CoCo SDC

About the cover image:

Darren Atkinson designed the CoCo SDC hardware and software.
http://cocosdc.blogspot.com

Printed circuit board manufactured by Ed Snider.
https://thezippsterzone.com

The clear acrylic case on the cover designed by tim lindner.
https://youtu.be/OdSOUcd60Ok

CoCo SDC now has a second source: https://retrorewind.ca

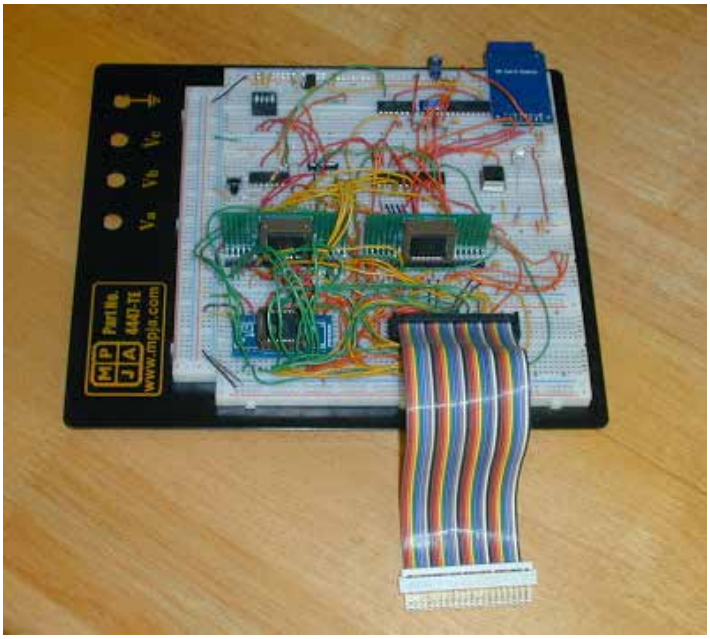
This manual takes inspiration from Brian Blake's original.

Manual Produced by tim Lindner: https://tlindner.macmess.org/

Fourth version: February, 2023.

Table of Contents

1 What is the CoCo SDC



CoCo SDC Prototyping.

A number of high capacity storage solutions have previously been developed for the CoCo, including a MicroSD card interface, a handful of IDE and SCSI interfaces and the very popular DriveWire server.

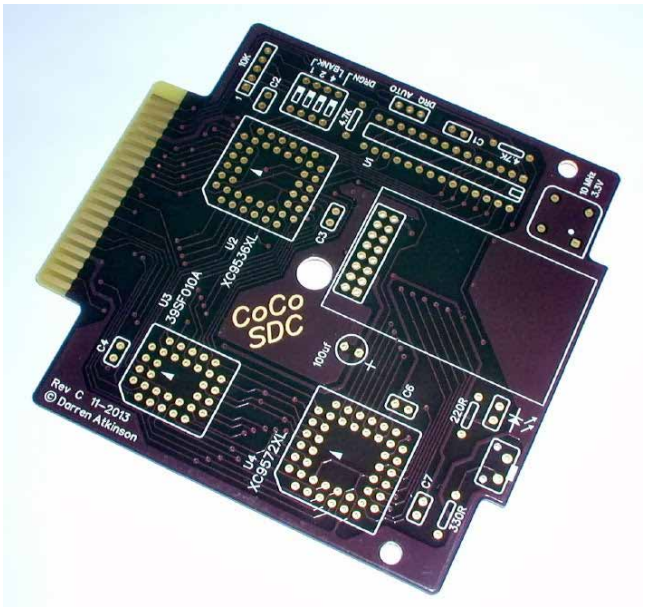
One drawback of these offerings has been that they aren't compatible with software that was written to interact directly with a floppy disk controller. This isn't so much a problem if you are primarily using the CoCo for BASIC programming or running OS9 software. There are however a number of titles (mostly commercial games) that fail to work with those other systems.

The CoCo SDC is a home-brew project for the TRS-80 Color Computer (CoCo). Darren Atkinson began development in 2009. Originally intended to just add floppy disk controller emulation to a DriveWire connection, that idea expanded over time to include emulating a floppy controller for an SD-card reader with DriveWire server access.

The CoCo SDC aims to solve the compatibility problem by combining the traditional "software hook" approach with a robust emulation of the floppy controller in hardware. This dual mode implementation provides excellent performance for the majority of software which "plays by the rules" while adding a high degree of compatibility with those titles that employ floppy-based copy protection schemes or simply choose to use their own floppy drivers.

Feature Overview

An enhanced LBA access mode has also been incorporated into the firmware, allowing the CoCo SDC to go beyond simply emulating floppy disks and interface with virtual hard disk images as large as 2 gigabytes. Two separate disk images (floppy or hard disk) contained on the same SD card may be "connected" simultaneously.



CoCo SDC Revision 3 Board.

Also on board is 128K of Flash memory which is divided into 8 banks of 16K. These 16K banks are both hardware and software selectable and occupy the cartridge ROM space from $C000 to $FEFF (or $FDFF for some CoCo modes). One bank of the Flash memory is used to hold the SDC-DOS code which is yet another patched version of Microsoft's Disk Extended Color BASIC 1.1.

Included in SDC-DOS are additional commands to mount disk image files on the SD card, program the Flash and execute ROM images contained in the Flash. DriveWire disk support is also included in SDC-DOS.

Features and Specifications

- Atmega 328P AVR micro controller @ 10MHz
- Custom 512-byte bootstrap allows firmware to be updated by the CoCo
- 128K In-System-Programmable Flash
- Accepts SD/SDHC cards formatted with FAT16 or FAT32 file system
- Emulates a Tandy Floppy Disk Controller
- Emulate Dragon DOS floppy controllers
- LBA access mode for virtual hard disk support
- Extensions to Disk BASIC in SDC-DOS for disk image manipulation
- DriveWire disk protocol with auto-speed configuration for CoCo 1, 2 or 3
- "Disk Switch" button to support multi-disk programs
- PCB can be mounted in a Tandy FD-502 enclosure
- Requires Extended Color BASIC.
- Works in as little as 16K.

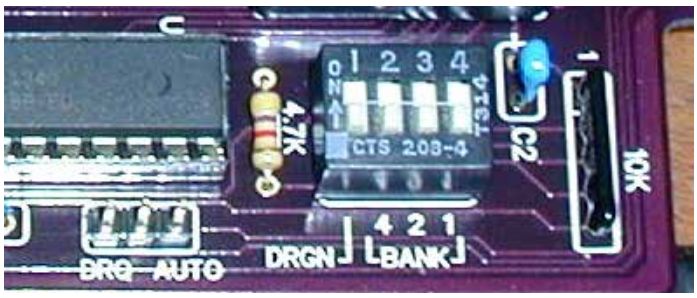
Jumper Settings

The three-pin jumper strip provides two mutually exclusive options for board configuration; Cartridge Auto-Start and Dragon DRQ Mode. The default setting has neither option enabled (no jumper installed).

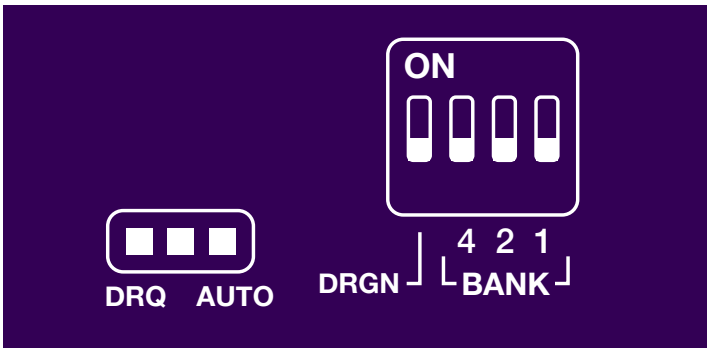
Installing a jumper between the center pin and the AUTO pin connects the Q clock to the CART interrupt pin. This causes the computer to automatically start executing the program in the selected Flash bank at power-up. Do NOT use this option to auto-start SDC-DOS or other Disk BASIC ROMs.

Installing a jumper between the center pin and the DRQ pin is required to support emulation of a Dragon DOS floppy controller. Do NOT install a jumper in this position when using the board with a CoCo.

CAUTION: Make sure the computer's power is off before making any changes to the jumpers.



Jumper & DIP Switches.



Installing a jumper in the DRQ position is necessary for emulating a Dragon floppy controller. Dragon controllers connect the DRQ signal from the Western Digital FDC chip to the CART interrupt (FIRQ) line on the cartridge port. The CoCo SDC provides an emulated DRQ signal for this purpose. You do not want to install the DRQ jumper when running on a CoCo since it only expects CART interrupts to be used for auto-starting a Program Pak.

DIP Switch Settings

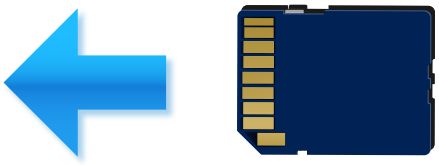
The board includes a 4-position DIP switch that is used to configure which bank of Flash is active at power-up or reset and which addressing scheme is used to communicate with the controller.

CAUTION: Make sure the computer's power is off before making any changes to the DIP switch settings!

Three of the switches specify the Flash bank to activate upon power-up or system reset. The switches are labeled on the board as 4, 2 and 1. The eight Flash banks are numbered 0 to 7. Place only those switches whose sum equals the desired bank number into the ON position. For example, to select bank 5, place the switches labeled 4 and 1 into the ON position and leave the switch labeled 2 in the OFF position. The board is provided with SDC-DOS in bank 0 of the Flash and all three switches in the OFF position.

The DRGN switch selects the address scheme for the controller. In the OFF position the controller will use the CoCo address scheme. In the ON position, the controller will use the Dragon DOS address scheme. The different schemes are summarized in the following table.

Hardware Guide



SD Card Direction.

Use only SD or SDHC cards with the CoCo SDC. It is recommended to use an SD Card with a capacity of 32GB or less. These cards generally are preinstalled with a FAT16 or FAT32 file system. Larger cards, such as a 64 GB SDXC, usually have an exFAT file system. These type of cards will have to be reformatted to work with the CoCo SDC.

The SD card socket is a Push-Push type. When removing the card, always push in to release the latching mechanism before sliding the card out. Never use force to pull the card out of the socket. The card must be inserted into the socket upside-down (label facing down, contacts facing up).

Insert the CoCo SDC into the expansion port before applying power to the CoCo or Multi-Pak Interface. The CoCo SDC, like other floppy disk controller generally go into slot 4 of a Multi-Pak Interface. When power is applied, the LED on the CoCo SDC board should light up momentarily. If the LED does not turn off after a few seconds then this is an indication that the card was not recognized by the hardware. This can happen if the card has not been formatted with a FAT16 or FAT32 file system. It could also indicate that the card was not inserted properly or that there is a problem with the CoCo SDC itself.

Although SD cards are hot-swappable, the CoCo SDC firmware does not handle that situation very well. It's recommended that you completely shutdown the CoCo and MPI before swapping cards.

Usage	CoCo Address	Dragon Address
Drive Control Latch	$FF40	$FF48
Flash Data Register	$FF42	$FF4A
Flash Control Register	$FF43	$FF4B
Command/Status	$FF48	$FF40
FDC Track Register I/O Register 1	$FF49	$FF41
FDC Sector Register I/O Register 2	$FF4A	$FF42
FDC Data Register I/O Register 3	$FF4B	$FF43

CoCo & Dragon Address Schemes.

How is the SDC different from competing products?

- No reliance on expensive third-party modules like the 4D systems uDrive.
- Does not use a slow serial interface based on an obsolete part (6551 ACIA).
- True emulation of the floppy controller hardware for maximum compatibility.
- Supports the popular DriveWire protocol for PC-based disk images.
- Eight banks of in-system-programmable Flash instead of an EPROM.
- Ability to "switch disks" for multi-disk programs via a button on the controller.
- SD cards are FAT-formatted and require no special imaging utility for a PC/Mac.

Probably the only drawback of the device is the fact that the SDC does not come with an enclosure.

John Strong has been known to make 3D printed cases available. Here is his website: http://strongware.net/author/johnstrong/

2 Getting Ready For Fun

The Basics

There are a few very important things that must be touched upon before we get into the actual operation of the CoCo SDC:

1. NEVER insert or remove the CoCo SDC into a CoCo that is turned on! Just like any other device that uses a Color Computer cartridge port, inserting the CoCo SDC into your Color Computer can damage the Color Computer, CoCo SDC, or both.

2. Although SD cards are hot-swappable, the CoCo SDC firmware does not handle that situation very well. It's recommended that you completely shutdown the CoCo and MPI before swapping cards.

3. The firmware in the CoCo SDC does not currently support long file names. You must ensure that the names of all files and directories which are to be accessible by the CoCo conform to the older 8.3 naming conventions.

D & E Compatibility Issues

The CoCo SDC is compatible with all versions of the Color Computer and Dragon Computer lines. However, after getting the CoCo SDC into the hands of some users, it was discovered that Flash programming does not work correctly on certain CoCo 1 motherboards. The two earliest CoCo 1 boards known as the 'D' and 'E' boards are the culprits.

Board Identifiers.

