Jef Raskin Information

SwyftCard Manual:

Theory of Operation

DOCUMENT NO. 25

COMPILED BY
DAVID T CRAIG
736 EDGEWATER, WICHITA, KANSAS 67230
U.S.A.

This Appendix is divided into three sections: interface, software, and hardware. The sections are not system documentation, but guides to the spirit of design in each area.

SwyftCard User Interface Theory of Operation

The paradigms used in SwyftCard were invented to cure a host of problems shared by almost all current systems, most of them small enough in their own right, but which taken together make learning and using conventional software far more time consuming than necessary: and make using computers a frustrating and annoying process.

We have always wondered why, for example, you have to format disks — isn't the computer smart enough to see if a disk isn't formatted and do it if necessary? We find cursor control keys far too slow, and when you consider the number of auxiliary commands they require (move to next/previous word, sentence, paragraph, page, move to beginning/end of line, document, file...) we find them too complex. The mouse, we find, is small improvment since it takes your hands away from the keyboard, and uses up much screen space for menus, scroll bars, and the rest of the associated mouse apparatus. We are annoyed when we are put through menus instead of being able to do what we want right now, and we are puzzled by the huge number of commands in most systems. We hate disk systems that allow you to lose work through trivial human error. We are amazed that many word processors can't keep up with human typing speed.

SwyftCard shows that, with proper design, all these questions and bothers, and many others that have plagued us for years, can be answered and fixed. And it works on an inexpensive computer with only one disk drive, with minimum memory requirements. Our product does what most people need, without operating system, expensive price tag, or bells and whistles.

The major design principles include a few innovations, and many examples of applying what we have learned from the work of others.

1. The cursor LEAP concept, whose average time to target is about one third that of the most advanced method in common use up to now: the mouse.

SwyftCard User's Manual — 1985

2. The cursor design itself, which shows both where what you type will appear and where delete will operate. The cursor also collapses upon being moved so that you do not have to aim "one off" if you want to delete.

3. Finding a very small set of fundamental operations that allow you to easily accomplish a wide range of tasks.

4. Eliminating the operating system and allowing all operations to be performed directly and immediately from the editor without having to go into different modes.

5. The elimination of modes in general, which makes habit formation easy since you do not have to think about what state the system is in to figure out what a command will do. This property is called "modelessness."

6. *Not* providing many ways to do a task — again so that you do not have to think about alternate strategies when you are about to do something. This, too, aids in habit formation. We call this principle "monotony."

7. The emphasis on habit formation is itself a fundamental principle of the design, and one often overlooked by others. We consider it important that after a brief period of learning, a user should not have to think about the system while using it.

8. The DISK command, which simplifies the usual complexities of a DOS (Disk Operating System) into a simple command. It also provides protection against most common mistakes that on other systems would cause loss of data. This kind of command is possible due to the technique of making one disk correspond to one text.

9. The emphasis on speed of operation proportional to frequency of use (often done tasks must be very fast, seldom done tasks can be slow).

10. What you see is what you get — the way it looks on-screen is the way it prints on paper. (This was violated for underlining due to a l imitation in the Apple display hardware.)

11. Noun-verb design of commands. You always specify what you are going to work on first (which gives you time to make sure you are right and to make corrections), and *then* you give the order as to what to do. Some systems work the other way around, or — what is even worse — mix the two styles.

12. It is very hard to louse yourself up or clobber something you are working on. Not impossible, but hard enough to do that it is not likely to happen by chance.

13. The inclusion of programming and communications within a general purpose environment, where the output is placed in the editor/retrieval environment.

14. The allowance of months of testing and re-working time in the schedule, so that purchasers of the system are not being used as test subjects.

This is only a barest sketch — the system specs run to some 50 pages — but we hope it gives you a feel for what led us to design SwyftCard the way we did.

SwyftCard Software Theory of Operation

The system is small and operates quickly partly because it is implemented in FORTH and assembly language, and partly because it has an inherently clean and simple design. There are few commands, and they operate uniformly. Text is not cluttered with special markers. All of this minimizes programming effort.

