELSE
000000r 1                     lda #0
000000r 1                     sta mem
000000r 1                  .ENDIF
000000r 1               .endmacro
000000r 1
000000r 1               .macro STZ_OR_STA_ATADataHigh_X
000000r 1                  .IF USE_65C02
000000r 1                     stz ATADataHigh,x           ; Clear high byte data latch
000000r 1                  .ELSE
000000r 1                     lda #0
000000r 1                     sta ATADataHigh,x           ; Clear high byte data latch
000000r 1                  .ENDIF
000000r 1               .endmacro
000000r 1
000000r 1               .macro PHY_OR_TYA_PHA
000000r 1                  .IF USE_65C02
000000r 1                     phy
000000r 1                  .ELSE
000000r 1                     tya
000000r 1                     pha                         ; save Y, it holds the $0n sratchpad RAM
offset
000000r 1                  .ENDIF
000000r 1               .endmacro
000000r 1
000000r 1               .macro PLY_OR_PLA_TAY
000000r 1                  .IF USE_65C02
000000r 1                     ply
```

```
000000r 1                  .ELSE
000000r 1                     pla
000000r 1                     tay
000000r 1                  .ENDIF
000000r 1               .endmacro
000000r 1
000000r 1               .macro ASSERT condition, string
000000r 1                  .if condition
000000r 1                  .else
000000r 1                     .error string
000000r 1                  .endif
000000r 1               .endmacro
000000r 1
000000r 1               .macro WARN condition, string
000000r 1                  .if condition
000000r 1                  .else
000000r 1                     .warning string
000000r 1                  .endif
000000r 1               .endmacro
000000r 1
000000r 1               ; String macro from <http://www.cc65.org/mailarchive/2003-05/2995.html>
000000r 1               .macro HiBitString str
000000r 1                  .repeat .strlen(str), i
000000r 1                     .byte .strat(str,i) | $80
000000r 1                  .endrepeat
000000r 1               .endmacro
000000r 1
000000r 1               ; SentinelString sets bit 7 on the final character in the string.
000000r 1               .macro SentinelString str
000000r 1                  .repeat .strlen(str) - 1, i
000000r 1                     .byte .strat(str,i)
000000r 1                  .endrepeat
000000r 1                  .byte .strat(str, .strlen(str) - 1) | $80
000000r 1               .endmacro
000000r 1
000000r 1
000000r 1               ;-------------------------------------------------------------------------
000000r 1               ; BEGIN ROM CONTENT
000000r 1               ;-------------------------------------------------------------------------
000000r 1
000000r 1               ;-------------------------------------------------------------------------
000000r 1               ; $C000
000000r 1               ;
000000r 1               ; The base address for this 4K of code is $C000.  The first page ($C0xx) is
000000r 1               ; not available in the Apple II address space, so we'll just fill it with
000000r 1               ; something you might want to see if you have a copy of the firmware on
000000r 1               ; disk.
000000r 1               ;-------------------------------------------------------------------------
000000r 1                  .ORG $C000
00C000  1  43 46 46 41     .byte "CFFA Firmware",CR,CR,CR
00C004  1  20 46 69 72
00C008  1  6D 77 61 72
00C010  1  53 65 65 20     .byte "See <http://dreher.net/CFforAppleII/>."
00C014  1  3C 68 74 74
00C018  1  70 3A 2F 2F
00C036  1  0D 0D 0D 0D     .byte CR,CR,CR,CR,CR,CR,CR,CR,CR,CR
00C03A  1  0D 0D 0D 0D
00C03E  1  0D 0D
00C040  1  56 65 72 73     .byte "Version "
00C044  1  69 6F 6E 20
00C048  1  32 2E 30        .byte $30+(FIRMWARE_VER/16), ".", $30+(FIRMWARE_VER & $F)
00C04B  1  20 66 6F 72     .byte " for "
00C04F  1  20
00C050  1               .IF USE_65C02
00C050  1  36 35 43 30     .byte "65C02"
00C054  1  32
00C055  1               .ELSE
00C055  1                  .byte "6502"
00C055  1               .ENDIF
00C055  1  20 6F 72 20     .byte " or later.",CR,CR
00C059  1  6C 61 74 65
00C05D  1  72 2E 0D 0D
00C061  1               .IF REQUIRES_EEPROM
00C061  1  52 65 71 75     .byte "Requires CFFA with EEPROM, not EPROM.",CR,CR
00C065  1  69 72 65 73
00C069  1  20 43 46 46
00C088  1               .ELSE
00C088  1                  .byte "Compatible with older EPROM-based CFFA cards.",CR,CR
00C088  1               .ENDIF
00C088  1  FF FF FF FF     .RES $C100-*, $FF            ; fill the rest of the $C0xx area (unused)
00C08C  1  FF FF FF FF
00C090  1  FF FF FF FF
00C100  1
00C100  1               ;------------------------------------------------------------------------------
00C100  1               ; $C100
00C100  1               ;
00C100  1               ; Start of Peripheral Card ROM Space $Cn00 to $CnFF
00C100  1               ;
00C100  1               ; A macro (CnXX) is used here so that this code can be easily generated for
00C100  1               ; each slot, with minor variations.
00C100  1               ;
00C100  1               ; This is done because the hardware does not overlay the C1xx address space at
00C100  1               ; C2xx, C3xx, etc. automatically. Instead, the base address for the ROM is
00C100  1               ; $C000, and a read from $Cnxx is mapped to $0nxx in our ROM.
00C100  1               ;
```

```
00C100  1               ; ROM offsets $0000 to $00FF are not used.  $0nxx is used for slot n.  From
00C100  1               ; $0800 to $0FFE is the 2K expansion ROM.
00C100  1               ;
00C100  1               ; In an older EPROM-based CFFA card, the expansion ROM stops at $CEFF, and any
00C100  1               ; access to $CFxx turns off the expansion ROM -- so $0F00 to $0FFF are not used.
00C100  1               ;
00C100  1               ; In a newer EEPROM-based CFFA card, the expansion ROM continues all the way to
00C100  1               ; $CFFE, except that $CFEF and $CFFF both disable the expansion ROM.
00C100  1               ;-----------------------------------------------------------------------------
00C100  1               .macro CnXX SLOT, SIGNATURE_BYTE_4
00C100  1                  .local P8DriverEntry
00C100  1                  .local SmartPortEntry
00C100  1                  .local commonEntry
00C100  1                  .local Boot
00C100  1                  .local MenuEntry
00C100  1                  .local notMenu
00C100  1                  .local PrepareForJumpToAuxROM
00C100  1                  .local InvalidBootBlock
00C100  1                  .local Error
00C100  1                  .local ShowErrorMessage
00C100  1                  .local ShowErrorMsgY
00C100  1                  .local ErrorMessages
00C100  1                  .local msgCheckTheDevice
00C100  1                  .local msgNoBootBlock
00C100  1                  .local msgCouldNotBootPartition
00C100  1
00C100  1                  lda #$20                    ; $20 is a signature for a drive to ProDOS
00C100  1                  ldx #$00                    ; $00 "
00C100  1                  lda #$03                    ; $03 "
00C100  1                  ASSERT (SIGNATURE_BYTE_4 < $80), "SIGNATURE_BYTE_4 must be <$80 (BPL)"
00C100  1                  lda #SIGNATURE_BYTE_4       ; $00 for SmartPort, $3C to look like a Disk II
00C100  1                  bpl Boot                    ; NOTE: this must be a 2 byte instuction to keep
the entry point offset correct
00C100  1
00C100  1               ;------------------- Non-boot P8 driver entry point ---------------------
00C100  1               ; ProDOS block device entry point (READ, WRITE, STATUS, FORMAT).
00C100  1               ;----------------------------------------------------------------------
00C100  1                  ASSERT (* = $C00A+$100*SLOT), "Should be at $Cn0A"
00C100  1               P8DriverEntry:                  ; At $Cn0A for best compatibility.
00C100  1                  clc
00C100  1                  bcc commonEntry
00C100  1                  ASSERT (* = $C00D+$100*SLOT), "Should be at $Cn0D"
00C100  1               SmartPortEntry:                 ; At $Cn0D: SmartPort entry must be 3 bytes
later
00C100  1                  sec
00C100  1               commonEntry:
00C100  1                  jsr PrepareForJumpToAuxROM
00C100  1                  jmp P8_SmartPort_Handler
00C100  1
00C100  1               ;----------------------------------------------------------------------
00C100  1               ; Boot
00C100  1               ;
00C100  1               ; Start up the Apple II from our device.  Typically we will get here
00C100  1               ; because of a "boot scan" (the Apple II is searching for a bootable
00C100  1               ; device), or because of a direct PR#n for our slot.
00C100  1               ;
00C100  1               ; If the user has pressed the "menu snag" key, we'll display an
00C100  1               ; interactive settings menu.
00C100  1               ;
00C100  1               ; Otherwise, we'll try to read the boot block (block 0) from the
00C100  1               ; configured boot partition and then jump to the boot code.
00C100  1               ;----------------------------------------------------------------------
00C100  1               Boot:
00C100  1                  jsr PrepareForJumpToAuxROM  ; also needed to access config params below
00C100  1                  stx pdUnitNumber            ; prepare to boot from "drive 1"
00C100  1
00C100  1               ;
00C100  1               ; Wait here (before the first CheckDevice) in case the CFFA RESET jumper
00C100  1               ; is enabled, or a Delkin Devices CF card never becomes ready.
00C100  1               ;
00C100  1                  ldy BootTimeDelayTenths
00C100  1               @wasteTime:
00C100  1                  lda #WAIT_100ms
00C100  1                  jsr ROMWAIT
00C100  1                  dey
00C100  1                  bne @wasteTime
00C100  1               ;
00C100  1               ; Pressing a certain key (normally "M") at boot time brings up
00C100  1               ; the interactive menu.
00C100  1               ;
00C100  1                  lda KEYBOARD
00C100  1                  and MenuSnagMask
00C100  1                  cmp MenuSnagKey
00C100  1                  bne notMenu
00C100  1
00C100  1                  .RES $C000+(SLOT*$100)+$30-*,$EA  ; fill with NOPs to $Cn30
00C100  1               ;
00C100  1               ; MenuEntry -- available from the monitor as Cn30G
00C100  1               ;
00C100  1               ; (Because this is a supported user entry point, we can't assume that
00C100  1               ; we have already called PrepareForJumpToAuxROM above.)
00C100  1               ;
00C100  1               MenuEntry:
00C100  1                  ASSERT MenuEntry=$C030+(SLOT*$100), "MenuEntry should be $Cn30"
00C100  1                  jsr PrepareForJumpToAuxROM
```

```
00C100  1                  jsr InteractiveMenu
00C100  1
00C100  1               notMenu:
00C100  1                  ldy #PRODOS_READ            ; Request: READ block
00C100  1                  sty pdCommandCode
00C100  1                  ASSERT PRODOS_READ=1, "PRODOS_READ must be 1"
00C100  1                  dey                         ; Y = 0
00C100  1                  sty pdBlockNumber           ; ProDOS block $0000 (the bootloader block)
00C100  1                  sty pdBlockNumberH
00C100  1                  lda #$08
00C100  1                  sta pdIOBufferH
00C100  1                  sty pdIOBuffer              ; Into Location $0800
00C100  1               ; Read the block by calling a special AuxROM entry point.
00C100  1               ; We could just use P8DriverEntry, but this allows future firmware
00C100  1               ; changes to the boot process without changes to the $Cn ROM.
00C100  1                  jsr PrepareForJumpToAuxROM
00C100  1                  jsr BootROM_ProDOS_Call
00C100  1                  sta bootError               ; store on zero page for error display
00C100  1                  bcs Error
00C100  1                  ldy $800                    ; Check the first byte of boot loader code.
00C100  1                  dey                         ; If bootload code is there, this byte = $01
00C100  1                  bne InvalidBootBlock
00C100  1                  lda $801                    ; If second byte is a 0, it's invalid
00C100  1                  beq InvalidBootBlock        ;   (we'd JMP to a BRK)
00C100  1
00C100  1                  ldx pdUnitNumber            ; X should contain the unit number when jumping
00C100  1                                              ;  to the bootloader
00C100  1                  jmp $801                    ; Jump to the bootloader we just read.
00C100  1
00C100  1
00C100  1               ; If any error occured, like drive not present, check to see if we are in a
00C100  1               ; boot scan, if so re-enter scan routine, else drop to Applesoft, aborting boot.
00C100  1               InvalidBootBlock:
00C100  1               Error:
00C100  1                  lda $00
00C100  1                  bne @notScanning
00C100  1                  lda $01
00C100  1                  cmp MSLOT
00C100  1                  bne @notScanning
00C100  1                  jmp ResumeBootScan          ; Re-enter Monitor's Autoscan Routine
00C100  1               @notScanning:
00C100  1               ;
00C100  1               ; The boot code must have been called manually because we are not in a slot
scan.
00C100  1               ;
00C100  1               ShowErrorMessage:
00C100  1                  jsr PrepareToShowErrorMessage   ; clear screen and show banner
00C100  1
00C100  1                  ldy #msgCheckTheDevice-ErrorMessages
00C100  1                  ldx bootError
00C100  1                  bne @someError
00C100  1                  ldy #msgNoBootBlock-ErrorMessages
00C100  1               @someError:
00C100  1                  jsr ShowErrorMsgY
00C100  1
00C100  1                  txa
00C100  1                  beq @skipDevNumber
00C100  1
00C100  1               ; show dev "0" or "1" after "Check device "
00C100  1                  lda DrvMiscFlags+SLOT
00C100  1                  ASSERT kMiscDev1=$40, "kMiscDev1 must be bit 6"
00C100  1                  asl a
00C100  1                  asl a
00C100  1                  lda #'0'+$80
00C100  1                  adc #0
00C100  1                  jsr COUT
00C100  1               @skipDevNumber:
00C100  1
00C100  1                  ldy #msgCouldNotBootPartition-ErrorMessages
00C100  1                  jsr ShowErrorMsgY
00C100  1
00C100  1                  jsr PrepareForJumpToAuxROM     ; set up X and Y for slot
00C100  1                  jmp FinishBootTimeErrorMsg
00C100  1
00C100  1               ;--------------------------------------------------------------------
00C100  1               ; ShowErrorMsgY
00C100  1               ;
00C100  1               ; Display a message from the boot-ROM table (ErrorMessages).
00C100  1               ;
00C100  1               ; Input:  Y = offset from ErrorMessages.
00C100  1               ;--------------------------------------------------------------------
00C100  1               ShowErrorMsgY:
00C100  1               @loop:
00C100  1                  lda ErrorMessages,y
00C100  1                  beq @done
00C100  1                  jsr CaseSafeCOUT
00C100  1                  iny
00C100  1                  bne @loop
00C100  1               @done:
00C100  1                  rts
00C100  1
00C100  1               ;--------------------------------------------------------------------
00C100  1               ; PrepareForJumpToAuxROM
00C100  1               ;
00C100  1               ; Output - MSLOT = $Cn, X = $n0, Y = $0n
```

```
00C100  1               ;          A = 0
00C100  1               ;          Carry - preserved
00C100  1               ;
00C100  1               ; Turns off any other card's Aux ROM.
00C100  1               ; Resets the CFFA's CS0 mode.
00C100  1               ;
00C100  1               ; Verifies that the 2.0-or-later Aux ROM is present (in case the firmware
00C100  1               ; has been incompletely upgraded, or incompletely downgraded).
00C100  1               ;------------------------------------------------------------------------
00C100  1               PrepareForJumpToAuxROM:
00C100  1                  sta ClearCSMask+SLOT*16     ; reset MASK bit in PLD for normal CS0 signaling
00C100  1
00C100  1               ; We must set MSLOT to $Cn *before* touching $CFFF, in case an interrupt
00C100  1               ; occurs.  The interrupt handler uses it to re-establish Aux ROM ownership.
00C100  1                  lda #$C0+SLOT
00C100  1                  sta MSLOT
00C100  1                  bit $cfff                   ; turn off other ROM that might be on.
00C100  1
00C100  1                  lda C820_Signature          ; must have $CF if the Aux ROM is present
00C100  1                  eor #$CF                    ; compare without disturbing the carry
00C100  1                  bne @AuxROMTrouble
00C100  1
00C100  1                  ldy #SLOT                   ; Y reg now has $0n for accessing scratchpad RAM
00C100  1                  ldx #SLOT*16                ; X reg now has $n0 for indexing I/O
00C100  1                  rts
00C100  1
00C100  1               ; The 2.0-and-later AuxROM is not there.  Just BRK into the monitor.
00C100  1               @AuxROMTrouble:
00C100  1                  jsr SetVID                  ; undo a PR#n to avoid an infinite reboot loop
00C100  1                  brk
00C100  1
00C100  1               ;--------------------------------------------------------------------------
00C100  1               ; ErrorMessages table
00C100  1               ;
00C100  1               ; Each string ends with a 0 byte.
00C100  1               ;--------------------------------------------------------------------------
00C100  1               ErrorMessages:
00C100  1               msgCheckTheDevice:
00C100  1                  .byte "Check device ",0
00C100  1               msgNoBootBlock:
00C100  1                  .byte "No boot block",0
00C100  1               msgCouldNotBootPartition:
00C100  1                  .byte ".",CR,CR,"Could not boot partition ",0
00C100  1
00C100  1               ;--------------------------------------------------------------------------
00C100  1               ; $CnF5 - $CnFF -- Boot ROM signature, version, and capability ID bytes
00C100  1               ;
00C100  1               ; $CnF5 to $CnFA were defined by CFFA, but should no longer be used.
00C100  1               ;     Instead, see the similar bytes in the Aux ROM ($C8xx) area.
00C100  1               ;
00C100  1               ; $CnFB to $CnFF are defined by ProDOS and SmartPort.
00C100  1               ;--------------------------------------------------------------------------
00C100  1                  .RES   $C000+(SLOT*$100)+$F5-*,$77  ; skip to $CnF5
00C100  1
00C100  1                  .byte $FF                   ; was GSOS_DRIVER value (see $C8xx version area)
00C100  1                  .byte "CFFA", $FF           ; $CnF6..CnFA: Card ID, old 1.x version #
00C100  1
00C100  1               ; $CnFB: SmartPort status byte
00C100  1                  .byte $0                    ; Not Extended; not SCSI; not RAM card
00C100  1                                              ; Even if not supporting SmartPort, we need a
00C100  1                                              ; zero at $CnFB so Apple's RAMCard driver
00C100  1                                              ; doesn't mistake us for a "Slinky" memory
00C100  1                                              ; card.
00C100  1
00C100  1               ; Data table for ProDOS drive scan
00C100  1               ; $CnFC/FD = disk capacity, if zero use status command to determine
00C100  1                  .word $0000                 ; $CnFC-D: A zero here will cause ProDOS to
00C100  1                                              ; rely on the STATUS command to determine volume
size
00C100  1
00C100  1               ; $CnFE = status bits (BAP p7-14)
00C100  1               ;  7 = medium is removable
00C100  1               ;  6 = device is interruptable
00C100  1               ;  5-4 = number of volumes (0..3 means 1..4)
00C100  1               ;  3 = device supports Format call
00C100  1               ;  2 = device can be written to
00C100  1               ;  1 = device can be read from (must be 1)
00C100  1               ;  0 = device status can be read (must be 1)
00C100  1                  .byte %00010111             ; $CnFE: support 2 ProDOS drives
00C100  1
00C100  1               ; $CnFF = LSB of block driver
00C100  1                  .byte <P8DriverEntry        ; $CnFF: low-order offset to ProDOS entry point
00C100  1
00C100  1               .endmacro
00C100  1               ;--------------------------------------------------------------------------
00C100  1               ;==========================================================================
00C100  1
00C100  1
00C100  1               ;--------------------------------------------------------------------------
00C100  1               ; For $C1xx through $C7xx, expand the CnXX macro 7 times to generate the
00C100  1               ; seven variants of the slot-ROM code.
00C100  1               ;
00C100  1               ; Parameters for following macro invocations:
00C100  1               ;     CnXX SLOT, SIGNATURE_BYTE_4
00C100  1               ;
```