The Cartridge Select Signal ($\overline{CTS}$) on these boards exhibits too slow of a rise-time which causes problems for the high-speed Flash chip. This does not affect normal operation of the CoCo SDC in terms of being able to read data or execute code from the Flash. When writing to the Flash however, the slow rise time often results in incorrect data being stored in the chip.

There are a few options to deal with this problem:

1. Do not use a CoCo 1 with one of the aforementioned motherboards to program the Flash. This option is not ideal, especially if you don't have another suitable CoCo in your possession.

2. Use a Multi-Pak Interface when programming the Flash. The signal buffering in the MPI acts as a kind of filter for the $\overline{CTS}$ line, producing a nice clean transition. This is a good option if you do not wish to modify your CoCo and you happen to own an MPI.

3. Perform a simple modification to the CoCo 1 motherboard to fix the problem (see details below).

Identifying the problem boards

To determine if your CoCo 1 has one of the problematic motherboards you will need to open the case and look inside. The boards in question have a large metal shielded area that encloses all of the main logic chips including the RAM, CPU, SAM, VDG and PIAs. There should be a number printed on the board just below the cartridge port which ends with "-D" or "-E" as seen in the photos below.

If your board has a smaller RF shield which only covers the SAM and RAM chips, or has a number printed on the board (near the front-left corner) that ends in '285' then this is what is often referred to as the 'F' board. The 'F' board does not exhibit the problem and needs no modification.

Motherboard Modification

Please note that any modification to the CoCo is performed at your own risk. Although it is highly unlikely that this modification will cause any problems with other hardware, I can't be held responsible for any damage or loss of functionality that may occur should you choose to go through with it.

The modification is rather simple and involves cutting just one leg of a capacitor. Be sure to disconnect power to the CoCo and discharge any static electricity from your body before touching any of the components inside the CoCo. The affected capacitor is located within the shielded area so you will need to remove the metal cover to gain access. Find the capacitor labeled C85 which is located next to the cartridge port (see photo).

Using an appropriate tool, cut the front leg (the one nearer the keyboard) of the capacitor to sever the connection. That's it! Replace the metal cover, close up the case and you are good to go.

CoCo 1 D & E board mod.

Finding a Suitable Enclosure

Before you plug in the CoCo SDC, you should consider an enclosure for the device. Ideally, an FD-502 enclosure is preferred, as the FD-501 enclosure is slightly different and requires some modifications to work properly. Both enclosures need to be modified to provide easier access to the DIP Switches, while the FD-502 already provides easy access to the SD card slot and the push button by the SD card slot.

The SD card slot and push button ARE accessible, with the FD-501, however, it's advisable to trim some excess material from the housing in order to make it easier to access these features of the CoCo SDC.

Updating SDC-DOS

The SDCSETUP.DSK[1] image contains a utility program that can be used to install the firmware for a CoCo SDC controller. Both the micro controller code and the SDC-DOS (Disk BASIC) ROM image can be installed using this utility. When using a CoCo 1 or 2 a minimum of 32K RAM is required to perform an installation of SDC-DOS and 64K RAM is required to install the MCU firmware.

The disk image may be copied to an SD card or accessed via DriveWire. With the disk image mounted, run the utility by entering:

```
RUN "SETUP"
```

You will be presented with the following menu options:

```
V  DISPLAY INSTALLED VERSIONS
F  INSTALL MCU FIRMWARE
D  INSTALL SDC-DOS
Q  QUIT
```

Press the V key to display the version information of the software currently installed in the CoCo SDC controller. This will display both the MCU firmware version and the SDC-DOS version.

If your firmware is older, press the F key to begin the process of installing the ATmega MCU firmware. This will first load the firmware into memory and perform a checksum validation. The version number of the firmware to be installed is also displayed. Before installation begins you will be asked for confirmation by pressing the Y key. After installation is complete the CoCo will re-boot.

Press the D key to perform an installation of SDC-DOS. This will first load the ROM image into memory and ask which of the 8 Flash banks should be used as the destination. You may install over the version of SDC-DOS that is currently running if so desired. When installation is complete the CoCo will re-boot using the newly installed version (switching banks if necessary).

1 Available from: http://cocosdc.blogspot.com

Boot screen for SDC-DOS.

Rescuing After a Failed Update

This is mostly for folks who have attempted an update on a D or E board CoCo 1, and ended up with a CoCo SDC that will only boot to DECB. The recovery steps should be performed on either a modified D or E board CoCo 1, a CoCo 2, or a CoCo3 – as long as the DIR command hasn't been issued with any arguments prior to attempting the update.

Before we get to the actual steps, it might help to understand a little about the boot process of the CoCo SDC. Mounting a disk image is actually a task performed by the Atmega micro-controller, not SDC-DOS. The commands embedded in SDC-DOS tells the Atmega to mount the disk images at the location you specify. However, if your CoCo SDC crashes after an update attempt, it's likely you will not be able boot to SDC-DOS if you chose to over-write bank 0 of your Flash memory.

The way around this is to use a file the Atmega will automatically mount when the CoCo is first powered on. The STARTUP.CFG file is this file. If you create an ASCII text file named:

STARTUP.CFG

…and save it to the root of your SD card, the Atmega micro-controller will read that file at boot. The STARTUP.CFG file must contain the following line in order cause the Atmega to automatically mount the SDC101.DSK file:

0=SDC101.DSK

When this file is read by the Atmega micro-controller at boot, the CoCo SDC is already mounted to the SDC101.DSK.

DO NOT COMPLETE THE FOLLOWING PROCEDURES ON AN UNMODIFIED D OR E BOARD CoCo 1!

Recovery Steps

In order to recover from this condition, use the following steps:

1. Create the STARTUP.CFG file as described earlier
2. Copy STARTUP.CFG to the root of the SD card on a PC or Macintosh
3. Change the CoCo SDC DIP Switches to select bank 1 (for DECB)
4. Place the SD card in the slot on the CoCo SDC.
5. Make sure your CoCo is turned OFF!
6. Insert the CoCo SDC into the cartridge port of your Color Computer.
7. Turn on your CoCo.
8. Enter the DIR command (with no arguments if you're using a CoCo3 – just to be safe).
9. Enter RUN "UPDATE" and follow the prompts.
10. Your CoCo SDC should now be back to the way it was when it was shipped to you.

You can test this by powering down your CoCo, setting the DIP Switches back to bank 0, and restarting your CoCo. It should boot up to the DECB message with the SDC-DOS and version label.

3 Using the SDC

If you had to go thru the CoCo SDC update process, then you already are familiar with one of the enhanced commands. As stated earlier, SDC-DOS is yet one more version of an extended or patched DECB.

DRIVE - The Status

DRIVE has several uses. It's first and most basic use is to provide a display of the current drive configuration and status.

Typing DRIVE on your CoCo with the SDC plugged into it should give you a display of the drive mappings that resembles this:

```
0: ON     GAMEPAK1.DSK      0
1: OFF    ----              0
2: ON     DW #0             2
3: ON     DW #0             3
```

Each line in the table shows the current mapping information for one of the four logical drive numbers. The first column represents that drive number.

The second column in the table indicates whether or not Disk Image mode is currently on or off. When Disk Image mode is off, the corresponding real floppy drive will be used instead. Assuming a floppy controller is present.

The third column identifies the current Disk Image being mapped to the drive number. There are two possible configurations for this:

1. For an image located on an SD card this will be the name of the image file. (Only the name of the disk image will appear here, not the full directory path)
2. For a DriveWire image this will be "DW #n" where n is a DriveWire server virtual drive number.

The fourth column shows the index of the current virtual floppy disk within the larger 'hard disk' image file (0 - 255).

DRIVE is also used to assign disk images to drives on the CoCo – real drives or virtual drives.

DRIVE – Mounting SD Based Images

The DRIVE command is multi-faceted. As stated above, issuing just the DRIVE command will give you a status on the CoCo SDC's drive mapping. However, the DRIVE command, as displayed in the SDC-DOS updating section, also instructs the Atmega micro-controller to map the disk images stored on the SD card to a virtual drive.

The proper context for the command is:

```
DRIVE n, "path name"
```

Where n is the drive number; path name is the folder that holds your target disk image.

For example, issuing:

```
DRIVE 0, "APPS/TW64/TW64.DSK"
```

Would tell the Atmega to map DRIVE 0 to the TW64.DSK image in the folder listed in the path name.

Another example; let's say you want to play Pitstop II, on the PITSTOP.DSK image, located in the Epyx folder under games. The command you would issue is:

```
DRIVE 0, "GAMES/EPYX/PITSTOP.DSK
```

Entering the DIR command would display a list of the files on the disk image, just like it would with real floppy hardware. You can LOAD & RUN or LOADM & EXEC just as you would with conventional hardware.

There is also an alternate form for the DRIVE Commnad:

```
DRIVE n, "path name", x
```

If the disk image is an HDB-DOS hard drive image, the extra parameter will let you choose which disk to mount.

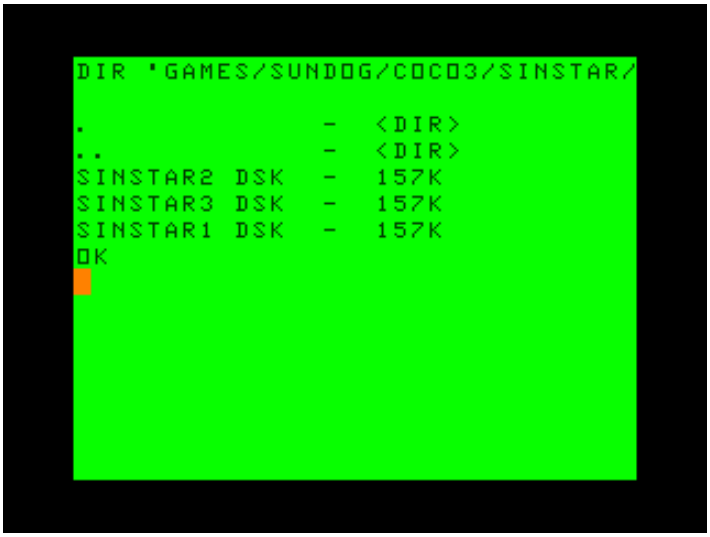
Multiple Disks

As discussed earlier, one of the features CoCo SDC has over products that preceded it to market is the ability to use software that contains multiple disks, as well as software that uses non-standard DSKCON routines.

In order to use games and applications that utilize multiple disks, the CoCo SDC has a way of knowing when this is necessary. The disk images for multi-disk games and applications need to be located in the same folder, and the last character of the disk title must be a number, 1 thruough 9. Entering the following command:

```
DIR "GAMES/SUNDOG/COCO3/SINSTAR/
```

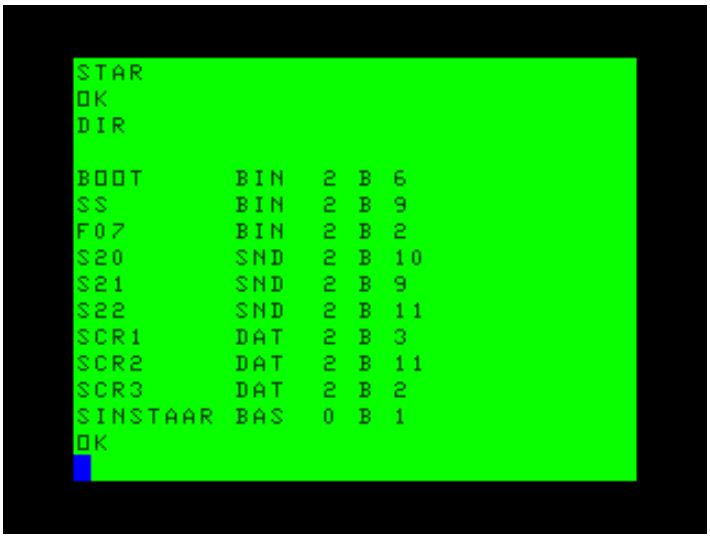
Returns the following:



Now, to mount the first disk image in the folder, simply enter:

```
DRIVE 0, "GAMES/SUNDOG/COCO3/SINSTAR"
```

When the CoCo SDC mounts this image, the red LED on the SDC will blink one time – indicating that the lowest numerical disk in the folder has been mounted – in this instance, one. Entering the DIR command now will result in SINSTAR1.DSK's directory:



From here, you would RUN "SINSTAAR" to load and run the game. When prompted for disk number two, simply push the button on the CoCo SDC, next to the SD slot, one time. The red LED will blink twice indicating that disk two is now selected. Follow the prompts on screen each time the game or application asks for the next disk, press the CoCo SDC button to select the next numerical disk and continue.

DRIVE – With Wildcards

Wildcard characters (* and ?) can be used in the file portion of the path name but not in the directory portion(s). Assuming there were no other files in the GAMES directory whose name started with the letters CH, the command could be shortened to:

```
DRIVE 0,"GAMES/CH*.DSK
```

You may also omit the extension from the file name. In this case the system will first try to mount a file with the given name that has no extension. If no such file exists then .* is substituted for the missing extension and the system uses the first wildcard match, if any. This means the above command could be further shortened to:

```
DRIVE 0,"GAMES/CH*
```

Startup Configuration File

Entering a DRIVE command every time you start up the CoCo can be inconvenient, especially if you tend to use a particular disk image file on a regular basis. To alleviate this problem you can add a startup configuration file to the SD card.

