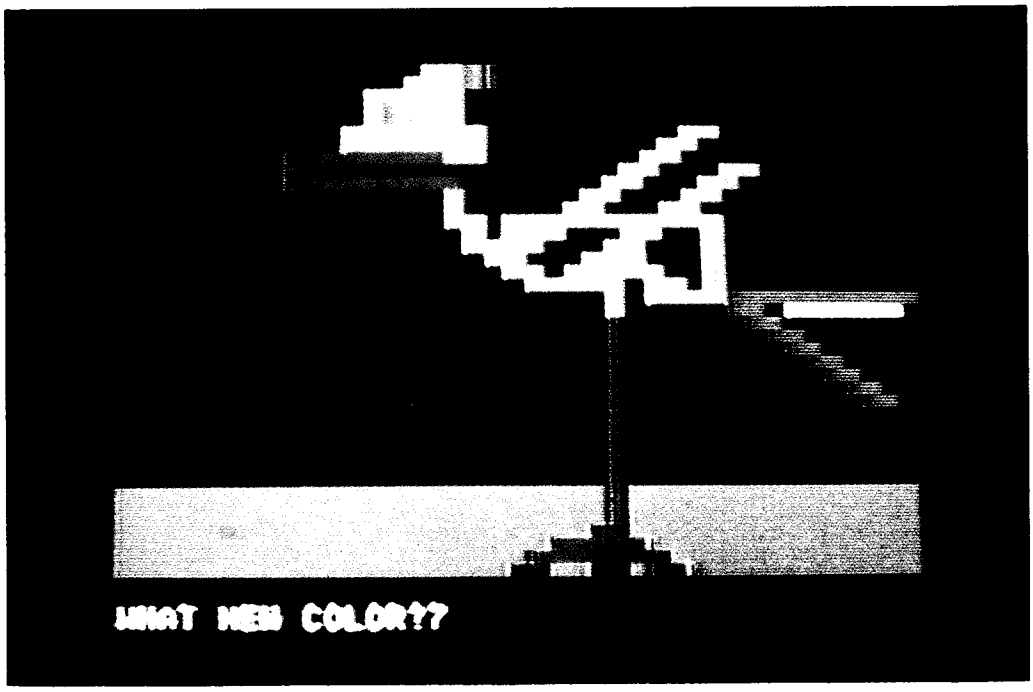


Photo 1: A concoction of color resembling a byrd. This color doodle was produced using a homebrew joystick plugged into the Apple II's game IO connector, and a BASIC program to implement interactive drawing on the television screen. The photograph was made using 1/15th second exposure and ASA 200 Kodachrome slide film in a 35mm camera with macro lens mounted on a tripod. All the color photographs in this article were taken using this setup.

An Apple to Byte

Carl Helmers, Editor

It has been a little over a year since I first became aware of the prototype of the Apple II computer on a visit to Palo Alto CA in November of 1976. At the time I first viewed the Apple II prototype, it was little more than a wire wrapped proof of concept in a homebrew masonite box. In the year since my first exposure to the machine Apple II has become what I consider to be one of the best examples of the concept of the complete "appliance" computer. This variety of computer is sold as a finished product off the shelves of the retailer's shop or by mail from the manufacturer's warehouse. In late October of 1977, I took delivery on an Apple II with 16 K bytes of programmable user memory. After removing it from its shipping box, I connected it to a color television with the cables and radio frequency modulator supplied, and also connected it to an inexpensive tape recorder with an "index" counter to keep track of position. I was able to turn on power and begin using the computer within five minutes of receipt. After one session in my basement laboratory wiring up some joystick hardware using Apple II's documentation as a guide, and after about three evenings of hacking with the built-in ROM BASIC interpreter, I was able to produce a program for a color sketchpad to provide an illustration of some of the potentials of such small computers for use in artistic contexts. While I treat the Apple II as one of the neatest "proofs of concept" of the idea of the personal computer yet to become available, it is by no means the only one on the market, so readers should judge for themselves with respect to their own values and preferences.