```
00C100  1               ; A signature byte of $3C is same as Disk ][, and turns SmartPort support off,
00C100  1               ; and allows autoboot on ][+, ][e.
00C100  1               ;
00C100  1               ; A signature byte of $00 signals SmartPort support, allow 4 drives in slot 5,
00C100  1               ; but disables autoboot on ][+, ][e.
00C100  1               ;------------------------------------------------------------------------
00C100  1               .IF USE_65C02
00C100  1  A9 20 A2 00     CnXX 1, $00         ; Slot PROM code for slot 1
00C104  1  A9 03 A9 00
00C108  1  10 0A 18 90
00C200  1  A9 20 A2 00     CnXX 2, $00         ; Slot PROM code for slot 2
00C204  1  A9 03 A9 00
00C208  1  10 0A 18 90
00C300  1  A9 20 A2 00     CnXX 3, $00         ; Slot PROM code for slot 3
00C304  1  A9 03 A9 00
00C308  1  10 0A 18 90
00C400  1  A9 20 A2 00     CnXX 4, $00         ; Slot PROM code for slot 4
00C404  1  A9 03 A9 00
00C408  1  10 0A 18 90
00C500  1  A9 20 A2 00     CnXX 5, $00         ; Slot PROM code for slot 5
00C504  1  A9 03 A9 00
00C508  1  10 0A 18 90
00C600  1  A9 20 A2 00     CnXX 6, $00         ; Slot PROM code for slot 6
00C604  1  A9 03 A9 00
00C608  1  10 0A 18 90
00C700  1               .listbytes unlimited   ; Show all of the code bytes for the 7th slot
00C700  1  A9 20 A2 00     CnXX 7, $00         ; Slot PROM code for slot 7
00C704  1  A9 03 A9 00
00C708  1  10 0A 18 90
00C70C  1  01 38 20 A1
00C710  1  C7 4C 28 C8
00C714  1  20 A1 C7 86
00C718  1  43 AC 0B C8
00C71C  1  A9 C5 20 A8
00C720  1  FC 88 D0 F8
00C724  1  AD 00 C0 2D
00C728  1  09 C8 CD 0A
00C72C  1  C8 D0 07 EA
00C730  1  20 A1 C7 20
00C734  1  2B C8 A0 01
00C738  1  84 42 88 84
00C73C  1  46 84 47 A9
00C740  1  08 85 45 84
00C744  1  44 20 A1 C7
00C748  1  20 37 C8 85
00C74C  1  FF B0 10 AC
00C750  1  00 08 88 D0
00C754  1  0A AD 01 08
00C758  1  F0 05 A6 43
00C75C  1  4C 01 08 A5
00C760  1  00 D0 0A A5
00C764  1  01 CD F8 07
00C768  1  D0 03 4C BA
00C76C  1  FA 20 2E C8
00C770  1  A0 00 A6 FF
00C774  1  D0 02 A0 0E
00C778  1  20 95 C7 8A
00C77C  1  F0 0C AD 7F
00C780  1  07 0A 0A A9
00C784  1  B0 69 00 20
00C788  1  ED FD A0 1C
00C78C  1  20 95 C7 20
00C790  1  A1 C7 4C 34
00C794  1  C8 B9 BC C7
00C798  1  F0 06 20 31
00C79C  1  C8 C8 D0 F5
00C7A0  1  60 8D F2 C0
00C7A4  1  A9 C7 8D F8
00C7A8  1  07 2C FF CF
00C7AC  1  AD 20 C8 49
00C7B0  1  CF D0 05 A0
00C7B4  1  07 A2 70 60
00C7B8  1  20 93 FE 00
00C7BC  1  43 68 65 63
00C7C0  1  6B 20 64 65
00C7C4  1  76 69 63 65
00C7C8  1  20 00 4E 6F
00C7CC  1  20 62 6F 6F
00C7D0  1  74 20 62 6C
00C7D4  1  6F 63 6B 00
00C7D8  1  2E 0D 0D 43
00C7DC  1  6F 75 6C 64
00C7E0  1  20 6E 6F 74
00C7E4  1  20 62 6F 6F
00C7E8  1  74 20 70 61
00C7EC  1  72 74 69 74
00C7F0  1  69 6F 6E 20
00C7F4  1  00 FF 43 46
00C7F8  1  46 41 FF 00
00C7FC  1  00 00 17 0A
00C800  1               .ELSE
00C800  1                  CnXX 1, $3C         ; Slot PROM code for slot 1. SmartPort disabled.
Autoboot.
00C800  1                  CnXX 2, $3C         ; Slot PROM code for slot 2. SmartPort disabled.
Autoboot.
```

```
00C800  1                  CnXX 3, $3C         ; Slot PROM code for slot 3. SmartPort disabled.
Autoboot.
00C800  1                  CnXX 4, $3C         ; Slot PROM code for slot 4. SmartPort disabled.
Autoboot.
00C800  1                  CnXX 5, $00         ; Slot PROM code for slot 5. SmartPort enabled. No
Autoboot.
00C800  1                  CnXX 6, $3C         ; Slot PROM code for slot 6. SmartPort disabled.
Autoboot.
00C800  1               .listbytes unlimited   ; Show all of the code bytes for the 7th slot
00C800  1                  CnXX 7, $3C         ; Slot PROM code for slot 7. SmartPort disabled.
Autoboot.
00C800  1               .ENDIF
00C800  1
00C800  1               .listbytes      12              ; revert back to normal listing mode for the
rest
00C800  1               ;-------------------- End of Peripheral Card ROM Space ----------------------
00C800  1
00C800  1
00C800  1               ;----------------- Start of I/O expansion ROM (AUX ROM) Space ---------------
00C800  1               ; $C800
00C800  1               ;
00C800  1               ; This code area is absolute and can use absolute addressing.  But it can't
00C800  1               ; easily call into the earlier code, because it might have been called from
00C800  1               ; any slot number.
00C800  1               ;------------------------------------------------------------------------
00C800  1
00C800  1               ;------------------------------------------------------------------------
00C800  1               ; User-configurable parameters ($C800..$C81F)
00C800  1               ;
00C800  1               ; If the CFFA card's CPLD is 2.1 or later, then these 32 bytes can be made
00C800  1               ; writable (even when the write-protect jumper is installed) by touching
00C800  1               ; location $C0x3 (WriteEEPROM,X) and put back to normal by touching $C0x4
00C800  1               ; (NoWriteEEPROM,X).
00C800  1               ;------------------------------------------------------------------------
00C800  1                  ASSERT *=$C800, "User-config section must start at $C800"
00C800  1               Max32MBPartitionsDev0:
00C800  1  04              .byte 4                  ; range 0 to 13
00C801  1               Max32MBPartitionsDev1:
00C801  1  00              .byte 0                  ; range 0 to 13
00C802  1               DefaultBootDevice:
00C802  1  00              .byte 0                  ; range 0 to 1
00C803  1               DefaultBootPartition:
00C803  1  01              .byte 1                  ; range 1 to 13
00C804  1
00C804  1               FourReservedBytes:
00C804  1  00 00 00 00     .byte 0, 0, 0, 0
00C808  1               WriteProtectBits:
00C808  1  00              .byte 0                  ; $80 = protect Dev1, $40 = protect Dev0
00C809  1               MenuSnagMask:
00C809  1                  ASSERT MenuSnagMask=$C809, "Frozen $Cn ROM knows MenuSnagMask is $C809"
00C809  1  DF              .byte $DF                ; allow "m" or "M" to activate the boot-time menu
00C80A  1               MenuSnagKey:
00C80A  1                  ASSERT MenuSnagKey=$C80A, "Frozen $Cn ROM knows MenuSnagKey is $C80A"
00C80A  1  CD              .byte 'M'+$80
00C80B  1               BootTimeDelayTenths:
00C80B  1                  ASSERT BootTimeDelayTenths=$C80B, "Frozen $Cn ROM knows BootTimeDelayTenths
is $C80B"
00C80B  1  05              .byte 5                  ; number of tenths of a second to wait on boot
before kbd check
00C80C  1               BusResetSeconds:
00C80C  1  1F              .byte 31                 ; number of seconds to wait for ready after bus
reset
00C80D  1               CheckDeviceTenths:
00C80D  1  64              .byte 100                ; number of tenths of a second to wait for device to
be ready
00C80E  1               ConfigOptionBits:
00C80E  1  00              .byte 0                  ; set high bit ($80) to skip bus reset
00C80F  1               BlockOffsetDev0:
00C80F  1  00 00 00        .byte 0,0,0              ; range 0 to 2^24 blocks (8 GB)
00C812  1               BlockOffsetDev1:
00C812  1  00 00 00        .byte 0,0,0              ; range 0 to 2^24 blocks (8 GB)
00C815  1
00C815  1               TenBytesAvailable:
00C815  1  00 00 00 00     .byte 0,0,0,0,0,0,0,0,0,0
00C819  1  00 00 00 00
00C81D  1  00 00
00C81F  1
00C81F  1               ConfigAreaVersion:
00C81F  1                  ASSERT ConfigAreaVersion=$C81F, "ConfigAreaVersion must be at $C81F"
00C81F  1  01              .byte 1
00C820  1
00C820  1                  .RES $C820-*,0
00C820  1               ;------------------------------------------------------------------------
00C820  1               ; Version & compatibility bytes
00C820  1               ;------------------------------------------------------------------------
00C820  1               C820_Signature:
00C820  1                  ASSERT C820_Signature=$C820, "Signature must be at $C820"
00C820  1  CF FA           .byte $CF,$FA
00C822  1               C822_VersionByte:
00C822  1                  ASSERT C822_VersionByte=$C822, "Version must be at $C822"
00C822  1  20              .byte FIRMWARE_VER
00C823  1
00C823  1               ; GS/OS driver compatibility byte. The GS/OS driver checks this byte to see
00C823  1               ; if it is compatible with this version of firmware. This way, firmware
00C823  1               ; changes that do not affect the GS/OS driver will not prevent the GS/OS
```

```
00C823  1               ; driver from loading and running.
00C823  1               C823_GSOS_Compatibility:
00C823  1                  ASSERT C823_GSOS_Compatibility=$C823, "Must be at $C823"
00C823  1  11              .byte GSOS_DRIVER
00C824  1
00C824  1               C824_FeatureBits:
00C824  1  40              .byte ($80*(1-USE_65C02))+($40*REQUIRES_EEPROM)+ \
00C825  1                        ($20*WRITE_PROTECT)+($10*SMARTPORT_RAW_BLOCK_MAGIC)
00C825  1
00C825  1               ; 2 bytes available here
00C825  1  00              .byte 0
00C826  1  00              .byte 0
00C827  1
00C827  1               C827_Release_Candidate_Rev:
00C827  1                  ASSERT C827_Release_Candidate_Rev=$C827, "Must be at $C827"
00C827  1  07              .byte 7   ;another minor version number to distinguish release candidates
00C828  1
00C828  1                  .res $C828-*,0
00C828  1               ;----------------------------------------------------------------------------
00C828  1               ; JMP vectors.  These are for use by our $Cn00 ROM.
00C828  1               ;
00C828  1               ; These are new for firmware 2.0, and they should remain in place for every
00C828  1               ; future firmware version.  The intent is for the $Cn00 ROM to not change
00C828  1               ; in future versions, for ease of updating the firmware in the field.
00C828  1               ;
00C828  1               ; So, don't rearrange these lightly!  Add more at the end, if necessary.
00C828  1               ;----------------------------------------------------------------------------
00C828  1                  ASSERT *=$C828, "Fatal error - JMP vectors must start at $C828"
00C828  1               P8_SmartPort_Handler:
00C828  1  4C A4 CA        jmp P8_SmartPort_Handler_Impl
00C82B  1               InteractiveMenu:
00C82B  1  4C EE CD        jmp InteractiveMenu_Impl
00C82E  1               PrepareToShowErrorMessage:
00C82E  1  4C 57 C8        jmp PrepareToShowErrorMessage_Impl
00C831  1               CaseSafeCOUT:
00C831  1  4C 97 CF        jmp CaseSafeCOUT_Impl
00C834  1               FinishBootTimeErrorMsg:
00C834  1  4C 3A C8        jmp FinishBootTimeErrorMsg_Impl
00C837  1               BootROM_ProDOS_Call:
00C837  1  4C A9 CA        jmp BootROM_ProDOS_Call_Impl
00C83A  1               ASSERT *=$C83A, "C83A"
00C83A  1
00C83A  1               ;-------------------------------------------------------------------------------
00C83A  1               ;-------------------------------------------------------------------------------
00C83A  1               ; FinishBootTimeErrorMsg
00C83A  1               ;
00C83A  1               ; Input: Y = SLOT, X = SLOT*16
00C83A  1               ;
00C83A  1               ; Called after the $Cn ROM displays an error message and "Could not boot
00C83A  1               ; partition ".  We should idenify the partition, beep, and jump to Applesoft.
00C83A  1               ;
00C83A  1               ; The error from reading block 0 is in bootError, in case we want to display it.
00C83A  1               ;-------------------------------------------------------------------------------
00C83A  1               FinishBootTimeErrorMsg_Impl:
00C83A  1  B9 F8 07        lda BootPartition,y
00C83D  1  1A              INC_A_OR_ADC1
00C83E  1  20 80 CF        jsr PrintSmallDecimal
00C841  1               ; It would be nice to display " (Dev0)" or " (Dev1)" here, if the code would
fit.
00C841  1  20 8E FD        jsr CROUT
00C844  1  20 DD FB        jsr BELL1
00C847  1
00C847  1  A6 FF           ldx bootError
00C849  1  F0 09           beq @noErrorNumber
00C84B  1  A0 12           ldy #MsgErrorNumber-Messages
00C84D  1  20 74 CF        jsr Message
00C850  1  8A              txa
00C851  1  20 DA FD        jsr PRBYTE
00C854  1
00C854  1               @noErrorNumber:
00C854  1  4C 00 E0        jmp AppleSoft
00C857  1
00C857  1               ;-------------------------------------------------------------------------------
00C857  1               ; PrepareToShowErrorMessage_Impl
00C857  1               ;
00C857  1               ; This is separate from ClearScreenAndIdentifyCFFA in case we want the boot-
00C857  1               ; time screen to act different (without changing the $Cn ROM).
00C857  1               ;-------------------------------------------------------------------------------
00C857  1               PrepareToShowErrorMessage_Impl:
00C857  1               ;-------------------------------------------------------------------------------
00C857  1               ; ClearScreenAndIdentifyCFFA
00C857  1               ;-------------------------------------------------------------------------------
00C857  1               ClearScreenAndIdentifyCFFA:
00C857  1  20 2F FB        jsr INIT             ; text mode, full screen, page 1
00C85A  1  20 84 FE        jsr SETNORM
00C85D  1  20 58 FC        jsr HOME
00C860  1  20 93 FE        jsr SetVID
00C863  1  20 89 FE        jsr SetKBD
00C866  1
00C866  1  20 7F C8        jsr LineOfDashes
00C869  1  A0 00           ldy #MsgVersion-Messages
00C86B  1  20 74 CF        jsr Message
00C86E  1
00C86E  1  A9 22           lda #34              ; room for "Slot N" at the right edge
00C870  1  85 24           sta CH
```