You will need to use a computer with an SD card reader to create a plain-text ASCII file in the root directory of the SD card. The name of the file must be "STARTUP.CFG". The contents of the file may contain lines of text which specify the initial mount points for drives 0 and/or 1 as shown in the example below.

```
0=Nos9Lev2.dsk
1=Utils.dsk
```

You can also specify the path name of the directory to be set as the Current Directory:

```
D=/CoCo/Games
```

Make sure all file and path name components conform to the 8.3 naming conventions.

DIR

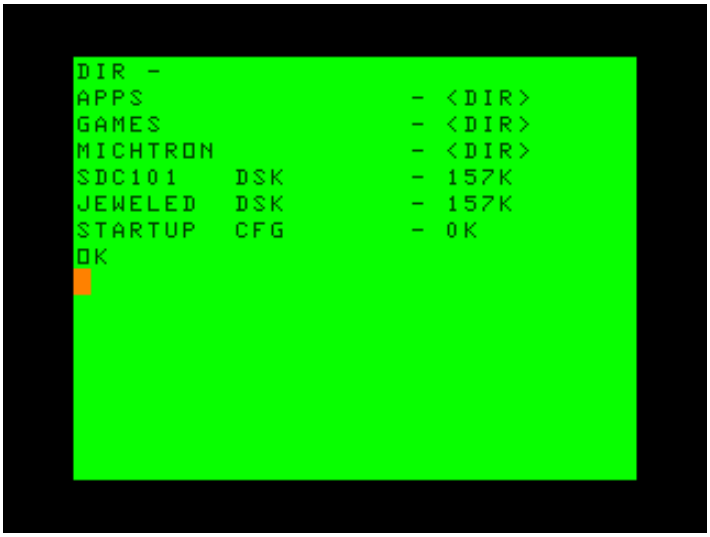
Let's say you have an SD card arranged into folders, and you want to load one of your favorite games, but you cannot remember for sure which folder the image is in. That's where the DIR command will come into play. The DIR command in SDC-DOS works very much like DIR in DECB, in that it gives you a directory listing of the current disk – be it a real floppy disk or a disk image assigned to a drive using the DRIVE command.

Just entering DIR after first powering on the CoCo, with the CoCo SDC inserted will most likely end in an I/O? error since no image was mounted. This may not be the situation if you had to use the STARTUP.CFG file discussed earlier, so we'll start with the assumption of no mounted disks.

When used with an SD/SDHC card, entering:

```
DIR -
```

Will result in a directory listing of the root of the card as shown below, including any files, disk images or folders:



DIR results of SD Card.

You can also look into folders to see what disk images reside inside them. For example, I know that Sinistaar by Sundog Software is a child folder to the Sundog folder. Using the following command, you can work down to find out what the disk names are for Sinistaar:

```
DIR "GAMES/"
```

Lists the files and folders in the GAMES folder.

```
DIR "GAMES/SUNDOG/"
```

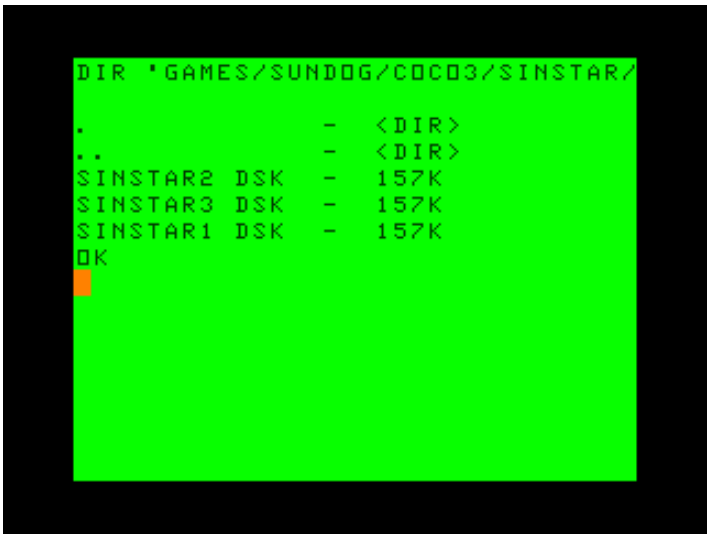
Lists files and folders in the GAMES/SUNDOG folder.

```
DIR "GAMES/SUNDOG/COCO3"
```

You should have the picture by now…

```
DIR "GAMES/SUNDOG/COCO3/SINSTAR/"
```

That last command will result in the following:



DIR results of subdirectory of SD Card.

That is a lot of typing to pull a directory, but, this depends solely on the file structure of your SD card.

DIR can also use wildcard characters (* and ?). For example, if you're looking for disk image and you only remember the first few characters, try:

```
DIR "GAMES/SIN*"
```

Which returns a list of games starting with the characters SIN.

Further, you can also search for specific files extensions.

For example:

```
DIR "MUSIC/*.ORC"
```

Returns a list of files with the ORC file extension. Likewise, entering:

```
DIR "APPS/*.DSK"
```

Returns a list of disk image files in the APPS folder.

Setting Current Directory

You can specify a Current Directory for commands that access the SD card. Once specified, all subsequent commands that refer to files or directories on the SD card are relative to the Current Directory unless the path name begins with a slash (/).

Examples:

```
DIR = "GAMES/ACTION"
DIR = ".."
DIR = "/"
```

Explaining DIR

Information displayed by the DIR command for each item is presented as 4 columns; Name, Extension, Lock Status and Size.

```
CASINO      DSK     -   157K
EGYPT       SDF     -   228K
GAMEPAK1    DSK     L   157K
GAMEPAK2    DSK     -   157K
GR2K        <DIR>
```

When an L appears in the third column instead of a hyphen (-), it indicates that the file is locked. A locked disk image may still be mounted, but you cannot make changes to its contents. Any attempt to use commands such as SAVE or KILL on a locked image will result in a ?WP ERROR.

For files, the fourth column displays the size of the file in kilobytes. For directories, the fourth column simply displays <DIR>.

Locking Disk Images

There are several way to mark a file as read only. Primarily, the CoCo SDC honors the FAT 16/32 read only attribute.

On Mac OS X this is accessible by choosing "Get Info" from the file menu while the file is selected. In the resulting window turn on the "Locked" check box.

On Windows, files can be lock by displaying the Properties and clicking the "Read-only" check box.

In addition, the SDF and VDK file formats include an internal value to mark the image as read only.

Creating New Disk Images

You can create a new, blank disk image file on the SD card by adding the word NEW as a final parameter to the DRIVE command. If the specified file already exists it will not be erased or replaced.

```
DRIVE 0,"SYSTOOLS.DSK",NEW
```

To create a single sided disk image in the SDF format use the NEW+[1] option.

```
DRIVE 0,"SEVENLNK.SDF",NEW+
```

To create a double-sided, 40 track, disk image in the SDF format use the NEW++ option.

```
DRIVE 0,"SEVENLNK.SDF",NEW++
```

A new SDF disk image is like an un-formatted floppy disk. You will need to use the DSKINI command to format the SDF image otherwise all disk operations will result in IO errors.

Ejecting a Disk Image

In most cases it is not necessary to eject disk images under SDC-DOS. To switch disks you can simply mount a new image in place of an existing one. One situation where the need to eject does arise is when you want to move an image to a different drive number.

For example, if you try to mount an image in drive 1 that is already mounted in drive 0, the system will produce an ?AO ERROR (already open). To accomplish this you must first eject the image from drive 0 by using the UNLOAD argument in the DRIVE command:

```
DRIVE 0,UNLOAD
```

Using the CoCo SDC with DriveWire

Having the functionality of storing and accessing your disk images right from an SD card is great. The possibilities are almost endless. However, the CoCo SDC is not a one trick pony.

Years ago DriveWire was released to act as a disk image file server for Color Computer users. With the DriveWire Server software running on a PC or Mac as a server application, a user could connect to their Color Computer to the server machine bit banger ←→ COM port and viola, you could change disk images at will on the server, while being able to run almost all of your favorite software.

The CoCo SDC has the DriveWire protocol already built in, and will communicate with a PC or Mac running the DriveWire server software.

Connecting via the Color Computer

To access disk images on the DriveWire server, you use the DRIVE command as previously explained; instead of a string argument identifying an image file on the SD card, you provide a DriveWire virtual drive number (prefixed with #) in the range of 0 to 63:

1 In SDC-DOS 1.2 the system was changed to create 40 track SDF images instead of 35 Track.

```
DRIVE 2,#0
```

If you have a virtual 'hard disk' image containing an array of up to 256 floppy images, you can specify the index of the desired floppy image as a third argument to the DRIVE command:

```
DRIVE 2,#0,125
```

What this does is assign whatever disk image you have pre-loaded in DriveWire to virtual floppy #2 in SDC-DOS. That is all. You cannot actually change the disk image from SDC-DOS.

Of course in DriveWire, you need to have disk images assigned to a floppy disk. Once the CoCo SDC has been assigned to a DriveWire disk position, you can switch disks at will with the DriveWire GUI on the host computer.

It should be stated here that DriveWire was created for use with HDB-DOS, a product developed and sold by Cloud 9.

There is no support in SDC-DOS or the CoCo SDC for sending a signal over the cable asking DriveWire to switch disk images. Of course if you have the server running on a nearby PC you should be able to use the server's UI to do this, allowing you to use games and applications that reside on multiple disks.

It should also be pointed out that DriveWire support in SDC-DOS was provided more as a convenience feature for transferring files between a PC and the SD card. You don't get quite the same feature-set that HDB-DOS provides.

An example of this would be if you wanted copy a disk image – let's say Flight Simulator II – from your PC to the CoCo SDC. The steps you would take are:

Mount the FSII disk image to Drive 0 in DriveWire.

Create a new floppy image on the CoCo SDC with the following command:

```
DRIVE 0, "GAMES/FSII.DSK",NEW
```

Assign the CoCo SDC drive 1 to the DriveWire server with the following command:

```
DRIVE 1, #0
```

Finish the process by entering the following command:

```
BACKUP 1 TO 0
```

Finally, the floppy controller emulation features of the CoCo SDC are only available to images located on the SD card. The DriveWire support is affected by the same compatibility issues that apply to HDB-DOS or CoCoNet since it is implemented completely in software by SDC-DOS.

If you run a program which has its own floppy I/O routines from an image on the DriveWire server, it will run until the point where those routines are first executed. At that time the CoCo SDC will detect that the floppy hardware is being accessed and try to translate it to the corresponding SD card image (if any). This would be indicated by the red LED on the CoCo SDC turning on and staying on when the disk access would normally occur, causing your Color Computer to enter into a locked state.

Accessing Real Floppy Disks

In addition to disk images located on SD cards and the DriveWire server, SDC-DOS will also provide access to real floppy disks if you have a Mult-Pak Interface and a separate floppy controller. When powering-up the system, the switch on the MPI must be set to the slot number containing the CoCo SDC board.

SDC-DOS will examine the hardware plugged into the MPI looking for the highest numbered slot containing a floppy controller. If found, the floppy controller will be used for any drive number in which Disk Image mode has been turned off.

To turn off Disk Image mode for a particular drive number and thereby utilize the floppy controller, specify OFF as the second argument in the DRIVE command:

```
DRIVE 1,OFF
```

Turning off Disk Image mode disables any SD card image or DriveWire image currently mounted under the specified drive number, but does not eject (unload) the image. You can reestablish access to the underlying image by simply turning Disk Image mode back on for that drive:

```
DRIVE 1,ON
```

Automatic Program Execution

When SDC-DOS starts following power-on or cold reset, it will search the RS-DOS formatted, mounted disk images (as specified in the STARTUP.CFG file) for a BASIC program file named "AUTOEXEC.BAS". If such a file is found, it will be automatically loaded and run. You can hold down the SHIFT key to bypass this feature.

Set Step Rate

When acessing the real floppy disk drives, use the following command to set the step rate:

```
DEF STEP=r
```

where r represents the stepping rate in miliseconds. Allowed values are 6, 12, 20, or 30. The default is 30.

EXP

The EXP command has been added to provide quick access to a program for browsing the contents of the SD card (Explorer utility). Entering the EXP command will cause the system to search the root directory of the SD card for a disk image named SDCEXP.DSK. If found, the disk image will be automatically mounted in drive 1. If the disk image contains a BASIC program named AUTOEXEC.BAS then it will be automatically loaded and run.

You can use this command to start The SDC Explorer program described in chapter 5 of this manual.

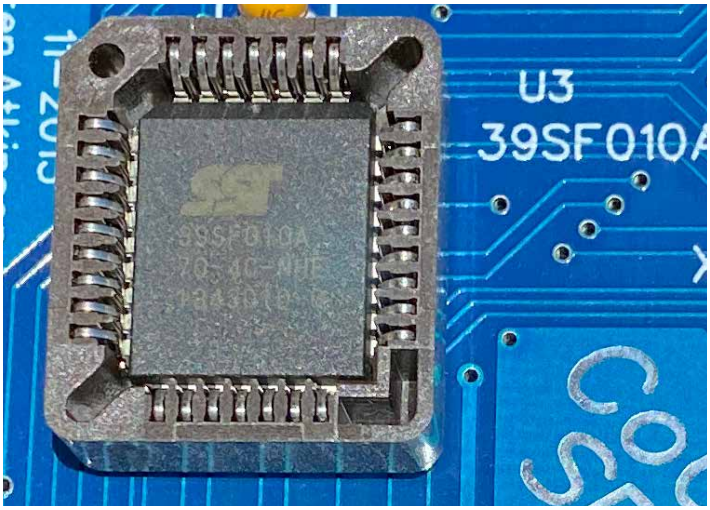
DEF DW = n

You can now change the DriveWire speed configuration by using the DEF DW=n command. Specify the CoCo platform number (1, 2 or 3) as the value for n to select the desired speed:

n	Speed	Comment
1	38,400 bps	
2	57,600 bps	Faster op-amp required to work correctly on a CoCo 1
3	115,200 bps	CoCo 3 only

When running on a CoCo 1 or 2 you cannot specify 3 as the parameter since those machines are not capable of running at true double-speed.