The potential for producing graphics like photo 1 illustrates why I became intoxicated with the Apple II concept from the first word of its existence. With a personal interest in the uses of computers for artistic

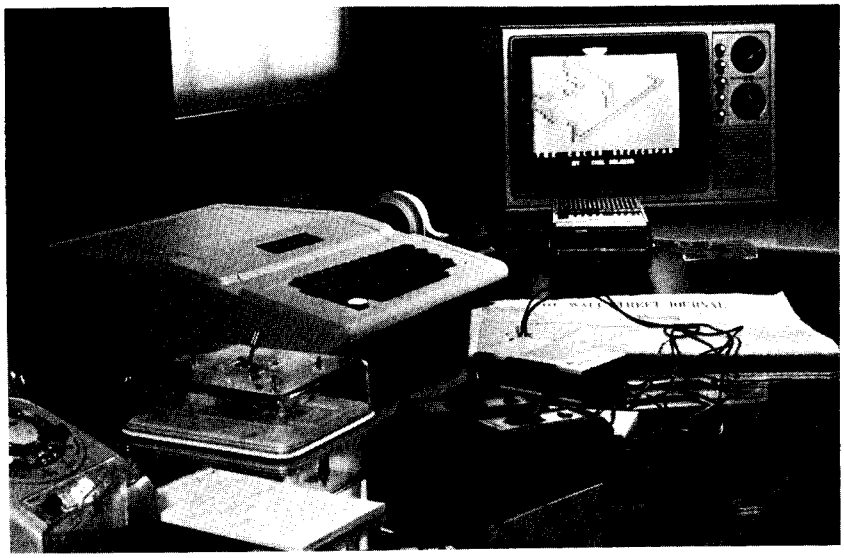


Photo 2: The Apple II in a typical use setting. The television is fairly far away from the computer itself in order to minimize interference and hash generated by the logic circuitry. An inexpensive cassette recorder with turns counter is used to provide mass storage for programs.

purposes, I knew that in principle I should be able to create something like photo 1. Later in this account, I will present the detail design of software which will enable the Apple II user to doodle as I did. I make no claims to great artistic genius, but the ability to do this sort of doodling, as well as much more serious computing in a utilitarian mode of operation, is what the personal computer is all about. Apple II is a self-contained package which talks to a color television set owned by its user, an audio cassette recorder supplied by the user, two control paddles supplied by the manufacturer, and a "kluge harp" style audible annunciator which can be programmed to play music or make a variety of noises. Much of the detail of the system design of the Apple II has already been presented in these pages in the form of an article by its designer, Stephen Wozniak, which appeared in the May 1977 BYTE on page 34. This article will survey the reactions I have had to using this system in the four weeks or so following its arrival.

Externals

The physical appearance of the Apple II in its normal usage context is illustrated in photo 2. The package in which the system is contained is a high strength injection molded plastic material with a metal bottom plate. The plastic parts are painted an off-white ivory color. The three piece case consists of a body with a door on the top for access to the internal works and peripheral sockets. The case is almost empty when the Apple II arrives, and when carrying the computer around in its optional leatherette bag my common practice has been to take the cover off and insert various paraphernalia cushioned by foam plastic to avoid damaging the circuit boards. (If I had any peripherals plugged into the IO bus of the machine, this would not necessarily be a good idea.) When zipped up in its carrying bag, the Apple II looks like an overgrown pocket calculator. I have taken this computer in its case to friends' homes and with me on trips by airplane (where it fits under the seat as "carry on" luggage). If the destination of one's travels has a color television, and a miniature cassette recorder is packed with the Apple II, then this computer can be considered to be truly portable and adds but one bag to the normal complement of travel luggage.

While the case is elegantly styled, as can be seen from photo 2, there are two minor problems related to the mechanical design of this case. The problems relate to the top of the case and how it fits into the main body of the computer. One problem is the fact that the adhesion between the paint of the case and the plastic is not strong enough to keep the fasteners in place. After I opened the top a few times both fasteners broke loose. The second mechanical design glitch is the fact that if the cover is moved more than about 5 or 10° from horizontal before it is slid out from under the front edge of the case, a leverage effect will tend to extract the keyboard from its moorings.

A usable configuration of the Apple II as emphasized in photo 2 is made up of the computer, a color television, and a cassette recorder. The standard game paddles which come with the system allow interactive graphic applications. The use of a color television is highly recommended, although a black and white set will certainly work, at the expense of one of the unique features of Apple II, its color display.