```
00C872  1  A0 09           ldy #MsgSlot-Messages
00C874  1  20 74 CF        jsr Message
00C877  1  AD F8 07        lda MSLOT
00C87A  1  49 70           eor #$70             ; change $Cx to $Bx (slot digit)
00C87C  1  20 ED FD        jsr COUT
00C87F  1
00C87F  1               LineOfDashes:
00C87F  1  A0 28           ldy #40
00C881  1               @loop:
00C881  1  A9 AD           lda #'-'+$80
00C883  1  20 ED FD        jsr COUT
00C886  1  88              dey
00C887  1  D0 F8           bne @loop
00C889  1  60              rts
00C88A  1
00C88A  1               ;-----------------------------------------------------------------------------
00C88A  1               ; SmartPort_Handler
00C88A  1               ;
00C88A  1               ; Called through $Cn0D entry point.
00C88A  1               ;
00C88A  1               ; Inputs:
00C88A  1               ;    Y = SLOT
00C88A  1               ;    X = SLOT*16
00C88A  1               ;
00C88A  1               ; 1) Push a block of zero-page locations onto the stack, creating a work space.
00C88A  1               ; 2) Get the request parameters that follow the call to this driver
00C88A  1               ; 3) Using request parameters as pointers get request specific information
00C88A  1               ;    and set up the P8 driver request registers $42-$47.
00C88A  1               ; 4) Call P8Driver code
00C88A  1               ; 5) On return from P8Driver code, restore zero page work space, and return to
00C88A  1               ;    caller.
00C88A  1               ;
00C88A  1               ;-----------------------------------------------------------------------------
00C88A  1               SmartPort_Handler:
00C88A  1               ; save the zero page locations we're going to use
00C88A  1  A0 0C           ldy #spZeroPgSize-1
00C88C  1               @save:
00C88C  1  B9 40 00        lda spZeroPgArea,y
00C88F  1  48              pha
00C890  1  88              dey
00C891  1  10 F9           bpl @save
00C893  1
00C893  1  86 41           stx spSlotX16
00C895  1
00C895  1               ; fetch the parameter list pointer from the code stream & adjust the returna
address
00C895  1  BA              tsx
00C896  1  BD 0E 01        lda $101+spZeroPgSize,x
00C899  1  85 48           sta spParamList
00C89B  1  18              clc
00C89C  1  69 03           adc #3
00C89E  1  9D 0E 01        sta $101+spZeroPgSize,x
00C8A1  1  BD 0F 01        lda $102+spZeroPgSize,x
00C8A4  1  85 49           sta spParamList+1
00C8A6  1  69 00           adc #0
00C8A8  1  9D 0F 01        sta $102+spZeroPgSize,x
00C8AB  1
00C8AB  1               ; set up Y and spSlot, and reload X from spSlotX16
00C8AB  1  AD F8 07        lda MSLOT
00C8AE  1  29 0F           and #$0f
00C8B0  1  85 40           sta spSlot
00C8B2  1  A8              tay
00C8B3  1  A6 41           ldx spSlotX16
00C8B5  1
00C8B5  1               ; reset the device if this is our first call
00C8B5  1  20 DE CB        jsr ResetBusIfFirstTime_ClearMiscFlags   ; needs X and Y
00C8B8  1
00C8B8  1               ; get command code from parameter list
00C8B8  1  A0 01           ldy #1
00C8BA  1  B1 48           lda (spParamList),y
00C8BC  1  85 42           sta spCommandCode
00C8BE  1  C8              iny
00C8BF  1  B1 48           lda (spParamList),y
00C8C1  1  AA              tax
00C8C2  1  C8              iny
00C8C3  1  B1 48           lda (spParamList),y
00C8C5  1  85 49           sta spParamList+1
00C8C7  1  86 48           stx spParamList
00C8C9  1
00C8C9  1  A9 01           lda #PRODOS_BADCMD          ; anticipate bad command error
00C8CB  1  A6 42           ldx spCommandCode
00C8CD  1  E0 0A           cpx #$09+1                  ; command too large
00C8CF  1  B0 1A           bcs @out
00C8D1  1
00C8D1  1               .IF USE_65C02
00C8D1  1  B2 48           lda (spParamList)           ; parameter count
00C8D3  1               .ELSE
00C8D3  1                  ldy #0
00C8D3  1                  lda (spParamList),y         ; parameter count
00C8D3  1               .ENDIF
00C8D3  1  DD 06 C9        cmp RequiredParamCounts,x   ; command number still in X
00C8D6  1  D0 27           bne @pCountMismatch
00C8D8  1
00C8D8  1  A0 01           ldy #1
00C8DA  1  B1 48           lda (spParamList),y
```

```
00C8DC  1  A4 40           ldy spSlot
00C8DE  1  99 F8 04        sta DriveNumber,y           ; don't remap yet -- Status call needs unit 0
00C8E1  1
00C8E1  1  8A              txa                         ; X is still the command code
00C8E2  1  0A              asl a
00C8E3  1  AA              tax
00C8E4  1  20 03 C9        jsr JumpToSPCommand         ; Y is still spSlot
00C8E7  1
00C8E7  1  B0 02           bcs @out
00C8E9  1  A9 00           lda #0
00C8EB  1
00C8EB  1               @out:
00C8EB  1  AA              tax                         ; error code in X
00C8EC  1
00C8EC  1               ; Restore zero page
00C8EC  1  A0 00           ldy #0
00C8EE  1               @restore:
00C8EE  1  68              pla
00C8EF  1  99 40 00        sta spZeroPgArea,y
00C8F2  1  C8              iny
00C8F3  1  C0 0D           cpy #spZeroPgSize
00C8F5  1  90 F7           bcc @restore
00C8F7  1
00C8F7  1  8A              txa
00C8F8  1  A0 02           ldy #2                     ; high byte of # bytes transferred
00C8FA  1                                             ;   (always (1) undefined, or
00C8FA  1                                             ;   (2) #bytes transferred to host)
00C8FA  1  A2 00           ldx #0                     ; low byte of # bytes transferred
00C8FC  1  C9 01           cmp #1                     ; C=1 if error code nonzero
00C8FE  1  60              rts
00C8FF  1
00C8FF  1               @pCountMismatch:
00C8FF  1  A9 04           lda #BAD_PARM_COUNT
00C901  1  D0 E8           bne @out
00C903  1
00C903  1               JumpToSPCommand:
00C903  1               .IF USE_65C02
00C903  1                  RTS_ADJUST = 0
00C903  1  7C 10 C9           jmp (spDispatch,x)
00C906  1               .ELSE
00C906  1                  RTS_ADJUST = 1
00C906  1                     lda spDispatch+1,x
00C906  1                     pha
00C906  1                     lda spDispatch,x
00C906  1                     pha
00C906  1                     rts
00C906  1               .ENDIF
00C906  1
00C906  1               RequiredParamCounts:
00C906  1  03              .byte 3   ; 0 = status
00C907  1  03              .byte 3   ; 1 = read
00C908  1  03              .byte 3   ; 2 = write
00C909  1  01              .byte 1   ; 3 = format
00C90A  1  03              .byte 3   ; 4 = control
00C90B  1  01              .byte 1   ; 5 = init
00C90C  1  01              .byte 1   ; 6 = open
00C90D  1  01              .byte 1   ; 7 = close
00C90E  1  04              .byte 4   ; 8 = read
00C90F  1  04              .byte 4   ; 9 = write
00C910  1
00C910  1               spDispatch:
00C910  1  24 C9           .word spStatus-RTS_ADJUST
00C912  1  1A CB           .word spReadBlock-RTS_ADJUST
00C914  1  12 CB           .word spWriteBlock-RTS_ADJUST
00C916  1  95 CA           .word spFormat-RTS_ADJUST
00C918  1  1D CA           .word spControl-RTS_ADJUST
00C91A  1  3F CA           .word spInit-RTS_ADJUST
00C91C  1  49 CA           .word spOpen-RTS_ADJUST
00C91E  1  49 CA           .word spClose-RTS_ADJUST
00C920  1  DE C9           .word spReadChars-RTS_ADJUST
00C922  1  DE C9           .word spWriteChars-RTS_ADJUST
00C924  1
00C924  1               ;------------------------------------------------------------------------------
00C924  1               ; SmartPort STATUS call
00C924  1               ;
00C924  1               ; We support the standard calls, plus an "Identify" command
00C924  1               ; for unit 0, which fills a buffer with the card's Identify
00C924  1               ; data.
00C924  1               ;------------------------------------------------------------------------------
00C924  1               spStatus:
00C924  1  20 4D CA        jsr SPSetUpControlOrStatus
00C927  1  B0 35           bcs statOut
00C929  1
00C929  1  A4 40           ldy spSlot
00C92B  1  B9 F8 04        lda DriveNumber,y
00C92E  1  D0 36           bne StatusForOneUnit
00C930  1
00C930  1               ; StatusCode = 0 && unit == 0: status for this entire SmartPort interface
00C930  1  A5 4C           lda spCSCode
00C932  1  F0 14           beq Status00
00C934  1
00C934  1               ; Check for device-specific status subcodes, only defined for the CFFA.
00C934  1  C9 49           cmp #SP_STATUS_IDENT_DEV0
00C936  1  F0 09           beq @ident
00C938  1  C9 48           cmp #SP_STATUS_IDENT_DEV1
```

```
00C93A  1  D0 08           bne BadStatusCode
00C93C  1  A9 40           lda #kMiscDev1
00C93E  1               ;  ora DrvMiscFlags,y     ; use ORA if we define new flag bits
00C93E  1  99 78 07        sta DrvMiscFlags,y
00C941  1               @ident:
00C941  1  4C C9 C9        jmp spStatusIdentify
00C944  1
00C944  1               ; Any other status code for unit 0 is an error.
00C944  1               BadStatusCode:
00C944  1  A9 21           lda #BAD_STATUS_CODE
00C946  1  38              sec
00C947  1  60              rts
00C948  1
00C948  1               Status00:
00C948  1  A4 40           ldy spSlot
00C94A  1  18              clc
00C94B  1  B9 78 06        lda PartitionsDev0,y
00C94E  1  79 F8 06        adc PartitionsDev1,y
00C951  1                  .IF USE_65C02
00C951  1  92 4A              sta (spCmdList)             ; byte +0 = number of drives
00C953  1                  .ELSE
00C953  1                     ldy #0
00C953  1                     sta (spCmdList),y           ; byte +0 = number of drives
00C953  1                  .ENDIF
00C953  1
00C953  1  A0 07           ldy #7
00C955  1               Stat00Loop:
00C955  1  B9 5E C9        lda Stat00Data-1,y
00C958  1  91 4A           sta (spCmdList),y
00C95A  1  88              dey
00C95B  1  D0 F8           bne Stat00Loop
00C95D  1  18              clc
00C95E  1               statOut:
00C95E  1  60              rts
00C95F  1
00C95F  1               Stat00Data:
00C95F  1               ; Interrupt flag = no interrupts caused by this interface
00C95F  1  40              .byte $40
00C960  1               ; Vendor ID assigned to Rich Dreher by www.syndicomm.com 3/16/2002
00C960  1  00 CC           .word $CC00
00C962  1  00 20           .word SPDRIVERVERSION        ; Our version number
00C964  1  00              .byte $00                    ; Reserved byte
00C965  1  00              .byte $00                    ; Reserved byte
00C966  1                  ASSERT (*-Stat00Data)=7, "There must be exactly 7 bytes of Stat00Data"
00C966  1
00C966  1               StatusForOneUnit:
00C966  1  3A              DEC_A_OR_SBC1
00C967  1  20 83 CD        jsr RemapAndStoreDriveNum
00C96A  1  B0 0E           bcs @error
00C96C  1
00C96C  1  A6 4C           ldx spCSCode
00C96E  1  F0 16           beq Status0or3
00C970  1  CA              dex
00C971  1  F0 08           beq StatusGetDCB
00C973  1  CA              dex
00C974  1  CA              dex
00C975  1  F0 0F           beq Status0or3
00C977  1  A9 21           lda #BAD_STATUS_CODE
00C979  1  38              sec
00C97A  1               @error:
00C97A  1  60              rts
00C97B  1               ;
00C97B  1               ; We have no interesting data to return for the device control
00C97B  1               ; block, so return the shortest one allowed: a length byte of
00C97B  1               ; 1 followed by a data byte of 0.
00C97B  1               ;
00C97B  1               StatusGetDCB:
00C97B  1  A9 01           lda #1
00C97D  1               .IF USE_65C02
00C97D  1  92 4A           sta (spCmdList)             ; Returned length = 1
00C97F  1               .ELSE
00C97F  1                  ldy #0
00C97F  1                  sta (spCmdList),y           ; Returned length = 1
00C97F  1               .ENDIF
00C97F  1  A8              tay
00C980  1  A9 00           lda #0
00C982  1  91 4A           sta (spCmdList),y           ; Returned data = $00
00C984  1  18              clc
00C985  1  60              rts
00C986  1
00C986  1               ;
00C986  1               ; Status code 0 and 3 start off the same; 3 returns extra data.
00C986  1               ;
00C986  1               ; 0: return device status (1 byte device status + 3-byte block count)
00C986  1               ; 3: return Device Information Block
00C986  1               ;
00C986  1               Status0or3:
00C986  1  A9 F8           lda #$F8                    ; Block device, write, read, format, online,
00C988  1                                              ; not write-prot
00C988  1               .IF USE_65C02
00C988  1  92 4A           sta (spCmdList)
00C98A  1               .ELSE
00C98A  1                  ldy #0
00C98A  1                  sta (spCmdList),y
00C98A  1               .ENDIF
```

```
00C98A  1  A6 41           ldx spSlotX16
00C98C  1  A4 40           ldy spSlot
00C98E  1  20 CF CA        jsr GetStatus               ; Returns block count in YX
00C991  1  B0 CB           bcs statOut
00C993  1
00C993  1  98              tya
00C994  1  A0 02           ldy #2
00C996  1  91 4A           sta (spCmdList),y           ; CmdList +2 = bits 8..15 of block count
00C998  1  88              dey
00C999  1  8A              txa                         ; CmdList +1 = bits 0..7 of block count
00C99A  1  91 4A           sta (spCmdList),y
00C99C  1  A0 03           ldy #3
00C99E  1  A9 00           lda #0
00C9A0  1  91 4A           sta (spCmdList),y           ; CmdList +3 = bits 16..23 of block count
00C9A2  1
00C9A2  1  A5 4C           lda spCSCode
00C9A4  1  F0 0C           beq statDone
00C9A6  1
00C9A6  1               ; status code 3: return 21 more bytes of data
00C9A6  1  A0 04           ldy #4
00C9A8  1               stat3Loop:
00C9A8  1  B9 B0 C9        lda stat3Data-4,y
00C9AB  1  91 4A           sta (spCmdList),y
00C9AD  1  C8              iny
00C9AE  1  C0 19           cpy #21+4
00C9B0  1  90 F6           bcc stat3Loop
00C9B2  1
00C9B2  1               .if SMARTPORT_STATUS_D0_D1
00C9B2  1               ; Doctor the "D0" to a "D1" in the status data, if we're on Device 1.
00C9B2  1                  ldy spSlot
00C9B2  1                  lda DrvMiscFlags,y
00C9B2  1                  asl a
00C9B2  1                  bpl statDone
00C9B2  1                  ldy #4+16
00C9B2  1                  lda #'1'
00C9B2  1                  sta (spCmdList),y
00C9B2  1               .endif
00C9B2  1
00C9B2  1               statDone:
00C9B2  1  18              clc
00C9B3  1  60              rts
00C9B4  1
00C9B4  1               stat3Data:
00C9B4  1               .if SMARTPORT_STATUS_D0_D1
00C9B4  1                  .byte 16,"COMPACT FLASH D0"  ; length byte + 16-byte ASCII, padded
00C9B4  1               .else
00C9B4  1  10 43 4F 4D     .byte 16,"COMPACT FLASH   "  ; length byte + 16-byte ASCII, padded
00C9B8  1  50 41 43 54
00C9BC  1  20 46 4C 41
00C9C5  1               .endif
00C9C5  1  02              .byte $02                    ; device type = hard disk
00C9C6  1  20              .byte $20                    ; subtype (no removable media, no extended,
00C9C7  1                                               ; no disk switched)
00C9C7  1  00 20           .word SPDRIVERVERSION
00C9C9  1                  ASSERT (*-stat3Data)=21, "There must be exactly 21 bytes of stat3Data"
00C9C9  1
00C9C9  1               ;------------------------------------------------------------------------------
00C9C9  1               ; Identify (Status subcode)
00C9C9  1               ;
00C9C9  1               ; The status list is a 512-byte buffer that we will fill with
00C9C9  1               ; the drive's IDE "Identify" data.  We post-process the data
00C9C9  1               ; by byte-swapping the model name and serial number strings
00C9C9  1               ; so they make sense.
00C9C9  1               ;
00C9C9  1               ; Input:  DrvMiscFlags,y has been set up
00C9C9  1               ;         spSlot
00C9C9  1               ;         spSlot16
00C9C9  1               ;         spCmdList
00C9C9  1               ;------------------------------------------------------------------------------
00C9C9  1               spStatusIdentify:
00C9C9  1  A4 40           ldy spSlot
00C9CB  1  A6 41           ldx spSlotX16
00C9CD  1
00C9CD  1               ; Make sure that Partition 0 exists on the specified device.
00C9CD  1  20 69 CD        jsr IsDeviceKnownToExist
00C9D0  1  D0 04           bne @ok
00C9D2  1  A9 28           lda #PRODOS_NO_DEVICE
00C9D4  1  38              sec
00C9D5  1  60              rts
00C9D6  1               @ok:
00C9D6  1
00C9D6  1  20 00 CD        jsr WaitReady_SetATAHeadFromMiscDrvFlags
00C9D9  1
00C9D9  1  20 E8 CC        jsr WaitReady_IssueIdentify_CheckErr
00C9DC  1  D0 04           bne iCommandOK
00C9DE  1
00C9DE  1               ReturnIOError:
00C9DE  1  A9 27           lda #PRODOS_IO_ERROR
00C9E0  1  38              sec
00C9E1  1  60              rts
00C9E2  1               ;
00C9E2  1               ; The "Identify" data is ready to read
00C9E2  1               ;
00C9E2  1               pageCount = spCommandCode       ; re-use a zero-page location
00C9E2  1
```