4 Using the Flash



An issue involving Flash programming on a CoCo 1 has been discovered.

Please see Chapter 2, section "D & E Compatibility Issues" for details.

The CoCo SDC contains 128K of Flash memory which is divided into eight banks of 16K. All eight banks of the Flash are user-programmable. The board is provided with SDC-DOS pre-programmed into bank 0 and stock Disk BASIC 1.1 in bank 1. SDC-DOS adds extensions to the Disk BASIC commands which make it easy to take advantage of the Flash memory.

Care should be taken when using these commands to avoid accidental destruction of data. You should always keep copies of the Flashed data elsewhere so it can be re-programmed if necessary.

Running a Cartridge Image

The Flash memory will typically be used to hold images of cartridge-based software (Program Paks).

The RUN command in SDC-DOS has been extended to facilitate the execution of a cartridge image from any of the 8 Flash banks. Simply pass the bank number, prefixed with the @ character, as an argument to the RUN command.

```
RUN @2
```

This has the effect of activating the specified bank and performing a cold re-boot of the CoCo. If the first two bytes of the cartridge image are "DK" then the normal start-up process occurs, allowing Extended BASIC to transfer control to the cartridge image at $C002 during initialization. If anything else appears in the first two bytes then control is transferred to $C000 immediately after the hardware is initialized.

Normally, pressing the RESET button on the CoCo will re-activate the Flash bank set by the DIP switches on the CoCo SDC board. You can force the system to retain the bank selection of the RUN command by suffixing the command with ,R:

```
RUN @2,R
```

After using this option you will need to fully power-down the CoCo (and Multi-Pak Interface) in order to restore the normal Reset behavior.

Erasing Banks and Sectors

The active Flash bank appears in the CoCo memory map from $C000 to $FEFF. Each of the 16K banks is further divided into four sectors of 4K:

Sector	Address Range
0	$C000 - $CFFF
1	$D000 - $DFFF
2	$E000 - $EFFF
3	$F000 - $FEFF

On a CoCo 3, the last 256 bytes ($FE00 to $FEFF) are not accessible using SDC-DOS 1.2 or earlier.

The Flash chip uses the sector divisions for erase operations. Before programming data into the Flash, the sectors to be programmed should first be erased. Erasing the flash has the effect of setting all bits to '1' resulting in byte values of $FF. The KILL MEM command can be used to erase a single sector or an entire bank.

Erase all four sectors of bank 3:

```
KILL MEM @3
```

Erase only sector 2 of bank 6:

```
KILL MEM @6,&HE000
```

When erasing a single sector you can specify any address from $C000 to $FEFF. The entire sector containing that address will be erased. You are not allowed to erase any part of the active bank (the one from which SDC-DOS is running).

Writing to the Flash

Writing data to the Flash involves a process where one or more bits in a byte are cleared to '0'. Once a bit has been cleared it can only be changed to a '1' through an Erase operation. The WRITE MEM command is used to program one or more bytes.

```
WRITE MEM @bank, source, destination, count
```

@bank	Bank number in which the data will be written (0-7)
source	Starting address of the source data
destination	Address in the Flash where the data will be written ($C000-$FEFF)
count	Number of bytes to write

The count argument will be clipped if necessary to prevent writing past the end of the Flash address space.

Copying a Block of Memory

When creating a utility program to manage the Flash, one feature that is often needed is the ability to quickly move a block of memory. Consider the situation where you wanted to copy the contents of one Flash bank to another. This would require that data from the source bank first be copied to a temporary buffer in RAM before writing it to the destination bank. The COPY MEM command has been provided for this purpose.

```
COPY MEM [@bank,] source, destination, count
[USING slot]
```

@bank	The Flash bank number to activate during the copy (0-7)
source	Starting address of the source block
destination	Address where the block will be copied to
count	Number of bytes in the block
USING slot	The Multi-Pak Interface slot to activate during the copy (1-4)

The bank and slot arguments are both optional and mutually exclusive. You can provide one or the other, but not both. The bank argument can be provided when you are copying data from a specific Flash bank on the CoCo SDC board. The slot argument allows you to copy data from the ROM of another cartridge when using a Multi-Pak Interface.

Be careful when using the COPY MEM command as it can easily crash the CoCo if a block is copied to a location used by the system.

The BASIC program listing below shows how to copy the contents of a ROM cartridge into one of the Flash banks of the CoCo SDC. The example assumes an MPI is attached and the CoCo has at least 32K RAM. It does not perform any validation of the input parameters.

```
10 CLEAR 200,&H3FFF ' RESERVE A 16K RAM
BUFFER AT $4000
20 INPUT "COPY FROM MPI SLOT";SL
30 COPY MEM &HC000,&H4000,16384 -256 USING
SL
40 INPUT "DESTINATION BANK";BK
50 KILL MEM @BK ' ERASE BEFORE WRITE
60 WRITE MEM @BK,&H4000,&HC000, 16384-256
```

The GUI editor

SDCFLASH 0.10, Guillaume Major

SDCFLASH is an utility to flash cartridges ROM files into any of the 8 available Coco SDC flash bank. It is compatible with all CoCos and requires 32K.

SDCFLASH is based on SideKick by Luis Antoniosi.

https://colorcomputerarchive.com/search?q=sdcflash

- Select ROM file to flash with a file browser
- Write to any of the 8 available flash banks
- Boot flash bank
- Quick file selection by the first 4 letters - select ROM file to flash with a file browser
- Write to any of the 8 available flash banks
- Boot flash bank
- Quick file selection by the first 4 letters

5 SDC Explorer

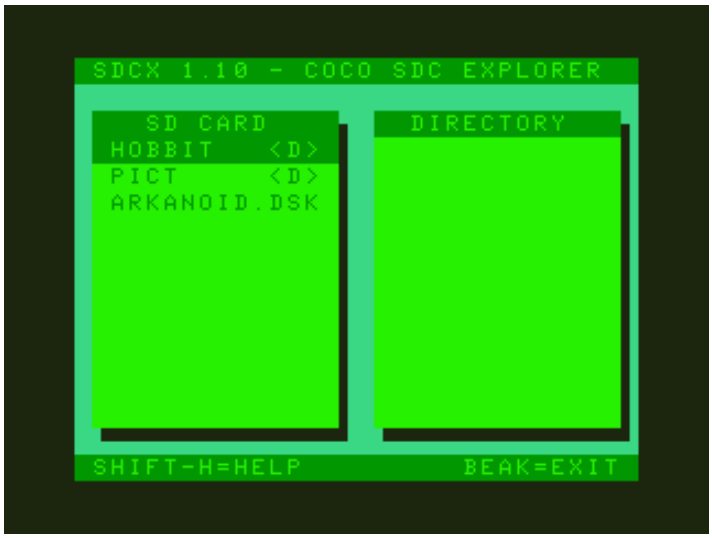
SDC Explorer

The SDC Explorer 1.10 (SDCX) is a free file browsing program created by Guillaume Major for the Coco SDC. It is compatible with all CoCos, the Dragon 32 and the Dragon 64. SDCX requires 32K. It is based on SideKick by Luis Antoniosi.

It can be downloaded from: The Color Computer Archive.[1]

Features

- List files and directories on the SD card
- Create, rename and delete disk images and directories
- Mount directories for multi-disks programs
- Launch ML and BASIC programs
- Detect and boot OS-9 disks
- Quick disk and file selection by the first 4 letters
- Sorted SDC and disk directory listings
- Read and write floppy disks[2]
- Display floppy disks directory[2]
- Format floppy disks[2]
- Double-sided floppy disks support[2]
- Joystick support



Command summary

SHIFT–B	Boot Flash Bank
SHIFT–C	Create disk
SHIFT–K	Create directory
SHIFT–N	Rename disk or directory
SHIFT–X	Delete disk or directory
SHIFT–1	Mount/unmount disk in drive 1 (Coco version)
SHIFT–2	Mount/unmount disk in drive 2 (Dragon version)
SHIFT–I	Display file information
SHIFT–S	Toggle directory sorting
SHIFT–M	Mount directory (multi-disks programs)
CLEAR	Refresh directory
SHIFT–CLEAR	Return to root of SD card.
SHIFT–R	Read floppy disk
SHIFT–W	Write disk image
SHIFT–D	Show floppy drive directory
SHIFT–F	Format floppy disk
SHIFT–H	Show help
ENTER	Launch program or boot disk
BREAK	Quit

Navigational keys

←/→	Switch between windows
SHIFT–↑/↓	Page up/page down
SHIFT–←/→	Home/end
A–Z, 0–9	Select next file matching up to 4 characters typed quickly

1 https://colorcomputerarchive.com/search?q=sdc+explorer
2 SDC-DOS, disk controller and MPI required

Joystick support

Press the joystick button of your choice (right of left) to enable joystick. Press the joystick button to change directory, launch a program or boot a disk. Press and hold the joystick button to navigate quickly through the lists.

Multi-disks Programs

SDCX can mount directories with a disk set to support multi-disks programs. Press (SHIFT)–M to mount a directory with a disk set. Files on the first disk will appear in the right window. Press (SHIFT)–F to refresh the directory list after switching disk with the button on the Coco SDC.

Auto execute SDCX at startup

To run SDCX automatically at startup you need SDC-DOS 1.3 or later. To download the latest version of SDC-DOS go to http://cocosdc.blogspot.ca/ and click on the "Latest Firmware" link in the Pages menu on the right. Run the SETUP.BAS program to update your SDC-DOS version.

Copy the SDCX.DSK file to the root of your SD card and rename it to SDCEXP.DSK.

Create or modify your startup.cfg file at the root of your SD card to mount the SDCEXP.DSK disk in drive 0 or 1 at startup. To do so, add the line #=SDCEXP.DSK where # is the drive number. Example:

```
0=SDCEXP.DSK
```

With this setup you can now use the EXP command to run SDCX.

Floppy drive commands (CoCo only)

Floppy drive commands require SDC-DOS, a Multi-Pak Interface, a disk controller and a floppy drive. 64K is required on a Coco 1 and Coco 2 for double-sided disk support.

Read/Write floppy disks

Select the disk file you want to use for the read or write operation and press:

(SHIFT)–R	To copy a floppy disk to the disk file
(SHIFT)–W	To copy the disk file to a floppy disk

Format floppy disk

Press (SHIFT)–F to format a floppy disk.

Options:

0-3	Drive number selection
T	Change number of tracks (35, 40 or 80)
S	Toggle between single, side 2 and double-sided disk
V	Verify disk (Format command only)

Side(s) option:

Single	Single sided disk
Side 2	Side 2 of a double-sided disk
Double	Double-sided disk (OS-9, FLEX)

NOTE: Drive 3 cannot be selected with Side 2 and Double side option. It is recommended that you use SDC-DOS 1.5 or later for floppy drive operations. On earlier versions of SDC-DOS, the floppy drive motor would fail to shut-off after a copy operation.

Floppy disk directory

The floppy disk directory can be displayed with the (SHIFT)–D command. Side selection can be toggled with the S key.

Limitations

- Creates 40 tracks disks only (no SDF yet)
- Can only delete empty directories (CoCo SDC limitation)
- Limit of 400 files per directory

6 About File Formats

The CoCo SDC supports four different disk image formats. The primary format is referred to as the DSK format and can be used for the imaging of both floppy disks and hard disks. The secondary format, known as SDF, was created specifically for the CoCo SDC and is used for imaging floppy disks only. The third is the JVC format. The fourth is VDK, popular for use with Dragon emulators.

No special name or extension need be assigned to an image file for the purpose of format determination. When a disk image is mounted the firmware detects which format the image uses by examining the file to see if it contains an SDF format signature. Nevertheless it is recommended that an extension which is indicative of the image format be used for identification by humans.

DSK Images

The DSK image format is named for the extension most commonly appearing on such files. Images in this format consist of a simple sector array with each sector being 256 bytes in length. This is the most common format used in the CoCo world.

In order to be recognized as a valid DSK image, the file size must be an exact multiple of 256 bytes. The minimum file size is 82,944 bytes which is equal to 324 sectors or 18 tracks of a single-sided CoCo disk (enough to accommodate the Disk BASIC directory track).

The disk geometry associated with a DSK image is determined by the file size. For floppy images the number of sectors per track is always 18. There are either one or two tracks per cylinder (equal to the number of sides) and a maximum of 80 cylinders. The largest file size for a floppy image is 737,280 bytes or 2880 sectors (double-sided 80 cylinders).