The structure of the text, although not unique to SwyftCard, is key. The beginning of the text (from the first character up to the last character of the highlight) is stored in the lower portion of the text area, and the end of the text (from the character at the cursor or one after the cursor to the last character of the text) is stored in the upper end of the text area. Between the upper and lower texts lies the gap. This means that typing just puts characters into the gap, and thus can proceed very rapidly. Updating the screen is the only task that need be done while typing is going on, which is one reason why it is possible for SwyftCard to do word wrap and unwrap on the fly. The other reason is the uniformity of text.

Searching is fast because text lies entirely in RAM, contains no codes or other obstructions, and is in only two contiguous areas. The disk operates quickly because we do not use Apple's encoding scheme, and because we only write or read as much text as necessary. As a consequence of this decision, a ProDos conversion routine is provided to establish a link to other Apple software. Formatting is done on the fly, since we write a whole track at a time, including sync bits. Not only is this fast, but it eliminates the need for a separate formatting step on the part of the user.

The system pointers are stored on disk so that SwyftCard texts come up in the exact state they were last saved. A serial number is written on each disk so that we can detect whether it is the same disk that was booted or if the user has changed disks. When backup disks are made, the same serial number is written on the master and all backups.

Updating the display after a leap is sped up by having a table of pointers to the places where pages begin. Thus, in order to figure out how the text should be formatted, the display algorithm has only to go back to the nearest page break prior to the text that is to be displayed at the beginning of the screen.

When inserting text, large areas of memory may have to be moved bodily. This "brute force" approach is surprisingly fast, but with very large texts does lead to a perceptible slowing. Still, SwyftCard is much faster than any other program that does a similar task on the Apple.

A deletion puts the deleted text at the beginning of the text area and moves the lower part of text up out of the way. This means that there is no limit to the amount of text that can be deleted, since we do not have to set aside room for a separate delete buffer.

Decisions of this sort abound, ultimately leaving an unusually large amount of space for the user — in this case over 80% of the memory that Apple does not dedicate to specific uses. SwyftCard does not use the extra memory afforded by the extended 80-column card, as the bank switching required would make the program operate too slowly for the high-quality interaction we think important.

The CALC, PRINT, and SEND command are all fundamentally the same. They take the highlighted chunk of text and transmit it to the BASIC interpreter, the printer port, and the serial port respectively.

Of these, dealing with the BASIC interpreter is the most difficult, since SwyftCard operates in the same address space as BASIC, and because of Applesoft's documentation.

The keyboard tables are in RAM so that software developers can redirect commands to execute code that they provide. BASIC programs that amplify SwyftCard's abilities can also be written. By changing the value of one word (BT%), the bottom of text can be moved up so that developers can have room for their own code.

SwyftCard's software amounts to less than 16 Kbytes. Approximately half is in a tokenized FORTH, and half in assembly code.

SwyftCard Hardware Theory of Operation

The SwyftCard is a plug-in card for the Apple //e that operates in Slot 3. The card contains three integrated circuits which provide a power-on reset circuit, storage for the SwyftCard program, and control signals for the card. The card operates by asserting the Apple //e bus signal INH' which disables the built-in ROM and enables the SwyftCard ROM. This permits the SwyftCard program to take over the system at power-on and run the SwyftCard program. (Please refer to the schematic.)

The LM311 voltage comparator is connected to provide the power-on reset function. When the Apple //e is first turned on, the power-on reset circuit resets the PAL, turning on the SwyftCard and disabling the Apple //e internal ROM. The power-on reset circuit must be provided because the existing Apple //e reset function is used by many Apple //e programs for a "warm start": if Apple //e reset always started the SwyftCard, other programs could not use the "warm start."