```
00C9E2  1               iCommandOK:
00C9E2  1  A0 01           ldy #1
00C9E4  1  84 42           sty pageCount
00C9E6  1  88              dey                          ; Y = 0
00C9E7  1               @iLoop:
00C9E7  1  BD 88 C0        lda ATADataLow,x
00C9EA  1  91 4A           sta (spCmdList),y
00C9EC  1  C8              iny
00C9ED  1  BD 80 C0        lda ATADataHigh,x
00C9F0  1  91 4A           sta (spCmdList),y
00C9F2  1  C8              iny
00C9F3  1  D0 F2           bne @iLoop
00C9F5  1  E6 4B           inc spCmdList+1
00C9F7  1  C6 42           dec pageCount
00C9F9  1  10 EC           bpl @iLoop
00C9FB  1
00C9FB  1               ; Swap ASCII text data of the Identify data to be readable.
00C9FB  1               ; These are both on the first page of the data.
00C9FB  1
00C9FB  1  C6 4B           dec spCmdList+1
00C9FD  1  C6 4B           dec spCmdList+1             ; Point to beginning of buffer
00C9FF  1
00C9FF  1  A0 2E           ldy #23*2                   ; Start at word 23 (firmware rev, model number)
00CA01  1  A2 18           ldx #24                     ; 24 words
00CA03  1  20 0A CA        jsr SwapBytes
00CA06  1
00CA06  1  A0 14           ldy #10*2                   ; Start at word 10 (serial number)
00CA08  1  A2 0A           ldx #10                     ; 10 words
00CA0A  1               ; Fall into SwapBytes and return with carry clear.
00CA0A  1               ; Swap X words of data starting at offset Y.
00CA0A  1               SwapBytes:
00CA0A  1  B1 4A           lda (spCmdList),y
00CA0C  1  48              pha                         ; Save the 1st byte
00CA0D  1  C8              iny
00CA0E  1  B1 4A           lda (spCmdList),y           ; Get the 2nd byte
00CA10  1  88              dey
00CA11  1  91 4A           sta (spCmdList),y           ; Store it in position 1
00CA13  1  C8              iny
00CA14  1  68              pla                         ; Finally, retrieve the 1st byte
00CA15  1  91 4A           sta (spCmdList),y           ; Put it in position 2
00CA17  1  C8              iny
00CA18  1  CA              dex
00CA19  1  D0 EF           bne SwapBytes
00CA1B  1  18              clc
00CA1C  1  60              rts
00CA1D  1
00CA1D  1               ;------------------------------------------------------------------------------
00CA1D  1               ; SmartPort CONTROL command
00CA1D  1               ;
00CA1D  1               ;------------------------------------------------------------------------------
00CA1D  1               spControl:
00CA1D  1  20 4D CA        jsr SPSetUpControlOrStatus
00CA20  1  B0 18           bcs ctrlDone
00CA22  1
00CA22  1  20 95 CA        jsr SPValidateAndRemapUnitNumber
00CA25  1  B0 13           bcs ctrlDone
00CA27  1               ; carry is clear
00CA27  1  A6 4C           ldx spCSCode
00CA29  1  F0 0F           beq ctlReset                 ; Control code 0 = Reset
00CA2B  1  CA              dex
00CA2C  1  F0 0C           beq ctlSetDCB                ; Control code 1 = SetDCB
00CA2E  1  CA              dex
00CA2F  1  F0 06           beq ctlSetNewline            ; Control code 2 = SetNewline
00CA31  1  CA              dex
00CA32  1  F0 07           beq ctlServiceInterrupt      ; Control code 3 = ServiceInterrupt
00CA34  1  CA              dex
00CA35  1  F0 03           beq ctlEject                 ; Control code 4 = Eject
00CA37  1
00CA37  1               ctlSetNewline:
00CA37  1  A9 21           lda #BAD_STATUS_CODE         ; Bad control code
00CA39  1  38              sec
00CA3A  1               ctlReset:
00CA3A  1               ctlSetDCB:
00CA3A  1               ctlEject:
00CA3A  1               ctrlDone:
00CA3A  1  60              rts
00CA3B  1
00CA3B  1               ctlServiceInterrupt:
00CA3B  1  A9 1F           lda #INTERRUPT_NOT_SUPT
00CA3D  1  38              sec
00CA3E  1  60              rts
00CA3F  1
00CA3F  1               ;------------------------------------------------------------------------------
00CA3F  1               ; SmartPort INIT command
00CA3F  1               ;
00CA3F  1               ; unit 0 = entire chain
00CA3F  1               ;
00CA3F  1               ;  SmartPort Technote #2 says you can't init an individual unit,
00CA3F  1               ;  so return error if DriveNumber is nonzero.
00CA3F  1               ;------------------------------------------------------------------------------
00CA3F  1               spInit:
00CA3F  1  B9 F8 04        lda DriveNumber,y
00CA42  1  18              clc
00CA43  1  F0 03           beq @initChain
00CA45  1  A9 11           lda #BAD_UNIT_NUMBER
```

```
00CA47  1  38              sec
00CA48  1               @initChain:
00CA48  1  60              rts
00CA49  1
00CA49  1               ;------------------------------------------------------------------------------
00CA49  1               ; SmartPort: Open and Close are for character devices only
00CA49  1               ;------------------------------------------------------------------------------
00CA49  1               spOpen:
00CA49  1               spClose:
00CA49  1  A9 01           lda #PRODOS_BADCMD
00CA4B  1  38              sec
00CA4C  1  60              rts
00CA4D  1
00CA4D  1               ;------------------------------------------------------------------------------
00CA4D  1               ; SmartPort: Read, Write
00CA4D  1               ;
00CA4D  1               ; We don't bother implementing Read and Write, although they are allowed
00CA4D  1               ; for block devices.  We would only support 512-byte transfers anyway.
00CA4D  1               ;------------------------------------------------------------------------------
00CA4D  1               spReadChars = ReturnIOError
00CA4D  1               spWriteChars = ReturnIOError
00CA4D  1               ;   lda #PRODOS_IO_ERROR
00CA4D  1               ;   sec
00CA4D  1               ;   rts
00CA4D  1
00CA4D  1
00CA4D  1               ;------------------------------------------------------------------------------
00CA4D  1               ; SPSetUpControlOrStatus
00CA4D  1               ;
00CA4D  1               ; fetch from parameter block:
00CA4D  1               ;    status/control list pointer (word)
00CA4D  1               ;    status/control code (byte)
00CA4D  1               ;------------------------------------------------------------------------------
00CA4D  1               SPSetUpControlOrStatus:
00CA4D  1  A0 02           ldy #2
00CA4F  1  B1 48           lda (spParamList),y
00CA51  1  85 4A           sta spCmdList
00CA53  1  C8              iny
00CA54  1  B1 48           lda (spParamList),y
00CA56  1  85 4B           sta spCmdList+1
00CA58  1  C8              iny
00CA59  1  B1 48           lda (spParamList),y
00CA5B  1  85 4C           sta spCSCode
00CA5D  1  18              clc
00CA5E  1  60              rts
00CA5F  1
00CA5F  1               ;------------------------------------------------------------------------------
00CA5F  1               ; SPSetUpReadWrite
00CA5F  1               ;
00CA5F  1               ; Input:
00CA5F  1               ;    DriveNumber,y is already a copy of SmartPort unit number
00CA5F  1               ;
00CA5F  1               ; fetch from SP parameter list:
00CA5F  1               ;   buffer pointer
00CA5F  1               ;   block number
00CA5F  1               ;
00CA5F  1               ; Validate unit number:
00CA5F  1               ;   127 = magic, allow any block number
00CA5F  1               ;   1..N:  Validate block number: $0..FFFE
00CA5F  1               ;
00CA5F  1               ; Output:
00CA5F  1               ;     DriveNumber,y: translated unit number in case unit was magic
00CA5F  1               ;                    or BootPartition is set
00CA5F  1               ;     DrvMiscFlags: bit 7 set if we should not use BlockOffset
00CA5F  1               ;     SEC = error
00CA5F  1               ;     CLC:
00CA5F  1               ;        Y = slot
00CA5F  1               ;        X = Slot*16
00CA5F  1               ;------------------------------------------------------------------------------
00CA5F  1               SPSetUpReadWrite:
00CA5F  1               ; copy pdIOBuffer from parameter list
00CA5F  1  A0 02           ldy #2
00CA61  1  B1 48           lda (spParamList),y
00CA63  1  85 44           sta pdIOBuffer
00CA65  1  C8              iny
00CA66  1  B1 48           lda (spParamList),y
00CA68  1  85 45           sta pdIOBuffer+1
00CA6A  1
00CA6A  1               ; copy pdBlockNumber from parameter list
00CA6A  1  C8              iny
00CA6B  1  B1 48           lda (spParamList),y
00CA6D  1  85 46           sta pdBlockNumber
00CA6F  1  C8              iny
00CA70  1  B1 48           lda (spParamList),y
00CA72  1  85 47           sta pdBlockNumber+1
00CA74  1
00CA74  1               ; Validate the unit number and block number.
00CA74  1  A4 40           ldy spSlot
00CA76  1  B9 F8 04        lda DriveNumber,y
00CA79  1               .if SMARTPORT_RAW_BLOCK_MAGIC
00CA79  1                  cmp #MagicRawBlocksDev0Unit
00CA79  1                  beq magicUnitDev0
00CA79  1                  cmp #MagicRawBlocksDev1Unit
00CA79  1                  beq magicUnitDev1
00CA79  1               .endif
```

```
00CA79  1
00CA79  1  20 95 CA        jsr SPValidateAndRemapUnitNumber
00CA7C  1  B0 16           bcs srwOut
00CA7E  1
00CA7E  1  A0 06           ldy #6
00CA80  1  B1 48           lda (spParamList),y          ; Bits 16..23 of block number must be $00
00CA82  1  D0 0D           bne badBlockNum
00CA84  1
00CA84  1  A5 47           lda pdBlockNumber+1          ; Block $FFFF is invalid
00CA86  1  25 46           and pdBlockNumber
00CA88  1               .IF USE_65C02
00CA88  1  1A              inc a
00CA89  1               .ELSE
00CA89  1                  cmp #$FF
00CA89  1               .ENDIF
00CA89  1  F0 06           beq badBlockNum
00CA8B  1
00CA8B  1               loadXYandReturnNoError:
00CA8B  1  A4 40           ldy spSlot
00CA8D  1  A6 41           ldx spSlotX16
00CA8F  1  18              clc
00CA90  1  60              rts
00CA91  1
00CA91  1               badBlockNum:
00CA91  1  A9 2D           lda #PRODOS_BADBLOCK         ; Bad block number
00CA93  1  38              sec
00CA94  1               srwOut:
00CA94  1  60              rts
00CA95  1               ;
00CA95  1               ; For the "magic raw blocks" units, allow a full 3-byte block number.
00CA95  1               ;
00CA95  1               .if SMARTPORT_RAW_BLOCK_MAGIC
00CA95  1               magicUnitDev1:
00CA95  1                  lda #kMiscRaw+kMiscDev1
00CA95  1                  bne magicUnitCommon
00CA95  1               magicUnitDev0:
00CA95  1                  lda #kMiscRaw
00CA95  1               magicUnitCommon:
00CA95  1               ;  ora DrvMiscFlags,y           ; use ORA if any other bits are defined
00CA95  1                  sta DrvMiscFlags,y           ; Raw block access
00CA95  1
00CA95  1                  ldy #6
00CA95  1                  lda (spParamList),y          ; Bits 16..23 of block number
00CA95  1                  sta DriveNumber,y
00CA95  1                  BRA_OR_JMP loadXYandReturnNoError
00CA95  1               .endif
00CA95  1
00CA95  1               ;-----------------------------------------------------------------------------
00CA95  1               ; SmartPort FORMAT command
00CA95  1               ;
00CA95  1               ; We don't actually do anything beyond validating the unit number.
00CA95  1               ;-----------------------------------------------------------------------------
00CA95  1               spFormat:
00CA95  1               ;   jmp SPValidateAndRemapUnitNumber
00CA95  1               ;
00CA95  1               ; v v v   FALL INTO   v v v
00CA95  1               ;-----------------------------------------------------------------------------
00CA95  1               ; SPValidateAndRemapUnitNumber
00CA95  1               ;
00CA95  1               ; Validate that DriveNumber is from 1 to N.
00CA95  1               ;
00CA95  1               ; Input: DriveNumber,Y
00CA95  1               ;
00CA95  1               ; Output: CLC = no error
00CA95  1               ;            DriveNumber,Y in range 0..N-1, remapped as needed
00CA95  1               ;            DrvMiscFlags,Y dev1 bit
00CA95  1               ;         SEC, A = error
00CA95  1               ;-----------------------------------------------------------------------------
00CA95  1               SPValidateAndRemapUnitNumber:
00CA95  1  A4 40           ldy spSlot
00CA97  1  B9 F8 04        lda DriveNumber,y
00CA9A  1  38              sec
00CA9B  1  F0 04           beq @badUnit
00CA9D  1  3A              DEC_A_OR_SBC1
00CA9E  1
00CA9E  1  20 83 CD        jsr RemapAndStoreDriveNum
00CAA1  1               @badUnit:
00CAA1  1  A9 11           lda #BAD_UNIT_NUMBER         ; only an error if SEC
00CAA3  1  60              rts
00CAA4  1
00CAA4  1               ;-----------------------------------------------------------------------------
00CAA4  1               ; P8_SmartPort_Handler_Impl
00CAA4  1               ;
00CAA4  1               ; Input: CLC if we should handle a ProDOS call
00CAA4  1               ;        SEC if we should handle a SmartPort call
00CAA4  1               ;        Y = SLOT
00CAA4  1               ;        X = SLOT * 16
00CAA4  1               ;
00CAA4  1               ; Other inputs as set up by the caller (ZP for ProDOS, inline parms for
00CAA4  1               ; SmartPort).
00CAA4  1               ;-----------------------------------------------------------------------------
00CAA4  1               P8_SmartPort_Handler_Impl:
00CAA4  1  90 03           bcc ProDOS_Handler
00CAA6  1  4C 8A C8        jmp SmartPort_Handler
00CAA9  1
```

```
00CAA9  1               ;-----------------------------------------------------------------------------
00CAA9  1               ; BootROM_ProDOS_Call_Impl
00CAA9  1               ;
00CAA9  1               ; Same as BootROM_ProDOS_Call_Impl, but we have a chance to do extra stuff
00CAA9  1               ; in the boot path, if needed, without changing the $Cn00 ROM.
00CAA9  1               ;-----------------------------------------------------------------------------
00CAA9  1               BootROM_ProDOS_Call_Impl:
00CAA9  1               ; v v v   FALL INTO   v v v
00CAA9  1               ;-----------------------------------------------------------------------------
00CAA9  1               ; ProDOS_Handler
00CAA9  1               ;
00CAA9  1               ; Input:
00CAA9  1               ;     Y = SLOT
00CAA9  1               ;     X = SLOT*16
00CAA9  1               ;     zero page pdUnitNumber, etc.
00CAA9  1               ;-----------------------------------------------------------------------------
00CAA9  1               ProDOS_Handler:
00CAA9  1  20 DE CB        jsr ResetBusIfFirstTime_ClearMiscFlags  ; needs X and Y
00CAAC  1               ;
00CAAC  1               ; If the ProDOS unit number doesn't match our slot number, add 2 to
00CAAC  1               ; the drive number.
00CAAC  1               ;
00CAAC  1               ; This actually happens: If we're in slot 5, we get calls for slot 2
00CAAC  1               ; that want to access our 3rd and 4th partitions.
00CAAC  1               ;
00CAAC  1  8A              txa                          ; A = $n0
00CAAD  1  45 43           eor pdUnitNumber
00CAAF  1  29 70           and #$70                     ; check only the slot-number bits
00CAB1  1  F0 02           beq @ourSlot
00CAB3  1  A9 02           lda #2                       ; Point to 3rd/4th partitions (2, 3)
00CAB5  1               @ourSlot:                       ; A is 0 or 2
00CAB5  1
00CAB5  1  24 43           bit pdUnitNumber
00CAB7  1  10 01           bpl @driveOne
00CAB9  1  1A              INC_A_OR_ADC1                ; A is 1 or 3
00CABA  1               @driveOne:
00CABA  1
00CABA  1  20 83 CD        jsr RemapAndStoreDriveNum
00CABD  1  B0 0E           bcs @error
00CABF  1
00CABF  1  A5 42           lda pdCommandCode
00CAC1  1                  ASSERT PRODOS_STATUS=0, "PRODOS_STATUS must be 0"
00CAC1  1               ;  cmp #PRODOS_STATUS         ; 0, so no "cmp" is needed
00CAC1  1  F0 0C           beq GetStatus
00CAC3  1
00CAC3  1  C9 01           cmp #PRODOS_READ          ; [OPT] (1 byte) DEC A to check for READ (1)
00CAC5  1  F0 58           beq ReadBlock
00CAC7  1
00CAC7  1               @chk1:
00CAC7  1  C9 02           cmp #PRODOS_WRITE        ; [OPT] (1 byte) DEC A to check for WRITE (2)
00CAC9  1  F0 4C           beq WriteBlock
00CACB  1
00CACB  1               @chk2:                         ; Invalid request number
00CACB  1  A9 27           lda #PRODOS_IO_ERROR
00CACD  1               @error:
00CACD  1  38              sec
00CACE  1  60              rts
00CACF  1
00CACF  1               ;-----------------------------------------------------------------------------
00CACF  1               ; GetStatus - Called by ProDOS and SmartPort to get device status and size
00CACF  1               ;
00CACF  1               ; Input:
00CACF  1               ;       DriveNumber,y
00CACF  1               ;       DrvMiscFlags,y
00CACF  1               ;       X = Slot*16 ($n0)
00CACF  1               ;       Y = slot
00CACF  1               ;
00CACF  1               ; Output:
00CACF  1               ;       A = ProDOS status return code
00CACF  1               ;       X = drive size LSB
00CACF  1               ;       Y = drive size MSB
00CACF  1               ;       Carry flag: 0 = Okay, 1 = Error
00CACF  1               ;-----------------------------------------------------------------------------
00CACF  1               GetStatus:
00CACF  1  B9 78 07        lda DrvMiscFlags,y
00CAD2  1  0A              asl a
00CAD3  1                  ASSERT kMiscDev1=$40, "kMiscDev1 must be $40"
00CAD3  1  30 0B           bmi @dev1
00CAD5  1
00CAD5  1  B9 F8 04        lda DriveNumber,y
00CAD8  1  1A              INC_A_OR_ADC1
00CAD9  1  D9 78 06        cmp PartitionsDev0,y
00CADC  1  F0 13           beq @finalPartition
00CADE  1  D0 09           bne @fullSize
00CAE0  1
00CAE0  1               @dev1:
00CAE0  1  B9 F8 04        lda DriveNumber,y
00CAE3  1  1A              INC_A_OR_ADC1
00CAE4  1  D9 F8 06        cmp PartitionsDev1,y
00CAE7  1  F0 08           beq @finalPartition
00CAE9  1
00CAE9  1               @fullSize:
00CAE9  1  A2 FF           ldx #$FF                  ; X gets low byte of size
00CAEB  1  A0 FF           ldy #$FF                  ; Y gets high byte of size
00CAED  1  A9 00           lda #0
```