Disk Geometry Table for DSK Images

File Size in Bytes	Sectors	Disk Type	Cylinders	Sides
Less than 82,944	<324	Invalid		
184,320 or less	≤ 720	FD	40	1
368,640 or less	≤ 1440	FD	40	2
737,280 or less	≤ 2880	FD	80	2
737,536 or more	> 2880	HD	80*	1*

* only when accessed through the floppy interface mode

An image with more than 2880 sectors is considered to be a hard disk. If a hard disk image is accessed using the floppy interface mode, only the first 1440 sectors can be used. In this situation those sectors are accessible as a single-sided 80 track floppy disk. The controller's LBA interface mode must be used to access sectors beyond the first 1440 in a hard disk image.

JVC Images

This disk image format is an array of sectors with a 1 to 4 byte header prepended to the front. The header bytes are described in the following table:

Byte Offset	Length	Description	Default Value
0	1	Sectors Per Track	18
1	1	Side Count	1
2	1	Sector Size Code	1 (256 bytes per sector)
3	1	First Sector ID	1

All of the bytes are optional, but are interpreted in the order listed in the table. If omitted their default values are assumed. The CoCo SDC requires the *Sectors Per Track* to be 18, and the *Sector Size Code* to be 1. It will honor a *Side Count* of 1 or 2.

Sector Size Code	Meaning
0	128 bytes per sector
1	256 bytes per sector
2	512 bytes per sector
3	1024 bytes per sector

VDK Images

This is also an array of sectors prepended by a header.

<table>
<tr><th>Byte Offset</th><th>Length</th><th>Description</th></tr>
<tr><td>0</td><td>2</td><td>ASCII ‘d’ and ‘k’.</td></tr>
<tr><td>2</td><td>2</td><td>Header size (little-endian).</td></tr>
<tr><td>4</td><td>1</td><td>Version of VDK format.</td></tr>
<tr><td>5</td><td>1</td><td>Backwards compatibility version.</td></tr>
<tr><td>6</td><td>1</td><td>Identity of file source.</td></tr>
<tr><td>7</td><td>1</td><td>Version of file source.</td></tr>
<tr><td>8</td><td>1</td><td>Number of tracks.</td></tr>
<tr><td>9</td><td>1</td><td>Number of sides.</td></tr>
<tr><td>10</td><td>1</td><td>Flags:
<table>
<tr><th>Bit</th><th>Meaning</th><th>Bit</th><th>Meaning</th></tr>
<tr><td>0</td><td>Write Protect</td><td>1</td><td>Advisory lock</td></tr>
<tr><td>2</td><td>Mandatory Lock</td><td>3</td><td>Disk Set</td></tr>
<tr><td>4</td><td>Unused</td><td>5</td><td>Unused</td></tr>
<tr><td>6</td><td>Unused</td><td>7</td><td>Unused</td></tr>
</table></td></tr>
<tr><td>11</td><td>1</td><td>Compression flags and name length.</td></tr>
</table>

The CoCo SDC will honor a *Number Of Sides* value of 1 or 2. It will also honor the *Write Protect* bit.

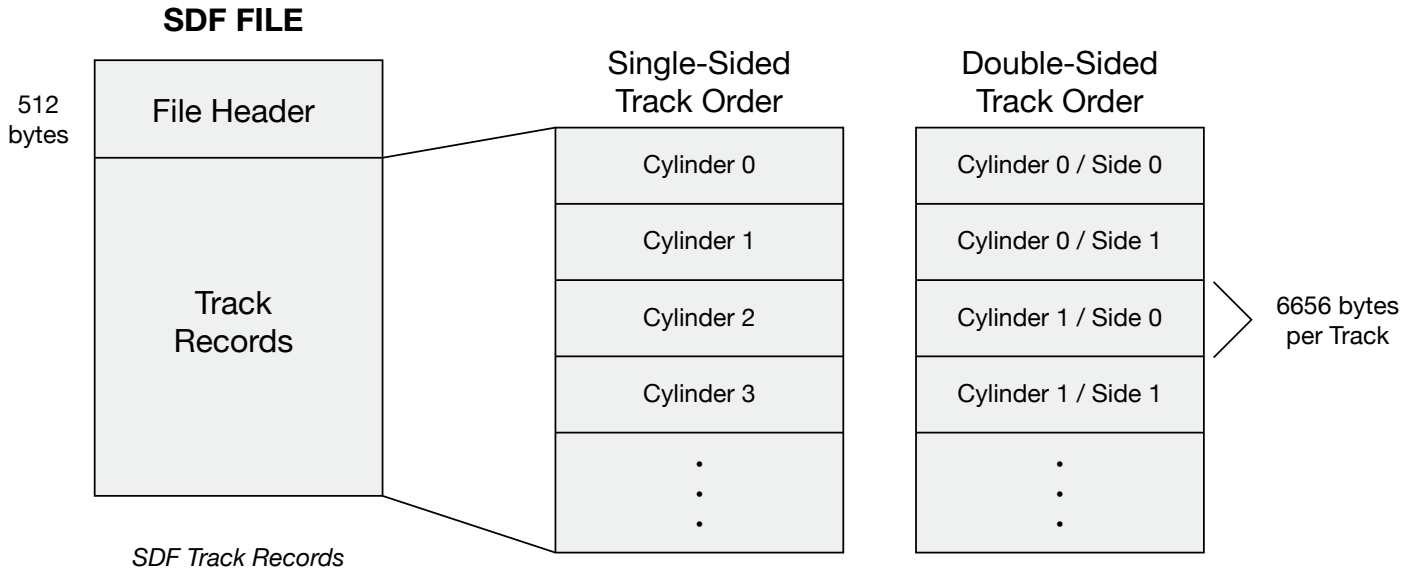
SDF Images

The SDF image format is used to represent floppy disks that have a non-standard layout which is anything other than 18 sectors per track and 256 bytes per sector using standard numbering of the tracks and sectors. The SDF format is similar to the DMK format supported by most CoCo emulators. It has been augmented to provide better performance within the limited resources of the Atmega328 micro controller.

The dmk2sdf program has been created for converting a DMK image to the SDF format. A Win32 command line executable along with the ANSI C source code can be downloaded using the link: https://goo.gl/q61D6s.

SDF File Format

An SDF file consists of a header followed by a variably-sized array of track records. The track records are arranged in ascending order corresponding to their physical position on the disk (cylinder and side).

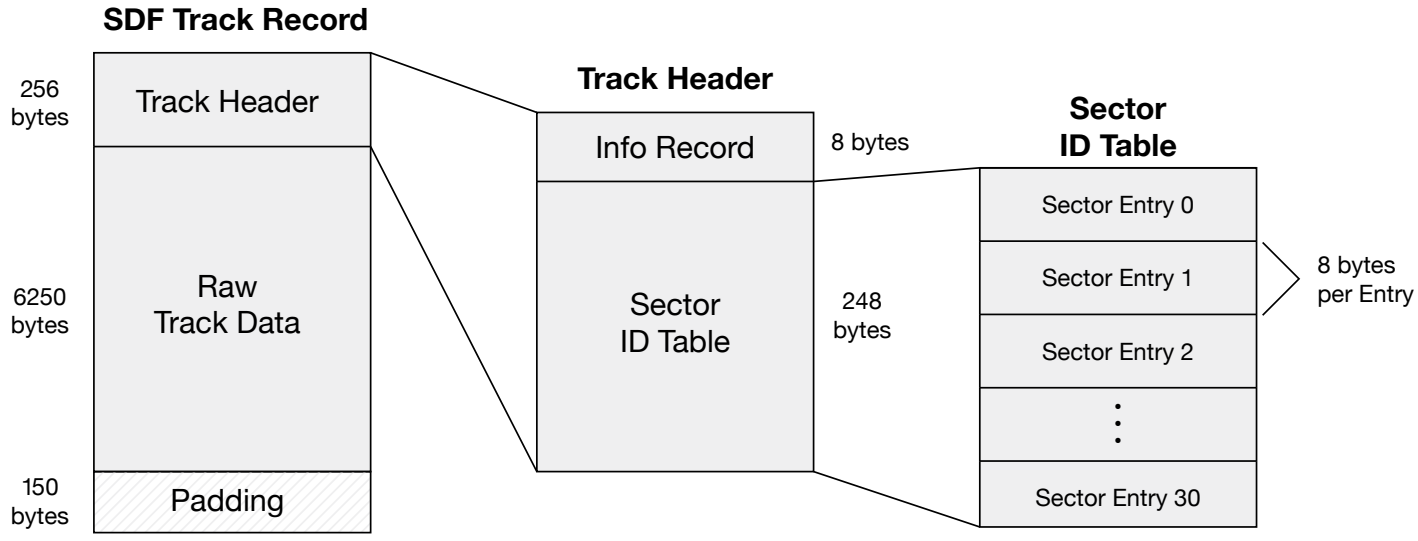


SDF Track Records

Contents of the SDF 512 byte File Header

Byte Offset	Length in Bytes	Description
0	4	Format signature and version string. The ASCII characters 'SDF1' appear at the beginning of the file to identify it as a version 1 SDF image. The numeric character may be incremented in future versions.
4	1	Number of cylinders (80 max).
5	1	Number of sides (1 or 2).
6	1	Write permission: 0x00 = Read/Write ; 0xFF = Read-Only.
7	1	Nested sectors flag: 0x00 = NO ; 0x01 = YES. This byte is set to 0x01 if the disk is known to use a copy-protection scheme in which the ID field for one sector is contained within the Data field of another.
8	504	Reserved. All remaining bytes in the header should be set to zero.

Following the File Header is the array of Track Records. Each track record begins with a 256 byte header and is then followed by 6250 bytes of raw track data. There are 150 bytes of unused padding at the end of a track record which are included to align every track on a 512 byte boundary within the file.



The fixed track size of 6250 bytes can accommodate either a single-density (125 kbps) or double-density (250 kbps) track at 300 rpm. The SDF format does NOT support 8 inch floppy disks or high density (500 kbps) images.

Any part of a track which is recorded in single-density has each byte written twice in succession. This preserves the correct spacing of data on mixed-density tracks.

Contents of the SDF 256 byte Track Header

Byte Offset	Length in Bytes	Description
0	1	Number of used entries in the Sector ID Table.
1	7	Reserved. All seven of the remaining Info Record bytes should be set to zero.
8	248	Sector ID Table.

Each entry in the Sector ID Table is 8 bytes in length and contains information about one sector recorded on the track. The total size of the table is 248 bytes and can accommodate a maximum of 31 sector entries for a single track. All used entries must appear sequentially from the beginning of the table. The unused entries must be filled with zeroes and placed at the end of the table.

Byte Offset	Length in Bytes	Description
0	2	The 14 low-order bits of this 16 bit field contain the offset from the beginning of the Track Record to the first byte of the sector's ID field within the raw track data. The two high-order bits (14 and 15) are used as flags. Bit 14 is set for a sector recorded in single-density. Bit 15 is set if the ID field has an incorrect CRC. This 16 bit integer field is stored in little-endian order (LSB first).
2	2	The 14 low-order bits of this 16 bit field contain the offset from the beginning of the Track Record to the first byte of the sector's Data field within the raw track data. The two high-order bits (14 and 15) are used as flags. Bit 14 is set if the sector's Data field uses a Deleted Data Mark. Bit 15 is set if the Data field has an incorrect CRC. This 16 bit integer field is stored in little-endian order (LSB first).
4	1	The Track Number byte copied from the sector's ID field.
5	1	The Side Number byte copied from the sector's ID field.
6	1	The Sector Number byte copied from the sector's ID field.
7	1	The Size Code byte copied from the sector's ID field.

7 Command Reference

This chapter describes the commands which can be sent to the CoCo SDC hardware from the Color Computer. The accompanying source code file (CommSDC.asm) contains an assembly language subroutine which is used to implement the low-level hardware communications protocol. This document should be used as a reference for making calls to the CommSDC subroutine.

Calling CommSDC to Send Commands and Receive Responses

A command sent to the CoCo SDC consists of a single-byte command code plus 0 to 3 parameter bytes depending on the specific command. Many commands require a 256 byte block of data to be sent as well. When required, the data block is typically used to provide a null-terminated ASCII command string. Any data following the NULL byte (0) in a command string will be ignored by the controller, but a complete block of 256 bytes is always sent.

In response to a command, the CoCo SDC provides a status byte indicating success or failure. Depending on the specific command, the CoCo SDC also returns 0 to 3 response bytes and/or a 256 byte block of data.

The command code and parameter/response bytes are transferred through hardware registers at addresses $FF48 to $FF4B. However, when calling the provided CommSDC sub-routine you do not write the values directly in the hardware registers. Instead, you pass the command code in accumulator A, the first parameter byte in accumulator B and the 2nd and 3rd bytes in index register X (2nd byte in the upper half of X, 3rd byte in the lower half).

For commands which accept or return a data block, you must also pass the address of a buffer in user stack pointer (U). For those commands which return a data block, you may instead pass $FFFF to indicate that you are not interested in receiving the data.

For all commands, the subroutine returns with the status bits in accumulator B. The Carry flag will also be set if any error occurred. If you need to obtain any of the other 3 response bytes then you must read them directly from the hardware registers.