There are two methods of sending video data to the television set from the Apple. The best option, which is often used by computer stores to show off the system, is use of direct video. However, stock color televisions or color monitors with direct video entry are rare and expensive. A less satisfactory but quite workable method is the use of a radio frequency modulator to generate a television "station" on channel 3, with connection via the antenna terminals and an FCC approved antenna isolation switch which mounts on the back of the set. This method of driving the television is the

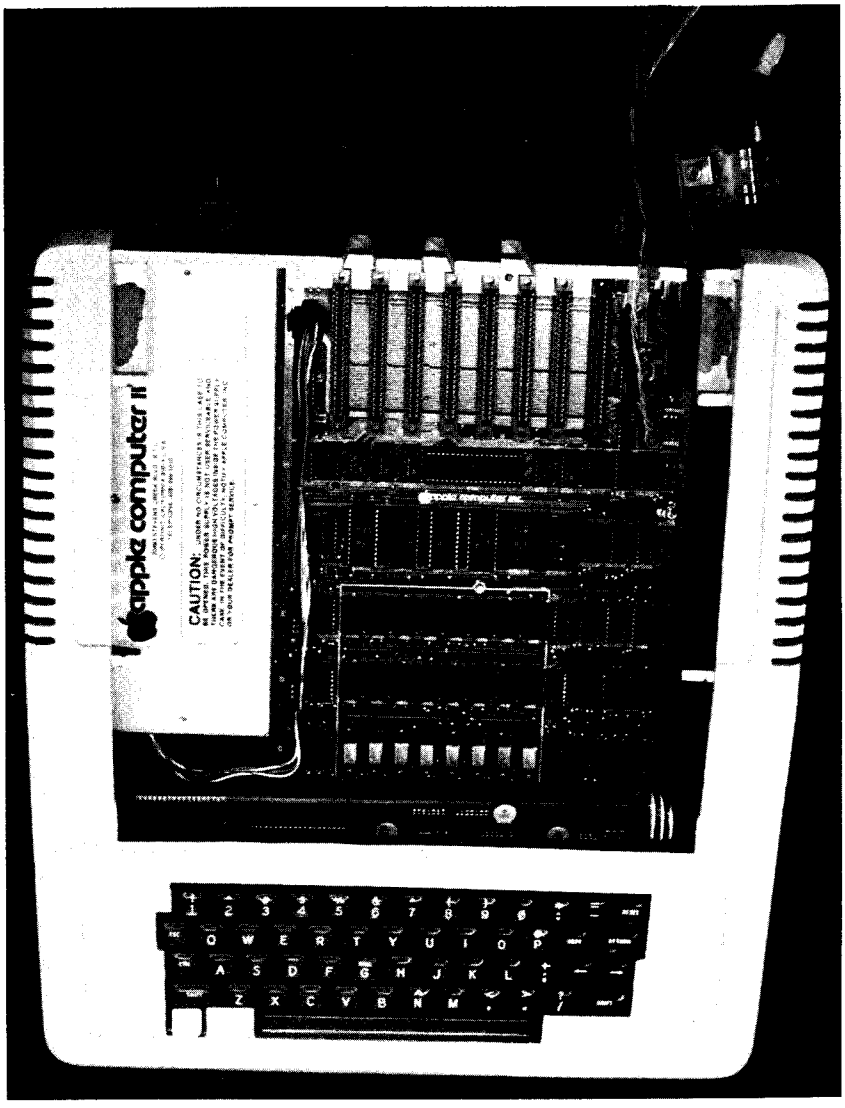


Photo 3: Removing the cover and looking straight down into the unit, the major internal subassemblies can be seen. At the left is the proprietary switching power supply unit; at the right along the bottom of the cabinet is the main processor board, with room for 48 K bytes of programmable user memory (using 16 K memory parts) and an 8 slot IO bus used to interconnect peripherals. At the right toward the rear of the cabinet (top of photo) is the RF modulator used to drive the television on channel 3. The keyboard can be seen toward the lower edge of the photo.

one used in all the photographs and listings accompanying this article.

The main problem with direct RF entry as a method is the tendency of the television set's tuner to pick up RF hash from the digital logic of the computer, a problem which was quite severe in my experience, using a Panasonic color television. The degree of interference is dependent upon the exact positioning of the cable, the television and the computer relative to its various power and peripheral cables. The problem can be minimized, as shown by the fact that in the listings and photos of this article I was able to get a clear picture; but I do not recommend use of a Panasonic television like mine with the Apple II. (Steve Jobs of Apple Computer suggests use of several models of Sony television sets due to superior RF shielding relative to the Panasonic set I used.) If you purchase an Apple II and a color television for it, I highly recommend testing the TV and computer combination prior to settling on a particular television. Some computer stores will convert TV sets to direct video entry, so this provides another option.

Internals

Photo 3 shows the Apple II's interior with the cover removed from the case. The computer is entirely contained on one large printed circuit board on the bottom of the case. At the left is a large metal box containing the switching regulated power supply. The printed circuit board of the keyboard assembly can be seen protruding past the lip at the front edge of the case (bottom of the photograph). Also seen in this picture is a homemade cable running from the game IO connector of the processor board to the joystick box which I built for use with the color sketchpad program to be described. At the top right in this photograph can be seen the RF modulator unit which plugs directly into the main board of the computer. Note that the RF cables contain large toroidal ferrite coil forms with several turns of the cables around them. These are used to minimize (but in my experience with the Panasonic television never totally eliminate) RF hash interference in the television set when the RF modulator technique of video data entry is used.