```
00CAEF  1  18              clc
00CAF0  1  60              rts
00CAF1  1
00CAF1  1               @finalPartition:
00CAF1  1  A5 08           lda blockCount+2
00CAF3  1  48              pha
00CAF4  1  A5 07           lda blockCount+1
00CAF6  1  48              pha
00CAF7  1  A5 06           lda blockCount
00CAF9  1  48              pha
00CAFA  1
00CAFA  1  20 10 CD        jsr GetDeviceBlockCount   ; SEC if error (check below)
00CAFD  1  A4 07           ldy blockCount+1
00CAFF  1  A6 06           ldx blockCount
00CB01  1
00CB01  1  68              pla
00CB02  1  85 06           sta blockCount
00CB04  1  68              pla
00CB05  1  85 07           sta blockCount+1
00CB07  1  68              pla
00CB08  1  85 08           sta blockCount+2
00CB0A  1
00CB0A  1  B0 03           bcs @ioErr
00CB0C  1  A9 00           lda #0
00CB0E  1  60              rts
00CB0F  1
00CB0F  1               @ioErr:
00CB0F  1  A9 27           lda #PRODOS_IO_ERROR
00CB11  1               readDone:
00CB11  1               writeDone:
00CB11  1  60              rts
00CB12  1
00CB12  1               ;------------------------------------------------------------------------------
00CB12  1               ; SmartPort WRITE BLOCK command
00CB12  1               ;------------------------------------------------------------------------------
00CB12  1               spWriteBlock:
00CB12  1  20 5F CA        jsr SPSetUpReadWrite
00CB15  1  B0 FA           bcs writeDone
00CB17  1               ;  jmp WriteBlock
00CB17  1               ;
00CB17  1               ; v v v   FALL INTO   v v v
00CB17  1               ;------------------------------------------------------------------------
00CB17  1               ; WriteBlock - Write a block in memory to device
00CB17  1               ;
00CB17  1               ; Input:
00CB17  1               ;       pd Command Block Data $42 - $47
00CB17  1               ;       X = slot * 16 ($n0)
00CB17  1               ;       Y = slot
00CB17  1               ;       DrvMiscFlags,y has been set up to specify Dev0 or Dev1
00CB17  1               ;       DriveNumber,y specifies a 32MB chunk (physical partition number)
00CB17  1               ;
00CB17  1               ; Output:
00CB17  1               ;       A = ProDOS write return code
00CB17  1               ;       Carry flag: 0 = Okay, 1 = Error
00CB17  1               ;
00CB17  1               ; ZeroPage Usage:
00CB17  1               ;       zpt1 is used but restored
00CB17  1               ;------------------------------------------------------------------------
00CB17  1               WriteBlock:
00CB17  1  38              sec
00CB18  1  B0 06           bcs ReadWriteCommon
00CB1A  1
00CB1A  1               ;------------------------------------------------------------------------------
00CB1A  1               ; SmartPort READ BLOCK command
00CB1A  1               ;------------------------------------------------------------------------------
00CB1A  1               spReadBlock:
00CB1A  1  20 5F CA        jsr SPSetUpReadWrite
00CB1D  1  B0 F2           bcs readDone
00CB1F  1               ;  jmp ReadBlock
00CB1F  1               ;
00CB1F  1               ; v v v   FALL INTO   v v v
00CB1F  1               ;------------------------------------------------------------------------------
00CB1F  1               ; ReadBlock - Read a block from device into memory
00CB1F  1               ;
00CB1F  1               ; Input:
00CB1F  1               ;       pd Command Block Data $42 - $47
00CB1F  1               ;       X = requested slot number in form $n0 where n = slot 1 to 7
00CB1F  1               ;       DrvMiscFlags,y has been set up to specify Dev0 or Dev1
00CB1F  1               ;       DriveNumber,y specifies a 32MB chunk (physical partition number)
00CB1F  1               ;
00CB1F  1               ; Output:
00CB1F  1               ;       A = ProDOS read return code
00CB1F  1               ;       Carry flag: 0 = Okay, 1 = Error
00CB1F  1               ;
00CB1F  1               ; ZeroPage Usage:
00CB1F  1               ;       zpt1 and pbBlockNumber - used but restored
00CB1F  1               ;------------------------------------------------------------------------------
00CB1F  1               ReadBlock:
00CB1F  1  18              clc
00CB20  1               ;
00CB20  1               ; CLC = Read
00CB20  1               ; SEC = Write
00CB20  1               ;
00CB20  1               ReadWriteCommon:
00CB20  1  A5 47           lda pdBlockNumberH           ; used and restored in WriteBlockCore
```

```
00CB22  1  48              pha
00CB23  1  A5 46           lda pdBlockNumber
00CB25  1  48              pha
00CB26  1
00CB26  1  A5 45           lda pdIOBufferH
00CB28  1  48              pha
00CB29  1  A5 EF           lda zpt1
00CB2B  1  48              pha
00CB2C  1
00CB2C  1  08              php
00CB2D  1  20 A1 CB        jsr Block2LBA               ; program IDE task file
00CB30  1  28              plp
00CB31  1  A9 20           lda #ATACRead
00CB33  1  90 02           bcc @issueCmd               ; CLC=read, SEC=write
00CB35  1               .if WRITE_PROTECT              ; 18 bytes
00CB35  1                  lda DrvMiscFlags,y
00CB35  1                  asl a
00CB35  1                  bmi :+
00CB35  1                  ora #$40                    ; assume that DrvMiscFlags bit 5 was 0
00CB35  1               :  and WriteProtectBits        ; only bits 7 and 6 are used
00CB35  1                  beq @writeEnabled
00CB35  1                  lda #PRODOS_WRITE_PROTECT
00CB35  1                  bne @outErrorA
00CB35  1               @writeEnabled:
00CB35  1                  sec
00CB35  1               .endif
00CB35  1  A9 30           lda #ATACWrite
00CB37  1               @issueCmd:
00CB37  1  08              php
00CB38  1  20 EA CC        jsr WaitReady_IssueCommand_CheckErr
00CB3B  1  F0 5F           beq @PLP_ioError
00CB3D  1  28              plp
00CB3E  1
00CB3E  1  A0 01           ldy #1
00CB40  1  84 EF           sty zpt1
00CB42  1  88              dey                          ; Y = 0
00CB43  1
00CB43  1  B0 16           bcs @write
00CB45  1
00CB45  1               ;--- ReadBlockCore:
00CB45  1               @rLoop:                         ; rLoop is 27 cycles/word
00CB45  1  BD 88 C0        lda ATADataLow,x
00CB48  1  91 44           sta (pdIOBuffer),y
00CB4A  1  C8              iny
00CB4B  1  BD 80 C0        lda ATADataHigh,x
00CB4E  1  91 44           sta (pdIOBuffer),y
00CB50  1  C8              iny
00CB51  1  D0 F2           bne @rLoop
00CB53  1                  WARN (*/256)=(@rLoop/256), "Timing citical rLoop should not cross a page."
00CB53  1
00CB53  1  E6 45           inc pdIOBufferH
00CB55  1  C6 EF           dec zpt1
00CB57  1  10 EC           bpl @rLoop
00CB59  1               ;--- end of ReadBlockCore
00CB59  1  80 29           BRA_OR_JMP @common
00CB5B  1
00CB5B  1               @write:
00CB5B  1               ;--- WriteBlockCore:
00CB5B  1               @ioBufferShifted = pdBlockNumber   ; saved & restored
00CB5B  1  A5 44           lda pdIOBuffer
00CB5D  1  18              clc
00CB5E  1  69 01           adc #1                          ; [OPT] (carefully) for 65C02
00CB60  1  85 46           sta @ioBufferShifted
00CB62  1  A5 45           lda pdIOBuffer+1
00CB64  1  69 00           adc #0
00CB66  1  85 47           sta @ioBufferShifted+1
00CB68  1
00CB68  1  BD 81 C0        lda SetCSMask,x             ; block IDE -CS0 on I/O read to drive
00CB6B  1
00CB6B  1               ; wLoop is 27 cycles/word (using a 2nd zero-page pointer, no PHA/PLA in loop)
00CB6B  1               @wLoop:
00CB6B  1  B1 46           lda (@ioBufferShifted),y
00CB6D  1  9D 80 C0        sta ATADataHigh,x
00CB70  1  B1 44           lda (pdIOBuffer),y
00CB72  1  9D 88 C0        sta ATADataLow,x
00CB75  1  C8              iny
00CB76  1  C8              iny
00CB77  1  D0 F2           bne @wLoop
00CB79  1                  WARN (*/256)=(@wLoop/256), "Timing citical wLoop should not cross a page."
00CB79  1
00CB79  1  E6 45           inc pdIOBuffer+1
00CB7B  1  E6 47           inc @ioBufferShifted+1
00CB7D  1  C6 EF           dec zpt1
00CB7F  1  10 EA           bpl @wLoop
00CB81  1               ; Set back to normal, allow CS0 assertions on read cycles
00CB81  1  BD 82 C0        lda ClearCSMask,x
00CB84  1               ;--- end of WriteBlockCore
00CB84  1
00CB84  1               @common:                        ; finish after a Read or Write
00CB84  1  20 F7 CC        jsr IDEWaitReadyCheckErr
00CB87  1  F0 14           beq @ioError
00CB89  1  A9 00           lda #0
00CB8B  1               @outErrorA:
00CB8B  1  C9 01           cmp #1                       ; SEC when A>0
00CB8D  1  AA              tax
```

```
00CB8E  1
00CB8E  1  68                  pla
00CB8F  1  85 EF               sta zpt1
00CB91  1  68                  pla
00CB92  1  85 45               sta pdIOBufferH
00CB94  1
00CB94  1  68                  pla
00CB95  1  85 46               sta pdBlockNumber
00CB97  1  68                  pla
00CB98  1  85 47               sta pdBlockNumberH
00CB9A  1
00CB9A  1  8A                  txa
00CB9B  1  60                  rts
00CB9C  1
00CB9C  1               @PLP_ioError:
00CB9C  1  28                  plp
00CB9D  1               @ioError:
00CB9D  1  A9 27               lda #PRODOS_IO_ERROR
00CB9F  1  D0 EA               bne @outErrorA
00CBA1  1
00CBA1  1               ;------------------------------------------------------------------------------
00CBA1  1               ; Block2LBA - Translates ProDOS block# into LBA and programs device's task file
00CBA1  1               ;            registers.
00CBA1  1               ;
00CBA1  1               ; Input:
00CBA1  1               ;       pd Command Block Data $42 - $47
00CBA1  1               ;       DrvMiscFlags,y has been set up to specify Dev0 or Dev1
00CBA1  1               ;       DriveNumber,y specifies a 32MB chunk (physical partition number)
00CBA1  1               ;       X = slot * 16 ($n0)
00CBA1  1               ;       Y = slot
00CBA1  1               ;
00CBA1  1               ; Ouput:
00CBA1  1               ;       None
00CBA1  1               ;
00CBA1  1               ; ZeroPage Usage:
00CBA1  1               ;       None
00CBA1  1               ;
00CBA1  1               ; CPU Registers changed:  A, P
00CBA1  1               ;
00CBA1  1               ; This function translates the block number sent in the ProDOS request
00CBA1  1               ; packet, into an ATA Logical Block Address (LBA).
00CBA1  1               ; The least significant 16 bits becomes the ProDOS block#.
00CBA1  1               ; The most significant 16 becomes the ProDOS Drive #
00CBA1  1               ;
00CBA1  1               ; A ProDOS block and a ATA sector are both 512 bytes.
00CBA1  1               ;
00CBA1  1               ; Logical Block Mode, the Logical Block Address is interpreted as follows:
00CBA1  1               ; LBA07-LBA00: Sector Number Register D7-D0.
00CBA1  1               ; LBA15-LBA08: Cylinder Low Register D7-D0.
00CBA1  1               ; LBA23-LBA16: Cylinder High Register D7-D0.
00CBA1  1               ; LBA27-LBA24: Drive/Head Register bits HS3-HS0.
00CBA1  1               ;------------------------------------------------------------------------------
00CBA1  1               Block2LBA:
00CBA1  1  20 00 CD            jsr WaitReady_SetATAHeadFromMiscDrvFlags
00CBA4  1
00CBA4  1  A9 01               lda #1
00CBA6  1  9D 8A C0            sta ATASectorCnt,x
00CBA9  1               ;
00CBA9  1               ; Add BlockOffset to the ProDOS block number to offset the first drive block we
00CBA9  1               ; use. This keeps the device's first BlockOffset blocks free, which usually
00CBA9  1               ; includes a MBR at block 0.
00CBA9  1               ;
00CBA9  1  B9 78 07            lda DrvMiscFlags,y          ; bit 7 = raw block access
00CBAC  1                      ASSERT kMiscRaw=$80, "kMiscRaw must be $80 so we can test it with BMI"
00CBAC  1               .if SMARTPORT_RAW_BLOCK_MAGIC
00CBAC  1                      bmi @rawBlocks
00CBAC  1               .endif
00CBAC  1                      ASSERT kMiscDev1=$40, "kMiscDev1 must be $40 for this test"
00CBAC  1  0A                  asl a                       ; since A<$80, this clears the carry
00CBAD  1  30 1A               bmi @dev1
00CBAF  1
00CBAF  1               ;  clc
00CBAF  1  A5 46               lda pdBlockNumber
00CBB1  1  6D 0F C8            adc BlockOffsetDev0
00CBB4  1  9D 8B C0            sta ATASector,x             ; store ProDOS Low block # into LBA 0-7
00CBB7  1
00CBB7  1  A5 47               lda pdBlockNumberH
00CBB9  1  6D 10 C8            adc BlockOffsetDev0+1
00CBBC  1  9D 8C C0            sta ATACylinder,x           ; store ProDOS High block # into LBA 15-8
00CBBF  1
00CBBF  1  AD 11 C8            lda BlockOffsetDev0+2
00CBC2  1               @addDriveAndStoreCylinderH:
00CBC2  1  79 F8 04            adc DriveNumber,y
00CBC5  1  9D 8D C0            sta ATACylinderH,x          ; store LBA bits 23-16
00CBC8  1  60                  rts
00CBC9  1
00CBC9  1               @dev1:
00CBC9  1               ;  clc
00CBC9  1  A5 46               lda pdBlockNumber
00CBCB  1  6D 12 C8            adc BlockOffsetDev1
00CBCE  1  9D 8B C0            sta ATASector,x             ; store ProDOS Low block # into LBA 0-7
00CBD1  1
00CBD1  1  A5 47               lda pdBlockNumberH
00CBD3  1  6D 13 C8            adc BlockOffsetDev1+1
00CBD6  1  9D 8C C0            sta ATACylinder,x           ; store ProDOS High block # into LBA 15-8
```

```
00CBD9  1
00CBD9  1  AD 14 C8        lda BlockOffsetDev1+2
00CBDC  1  80 E4           BRA_OR_JMP @addDriveAndStoreCylinderH
00CBDE  1
00CBDE  1               .if SMARTPORT_RAW_BLOCK_MAGIC
00CBDE  1               @rawBlocks:
00CBDE  1                  lda pdBlockNumber
00CBDE  1                  sta ATASector,x             ; store ProDOS Low block # into LBA 0-7
00CBDE  1                  lda pdBlockNumberH
00CBDE  1                  sta ATACylinder,x           ; store ProDOS High block # into LBA 15-8
00CBDE  1                  lda DriveNumber,y
00CBDE  1                  BRA_OR_JMP @storeCylinderH
00CBDE  1               .endif
00CBDE  1
00CBDE  1               ;------------------------------------------------------------------------------
00CBDE  1               ; ResetBusIfFirstTime_ClearMiscFlags
00CBDE  1               ;
00CBDE  1               ; Reset the bus (both devices) once, the first time the driver is called.
00CBDE  1               ;
00CBDE  1               ; ide_devctrl Bit 2 = Software Reset, Bit 1 = nIEN (enable assertion of INTRQ)
00CBDE  1               ;
00CBDE  1               ; Input:
00CBDE  1               ;       X = slot * 16 ($n0)
00CBDE  1               ;       Y = slot
00CBDE  1               ;
00CBDE  1               ; ZeroPage Usage:
00CBDE  1               ;       None
00CBDE  1               ;
00CBDE  1               ; CPU Registers changed:  A, P
00CBDE  1               ;------------------------------------------------------------------------------
00CBDE  1               INIT_DONE_SIG = $A5
00CBDE  1
00CBDE  1               ResetBusIfFirstTime_ClearMiscFlags:
00CBDE  1  B9 78 04        lda DriveResetDone,Y
00CBE1  1  C9 A5           cmp #INIT_DONE_SIG
00CBE3  1  F0 45           beq noNeedToResetBus
00CBE5  1
00CBE5  1               ResetBusAlways_ClearMiscFlags:
00CBE5  1  AD 03 C8        lda DefaultBootPartition
00CBE8  1  3A              DEC_A_OR_SBC1
00CBE9  1  99 F8 07        sta BootPartition,y
00CBEC  1  AD 02 C8        lda DefaultBootDevice
00CBEF  1  99 78 05        sta BootDevice,y
00CBF2  1
00CBF2  1  AD 00 C8        lda Max32MBPartitionsDev0
00CBF5  1  99 78 06        sta PartitionsDev0,y
00CBF8  1  AD 01 C8        lda Max32MBPartitionsDev1
00CBFB  1  99 F8 06        sta PartitionsDev1,y
00CBFE  1  19 78 06        ora PartitionsDev0,y
00CC01  1  F0 22           beq @totalMaxPartitionsZero
00CC03  1               ;
00CC03  1               ; Reset the ATA bus (applies to both devices at once)
00CC03  1               ;
00CC03  1  BD 82 C0        lda ClearCSMask,x           ; reset MASK bit in PLD for normal CS0 signaling
00CC06  1
00CC06  1  9E 80 C0        STZ_OR_STA_ATADataHigh_X
00CC09  1
00CC09  1  AD 0E C8        lda ConfigOptionBits
00CC0C  1  30 14           bmi @skipBusReset
00CC0E  1
00CC0E  1  A9 06           lda #$06                    ; SRST (software reset) = 1, Disable INTRQ=1
00CC10  1  9D 86 C0        sta ATADevCtrl,x
00CC13  1               ;
00CC13  1               ; Per ATA-6 sec 9.2, need to wait 5us minimum. Use a delay of 100us to
00CC13  1               ; cover accelerated Apples up to 20 MHz.
00CC13  1               ;
00CC13  1  A9 04           lda #WAIT_100us
00CC15  1  20 77 CD        jsr Wait
00CC18  1
00CC18  1  A9 02           lda #$02                    ; SRST=0, Disable INTRQ=1
00CC1A  1  9D 86 C0        sta ATADevCtrl,x
00CC1D  1               ;
00CC1D  1               ; Per ATA-6 sec 9.2, need to wait 2ms minimum. Use a delay of 40ms to
00CC1D  1               ; cover accelerated Apples up to 20 MHz.
00CC1D  1               ;
00CC1D  1  A9 7C           lda #WAIT_40ms
00CC1F  1  20 77 CD        jsr Wait
00CC22  1
00CC22  1               @skipBusReset:
00CC22  1  20 30 CC        jsr DetermineDevPartitionCounts
00CC25  1
00CC25  1               ; Set the init done flag so init only happens once.
00CC25  1               @totalMaxPartitionsZero:
00CC25  1  A9 A5           lda #INIT_DONE_SIG
00CC27  1  99 78 04        sta DriveResetDone,Y
00CC2A  1
00CC2A  1               noNeedToResetBus:
00CC2A  1  A9 00           lda #0
00CC2C  1  99 78 07        sta DrvMiscFlags,Y
00CC2F  1  60              rts
00CC30  1
00CC30  1
00CC30  1               ;------------------------------------------------------------------------------
00CC30  1               ; DetermineDevPartitionCounts
00CC30  1               ;
```