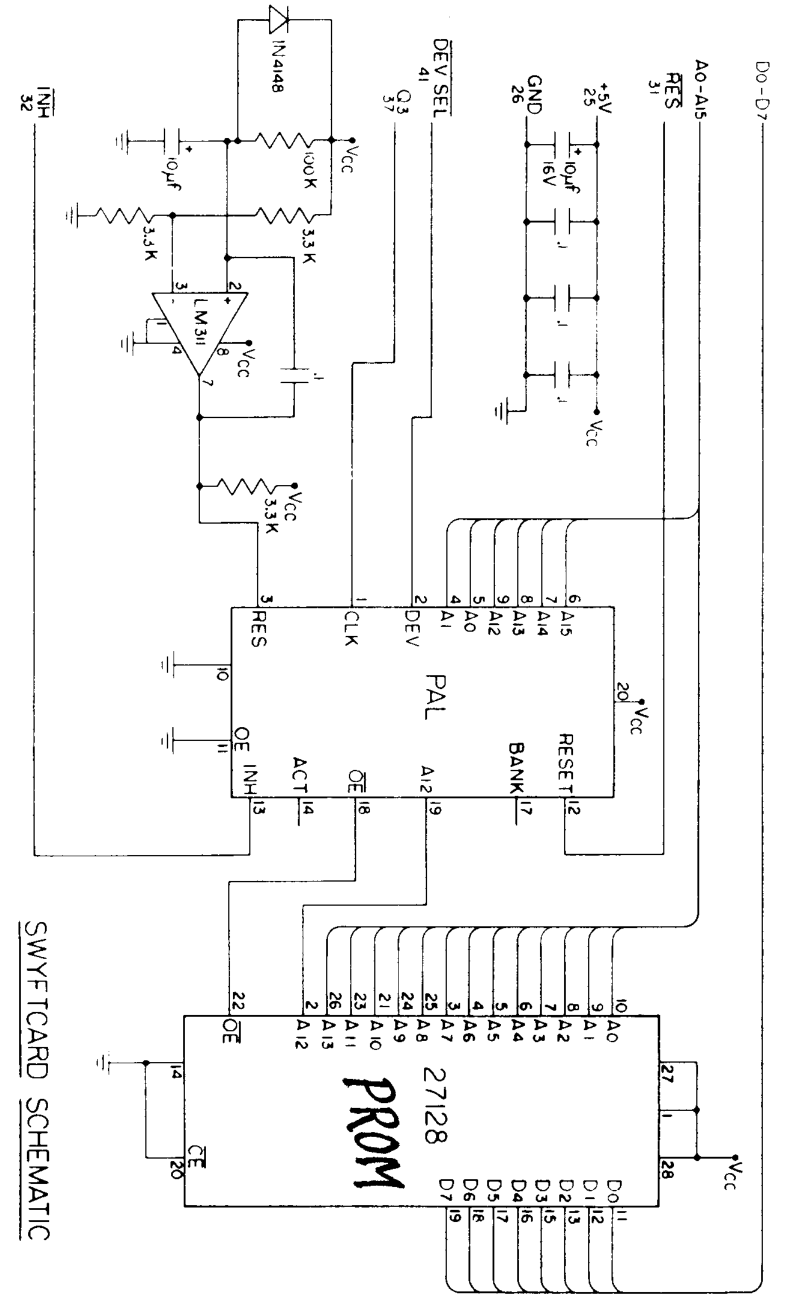
The 27128 PROM is used to store the SwyftCard program. The PROM contains 16384 bytes which are mapped into the address space $D000 - $FFFF. Since the address space is only 12 Kbytes, there are two 4 Kbyte sections of the PROM mapped into the address space $D000-$DFFF.

The card is controlled by the PAL. When the SwyftCard is active, the PAL asserts the INH' signal, enables the PROM, and bank switches the $D000-$DFFF address space. The card is controlled by two soft switches. The soft switches are controlled by accessing the following memory locations with either a read or a write operation.

$C0B0 - SwyftCard active, Bank 1
$C0B1 - SwyftCard inactive, Bank 1
$C0B2 - SwyftCard active, Bank 2

When the power-on reset circuit asserts the RES signal on Pin 3 of the PAL, the SwyftCard is made active in Bank 1. Accessing location $C0B1 deactivates the SwyftCard for normal Apple //e operation.

The INH' line is driven by a tri-state driver, so if another card in the Apple //e asserts the INH' signal there will not be a bus contention. However, there will be a bus contention on the data bus if another card attempts to control the bus while the SwyftCard is active.



SWYFTCARD SCHEMATIC