The Apple II provides a decoded set of eight IO sockets which also feature all the processor bus signals. At the time the Apple II was delivered to me no peripherals were available which used this bus, but I have since seen several advertisements for products to plug into the Apple II bus and the Apple Computer Company is working on peripherals to extend the power of the machine. This bus is completely documented and should work out well for the advanced experimenter.

The Apple hardware includes two fairly gamey peripherals for use in family entertainment situations. One such peripheral is a pair of game IO paddles consisting of an analog input lever and pushbutton switch for each channel. A second such peripheral is the annunciator output which in addition to sounding the "bell" character of ASCII can be programmed by the user with arbitrary pitches, as a sort of music synthesizer.

The Processor

As documented by its designer Stephen Wozniak in the May 1977 BYTE, the design of the Apple II system uses the 6502 processor created by MOS Technology and now available from several sources. The design has a certain elegance which comes from a simple combination of the video display generation, dynamic memory refresh and processor clock timing based on a single crystal oscillator. I'll not repeat the details here, but simply summarize: the two phase nature of the 6502 clock is such that the processor turns itself off with respect to the outside world during one phase, and accesses memory during the other phase. By using the phase unused by the processor for access of memory by the video display generation logic, there is never any conflict between the display and the processor's access of memory. As a side effect, since the display generator is always cycling through the low order address bits of the dynamic memory content of the machine, the dynamic memory refresh requirements are met by this regular access of memory for display purposes.

The memory address space of the Apple II is partitioned into three major segments. The region from addresses hexadecimal 0 to BFFF (48 K bytes) is reserved for programmable user memory, implemented with dynamic memory parts. The region from D000 hexadecimal to FFFF hexadecimal is reserved for systems software in read only memory, and IO ports are found in the C000 to CFFF region.

The user memory region can have any combination of three 4 K or 16 K byte regions depending upon which memory chips one plugs into three sets of eight sockets. Thus the Apple II can be had with 4 K, 8 K, 12 K, 16 K, 20 K, 24 K, 32 K, 36 K or 48 K bytes of memory at the user's option. For full use of the capabilities of the machine I would not recommend purchasing less than 16 K bytes of memory.

The read only memory regions cover a total of 12 K bytes in the address space, starting at D000 and extending through FFFF. In the versions of Apple II currently being delivered, four 2 K byte read only memory parts are plugged into addresses E000 through FFFF, giving a total of 8 K bytes of systems software and leaving two 2 K byte sockets unused. The present ROM load includes 5 K for the integer BASIC interpreter, 1 K for miscellaneous utility routines, and 2 K for the system monitor program.

Systems Software

As with all the self contained "complete" computer systems, Apple II is ready and willing to act as a personal computer servant as soon as the power is turned on and the "reset" button on the keyboard is pushed. This capability for instant use is achieved by the systems software contained

Continued on page 30

Table 1: Summary of the Apple II 5 K BASIC interpreter.

Variables: Names may be from 1 to 100 characters in length. Data type is numeric unless name is followed by the character "$".

Numeric Variables: May be arrayed up to the limits of available memory with DIM statement. Precision is 16 bits, signed two's complement representation with range −32768 to +32767.

String Variables: DIM statement sets length other than default, from 1 to 255 characters per string.

Speed: Executes the loop 100 FOR I=1 TO 10000,110 NEXT I in about 14 seconds.

Statements:

CALL
DIM
DSP
END
FOR ... = ... TO ... STEP ...
GOSUB
GOTO
IF ... THEN ...
INPUT
IN#
LET ... = (LET optional)
LIST
NEXT
NO DSP
NO TRACE
POKE
POP
PRINT
PR#
REM
RETURN
TAB
TEXT
TRACE
VTAB

Graphic Statements:

COLOR=
GR
HLIN
PLOT
VLIN

Operators:

\+
−
NOT
↑ (exponentiate)
*
/
MOD
=
or < >
<
>
<=
>=
AND
OR

Functions:

ABS
ASC
LEN
PDL
PEEK
RND
SCRN
SGN

Other features:

Branch addresses in GOSUB and GOTO and CALL can be any arbitrary numeric expression which results in a positive value. Legal line numbers are the positive integers 1 to 32767.