```
00CC30  1               ; For each device that has a MaxPartitions > 0 and is actually present,
00CC30  1               ; find out how many partitions it contains (up to its configured max).
00CC30  1               ;
00CC30  1               ; Inputs:
00CC30  1               ;    X = slot * 16
00CC30  1               ;    Y = slot
00CC30  1               ;    PartitionsDev0,y = Max32MBPartitionsDev0
00CC30  1               ;    PartitionsDev1,y = Max32MBPartitionsDev1
00CC30  1               ;
00CC30  1               ; Outputs:
00CC30  1               ;    PartitionsDev0,y
00CC30  1               ;    PartitionsDev1,y
00CC30  1               ;------------------------------------------------------------------------------
00CC30  1               DetermineDevPartitionCounts:
00CC30  1  BD 82 C0        lda ClearCSMask,x           ; reset MASK bit in PLD for normal CS0 signaling
00CC33  1
00CC33  1  20 66 CC        jsr WaitForBusReadyAfterReset
00CC36  1  90 09           bcc @busReady
00CC38  1
00CC38  1  A9 00           lda #0
00CC3A  1  99 78 06        sta PartitionsDev0,y
00CC3D  1  99 F8 06        sta PartitionsDev1,y
00CC40  1  60              rts
00CC41  1
00CC41  1               @busReady:
00CC41  1  B9 78 06        lda PartitionsDev0,y
00CC44  1  F0 0D           beq @skipDev0
00CC46  1  A9 E0           lda #$E0                    ; $E0 = [1, LBA=1, 1, Drive=0, LBA 27-24]
00CC48  1  20 84 CC        jsr CheckOneDevice_GetPartitionCount
00CC4B  1  D9 78 06        cmp PartitionsDev0,y
00CC4E  1  B0 03           bcs @skipDev0                ; don't increase beyond the configured number
00CC50  1  99 78 06        sta PartitionsDev0,y
00CC53  1               @skipDev0:
00CC53  1
00CC53  1  B9 F8 06        lda PartitionsDev1,y
00CC56  1  F0 0D           beq @skipDev1
00CC58  1  A9 F0           lda #$F0                    ; $F0 = [1, LBA=1, 1, Drive=1, LBA 27-24]
00CC5A  1  20 84 CC        jsr CheckOneDevice_GetPartitionCount
00CC5D  1  D9 F8 06        cmp PartitionsDev1,y
00CC60  1  B0 03           bcs @skipDev1                ; don't increase beyond the configured number
00CC62  1  99 F8 06        sta PartitionsDev1,y
00CC65  1               @skipDev1:
00CC65  1  60              rts
00CC66  1
00CC66  1
00CC66  1               ;------------------------------------------------------------------------------
00CC66  1               ; WaitForBusReadyAfterReset
00CC66  1               ;
00CC66  1               ; Inputs:
00CC66  1               ;   X = slot * 16 ($n0)
00CC66  1               ;   Y = slot
00CC66  1               ;
00CC66  1               ; Output:
00CC66  1               ;   CLC if the ATA bus is available
00CC66  1               ;   SEC if the bus remained busy -- no devices present?
00CC66  1               ;------------------------------------------------------------------------------
00CC66  1               WaitForBusReadyAfterReset:
00CC66  1  5A              PHY_OR_TYA_PHA
00CC67  1  AC 0C C8        ldy BusResetSeconds
00CC6A  1               @check:
00CC6A  1  BD 8F C0        lda ATAStatus,x
00CC6D  1  10 12           bpl @ready
00CC6F  1
00CC6F  1               ; Wait1Second
00CC6F  1  A9 0A           lda #10
00CC71  1  48           :  pha
00CC72  1  A9 C5           lda #WAIT_100ms
00CC74  1  20 77 CD        jsr Wait
00CC77  1  68              pla
00CC78  1  3A              DEC_A_OR_SBC1
00CC79  1  D0 F6           bne :-
00CC7B  1
00CC7B  1  88              dey
00CC7C  1  D0 EC           bne @check
00CC7E  1  38              sec
00CC7F  1  B0 01           bcs @exit
00CC81  1               @ready:
00CC81  1  18              clc
00CC82  1               @exit:
00CC82  1  7A              PLY_OR_PLA_TAY
00CC83  1  60              rts
00CC84  1
00CC84  1               ;------------------------------------------------------------------------------
00CC84  1               ; CheckOneDevice_GetPartitionCount
00CC84  1               ;
00CC84  1               ; Check to see if a device is attached to the interface.
00CC84  1               ;
00CC84  1               ; Input:
00CC84  1               ;       X = slot * 16 ($n0)
00CC84  1               ;       Y = slot
00CC84  1               ;       A = value for ATAHead ($E0 for Dev0, $F0 for Dev1)
00CC84  1               ;
00CC84  1               ; Output:
00CC84  1               ;       A = number of partitions on device
00CC84  1               ;
```

```
00CC84  1               ;  Checks to see if the drive status register is readable and equal to $50
00CC84  1               ;  If so, return with the Carry clear, otherwise return with the carry set.
00CC84  1               ;  Waits up to 10sec on a standard 1 MHz Apple II for drive to become ready
00CC84  1               ;
00CC84  1               ;  If we determine (by timing out) that the specified device is not present,
00CC84  1               ;  we set ATAHead to the *other* device.
00CC84  1               ;-----------------------------------------------------------------------------
00CC84  1               CheckOneDevice_GetPartitionCount:
00CC84  1  9D 8E C0        sta ATAHead,x
00CC87  1  99 F8 05        sta TempScreenHole,y
00CC8A  1  5A              PHY_OR_TYA_PHA
00CC8B  1  AC 0D C8        ldy CheckDeviceTenths
00CC8E  1  BD 8F C0     :  lda ATAStatus,x
00CC91  1  29 D0           and #%11010000
00CC93  1  C9 50           cmp #%01010000              ; if BUSY=0 and RDY=1 and DSC=1
00CC95  1  F0 14           beq @found
00CC97  1  A9 C5           lda #WAIT_100ms
00CC99  1  20 77 CD        jsr Wait                    ; Wait 100ms for device to be ready
00CC9C  1  88              dey
00CC9D  1  D0 EF           bne :-
00CC9F  1  7A              PLY_OR_PLA_TAY
00CCA0  1               ; We timed out - the specified device is not present, so select the other one.
00CCA0  1  B9 F8 05        lda TempScreenHole,y
00CCA3  1  49 10           eor #$10
00CCA5  1  9D 8E C0        sta ATAHead,x
00CCA8  1
00CCA8  1  A9 00           lda #0
00CCAA  1  60              rts
00CCAB  1
00CCAB  1               @found:
00CCAB  1  7A              PLY_OR_PLA_TAY
00CCAC  1  BD 8E C0        lda ATAHead,x
00CCAF  1  29 10           and #$10
00CCB1  1  0A              asl a
00CCB2  1  0A              asl a
00CCB3  1                  ASSERT kMiscDev1=$40, "kMiscDev1 must be bit 6 for this code"
00CCB3  1  99 78 07        sta DrvMiscFlags,y
00CCB6  1
00CCB6  1  A5 08           lda blockCount+2
00CCB8  1  48              pha
00CCB9  1  A5 07           lda blockCount+1
00CCBB  1  48              pha
00CCBC  1  A5 06           lda blockCount
00CCBE  1  48              pha
00CCBF  1
00CCBF  1  20 10 CD        jsr GetDeviceBlockCount
00CCC2  1  A9 00           lda #0
00CCC4  1  B0 0A           bcs @exit
00CCC6  1
00CCC6  1               ; Round *up* to the next whole number of 32MB partitions.
00CCC6  1               ; Start with blockCount+2 (number of 32MB multiples), and
00CCC6  1               ; add one if the two lower bytes are anything but $0000.
00CCC6  1  A5 06           lda blockCount
00CCC8  1  05 07           ora blockCount+1
00CCCA  1  C9 01           cmp #1                     ; SEC if any bits set in blockCount, blockCount+1
00CCCC  1  A5 08           lda blockCount+2
00CCCE  1  69 00           adc #0
00CCD0  1
00CCD0  1               @exit:
00CCD0  1  99 F8 05        sta TempScreenHole,y
00CCD3  1
00CCD3  1  68              pla
00CCD4  1  85 06           sta blockCount
00CCD6  1  68              pla
00CCD7  1  85 07           sta blockCount+1
00CCD9  1  68              pla
00CCDA  1  85 08           sta blockCount+2
00CCDC  1
00CCDC  1  B9 F8 05        lda TempScreenHole,y
00CCDF  1  60              rts
00CCE0  1
00CCE0  1               ;-----------------------------------------------------------------------------
00CCE0  1               ; IDEWaitReady - Waits for BUSY flag to clear, and returns DRQ bit status
00CCE0  1               ;
00CCE0  1               ; Input:
00CCE0  1               ;       X = slot*16 ($n0)
00CCE0  1               ; Ouput:
00CCE0  1               ;       BNE if DRQ is set
00CCE0  1               ;       BEQ if DRQ is clear
00CCE0  1               ;       (used to be: Carry flag = DRQ status bit)
00CCE0  1               ;
00CCE0  1               ; CPU Registers changed:  A, P
00CCE0  1               ;-----------------------------------------------------------------------------
00CCE0  1               IDEWaitReady:
00CCE0  1  BD 8F C0     :  lda ATAStatus,x
00CCE3  1  30 FB           bmi :-                       ; Wait for BUSY (bit 7) to be zero
00CCE5  1  29 08           and #$08
00CCE7  1  60              rts
00CCE8  1
00CCE8  1               ;-----------------------------------------------------------------------------
00CCE8  1               ; WaitReady_IssueCommand_CheckErr
00CCE8  1               ;
00CCE8  1               ; Input:  A = ATA command
00CCE8  1               ;         X = slot*16
00CCE8  1               ;-----------------------------------------------------------------------------
```

```
00CCE8  1               WaitReady_IssueIdentify_CheckErr:
00CCE8  1  A9 EC           lda #ATAIdentify
00CCEA  1               WaitReady_IssueCommand_CheckErr:
00CCEA  1  48              pha
00CCEB  1  BD 8F C0     :  lda ATAStatus,x
00CCEE  1  30 FB           bmi :-                       ; Wait for BUSY (bit 7) to be zero
00CCF0  1  9E 80 C0        STZ_OR_STA_ATADataHigh_X
00CCF3  1  68              pla
00CCF4  1  9D 8F C0        sta ATACommand,x            ; Issue the read command to the drive
00CCF7  1               ;  jmp IDEWaitReadyCheckErr
00CCF7  1               ;
00CCF7  1               ; v v v   FALL INTO  v v v
00CCF7  1               ;-------------------------------------------------------------------------------
00CCF7  1               ; IDEWaitReadyCheckErr
00CCF7  1               ;
00CCF7  1               ; Input:
00CCF7  1               ;    X = slot * 16
00CCF7  1               ;
00CCF7  1               ; Output:
00CCF7  1               ;    BEQ error    (DRQ=0, ERR=1)
00CCF7  1               ;    BNE noError
00CCF7  1               ;-------------------------------------------------------------------------------
00CCF7  1               IDEWaitReadyCheckErr:
00CCF7  1  BD 8F C0     :  lda ATAStatus,x
00CCFA  1  30 FB           bmi :-                      ; Wait for BUSY (bit 7) to be zero
00CCFC  1  29 09           and #$09
00CCFE  1               .IF USE_65C02
00CCFE  1  3A              dec a                       ; if DRQ=0 and ERR=1 (A=$01) an error occured
00CCFF  1               .ELSE
00CCFF  1                  cmp #$01                    ; if DRQ=0 and ERR=1 an error occured
00CCFF  1               .ENDIF
00CCFF  1  60              rts
00CD00  1
00CD00  1
00CD00  1               ;-------------------------------------------------------------------------------
00CD00  1               ; WaitReady_SetATAHeadFromMiscDrvFlags
00CD00  1               ;
00CD00  1               ; Input:  X = slot*16 ($n0)
00CD00  1               ;         DrvMiscFlags,y has been set up
00CD00  1               ;
00CD00  1               ; Output: A = the value stored into the ATAHead register
00CD00  1               ;-------------------------------------------------------------------------------
00CD00  1               WaitReady_SetATAHeadFromMiscDrvFlags:
00CD00  1  20 E0 CC        jsr IDEWaitReady
00CD03  1  B9 78 07        lda DrvMiscFlags,y
00CD06  1                  ASSERT kMiscDev1=$40, "kMiscDev1 must be $40 for the following code"
00CD06  1  4A              lsr a
00CD07  1  4A              lsr a
00CD08  1  29 10           and #$10
00CD0A  1  09 E0           ora #$E0                    ; $E0 = [1, LBA=1, 1, Drive, LBA 27-24]
00CD0C  1  9D 8E C0        sta ATAHead,x
00CD0F  1  60              rts
00CD10  1
00CD10  1               ;-------------------------------------------------------------------------------
00CD10  1               ; GetDeviceBlockCount
00CD10  1               ;
00CD10  1               ; Inputs:
00CD10  1               ;    DrvMiscFlags,y has been set up
00CD10  1               ;    X = slot * 16
00CD10  1               ;    Y = slot
00CD10  1               ;
00CD10  1               ; Output:
00CD10  1               ;    CLC, blockCount (3 bytes) -- size from device minus BlockOffset
00CD10  1               ;         (Caller must save/restore these locations)
00CD10  1               ;    SEC if error
00CD10  1               ;-------------------------------------------------------------------------------
00CD10  1               GetDeviceBlockCount:
00CD10  1  20 00 CD        jsr WaitReady_SetATAHeadFromMiscDrvFlags
00CD13  1  20 E8 CC        jsr WaitReady_IssueIdentify_CheckErr
00CD16  1  D0 04           bne @ok
00CD18  1  A9 27           lda #PRODOS_IO_ERROR
00CD1A  1  38              sec
00CD1B  1  60              rts
00CD1C  1
00CD1C  1               @ok:
00CD1C  1  5A              PHY_OR_TYA_PHA
00CD1D  1  A0 00           ldy #0                      ; zero loop counter
00CD1F  1               @tossWord:
00CD1F  1  BD 88 C0        lda ATADataLow,x            ; Read words 0 thru 56 but throw them away
00CD22  1  C8              iny
00CD23  1  C0 39           cpy #57                     ; Number of the first word to keep
00CD25  1  D0 F8           bne @tossWord
00CD27  1  7A              PLY_OR_PLA_TAY
00CD28  1
00CD28  1  B9 78 07        lda DrvMiscFlags,y
00CD2B  1                  ASSERT kMiscDev1=$40, "kMiscDev1 must be $40 for this test"
00CD2B  1  0A              asl a
00CD2C  1  0A              asl a
00CD2D  1  5A              PHY_OR_TYA_PHA
00CD2E  1  A0 00           ldy #0
00CD30  1  90 02           bcc @dev0
00CD32  1                  ASSERT BlockOffsetDev1>BlockOffsetDev0, "BlockOffsetDev1 must come after
Dev0"
00CD32  1  A0 03           ldy #BlockOffsetDev1-BlockOffsetDev0
00CD34  1               @dev0:
```

```
00CD34  1
00CD34  1  38              sec
00CD35  1  BD 88 C0        lda ATADataLow,x            ; Read the current capacity in sectors (LBA)
00CD38  1  F9 0F C8        sbc BlockOffsetDev0,y
00CD3B  1  85 06           sta blockCount
00CD3D  1
00CD3D  1  BD 80 C0        lda ATADataHigh,x
00CD40  1  F9 10 C8        sbc BlockOffsetDev0+1,y
00CD43  1  85 07           sta blockCount+1
00CD45  1
00CD45  1  BD 88 C0        lda ATADataLow,x
00CD48  1  F9 11 C8        sbc BlockOffsetDev0+2,y
00CD4B  1  85 08           sta blockCount+2
00CD4D  1
00CD4D  1  BD 80 C0        lda ATADataHigh,x
00CD50  1  E9 00           sbc #0
00CD52  1  F0 08           beq @lessThan8GB
00CD54  1
00CD54  1  A9 FF           lda #$ff
00CD56  1  85 06           sta blockCount
00CD58  1  85 07           sta blockCount+1
00CD5A  1  85 08           sta blockCount+2
00CD5C  1
00CD5C  1               @lessThan8GB:
00CD5C  1  7A              PLY_OR_PLA_TAY
00CD5D  1
00CD5D  1               @tossRemaining:
00CD5D  1  20 E0 CC        jsr IDEWaitReady            ; read the rest of the words, until command ends
00CD60  1  F0 05           beq @done
00CD62  1  BD 88 C0        lda ATADataLow,x
00CD65  1  80 F6           BRA_OR_JMP @tossRemaining
00CD67  1               @done:
00CD67  1  18              clc
00CD68  1  60              rts
00CD69  1
00CD69  1
00CD69  1               ;------------------------------------------------------------------------------
00CD69  1               ; IsDeviceKnownToExist
00CD69  1               ;
00CD69  1               ; Input:
00CD69  1               ;    DrvMiscFlags,y is set up
00CD69  1               ;    Y = slot
00CD69  1               ; Output:
00CD69  1               ;    BNE = device exists
00CD69  1               ;    BEQ = device does not exist
00CD69  1               ;------------------------------------------------------------------------------
00CD69  1               IsDeviceKnownToExist:
00CD69  1  B9 78 07        lda DrvMiscFlags,y
00CD6C  1                  ASSERT kMiscDev1=$40, "kMiscDev1 must be bit 6 for this ASL"
00CD6C  1  0A              asl a
00CD6D  1  30 04           bmi @dev1
00CD6F  1
00CD6F  1  B9 78 06        lda PartitionsDev0,y
00CD72  1  60              rts
00CD73  1
00CD73  1               @dev1:
00CD73  1  B9 F8 06        lda PartitionsDev1,y
00CD76  1  60              rts
00CD77  1
00CD77  1
00CD77  1               ;------------------------------------------------------------------------------
00CD77  1               ;  Wait - Copy of Apple's wait routine. Can't use ROM based routine in case
00CD77  1               ;         ROM is not active when we need it.
00CD77  1               ;
00CD77  1               ; Input:
00CD77  1               ;       A = desired delay time, where Delay(us) = .5(5A^2 + 27A + 26)
00CD77  1               ;       or more usefully: A = (Delay[in uS]/2.5 + 2.09)^.5 - 2.7
00CD77  1               ;
00CD77  1               ; CPU Registers changed:  A, P
00CD77  1               ;------------------------------------------------------------------------------
00CD77  1               Wait:
00CD77  1  38              sec
00CD78  1               @Wait2:
00CD78  1  48              pha
00CD79  1               @Wait3:
00CD79  1  E9 01           sbc #1
00CD7B  1  D0 FC           bne @Wait3
00CD7D  1  68              pla
00CD7E  1  E9 01           sbc #1
00CD80  1  D0 F6           bne @Wait2
00CD82  1  60              rts
00CD83  1
00CD83  1               ;------------------------------------------------------------------------------
00CD83  1               ; RemapAndStoreDriveNum
00CD83  1               ;
00CD83  1               ; Input:  A = logical drive number (0 = first partition, 1 = second, ...)
00CD83  1               ;         Y = slot
00CD83  1               ;         DrvMiscFlags,Y does not have the kMiscDev1 bit set yet
00CD83  1               ;         BootDevice,Y indicates which device's partitions come first
00CD83  1               ;         BootPartition,Y indicates the boot partition (0=first)
00CD83  1               ;
00CD83  1               ; Output: SEC = error (no such partition)
00CD83  1               ;            A = error code
00CD83  1               ;         CLC = no error
00CD83  1               ;            DriveNumber,Y = A = physical drive number
```