Path Names for Files and Directories on the SD Card

Many commands require that you pass a path name for a file or directory on the SD card as part of the command string. Each component in the path (file or directory) must conform to the MS-DOS 8.3 naming conventions. Each component is separated by a single forward slash character (/). A path is considered to be an absolute path (starting in the root directory) if the very first character is a slash, otherwise the path is considered to be relative to the **Current Directory**. The "." and ".." entries of a sub-directory may also be used in a path name to represent the current path location and the parent directory respectively.

Mount Image

To mount a disk image into one of the virtual drive slots you send a data block to the controller containing an ASCII command string which identifies the target file. The first two characters of the command string are "M:" and are followed by the path name of the file. The string is terminated with a null byte (0). The total size of the string, including "M:" and the null terminator cannot be more than 256 bytes.

Passing a command string without a path name (consisting only of "M:") will effectively eject whatever image may be mounted in the virtual drive at that time.

There is an additional option for mounting a file as a raw array of blocks rather than specifically as a disk image. This is achieved by using a lowercase 'm' in the command string.

When mounting a file of raw blocks, no attempt is made to recognize any JVC, VDK or SDF header within the file and no minimum file size restriction is enforced. The file's data can be accessed through the READ/WRITE LOGICAL SECTOR commands or the STREAM command. Access through FDC emulation is not supported.

Mount Image

Command Code: $E0 or $E1 (drive number in bit 0)

Additional Parameters: None

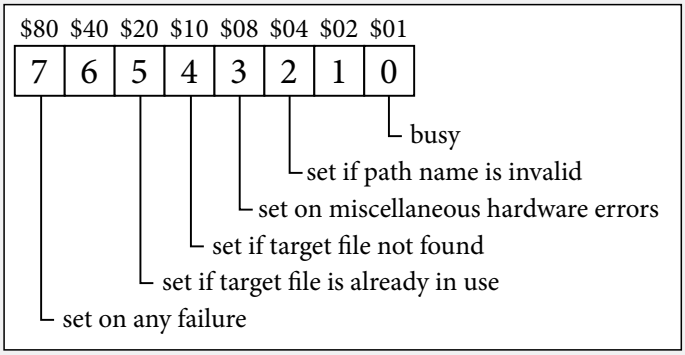
Data Block Sent:

0-1	$4D $3A (M:) or $6D $3A (m:)
2-255	254 bytes containing the null-terminated path string.

Data Block Returned: None

Response Bytes Returned: None

Status:



Mount New Image

Possibly creates and mounts a disk image into one of the virtual drive slots. This command is nearly the same as the Mount Image command with the difference being that the specified image file will be created if it does not already exist. You send a data block to the controller containing an ASCII command string which identifies the target file. The first two characters of the command string are "N:" and are followed by the path name of the file. The string is terminated with a null byte (0). The total size of the string, including "N:" and the null terminator cannot be more than 256 bytes.

Two additional values must be passed as parameters. To create a DSK file that initially contains 630 sectors (Single sided 35 tracks) pass 0 in both the B and X registers. To create an SDF image you pass the number of cylinders (1-80) in B and the number of sides (1-2) in the high-order half of X.

There is an additional option for mounting a new file as a raw array of blocks rather than specifically new a disk image. Using a lowercase 'n' ($6E) in the command string will create a new empty file and mount it for access as a raw array of blocks.

Mount New Image

Command Code: $E0 or $E1 (drive number in bit 0)

Additional Parameters:

$FF49	B	0 for DSK image, number of cylinders for SDF
$FF4A	X.H	0 for DSK image, number of sides for SDF image
$FF4B	X.L	0

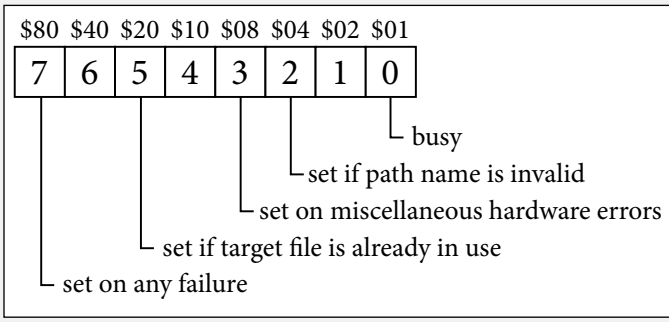
Data Block Sent:

0-1	$4E $3A (N:) or $6E $3A (n:)
2-255	254 bytes containing the null-terminated path string.

Data Block Returned: None

Response Bytes Returned: None

Status:



Get Info for Mounted Image

Retrieves information about the disk image file that is currently mounted in one of the virtual drive slots. This command returns a 256 byte data block. The first 32 bytes of the block contain a copy of the image file's directory entry record from the SD card. The remaining bytes are filled with zeroes.

Get Info for Mounted Image

Command Code: $C0 or $C1 (drive number in bit 0)

Additional Parameters:

$FF49	B	$49 (I)

Data Block Sent: None

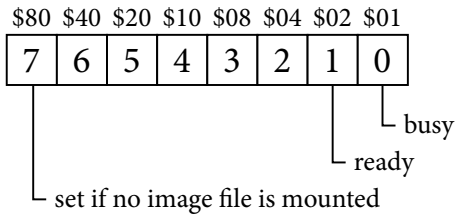
Data Block Returned:

256 bytes with a directory record in the first 32 bytes	
0-7	File Name
8-10	Extension
11	Attribute Bits:
28-31	File Size in bytes (LSB first)

$10	Directory
$04	SDF Format
$02	Hidden
$01	Locked

Response Bytes Returned: None

Status:



Query the Size of a DSK Image

Retrieve the number of 256 byte sectors contained in the disk image file that is currently mounted in one of the virtual drive slots. Note that the value returned is simply the size of the file divided by 256 and is therefore not accurate for images using the SDF format.

Query the Size of a DSK Image

Command Code: $C0 or $C1 (drive number in bit 0)

Additional Parameters:

$FF49	B	$51 (Q)

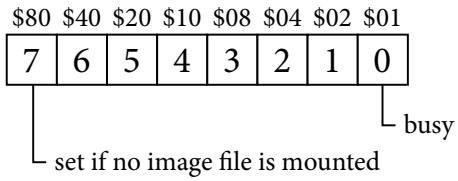
Data Block Sent: None

Data Block Returned: None

Response Bytes Returned:

$FF49	Sector count high byte
$FF4A	Sector count middle byte
$FF4B	Sector count low byte

Status:



Set Current Directory

To set the Current Directory for the SD card (the directory from which relative path names originate), you send a data block to the controller containing an ASCII command string which identifies the target directory. The first two characters of the command string are "D:" and are followed by the path name of the directory. The string is terminated with a null byte (0). The total size of the string, including "D:" and the null terminator cannot be more than 256 bytes.

Set Current Directory

Command Code: $E0

Additional Parameters: None

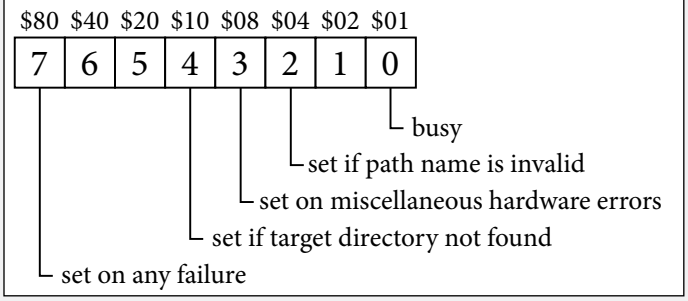
Data Block Sent:

0-1	$44 $3A (D:)
2-255	254 bytes containing the null-terminated path string.

Data Block Returned: None

Response Bytes Returned: None

Status:



Get Current Directory

Retrieves information about the Current Directory for the SD card. This command returns a 256 byte data block. The first 32 bytes of the block contain a copy of the directory entry record from the SD card. The remaining bytes are filled with zeroes.

If the Current Directory is the root directory then the 11 bytes which make up the name and extension fields will all be zeroes and bits 4 and 7 of the Status register will be set.

Note that this command only retrieves the leaf name of the current directory, not the full path. It is possible to construct the full path of the current directory by using the Set Current Directory command to walk up the directory hierarchy one step at a time (using ".."), retrieving each name along the way. Once you have the full path you can use it to restore the Current Directory to its original location.

Get Current Directory

Command Code: $C0

Additional Parameters:

$FF49	B	$43 (C)

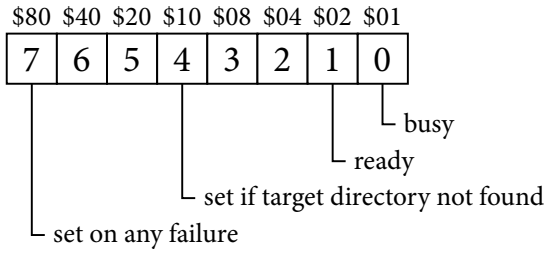
Data Block Sent: None

Data Block Returned:

256 bytes with a directory record in the first 32 bytes	
0-7	File Name
8-10	Extension
12-31	Private

Response Bytes Returned: None

Status:



Initiate Directory Listing

Begins the process of retrieving a list of items from a directory on the SD card. This command accepts an ASCII command string which identifies the target directory whose contents you wish to retrieve. The first two characters of the command string are "L:" and are followed by the path name of the target directory. The final component of the path name should be a wildcard pattern that will be used to filter the list of returned items. Examples of a valid command string are:

L:*.*	All items in the current directory
L:GAMES/*.SDF	All items with an SDF extension in the GAMES sub-directory.

After successfully sending this command you will need to execute one or more Directory Page commands to retrieve the requested items.

Initiate Directory Listing

Command Code: $E0

Additional Parameters: None

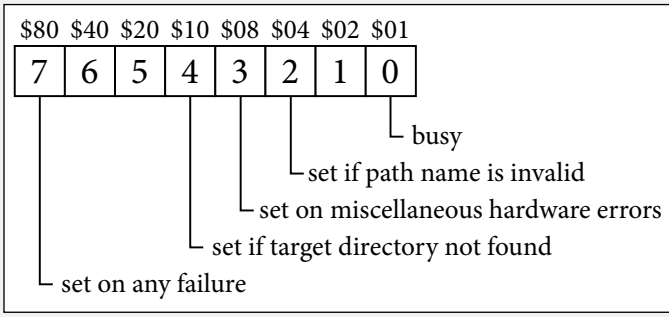
Data Block Sent:

0-1	$4C $3A (L:)
2-255	254 bytes containing the null-terminated path string.

Data Block Returned: None

Response Bytes Returned: None

Status:



Directory Page

Retrieves one page of items for a directory listing. After successfully sending an Initiate Directory Listing command, you will typically send one or more Directory Page commands to retrieve the records which describe the matching items from the directory.

This command returns a 256 byte data block which is divided into 16 records of 16 bytes each. Each record describes one item.

If there are not enough items to fill the entire page then unused records are filled with zeroes. You may continue to send commands for additional pages until a page containing at least one unused record is returned. You are not required to send a command for every page of the listing. You can stop at any time.

Directory Page

Command Code: $C0

Additional Parameters:

$FF49	B	$3E (>)

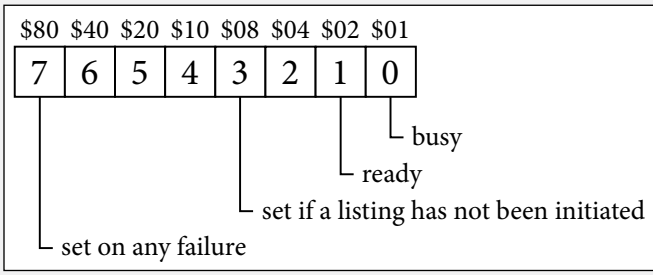
Data Block Sent: None

Data Block Returned:

256 bytes containing an array of 16 directory records	
0-7	File Name
8-10	Extension
11	Attribute Bits: $10 Directory $02 Hidden $01 Locked
12-15	Size in bytes (MSB first)

Response Bytes Returned: None

Status:



Create New Directory

Creates a new directory on the SD card. This command accepts a data block containing an ASCII command identifying the name and location of the new directory. The first two characters of the command string are "K:" and are followed by the path name for the directory. The string is terminated with a null byte (0). The total size of the string, including "K:" and the null terminator cannot be more than 256 bytes.

The command will only create the single directory identified by the leaf component in the path name. Any directories listed in the path name preceding the leaf must already exist.

Create New Directory

Command Code: $E0

Additional Parameters: None

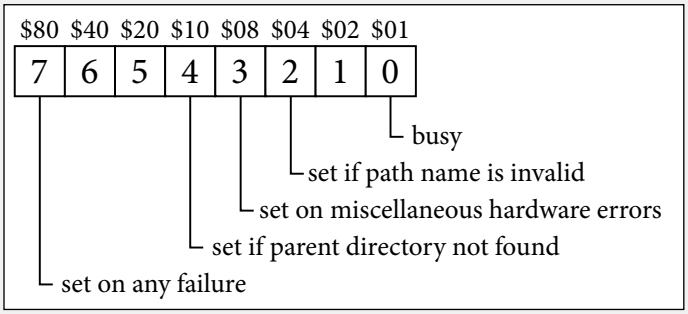
Data Block Sent:

0-1	$4B $3A (K:)
2-255	254 bytes containing the null-terminated path string.