Documentation includes several PEEK and POKE strategies to access hardware such as speaker, paddle pushbutton switch inputs, etc.

Other Features: Interpreter Control

AUTO
CLR
CON (or control C)
DEL
DSP
HIMEM
GOTO
GR
LIST
LOAD
LOMEM
MAN
NEW
NO DSP
NO TRACE
RUN
SAVE
TEXT
TRACE

Continued from page 24

in the 8 K byte read only memory. On reset, the system monitor program is entered, with an asterisk (*) returned to the video display as a prompting character. The functions available from this program include hexadecimal manipulations of memory contents: displaying memory, changing memory contents, moving blocks from place to place in memory, comparing blocks in memory, reading or writing memory blocks to tape. At a slightly higher level, there is a "mini-assembler" which does operation code lookup and branch address calculation, and a disassembler which inverts the operation of the assembler. The monitor also includes provisions for machine language tracing of programs, single step execution of machine language programs, and hexadecimal arithmetic of addition and subtraction. This software provides the basis for effective low level use of the 6502 processor, and in fact was used by Apple II's designer Stephen Wozniak as one of the key software development tools in implementation of the 5 K Apple BASIC interpreter which makes up the remainder of the 8 K software in read only memory.

The 5 K Apple BASIC interpreter is entered from the system monitor through use of a "control B" command followed by a carriage return. This sequence results in the 5 K BASIC interpreter's prompt of an angle bracket (>). The 5 K interpreter built into the Apple II is an integer BASIC with 16 bit precision and a signed two's complement number representation. Table 1 summarizes the characteristics of this BASIC. Built into the language are a number of extensions which are used to control special hardware and the graphics of the Apple II color display. These extensions include the commands GR, TEXT, PDL (read a control paddle), SCRN (extract the current color of a point on the screen), COLOR=, PLOT, VLIN (draw vertical line) and HLIN (draw horizontal line). Using this 5 K BASIC I was able to implement the color sketchpad program shown in the listings of this article, in about three evenings of experimentation which mostly concerned defining just what the program must do.

The 5 K BASIC interpreter which is built into the Apple is all that is needed for implementation of most types of interactive games involving color graphics and reaction times on the part of the user. But the 5 K BASIC, even given its string capabilities, is not what one would want to use to do a simulation of a physical system or calculate quantities other than integers.

As an answer to the need for an extended BASIC as a language for the Apple II, there is the "Applesoft" extended BASIC interpreter which can be used in systems with 16 K bytes of memory or more. The name "Applesoft" is a cross between the source of the interpreter, the Microsoft company, and a gross pun ("applesauce"). This interpreter is nearly identical to the Microsoft extended BASIC interpreters which have been made available for a number of personal computer products. The people at Apple have hacked the interpreter to include a few variations on the standard version which address the color display hardware. The only relative novelty of this hack is that in order to get graphics extensions, they had to sacrifice two statements: LET and REM. Thus, on initialization of Applesoft, the user is given the option of having LET and REM but no built-in graphics primitives, or having graphics but no LET and REM statements. Since LET is totally optional in assignments, its loss is hardly felt; but the lack of remarks may be felt by self-documenting code purists who want to use the graphics mode of Applesoft. Of course not having the primitives does not prevent use of the graphics hardware, since like all Microsoft interpreters, Applesoft has PEEK and POKE

Table 2: Summary of Apple II features.

Processor: 6502 running at 1.023 MHz

User Memory Capacity: Three banks of eight 4 K or 16 K dynamic memory parts
4 K, 8 K, 12 K, 16 K, 20 K, 24 K, 32 K, 36 K or 48 K bytes

Read Only Memory Capacity:

12 K bytes using 2 K by 8 bit ROM parts, 8 K installed

Standard Peripherals:

Two game paddles (one switch, one variable analog input per paddle)
Programmable annunciator
ASCII keyboard
Audio tape mass storage (approximately 1500 bps)
NTSC color video generation for primary display

Optional:

RF modulator for video coupling to standard television

Expansion Capabilities:

Eight IO connectors with 50 pins:
Full address bus (16 pins)
Full data bus (8 pins)
Timing signals
DMA signals
Device select signals

Software:

System monitor (2 K bytes, ROM)
Utility routines (1 K bytes, ROM)
Integer BASIC interpreter (5 K bytes, ROM)
Full extended BASIC (Applesoft, loaded from tape, requires 16 K user memory)
Applications software examples including games, accounting, etc