```
00CD83  1               ;            DrvMiscFlags,Y --> set kMiscDev1 if Dev1
00CD83  1               ;         X and Y are preserved
00CD83  1               ;
00CD83  1               ; Bring the desired partition to #0 by swapping it with the first one
00CD83  1               ;    (for example, if you boot from partition #3 then eight 32MB chunks
00CD83  1               ;    on the device appear will in the order: 3, 1, 2, 0, 4, 5, 6, 7).
00CD83  1               ;
00CD83  1               ; If you have two devices, all the partitions on the boot device come
00CD83  1               ; before all the partitions on the non-boot device.
00CD83  1               ;
00CD83  1               ; Algorithm:
00CD83  1               ;
00CD83  1               ;   If BootDevice == Dev0:
00CD83  1               ;      ; Dev0 comes first
00CD83  1               ;      swap 0 with BootPartition
00CD83  1               ;      If P >= NumPartitionsDev0:
00CD83  1               ;         MiscFlags = Dev1
00CD83  1               ;         P -= NumPartitionsDev0
00CD83  1               ;         If P >= NumPartitionsDev1, error.
00CD83  1               ;   Else
00CD83  1               ;      ; Dev1 comes first
00CD83  1               ;      swap 0 with BootPartition
00CD83  1               ;      If P >= NumPartitionsDev1:
00CD83  1               ;         P -= NumPartitionsDev1
00CD83  1               ;         if P >= NumPartitionsDev0, error.
00CD83  1               ;      else
00CD83  1               ;         MiscFlags = Dev1
00CD83  1               ;
00CD83  1               ;------------------------------------------------------------------------------
00CD83  1               RemapAndStoreDriveNum:
00CD83  1  48              pha
00CD84  1  B9 78 05        lda BootDevice,y
00CD87  1  D0 22           bne @dev1BeforeDev0
00CD89  1
00CD89  1               ; Dev0 partitions come first (the normal order)
00CD89  1               ;
00CD89  1               ; (1)  Swap 0 with BootPartition
00CD89  1               ;
00CD89  1               ; (2)  if P >= PartitionsDev0:
00CD89  1               ;           MiscFlags = Dev1
00CD89  1               ;           P -= PartitionsDev0
00CD89  1               ;           if P >= PartitionsDev1, error.
00CD89  1  68              pla
00CD8A  1  F0 09           beq @swapToBootPartition
00CD8C  1  D9 F8 07        cmp BootPartition,y
00CD8F  1  D0 07           bne @haveAdjustedPartitionIndex
00CD91  1               @physicalZero:
00CD91  1  A9 00           lda #0      ; map the boot partition's "normal" location to physical 0
00CD93  1  F0 03           beq @haveAdjustedPartitionIndex
00CD95  1               @swapToBootPartition:
00CD95  1  B9 F8 07        lda BootPartition,y     ; map 0 to physical location of boot partition
00CD98  1               @haveAdjustedPartitionIndex:
00CD98  1  99 F8 04        sta DriveNumber,y
00CD9B  1  38              sec
00CD9C  1  F9 78 06        sbc PartitionsDev0,y    ; SBC affects Carry just like CMP
00CD9F  1  90 32           bcc @noError
00CDA1  1  99 F8 04        sta DriveNumber,y
00CDA4  1  D9 F8 06        cmp PartitionsDev1,y
00CDA7  1  B0 22           bcs @tooLarge
00CDA9  1  90 23           bcc @onDev1
00CDAB  1
00CDAB  1               ; Dev1 partitions come first
00CDAB  1               ;
00CDAB  1               ; (1) Swap 0 with BootPartition
00CDAB  1               ;
00CDAB  1               ; (2) if P >= PartitionsDev1:
00CDAB  1               ;        P -= PartitionsDev1
00CDAB  1               ;        if P >= PartitionsDev0, error.
00CDAB  1               ;     else
00CDAB  1               ;        MiscFlags = Dev1
00CDAB  1               @dev1BeforeDev0:
00CDAB  1  68              pla
00CDAC  1  F0 09           beq @swapToBootPartitionDev1
00CDAE  1  D9 F8 07        cmp BootPartition,y
00CDB1  1  D0 07           bne @haveAdjustedPartitionIndexDev1
00CDB3  1               @physicalZeroDev1:
00CDB3  1  A9 00           lda #0
00CDB5  1  F0 03           beq @haveAdjustedPartitionIndexDev1
00CDB7  1               @swapToBootPartitionDev1:
00CDB7  1  B9 F8 07        lda BootPartition,y
00CDBA  1               @haveAdjustedPartitionIndexDev1:
00CDBA  1  99 F8 04        sta DriveNumber,y
00CDBD  1  38              sec
00CDBE  1  F9 F8 06        sbc PartitionsDev1,y     ; SBC affects Carry just like CMP
00CDC1  1  90 0B           bcc @onDev1
00CDC3  1  99 F8 04        sta DriveNumber,y
00CDC6  1  D9 78 06        cmp PartitionsDev0,y
00CDC9  1  90 08           bcc @noError
00CDCB  1               @tooLarge:                  ; carry is already set
00CDCB  1  A9 28           lda #PRODOS_NO_DEVICE
00CDCD  1               ;  sec
00CDCD  1  60              rts
00CDCE  1
00CDCE  1               @onDev1:                    ; carry is already clear
00CDCE  1  A9 40           lda #kMiscDev1
```

```
00CDD0  1               ;  ora DrvMiscFlags,y
00CDD0  1  99 78 07        sta DrvMiscFlags,y
00CDD3  1               @noError:
00CDD3  1               ;  clc
00CDD3  1  B9 F8 04        lda DriveNumber,y
00CDD6  1  60              rts
00CDD7  1
00CDD7  1               ;===============================================================================
00CDD7  1               ;===============================================================================
00CDD7  1
00CDD7  1               .IF FULL_MENU=0
00CDD7  1               ;-------------------------------------------------------------------------------
00CDD7  1               ; InteractiveMenu for 6502
00CDD7  1               ;
00CDD7  1               ; INPUT:  X = slot*16, Y = slot
00CDD7  1               ;
00CDD7  1               ; RTS to caller to continue booting.
00CDD7  1               ;-------------------------------------------------------------------------------
00CDD7  1               InteractiveMenu_Impl:
00CDD7  1                  rts
00CDD7  1               .ELSE
00CDD7  1               ;-------------------------------------------------------------------------------
00CDD7  1               ; InteractiveMenu_Impl
00CDD7  1               ;
00CDD7  1               ; INPUT:  X = slot*16, Y = slot
00CDD7  1               ;
00CDD7  1               ; RTS to caller to continue booting.
00CDD7  1               ;-------------------------------------------------------------------------------
00CDD7  1               kMenuInteractiveRows = 4
00CDD7  1               kMenuTotalRows = 5
00CDD7  1
00CDD7  1               kRowPartitionsDev0 = 0
00CDD7  1               kRowPartitionsDev1 = 1
00CDD7  1               kRowBootDevice = 2
00CDD7  1               kRowBootPartition = 3
00CDD7  1               kRowBootSaveQuit = 4
00CDD7  1
00CDD7  1                  ASSERT (Max32MBPartitionsDev0-Max32MBPartitionsDev0)=kRowPartitionsDev0,
"oops"
00CDD7  1                  ASSERT (Max32MBPartitionsDev1-Max32MBPartitionsDev0)=kRowPartitionsDev1,
"oops"
00CDD7  1                  ASSERT (DefaultBootDevice-Max32MBPartitionsDev0)=kRowBootDevice, "oops"
00CDD7  1                  ASSERT (DefaultBootPartition-Max32MBPartitionsDev0)=kRowBootPartition, "oops"
00CDD7  1
00CDD7  1               kMenuTop = 10
00CDD7  1               kMenuBootSaveQuitLine = 21
00CDD7  1
00CDD7  1               kMenuAlignment = 20
00CDD7  1
00CDD7  1               menuVTAB:
00CDD7  1  0A              .byte kMenuTop
00CDD8  1  0B              .byte kMenuTop+1
00CDD9  1  0D              .byte kMenuTop+3
00CDDA  1  0E              .byte kMenuTop+4
00CDDB  1  15              .byte kMenuBootSaveQuitLine
00CDDC  1               menuHTAB:
00CDDC  1  03              .byte kMenuAlignment-15-2
00CDDD  1  0E              .byte kMenuAlignment-4-2
00CDDE  1  0A              .byte kMenuAlignment-8-2
00CDDF  1  09              .byte kMenuAlignment-9-2
00CDE0  1  08              .byte 8
00CDE1  1               menuRowText:
00CDE1  1  19              .byte MsgMenuRow0-Messages
00CDE2  1  28              .byte MsgMenuRow1-Messages
00CDE3  1  2C              .byte MsgMenuRow2-Messages
00CDE4  1  34              .byte MsgMenuRow3-Messages
00CDE5  1  3D              .byte MsgBootSaveQuit-Messages
00CDE6  1               menuMinimumValues:
00CDE6  1  00 00 00 01     .byte 0, 0, 0, 1
00CDEA  1               menuMaximumValuesPlusOne:
00CDEA  1  0E 0E 02 0E     .byte 13+1, 13+1, 1+1, 13+1
00CDEE  1
00CDEE  1
00CDEE  1               InteractiveMenu_Impl:
00CDEE  1  86 41           stx menuSlot16
00CDF0  1  84 40           sty menuSlot
00CDF2  1               ; Intentionally change $01 so that monitor SLOOP won't continue the boot scan.
00CDF2  1  84 01           sty $01                      ; store anything *other* than $Cn
00CDF4  1
00CDF4  1               ; Reset goes to BASIC
00CDF4  1               .if USE_65C02
00CDF4  1  9C F2 03        stz ResetVector
00CDF7  1               .else
00CDF7  1                  lda #<AppleSoft
00CDF7  1                  sta ResetVector
00CDF7  1               .endif
00CDF7  1  A9 E0           lda #AppleSoft/256
00CDF9  1  8D F3 03        sta ResetVector+1
00CDFC  1  20 6F FB        jsr SETPWRC
00CDFF  1
00CDFF  1  20 57 C8        jsr ClearScreenAndIdentifyCFFA
00CE02  1
00CE02  1               ; Call ResetBus to set up default screen-hole parameters.
00CE02  1  A6 41           ldx menuSlot16
00CE04  1  A4 40           ldy menuSlot
```

```
00CE06  1  20 E5 CB        jsr ResetBusAlways_ClearMiscFlags   ; needs X and Y
00CE09  1
00CE09  1  8D 10 C0        sta KBDSTROBE
00CE0C  1
00CE0C  1               .if MENU_IDENTIFY_DEVS
00CE0C  1  18              clc                          ; ask for dev0
00CE0D  1  20 34 CF        jsr MenuIdentifyDevice
00CE10  1
00CE10  1  AD 01 C8        lda Max32MBPartitionsDev1
00CE13  1  F0 04           beq @skipDev1
00CE15  1  38              sec                          ; ask for dev1
00CE16  1  20 34 CF        jsr MenuIdentifyDevice
00CE19  1               @skipDev1:
00CE19  1               .endif
00CE19  1
00CE19  1  A4 40           ldy menuSlot
00CE1B  1  B9 78 05        lda BootDevice,Y
00CE1E  1  85 87           sta menuValues+kRowBootDevice
00CE20  1  B9 F8 07        lda BootPartition,Y
00CE23  1  1A              INC_A_OR_ADC1
00CE24  1  85 88           sta menuValues+kRowBootPartition
00CE26  1
00CE26  1  A2 01           ldx #kRowPartitionsDev1
00CE28  1  D0 02           bne ResetDownFromX
00CE2A  1
00CE2A  1               ; Pressing Esc in the menu comes here.
00CE2A  1               ; Reset the menu values to the saved defaults.
00CE2A  1               ResetMenuValues:
00CE2A  1  A2 03           ldx #kMenuInteractiveRows-1
00CE2C  1               ResetDownFromX:
00CE2C  1  BD 00 C8     :  lda Max32MBPartitionsDev0,x
00CE2F  1  95 85           sta menuValues,x
00CE31  1  CA              dex
00CE32  1  10 F8           bpl :-
00CE34  1
00CE34  1  A9 03           lda #kRowBootPartition
00CE36  1  85 82           sta menuRow
00CE38  1
00CE38  1               RedisplayValues:
00CE38  1  64 83           STZ_OR_LDA_STA menuMustSave
00CE3A  1  A2 04           ldx #kMenuTotalRows-1
00CE3C  1  BD D7 CD     :  lda menuVTAB,x
00CE3F  1  20 5B FB        jsr TABV
00CE42  1  BD DC CD        lda menuHTAB,x
00CE45  1  85 24           sta CH
00CE47  1  BD E1 CD        lda menuRowText,x
00CE4A  1  A8              tay
00CE4B  1  20 74 CF        jsr Message
00CE4E  1  E0 04           cpx #kRowBootSaveQuit
00CE50  1  F0 22           beq @noValueForThisRow
00CE52  1  A0 0E           ldy #MsgColonSpace-Messages
00CE54  1  20 74 CF        jsr Message
00CE57  1  B5 85           lda menuValues,x
00CE59  1  DD 00 C8        cmp Max32MBPartitionsDev0,x
00CE5C  1  F0 0A           beq @valueUnchanged
00CE5E  1  E0 02           cpx #kRowPartitionsDev1+1
00CE60  1  B0 02           bcs @saveIsOptional
00CE62  1  E6 83           inc menuMustSave
00CE64  1               @saveIsOptional:
00CE64  1  A0 3F           ldy #kInverseText
00CE66  1  84 32           sty INVFLG
00CE68  1               @valueUnchanged:
00CE68  1  20 80 CF        jsr PrintSmallDecimal
00CE6B  1  A0 FF           ldy #kNormalText
00CE6D  1  84 32           sty INVFLG
00CE6F  1  A9 A0           lda #' '+$80
00CE71  1  20 ED FD        jsr COUT
00CE74  1               @noValueForThisRow:
00CE74  1  CA              dex
00CE75  1  10 C5           bpl :-
00CE77  1
00CE77  1               ; Keep the screen holes up to date with the latest settings.
00CE77  1  A4 40           ldy menuSlot
00CE79  1  A5 87           lda menuValues+kRowBootDevice
00CE7B  1  99 78 05        sta BootDevice,Y
00CE7E  1  A5 88           lda menuValues+kRowBootPartition
00CE80  1  3A              DEC_A_OR_SBC1
00CE81  1  99 F8 07        sta BootPartition,Y
00CE84  1
00CE84  1               MenuKeyFreshRow:
00CE84  1  A6 82           ldx menuRow
00CE86  1  BD D7 CD        lda menuVTAB,x
00CE89  1  20 5B FB        jsr TABV
00CE8C  1  A9 16           lda #kMenuAlignment+2
00CE8E  1  85 24           sta CH
00CE90  1  D0 03           bne MenuKey
00CE92  1
00CE92  1               BeepMenuKey:
00CE92  1  20 DD FB        jsr BELL1
00CE95  1               ;
00CE95  1               ; Get a key
00CE95  1               ;
00CE95  1               MenuKey:
00CE95  1  20 0C FD        jsr KEYIN
00CE98  1  C9 9B           cmp #kEscape+$80
```