Data Block Returned: None

Response Bytes Returned: None

Status:



Delete File or Directory

Deletes a file or an *empty* directory from the SD card. This command accepts a data block containing an ASCII command identifying the name and location of the target object to delete. The first two characters of the command string are "X:" and are followed by the path name of the file or directory. The string is terminated with a null byte (0). The total size of the string, including "X:" and the null terminator cannot be more than 256 bytes.

When deleting a directory, the command will fail if any files or sub-directories currently reside within the target.

Delete File or Directory

Command Code: $E0

Additional Parameters: None

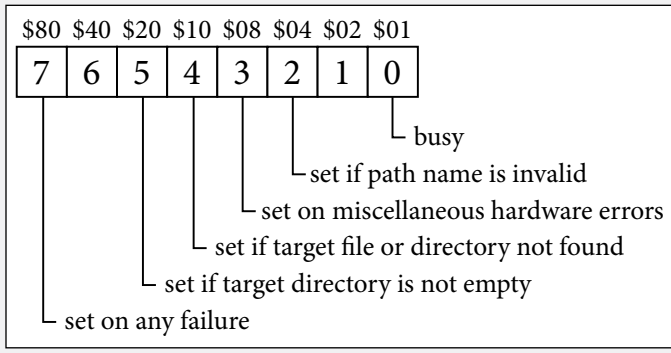
Data Block Sent:

0-1	$58 $3A (X:)
2-255	254 bytes containing the null-terminated path string.

Data Block Returned: None

Response Bytes Returned: None

Status:



Read Logical Sector

Reads a single 256 byte sector from a mounted disk image. This command requires a 24-bit Logical Sector Number for the input parameters and returns the 256 bytes of data from the corresponding sector of the disk image file mounted in virtual drive 0 or 1.

Setting bit 1 of the command code tells the controller that the Logical Sector Number provided assumes the disk image is that of a single-sided floppy disk. That is to say, an LSN of 18 is meant to refer to sector 1 on the first side of track 1 rather than sector 1 on the 2nd sided of track 0. If the mounted disk image is actually that of a double-sided floppy disk, the controller will automatically adjust the LSN to reference the intended sector. That is to say, an LSN of 18 would be internally adjusted to 36 if the disk image is for a double-sided floppy disk.

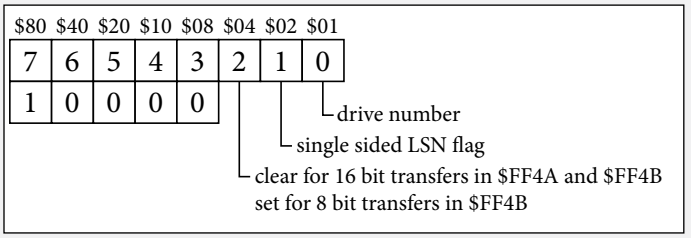
Setting bit 2 of the command code is not supported by the provided subroutine in CommSDC.asm. But if you are writing your own low level I/O subroutine this may be useful on the HD6309. That microprocessor's TFM instruction can only process 1 byte at a time.

If a CRC error occurs in the ID field, then bits 4 and 7 are also set indicating that the sector could not be found. With SDF images, a CRC error occurring in the sector's Data Field will result in status bits 3 and 7 being set, but only after the data has been read.

If the sector uses a Deleted Data Mark then status bit 5 is set immediately and the data is made available for reading. Since this is not actually an error condition, status bit 7 is not set in this circumstance.

Read Logical Sector

Command Code: $8x



Additional Parameters:

$FF49	B	high-order byte of LSN
$FF4A:$FF4B	X	low-order word of LSN

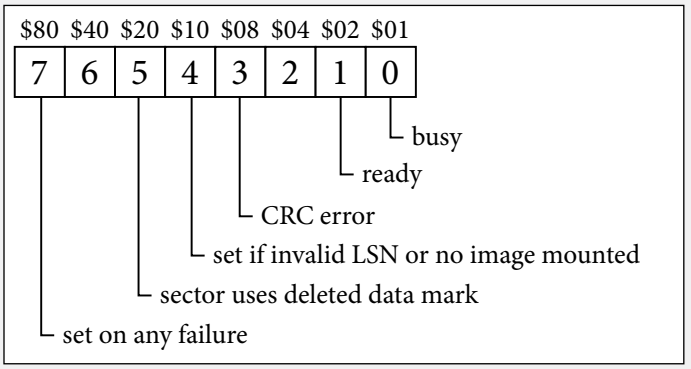
Data Block Sent: None

Data Block Returned:

256 bytes containing the sector data

Response Bytes Returned: None

Status:



Write Logical Sector

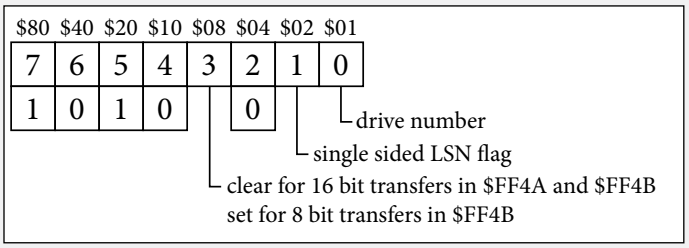
Writes a single 256 byte sector to a mounted disk image. This command accepts a 24-bit Logical Sector Number as the input parameters and the 256 byte data block containing the sector data .

Setting bit 1 of the command code tells the controller that the Logical Sector Number provided assumes the disk image is that of a single-sided floppy disk. That is to say, an LSN of 18 is meant to refer to sector 1 on the first side of track 1 rather than sector 1 on the 2nd sided of track 0. If the mounted disk image is actually that of a double-sided floppy disk, the controller will automatically adjust the LSN to reference the intended sector. That is to say, an LSN of 18 would be internally adjusted to 36 if the disk image is for a double-sided floppy disk.

Setting bit 3 of the command code is not supported by the provided subroutine in CommSDC.asm. But if you are writing your own low level I/O subroutine this may be useful on the HD6309. That microprocessor's TFM instruction can only process 1 byte at a time.

Write Logical Sector

Command Code: $Ax



Additional Parameters:

$FF49	B	high-order byte of LSN
$FF4A:$FF4B	X	low-order word of LSN

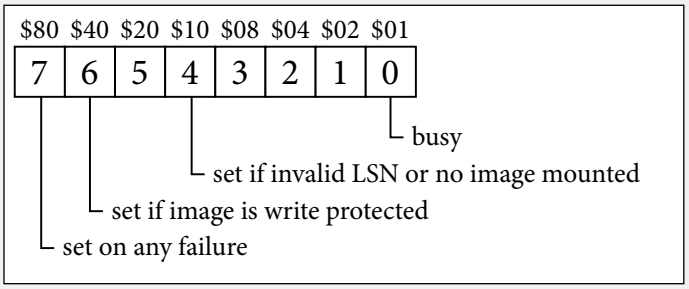
Data Block Sent:

256 bytes containing the sector data

Data Block Returned: None

Response Bytes Returned: None

Status:



Low-Level Stream

There is a special streaming mode that exists within the SDC. First a file has to be mounted in drive 0 or 1. It then persorms a continous read of the file's data in blocks of 512 bytes. This is the SD card's native block size.

The command codes $90 and $91 will let you turn on streaming for either drive 0 or drive 1. When this mode is entered, 512 byte low level sectors will be made available on the data port. One sector after another. The first block can be set by writing the 24 bit block number to the three parameter registers. Polling for the READY status bit is required for the first byte of each 512 byte block.

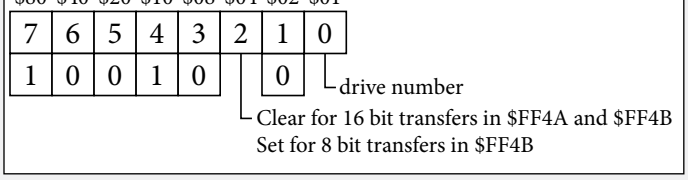
The process will end when the last sector is transferred and the busy bit is cleared.

To stop transferring early you can send the abort stream command: $D0.

The StreamTest.asm file at the end of this chapter to can be used as an example.

Stream

Command Code: $9X



Additional Parameters:

$FF49	High byte of block number
$FF4A	Middle byte of block number
$FF4B	Low byte of block number

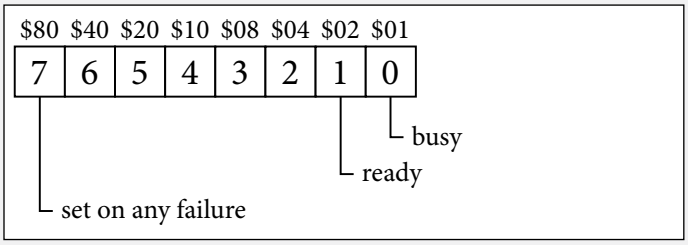
Data Block Sent: None

Data Block Returned:

512 bytes containing the sector data of the SD Card

Response Bytes Returned: None

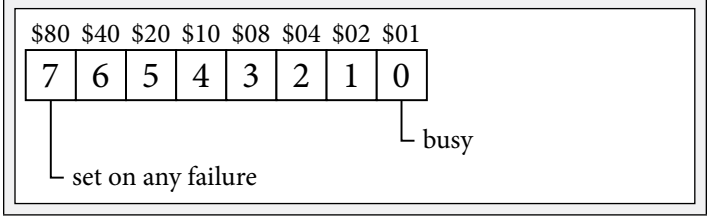
Status:



Abort Stream

Issue this command to abort the stream.

Abort Stream
Command Code: $D0
Additional Parameters: None
Data Block Sent: None
Data Block Returned: None
Response Bytes Returned: None
Status:



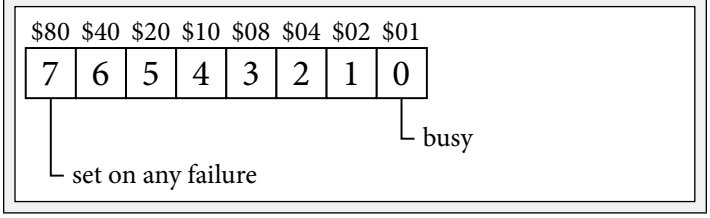
Mount Next Disk In Set

Searches the current Disk Set folder for a file with the next higher numeric suffix ('9' max). If none found, searches for a file with a numeric suffix of '1'. When a file is found, an attempt is made to mount the file as a disk image. On success, the LED is blinked the same number of times as the file's numeric suffix. This command is equivalent to pressing the button on the CoCo-SDC hardware.

Mount Next Disk in Set
Command Code: $C0
Additional Parameters:

$FF49	B	$2B (+)

Data Block Sent: None
Data Block Returned: None
Response Bytes Returned: None
Status:



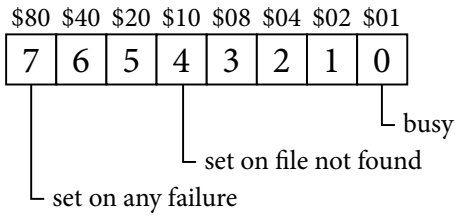
Mount Disk In Set

Searches the current Disk Set folder for a file either with the next higher numeric suffix (FF4A=0) or a specific numeric suffix (FF4A=1..9). When a file is found, an attempt is made to mount the file as a disk image. On success, the LED may optionally blink (FF4B=1) the same number of times as the file's numeric suffix.

Mount Disk in Set
Command Code: $D0
Additional Parameters: None

$FF49	B	$23 (#)
$FF4A	X.H	0:Next Disk, 1-9:Specific Disk
$FF4B	X.L	bit 0: 1 = Blink enable

Data Block Sent: None
Data Block Returned: None
Response Bytes Returned: None
Status:



Version Number

Returns the firmware's version number as a 16-bit BCD value. Versions prior to 113 did not support this command and will fail with status bit 7 set. This is useful for determining if a particular command or feature is available. Refer to the Firmware Version History table at the end of this manual.

Version

Command Code: $C0

Additional Parameters: None

$FF49	B	$56 (V)

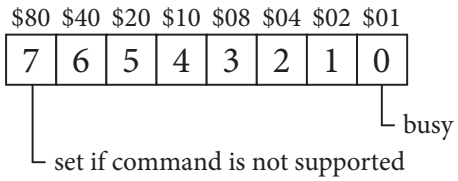
Data Block Sent: None

Data Block Returned: None

Response Bytes Returned:

$FF49	undefined
$FF4A	first two digits of version number (BCD)
$FF4B	last two digits of version number (BCD)

Status:



Low-Level Hardware Interface

This section describes the low-level details for communicating directly with the CoCo SDC hardware. Source code for the CommSDC sub-routine which handles these details has been made available. You should refer to that source code as an example of the information presented here. It is not necessary to understand this information in order to simply use the CommSDC code. It is provided for completeness.

The following steps outline the protocol used to execute a command:

1. Place controller in Command Mode.
2. Wait for the controller to clear the BUSY status bit.
3. Write any required command parameters to the hardware registers.
4. Write the command code to the command register.
5. If required, send 256 bytes of command data or retrieve 256 bytes of response data.
6. Wait for the command to complete by polling the BUSY bit in the status register.
7. Return the controller to FDC Emulation Mode.
8. Examine the status value and read any response bytes returned in the hardware registers.