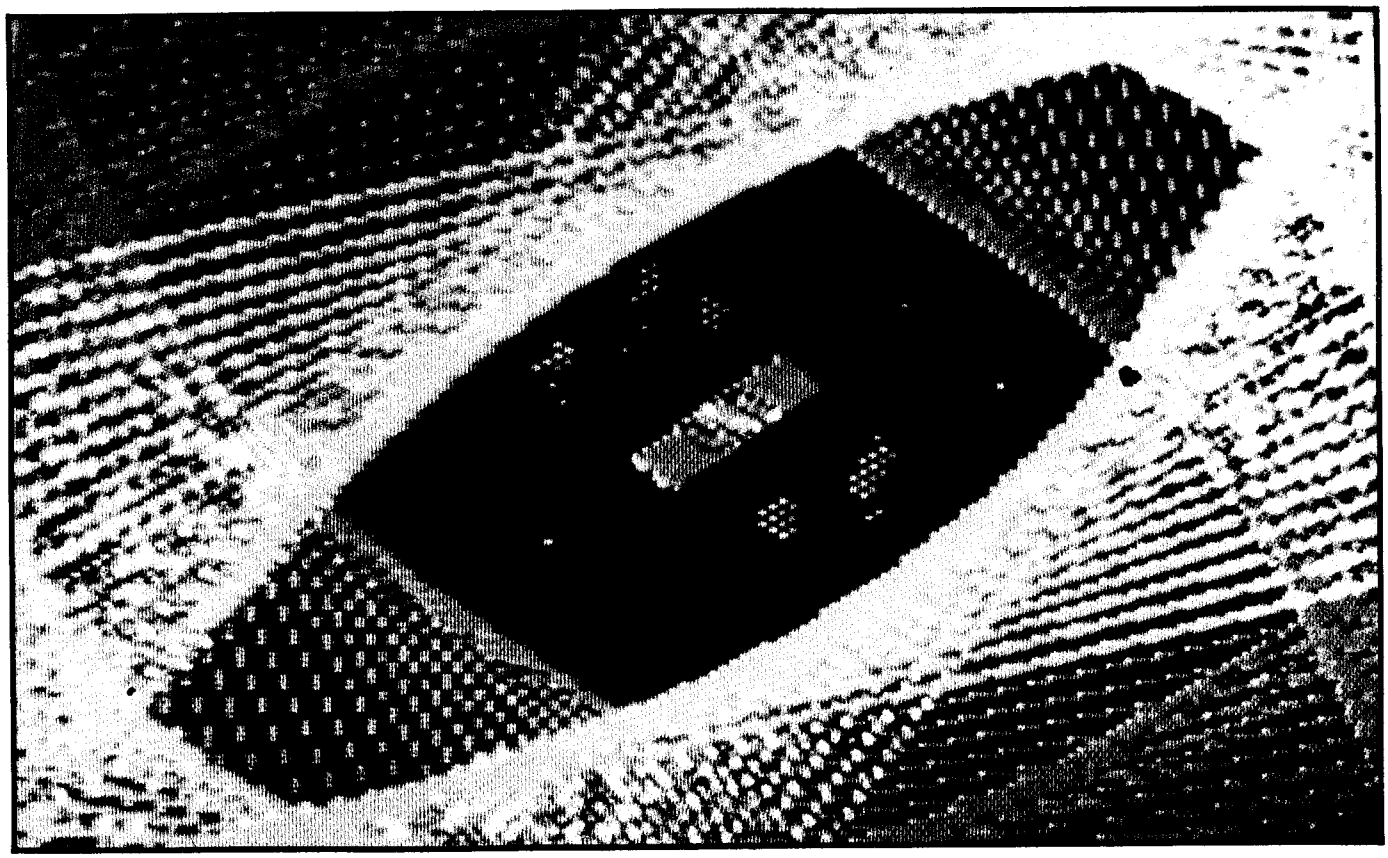


Photo 4: An example of the high resolution graphics hardware of the Apple II is provided by this photo. Although four nominal colors are available, this photo illustrates how some subtleties of color can be obtained by taking advantage of the distortions inherent in the television set when manipulating controls. This demonstration is one of several options available in the high resolution graphics demonstration program of Apple II.

primitives and all hardware addresses are documented in the Apple II.

The Applesoft interpreter is loaded from its cassette tape using the LOAD command of the 5 K interpreter. It takes about one minute and 30 seconds to do this load, which results in what 5 K BASIC thinks is a really big application program, but what is in effect the object text of the new interpreter plus a 5 K BASIC program which presents an interactive setup sequence. After loading this pseudo 5 K BASIC text, the user types RUN, and the interactive setup sequence is entered. This sequence includes the option to display a summary of Applesoft BASIC commands which is also available on a reference card. After the setup sequence, the Applesoft interpreter is entered, as indicated by a right square bracket prompt character (]). With a 16 K Apple II system, the user gets the message "5615 BYTES FREE" at the conclusion of the setup sequence.