```
00CE9A  1  F0 8E           beq ResetMenuValues
00CE9C  1
00CE9C  1  C9 88           cmp #kLeftArrow+$80
00CE9E  1  D0 12           bne notLeft
00CEA0  1  A9 FE           lda #-2
00CEA2  1               AdjustMenuValueByAPlusOne:
00CEA2  1               ;  sec
00CEA2  1  75 85           adc menuValues,x
00CEA4  1               ; X = menuRow, A = new value
00CEA4  1               SetValueFromA:
00CEA4  1  DD E6 CD        cmp menuMinimumValues,x
00CEA7  1  90 E9           bcc BeepMenuKey
00CEA9  1  DD EA CD        cmp menuMaximumValuesPlusOne,x
00CEAC  1  B0 E4           bcs BeepMenuKey
00CEAE  1  95 85           sta menuValues,x
00CEB0  1  90 86           bcc RedisplayValues
00CEB2  1
00CEB2  1               notLeft:
00CEB2  1               .if MENU_DIGIT_INPUT
00CEB2  1                  cmp #'0'+$80
00CEB2  1                  bcc notDigit
00CEB2  1                  cmp #'9'+1+$80
00CEB2  1                  bcs notDigit
00CEB2  1                  and #$0F
00CEB2  1                  bpl SetValueFromA
00CEB2  1               notDigit:
00CEB2  1               .endif
00CEB2  1
00CEB2  1  C9 95           cmp #kRightArrow+$80
00CEB4  1  D0 04           bne @notRight
00CEB6  1  A9 00           lda #0
00CEB8  1  F0 E8           beq AdjustMenuValueByAPlusOne
00CEBA  1
00CEBA  1               @notRight:
00CEBA  1               .if MENU_STAR_MONITOR
00CEBA  1                  cmp #'*'+$80
00CEBA  1                  bne @notMonitor
00CEBA  1                  jsr HOME
00CEBA  1                  jmp MONITOR
00CEBA  1               @notMonitor:
00CEBA  1               .endif
00CEBA  1
00CEBA  1  C9 8D           cmp #CR+$80
00CEBC  1  F0 04           beq @down
00CEBE  1  C9 8A           cmp #kDownArrow+$80
00CEC0  1  D0 0D           bne @notDown
00CEC2  1               @down:
00CEC2  1  8A              txa
00CEC3  1  1A              INC_A_OR_ADC1
00CEC4  1  C9 04           cmp #kMenuInteractiveRows
00CEC6  1  90 02           bcc @row
00CEC8  1  A9 00           lda #0
00CECA  1               @row:
00CECA  1  85 82           sta menuRow
00CECC  1               @ok:
00CECC  1               @notQuit:
00CECC  1  4C 84 CE        jmp MenuKeyFreshRow
00CECF  1
00CECF  1               @notDown:
00CECF  1  C9 8B           cmp #kUpArrow+$80
00CED1  1  D0 08           bne @notUp
00CED3  1  C6 82           dec menuRow
00CED5  1  10 F5           bpl @ok
00CED7  1  A9 03           lda #kMenuInteractiveRows-1
00CED9  1  D0 EF           bne @row
00CEDB  1
00CEDB  1               @notUp:
00CEDB  1  29 DF           and #$DF    ; fold lowercase letters to uppercase
00CEDD  1
00CEDD  1  C9 C2           cmp #'B'+$80
00CEDF  1  D0 0F           bne @notBoot
00CEE1  1
00CEE1  1               @MenuBoot:
00CEE1  1  A5 83           lda menuMustSave
00CEE3  1  D0 AD           bne BeepMenuKey
00CEE5  1  20 58 FC        jsr HOME
00CEE8  1                  ASSERT menuJumpVectorH = menuMustSave+1, "sorry"
00CEE8  1  AD F8 07        lda MSLOT
00CEEB  1  85 84           sta menuJumpVectorH
00CEED  1  6C 83 00        jmp (menuMustSave)
00CEF0  1
00CEF0  1               @notBoot:
00CEF0  1  C9 D3           cmp #'S'+$80
00CEF2  1  F0 0A           beq MenuSave
00CEF4  1
00CEF4  1  C9 D1           cmp #'Q'+$80
00CEF6  1  D0 D4           bne @notQuit
00CEF8  1  20 58 FC        jsr HOME
00CEFB  1  4C 00 E0        jmp AppleSoft
00CEFE  1
00CEFE  1               ;
00CEFE  1               ; Save the menuValues settings to the EEPROM.
00CEFE  1               ;
00CEFE  1               ; Only write the changed values.
00CEFE  1               ;
```

```
00CEFE  1               ; If the max # of partitions changes for either device,
00CEFE  1               ; force a device reset on the next boot.
00CEFE  1               ;
00CEFE  1               MenuSave:
00CEFE  1  A2 0E           ldx #Image_WOB_End-Image_WriteOneByteToEEPROM-1
00CF00  1  BD F0 CF     :  lda Image_WriteOneByteToEEPROM,x
00CF03  1  95 90           sta WriteOneByteToEEPROM,x
00CF05  1  CA              dex
00CF06  1  10 F8           bpl :-
00CF08  1
00CF08  1  A6 41           ldx menuSlot16
00CF0A  1  9D 83 C0        sta WriteEEPROM,x
00CF0D  1
00CF0D  1  A2 03           ldx #kMenuInteractiveRows-1
00CF0F  1               @nextByte:
00CF0F  1  B5 85           lda menuValues,x
00CF11  1  DD 00 C8        cmp Max32MBPartitionsDev0,x
00CF14  1  F0 0E           beq @noNeedToWrite
00CF16  1  E0 02           cpx #kRowPartitionsDev1+1
00CF18  1  B0 05           bcs @noNeedToReinit
00CF1A  1  A4 40           ldy menuSlot
00CF1C  1  99 78 04        sta DriveResetDone,y   ; force device reset later
00CF1F  1               @noNeedToReinit:
00CF1F  1  20 90 00        jsr WriteOneByteToEEPROM
00CF22  1  F0 0A           beq @checkWriteProt
00CF24  1               @noNeedToWrite:
00CF24  1  CA              dex
00CF25  1  10 E8           bpl @nextByte
00CF27  1
00CF27  1  A6 41           ldx menuSlot16
00CF29  1  9D 84 C0        sta NoWriteEEPROM,x
00CF2C  1  D0 03           bne @toMenuKey
00CF2E  1
00CF2E  1               @checkWriteProt:
00CF2E  1  20 DD FB        jsr BELL1
00CF31  1               @toMenuKey:
00CF31  1  4C 38 CE        jmp RedisplayValues
00CF34  1               .endif
00CF34  1
00CF34  1               .if MENU_IDENTIFY_DEVS
00CF34  1               ;--------------------------------------------------------------------------------
00CF34  1               ; MenuIdentifyDevice
00CF34  1               ;
00CF34  1               ; Input: CLC for Dev0, SEC for Dev1
00CF34  1               ;--------------------------------------------------------------------------------
00CF34  1               menuIdentifyData       = $0C00
00CF34  1
00CF34  1               MenuIdentifyDevice:
00CF34  1  08              php
00CF35  1  A0 31           ldy #MsgDev-Messages
00CF37  1  20 74 CF        jsr Message
00CF3A  1  28              plp
00CF3B  1  08              php
00CF3C  1  A9 00           lda #0
00CF3E  1  2A              rol a
00CF3F  1  20 80 CF        jsr PrintSmallDecimal
00CF42  1  A0 0E           ldy #MsgColonSpace-Messages
00CF44  1  20 74 CF        jsr Message
00CF47  1
00CF47  1                  ASSERT menuSlot=spSlot, "oops"
00CF47  1                  ASSERT menuSlot16=spSlotX16, "oops"
00CF47  1
00CF47  1  28              plp
00CF48  1  A9 00           lda #0               ; for RORs below as well as the STA
00CF4A  1  85 4A           sta spCmdList
00CF4C  1  6A              ror a
00CF4D  1  6A              ror a
00CF4E  1  A4 40           ldy menuSlot
00CF50  1  99 78 07        sta DrvMiscFlags,y
00CF53  1
00CF53  1  A9 0C           lda #menuIdentifyData/256
00CF55  1  85 4B           sta spCmdList+1
00CF57  1
00CF57  1  20 C9 C9        jsr spStatusIdentify
00CF5A  1               ;  jmp PrintIdentifyData
00CF5A  1               ;
00CF5A  1               ; v v v   FALL INTO   v v v
00CF5A  1               ;--------------------------------------------------------------------------------
00CF5A  1               ; PrintIdentifyData
00CF5A  1               ;
00CF5A  1               ; Display a device's Identify string (the model, not the serial number).
00CF5A  1               ;
00CF5A  1               ; Input:  SEC if there was an error reading the Identify string
00CF5A  1               ;         CLC if menuIdentifyData has been filled in
00CF5A  1               ;--------------------------------------------------------------------------------
00CF5A  1               PrintIdentifyData:
00CF5A  1  B0 16           bcs @error
00CF5C  1               ; show model string (40 chars)
00CF5C  1  A0 36           ldy #27*2
00CF5E  1               @loop:
00CF5E  1  B9 00 0C        lda menuIdentifyData,y
00CF61  1  09 80           ora #$80
00CF63  1  C9 A0           cmp #' '+$80
00CF65  1  90 03           bcc @skip
00CF67  1  20 31 C8        jsr CaseSafeCOUT
```

```
00CF6A  1               @skip:
00CF6A  1  C8              iny
00CF6B  1  C0 5E           cpy #(23*2)+(24*2)
00CF6D  1  90 EF           bcc @loop
00CF6F  1  4C 8E FD        jmp CROUT
00CF72  1
00CF72  1               @error:
00CF72  1  A0 10           ldy #MsgQuestionReturn-Messages
00CF74  1               .endif
00CF74  1               ;  jmp Message
00CF74  1               ; v v v   FALL INTO   v v v
00CF74  1               ;-------------------------------------------------------------------------------
00CF74  1               ; Message
00CF74  1               ;
00CF74  1               ; Input:  Y = offset from Messages table
00CF74  1               ;
00CF74  1               ; Prints a message through CaseSafeCOUT.
00CF74  1               ;-------------------------------------------------------------------------------
00CF74  1               Message:
00CF74  1               @loop:
00CF74  1  B9 9C CF        lda Messages,y
00CF77  1  48              pha
00CF78  1  20 31 C8        jsr CaseSafeCOUT
00CF7B  1  C8              iny
00CF7C  1  68              pla
00CF7D  1  10 F5           bpl @loop
00CF7F  1  60              rts
00CF80  1
00CF80  1               ;-------------------------------------------------------------------------------
00CF80  1               ; PrintSmallDecimal
00CF80  1               ;
00CF80  1               ; Input:  A = 0..19
00CF80  1               ;
00CF80  1               ; Prints 1 or 2 digits, or a "?" if the value is too large.
00CF80  1               ;-------------------------------------------------------------------------------
00CF80  1               MAX_DECIMAL_TO_PRINT = 19
00CF80  1
00CF80  1               PrintSmallDecimal:
00CF80  1  C9 0A           cmp #10
00CF82  1  90 11           bcc PrintDigit
00CF84  1  C9 14           cmp #MAX_DECIMAL_TO_PRINT+1
00CF86  1  90 04           bcc :+
00CF88  1  A9 3F           lda #'?'
00CF8A  1  D0 0B           bne CaseSafeCOUT_Impl
00CF8C  1
00CF8C  1  E9 09        :  sbc #10-1      ; CLC subtract an extra 1
00CF8E  1  48              pha
00CF8F  1  A9 B1           lda #'1'+$80
00CF91  1  20 ED FD        jsr COUT
00CF94  1  68              pla
00CF95  1               PrintDigit:
00CF95  1  09 B0           ora #'0'+$80
00CF97  1               ;   jmp COUT
00CF97  1               ; v v v   FALL INTO   v v v
00CF97  1               ;-------------------------------------------------------------------------------
00CF97  1               ; CaseSafeCOUT
00CF97  1               ;
00CF97  1               ; Print a character, but first change it to uppercase if necessary.
00CF97  1               ;
00CF97  1               ; Check a ROM ID byte to see if we are on an Apple IIe or later, where lowercase
00CF97  1               ; is available.   $FBB3 = $06 on Apple IIe and later.
00CF97  1               ;-------------------------------------------------------------------------------
00CF97  1               CaseSafeCOUT_Impl:
00CF97  1  09 80           ora #$80
00CF99  1               .IF USE_65C02
00CF99  1               .ELSE
00CF99  1                  cmp #'a'+$80
00CF99  1                  bcc @notLC
00CF99  1                  pha
00CF99  1                  lda $FBB3
00CF99  1                  eor #6
00CF99  1                  cmp #1         ; SEC if $FBB3 was not $06
00CF99  1                  pla
00CF99  1                  bcc @lowercaseAvailable
00CF99  1                  and #$DF
00CF99  1               @lowercaseAvailable:
00CF99  1               @notLC:
00CF99  1               .ENDIF
00CF99  1  4C ED FD        jmp COUT
00CF9C  1
00CF9C  1               ;-------------------------------------------------------------------------------
00CF9C  1               ; Messages (up to 256 bytes)
00CF9C  1               ;
00CF9C  1               ; It's OK to use lowercase here, because the code uppercases characters at
00CF9C  1               ; runtime if needed.
00CF9C  1               ;-------------------------------------------------------------------------------
00CF9C  1               Messages:
00CF9C  1               MsgVersion:
00CF9C  1  43 46 46 41     .byte "CFFA "
00CFA0  1  20
00CFA1  1               .IF USE_65C02
00CFA1  1               .ELSE
00CFA1  1                  .byte "6502 "
00CFA1  1               .ENDIF
00CFA1  1               .if IN_DEVELOPMENT
```

```
00CFA1  1                     .byte "d"    ; indicate development versions by changing this letter
00CFA1  1                  .else
00CFA1  1  76                 .byte "v"
00CFA2  1                  .endif
00CFA2  1  32 2E B0           .byte $30+(FIRMWARE_VER/16), ".", $30+(FIRMWARE_VER & $F)+$80   ; "X.Y"
00CFA5  1
00CFA5  1                  MsgSlot:
00CFA5  1  53 6C 6F 74        SentinelString "Slot "
00CFA9  1  A0
00CFAA  1                  MsgColonSpace:
00CFAA  1  3A A0              SentinelString ": "
00CFAC  1                  .if MENU_IDENTIFY_DEVS
00CFAC  1                  MsgQuestionReturn:
00CFAC  1  3F 8D              .byte "?",CR+$80
00CFAE  1                  .endif
00CFAE  1                  MsgErrorNumber:
00CFAE  1  0D 0D              .byte CR,CR
00CFB0  1  45 72 72 20        SentinelString "Err $"
00CFB4  1  A4
00CFB5  1                  .if FULL_MENU
00CFB5  1                  MsgMenuRow0:
00CFB5  1  50 61 72 74        SentinelString "Partitions Dev0"
00CFB9  1  69 74 69 6F
00CFBD  1  6E 73 20 44
00CFC4  1                  MsgMenuRow1:
00CFC4  1  44 65 76 B1        SentinelString "Dev1"
00CFC8  1                  MsgMenuRow2:
00CFC8  1  42 6F 6F 74        .byte "Boot "       ; "Boot Dev" (continues in next message)
00CFCC  1  20
00CFCD  1                  MsgDev:
00CFCD  1  44 65 F6           SentinelString "Dev"
00CFD0  1                  MsgMenuRow3:
00CFD0  1  50 61 72 74        SentinelString "Partition"
00CFD4  1  69 74 69 6F
00CFD8  1  EE
00CFD9  1                  MsgBootSaveQuit:
00CFD9  1  42 29 4F 4F        SentinelString "B)OOT S)AVE Q)UIT"
00CFDD  1  54 20 53 29
00CFE1  1  41 56 45 20
00CFEA  1                  .endif
00CFEA  1                     ASSERT (*-Messages)<256
00CFEA  1
00CFEA  1                  ;-----------------------------------------------------------------------------
00CFEA  1
00CFEA  1                  ALLOW_OVERFLOWING_ROM = 0
00CFEA  1
00CFEA  1                  .if REQUIRES_EEPROM
00CFEA  1
00CFEA  1                  .if ALLOW_OVERFLOWING_ROM
00CFEA  1                     .warning "Allowing ROM overflow, just to see how big the code is."
00CFEA  1                     .byte $FF ; for $CFEF
00CFEA  1                     .byte $FF ; for $CFFF
00CFEA  1                  .else
00CFEA  1                     .out .concat("Free bytes in EEPROM = ", .string($CFEF-*))
00CFEA  1  77 77 77 77        .RES $CFEF-*,$77    ; 238 bytes avail EEPROM from $CF00 to $CFEE.
00CFEE  1  77
00CFEF  1                     CFEF_Unavailable:
00CFEF  1  FF                    .byte $FF           ; touching $CFEF turns off the AUX ROM
00CFF0  1                  .endif
00CFF0  1
00CFF0  1                  ;-----------------------------------------------------------------------------
00CFF0  1                  ; This "image" of WriteOneByteToEEPROM can't be executed in place,
00CFF0  1                  ; because after writing to EEPROM, you can't read from it for the
00CFF0  1                  ; next 200 microseconds.  This get copied into RAM and called there.
00CFF0  1                  ;
00CFF0  1                  ; 15 bytes available (carefully) on EEPROM ($CFF0..CFFE).
00CFF0  1                  ;-----------------------------------------------------------------------------
00CFF0  1                  .if FULL_MENU
00CFF0  1                  Image_WriteOneByteToEEPROM:
00CFF0  1  9D 00 C8           sta Max32MBPartitionsDev0,x
00CFF3  1  A0 00              ldy #0
00CFF5  1  88              :  dey
00CFF6  1  F0 05              beq @fail
00CFF8  1  DD 00 C8           cmp Max32MBPartitionsDev0,x
00CFFB  1  D0 F8              bne :-
00CFFD  1                  @fail:
00CFFD  1  98                 tya     ; out:  BEQ failure, BNE success
00CFFE  1  60                 rts
00CFFF  1                  Image_WOB_End = *
00CFFF  1                  .endif
00CFFF  1                  ;-----------------------------------------------------------------------------
00CFFF  1                  .if ALLOW_OVERFLOWING_ROM
00CFFF  1                     .out .concat("Free bytes in EEPROM = ", .string($D000-*))
00CFFF  1                  .else
00CFFF  1                     .RES $CFFF-*,$33
00CFFF  1                     CFFF_Unavailable:      ; touching $CFFF turns off the AUX ROM
00CFFF  1  FF                    .byte $FF
00D000  1                        ASSERT *=$D000, "Fatal error - We should be at exactly $D000 here."
00D000  1                  .endif
00D000  1                  ;-----------------------------------------------------------------------------
00D000  1                  .else
00D000  1                     ; EPROM only, no EEPROM.  We have to stop before $CF00.
00D000  1                     .out .concat("Free bytes in EPROM = ", .string($CF00-*))
00D000  1                     .if ALLOW_OVERFLOWING_ROM
00D000  1                        .warning "Allowing ROM overflow, just to see how big the code is."
```

```
00D000  1                   .else
00D000  1                      .RES $CF00-*,$77
00D000  1                      ASSERT *=$CF00, "Fatal error - We should be at exactly $CF00 here."
00D000  1                   .endif
00D000  1                .endif
00D000  1
00D000  1
00D000  1
```