1. Enabling Command Mode

The CoCo SDC normally operates in FDC Emulation Mode. This makes it appear to the CoCo that a standard floppy disk controller is present. To execute any of the extended commands, the hardware must first be placed in Command Mode. To do this you store the value $43 in the control latch at $FF40. This value would not normally be used with a real floppy controller and so it is used to signal to the controller that it should treat any commands it receives as SDC-specific.

2. Waiting for the Controller to be Not Busy.

After enabling Command Mode the BUSY bit may be set for as long a 2.5 seconds (worst case scenario). You must wait for this bit to clear before issuing a command to the controller.

3. Setup the Command Parameters.

Store any required parameter bytes related to the specific command into the hardware registers at $FF49, $FF4A and $FF4B. These registers can be set in any order and you may use a 16-bit instruction like STD to write to two consecutive register locations.

4. Invoke the Command.

Once the parameter registers have been set up you can write the command code to the command register at $FF48. After writing to the command register you should not access any of the hardware registers for at least 20 microseconds.

5. Perform a Data Block transfer.

Most commands require more data than can be passed in the three parameter registers. These commands accept a 256 byte data block which is sent to the controller two bytes at a time through a pair of data registers at $FF4A and $FF4B. Before sending the first pair of bytes you must poll the status register for the READY or FAILED bits. If the FAILED bit is set then something went wrong and the command was terminated prematurely.

When the READY bit is set you may proceed to write the data. For each pair of bytes, the first must be written to $FF4A before the second is written to $FF4B. You may also use a 16-bit instruction like STD to write both bytes at the same time. It is not necessary to poll for READY on each subsequent pair of bytes.

Commands which do not accept a data block as input may instead provide a data block as a response. In these cases, the READY bit indicates that the data block is available to be read, two bytes at a time through the same pair of data registers ($FF4A and $FF4B).

When the READY bit is set you may proceed to read the data. For each pair of bytes, the first must be read from $FF4A before the second is read from $FF4B. You may also use a 16-bit instruction like LDD to read both bytes at the same time. It is not necessary to poll for READY on each subsequent pair of bytes.

6. Wait for Command Completion.

The controller will require some time to complete execution of the command. You should wait for the BUSY bit in the status register to clear before proceeding. If an error occurs during execution, the FAILED bit will be set in the status register. Other bits in the status register may also be set to indicate a specific type of failure (see the command descriptions).

7. Put Controller back into Emulation Mode.

After execution of each command, you should put the controller back into FDC emulation mode by writing 0 to the control latch at $FF40.

8. Retrieve Status and Response Bytes.

Examine the value in the status register to determine if the command was successful. If the command provides any response bytes you may now read them from the three register locations at $FF49, $FF4A and $FF4B.

```
*******************************************************************
* Filename: CommSDC.asm
* CoCo SDC Low-level interface routine
*
* Hardware Addressing - CoCo Scheme
CTRLATCH        equ       $FF40             controller latch (write)
CMDREG          equ       $FF48             command register (write)
STATREG         equ       $FF48             status register (read)
PREG1           equ       $FF49             param register 1
PREG2           equ       $FF4A             param register 2
PREG3           equ       $FF4B             param register 3
DATREGA         equ       PREG2             first data register
DATREGB         equ       PREG3             second data register

* Status Register Masks
BUSY            equ       %00000001         set while a command is executing
READY           equ       %00000010         set when ready for a data transfer
FAILED          equ       %10000000         set on command failure

* Mode and Command Values
CMDMODE         equ       $43               control latch value to enable command mode
CMDREAD         equ       $80               read logical sector
CMDWRITE        equ       $A0               write logical sector
CMDEX           equ       $C0               extended command
CMDEXD          equ       $E0               extended command with data block


*-------------------------------------------------------------------------
* CommSDC
*
*    This is the core routine used for all
*    transactions with the SDC controller.
*
* Entry:
*    A = Command code
*    B = LSN hi byte   or  First parameter byte
*    X = LSN lo word   or  2nd and third parameter bytes
*    U = Address of 256 byte I/O buffer ($FFFF = none)
*
* Exit:
*    Carry set on error.
*    B = controller status code.
*    A, X, Y and U are preserved.
*
CommSDC         pshs      u,y,x,a,cc        preserve registers
                lsr       ,s                shift carry flag out of saved CC

* Put controller in Command mode
                ldy       #DATREGA          setup Y for hardware addressing
                lda       #CMDMODE          the magic number
                sta       -10,y             send to control latch (FF40)
```

```
* Put input parameters into the hardware registers.
* It does no harm to put random data in the
* registers for commands which do not use them.
               stb       -1,y              high byte to param reg 1
               stx       ,y                low word to param regs 2 and 3

* Wait for Not Busy.
               bsr       waitForIt         run polling loop
               bcs       cmdExit           exit if error or timeout

* Send command to controller
               lda       1,s               get preserved command code from stack
               sta       -2,y              send to command register (FF48)

* Determine if a data block needs to be sent.
* Any command which requires a data block will
* have bit 5 set in the command code.
               bita      #$20              test the "send block" command bit
               beq       rxBlock           branch if no block to send

* Wait for Ready to send
               bsr       waitForIt         run polling loop
               bcs       cmdExit           exit if error or timeout
               leax      ,u                move data address to X

* Send 256 bytes of data
               ldd       #32*256+8         32 chunks of 8 bytes
txChunk        ldu       ,x                send one chunk...
               stu       ,y
               ldu       2,x
               stu       ,y
               ldu       4,x
               stu       ,y
               ldu       6,x
               stu       ,y
               abx                         point X at next chunk
               deca                        decrement chunk counter
               bne       txChunk           loop until all 256 bytes sent

* Wait for command completion
               lda       #5                timeout retries
waitCmplt      bsr       waitForIt         run polling loop
               bitb      #BUSY             test BUSY bit
               beq       cmdExit           exit if completed
               deca                        decrement retry counter
               bne       waitCmplt         repeat until 0
               coma                        set carry for timeout error
               bra       cmdExit           exit

* For commands which return a 256 byte response block the
```

```
* controller will set the READY bit in the Status register
* when it has the data ready for transfer.   For commands
* which do not return a response block the BUSY bit will
* be cleared to indicate that the command has completed.
*
rxBlock         bsr       longWait          run long status polling loop
                bls       cmdExit           exit if error, timeout or completed
                leax      1,u               test the provided buffer address
                beq       cmdExit           exit if "no buffer" ($FFFF)
                leax      ,u                move data address to X

* Read 256 bytes of data
                ldd       #32*256+8         32 chunks of 8 bytes
rxChunk         ldu       ,y                read one chunk...
                stu       ,x
                ldu       ,y
                stu       2,x
                ldu       ,y
                stu       4,x
                ldu       ,y
                stu       6,x
                abx                         update X for next chunk
                deca                        decrement chunk counter
                bne       rxChunk           loop until all 256 bytes transferred
                clrb                        status code for SUCCESS, clear carry

* Exit
cmdExit         rol       ,s                rotate carry into saved CC on stack
                clr       -10,y             end command mode
                puls      cc,a,x,y,u,pc     restore irq masks, update carry and return


*-------------------------------------------------------------------------
* Wait for controller status to indicate either "Not Busy" or "Ready".
* Will time out if neither condition satisfied within a suitable period.
*
* Exit:
*    CC.C set on error or time out.
*    CC.Z set on "Not Busy" status (if carry cleared).
*    B = status
*    X is clobbered.  A, Y and U are preserved.
*
longWait        bsr       waitForIt         enter here for doubled timeout
                bcc       waitRet           return if cleared in 1st pass
waitForIt       ldx       #0                setup timeout counter
waitLp          comb                        set carry for assumed FAIL
                ldb       -2,y              read status
                bmi       waitRet           return if FAILED
                lsrb                        BUSY --> Carry
                bcc       waitDone          branch if not busy
                bitb      #READY/2          test READY (shifted)
```

```
                bne     waitRdy         branch if ready for transfer
                bsr     waitRet         consume some time
                ldb     #$81            status = timeout
                leax    ,-x             decrement timeout counter
                beq     waitRet         return if timed out
                bra     waitLp          try again

waitDone        clrb                    Not Busy: status = 0, set Z
waitRdy         rolb                    On Ready: clear C and Z
waitRet         rts                     return


*-------------------------------------------------------------------------

                END

*****************************************************************
*  Filename: StreamTest.asm
*  Test routine for the SDC Continuous Stream.
*


* Hardware Addressing - CoCo Scheme
CTRLATCH    equ         $FF40               controller latch (write)
CMDREG      equ         $FF48               command register (write)
STATREG     equ         $FF48               status register (read)
PREG1       equ         $FF49               param register 1
PREG2       equ         $FF4A               param register 2
PREG3       equ         $FF4B               param register 3
DATREGA     equ         PREG2               first data register
DATREGB     equ         PREG3               second data register


* Status Register Masks
BUSY        equ         %00000001           set while a command is executing
READY       equ         %00000010           set when ready for a data transfer
FAILED      equ         %10000000           set on command failure


* Mode and Command Values
CMDMODE     equ         $43                 command mode setting for control latch
CMDSTREAM   equ         $90                 continuous read of 512 byte blocks
CMDABORT    equ         $D0                 abort I/O command


            org         $4000

StreamSDC   ldy         #DATREGA            setup Y for hardware addressing
            lda         #CMDMODE            the magic number
            sta         CTRLATCH            send to control latch (FF40)
```

```
            lda         #BUSY                   status mask
mPoll       bita        -2,y                    poll BUSY flag
            bne         mPoll                   loop while controller is busy

* Set starting block number to 0.
            clr         -1,y                    high param
            clr          0,y                    mid param
            clr          1,y                    low param

* Send command to the controller
            lda         #CMDSTREAM+1            stream from drive 1 using classic method
            sta         CMDREG                  send to command register

* Prepare for BREAK key tests
            ldb         #$FB                    strobe the keyboard column..
            stb         $FF02                   ..which contains the BREAK key

* Loop to read the file blocks
blockLoop   ldx         #$0400+16               buffer address + 16
bPoll       ldb         -2,y                    poll status
            asrb                                BUSY --> carry
            bcc         streamDone              exit if BUSY cleared
            beq         bPoll                   continue polling if not READY

*
* Partially unrolled transfer loop (32 bytes per iteration).
* Displacements of -16 to +14 utilize full range of 5 bit indexing.
*
            ldd         #16*256+32              A = chunk count, B = bytes per chunk
chunkLoop   ldu         ,y
            stu         -16,x
            ldu         ,y
            stu         -14,x
            ldu         ,y
            stu         -12,x
            ldu         ,y
            stu         -10,x
            ldu         ,y
            stu         -8,x
            ldu         ,y
            stu         -6,x
            ldu         ,y
            stu         -4,x
            ldu         ,y
            stu         -2,x
            ldu         ,y
            stu         ,x
            ldu         ,y
            stu         2,x
            ldu         ,y
            stu         4,x
```

```
            ldu         ,y
            stu         6,x
            ldu         ,y
            stu         8,x
            ldu         ,y
            stu         10,x
            ldu         ,y
            stu         12,x
            ldu         ,y
            stu         14,x
            abx                             update buffer address for next chunk
            deca                            decrement chunk counter
            bne         chunkLoop           loop until all chunks transferred

* Test for BREAK key to abort
            ldb         $FF00               get keyboard state
            bitb        #$40                test row with the BREAK key
            bne         blockLoop           loop if BREAK not pressed
            ldb         #CMDABORT           send abort I/O command..
            stb         CMDREG              ..to the controller
            asrb                            will use CMDABORT for the status result

* Exit
streamDone  clr         CTRLATCH            put controller back in floppy mode
            aslb                            save controller status in..
            stb         $00F0               ..the DSKCON status variable
            rts
            end
```

Glossary

ACIA	Asynchronous Communications Interface Adapter
AVR	Alf and Vegard's RISC
BASIC	Beginners All-purpose Symbolic Instruction Code
CoCo	Color Computer
DIP	Dual In-line Package switches and chips
DOS	Disk Operating System
DriveWire	A tool to network a CoCo with an external computer over serial or emulation ports
DRQ	Data Request
EPROM	Electronically Programmable Read Only Memory
IDE	Integrated Drive Electronics
FAT	File Allocation Table
FDC	Floppy Disk Controller
FD	Floppy Disk
LBA	Logical Block Addressing
LED	Light Emitting Diode
MicroSD	Micro Secure Digital card
MPI	Multi-Pak Interface
PC	IBM Personal Computer
PCB	Printed Circuit Board
RISC	Reduced Instruction Set Computing
SCSI	Small Computer Systems Interface
SDC	Secure Digital Card Controller
SDHC	Secure Digital High Capacity
SD	Secure Digital Card
SDXC	Secure Digital Extended Capacity

Firmware Version History

113	Added the VERSION command.
115	Added the DELETE, RENAME and QUERY DISK SIZE commands as well as the ability to recognize Dragon VDK images (but only with a 12-byte header).
116	Fixed the recognition of Dragon VDK headers to include headers up to 256 bytes in length.
117	Added the STREAM command.
120	Added the MOUNT NEXT DISK command. It also fixed the QUERY DISK SIZE command. Prior versions only returned a 16-bit value rather than a 24-bit value (FF49 was always returning 0).
124	Introduced the MOUNT DISK # command and the 8-bit option for WRITE SECTOR.

Index