Running the same rather incomplete benchmark as is found in table 1, I found that the Applesoft interpreter running in an Apple II took 12 seconds for 10000 iterations of a null loop.

Tape Mass Storage

The Apple II tape mass storage system accomplishes its purposes of storing and recovering files of data. I have proven this to my own satisfaction by using the system. For both the 5 K BASIC and Applesoft BASIC, SAVE and LOAD operations feature a "beep" from the Apple II's annunciator following the leader at the beginning of the file, and at the end of the operation. There is no such audible feedback when using the tape with system monitor commands.

It is the user's responsibility with the Apple II tape system to keep track of files on a piece of paper or in a log book. The software of the tape system does not include any naming of files in file headers. It also does not include any "verify" command for those of us who would like to make sure that a file is properly written before pulling the plug and closing down at the end of an evening's programming.

The act of loading a file involves the user positioning the tape just past the start of the leader tone for the file, issuing the proper command up to the point of the carriage return which completes the command line, then simultaneously starting the recorder and hitting the carriage return key.

High Resolution Graphics Software?

The one item of systems software which has been promised but not yet (December

Continued on page 35

Continued from page 32

1977) delivered to me in a documented form is the "high resolution" graphics software package. This is perhaps the best attribute of the Apple II's unique color graphics orientation; its potential is shown by a "high resolution demo" program which I was able to get from the company, including a number of graphic whizbangs selected from a list of choices. Perhaps the best in my opinion is something called the "Spirograph," a constantly changing color graphic implementation of a mandala. The illustration of photo 4 is taken from one state of this program's execution approximately six hours into its evolution one evening. The program uses a random number generator which appears truly random (unrepeatable upon reloading from the same tape) to control the course of the pattern. The price of the graphics routines is listed as $10.

Applications Software

A mixed bag of user oriented software is available with Apple as a means of demonstrating the system. This includes a number of tapes with games using the display and paddles, a "Checkbook-Home Management" package, an excellent 16 K Star Trek game which I have used on my system several times, etc. Typical prices are around $10 per tape, with four tapes currently listed in the catalog sheet. Users can expect more offerings as time goes on. It is these applications programs, games and graphic whizbangs, which provide the greatest value to me when demonstrating the concept of a personal computer to friends.

Documentation

Aside from having a well designed hardware configuration, the Apple II is not hidden from the user who wants to figure out what is in his computer. The "preliminary" documentation which I have on the Apple II consists of a loose leaf file bound in a report folder which has proved anything but preliminary in terms of completeness and usefulness. Its contents are not excessively verbose, but all the essentials are present: the specification of the syntax of the 5 K BASIC, how to access systems software hooks from BASIC, the complete address space map of the hardware, and neat comprehensive drawings of all of the system's electronics. When it came time for me to wire up a version of the game paddles in the form of my own joystick hack, I was able to turn to a page of the manual containing a specification of all the signals on the 16 pin game IO connector. This manual

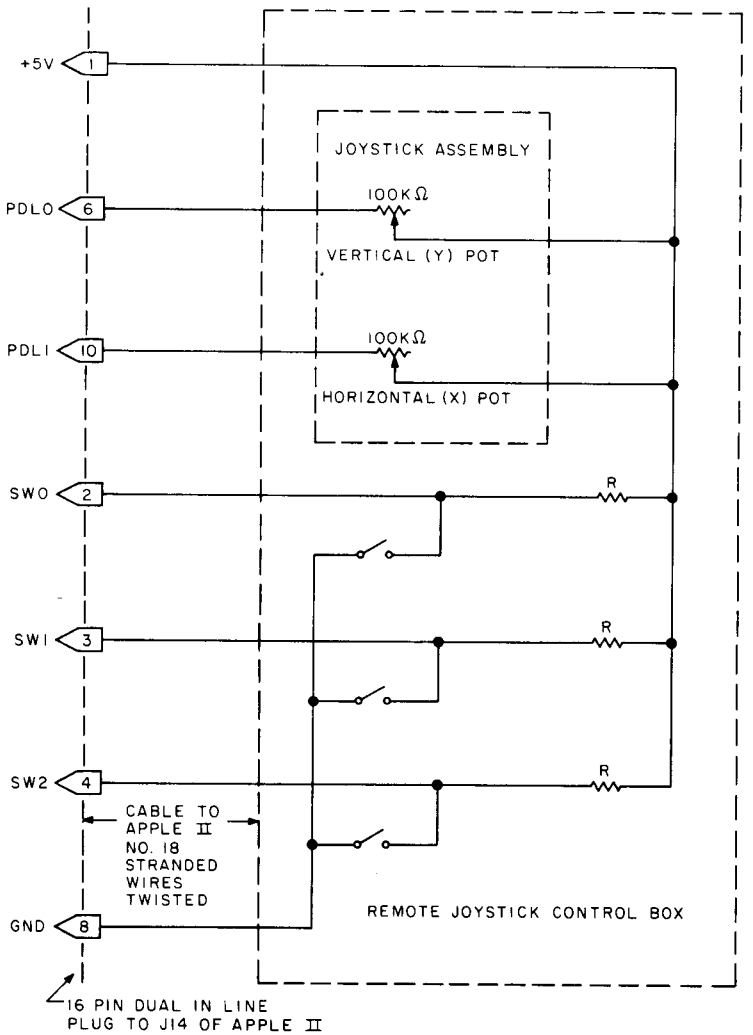


Figure 1: Wiring of the joystick control box and cable. The box is shown in photo 5. The cable used was made up of separate #18 stranded wires, twisted after all interconnection to the plug had been completed.

is not a tutorial on how the computer works, but it does contain all the information needed for the experienced experimenter to add custom hardware to the system. A "final" version of the manual is in preparation according to the Apple Computer Co, but this preliminary manual is complete enough to stand on its own.

The only items where I found documentation somewhat scanty were the Applesoft BASIC interpreter (documentation limited to a reference card quoting the initialization texts on how to use it), and the lack of documentation of high resolution graphics to date.

Using the Apple II: A Color Sketchpad Program and Joystick

As a means of trying out the Apple II system and its documentation, I set a goal of

Continued on page 40

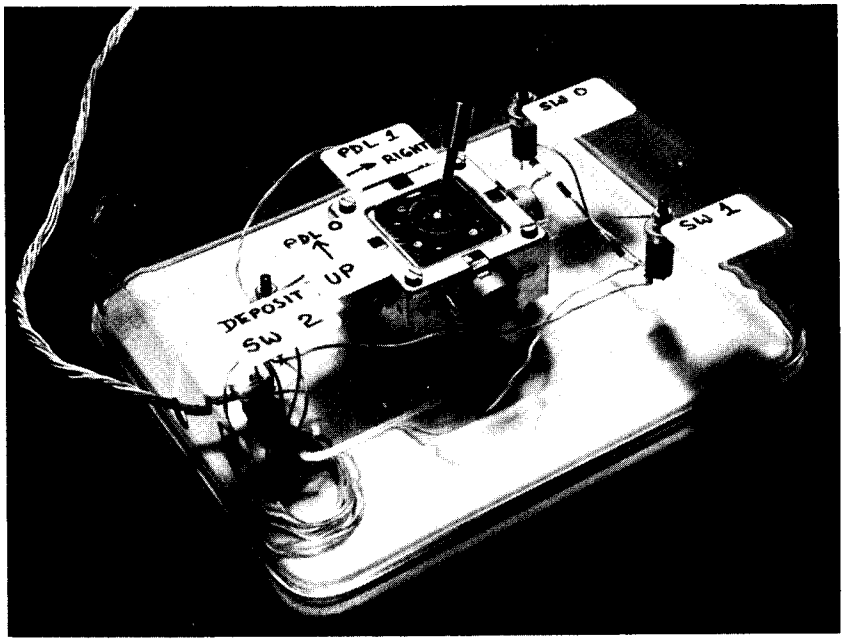


Photo 5: The joystick box which was implemented in order to create the color sketchpad application of listing 1. The cabinet is a gourmet plastic food container obtained at a retail outlet. The box is wired according to figure 1, which was created using the Apple II documentation.

Continued from page 35

implementing a version of what was called a "Cybernetic Crayon" by Thomas Dwyer, Leon Sweer and Margot Critchfield in an article we published in November 1976. The first requirement was that of creating a joystick input to replace the separate control paddles which are standard from the manufacturer of Apple II. Then with the joystick box tested and working, I would create software which would use the joystick and the Apple II keyboard to create color doodles within the 40 by 40 matrix of the television screen in the low resolution graphics mode of operation.

The creation of a joystick box with the Apple II is a simple matter of wiring. The manual lists the complete set of signals available at the game IO connector, a 16 pin dual in line socket located near the rear of the main board of the computer. The wiring is given in figure 1, showing the essentials of two resistance measurements inputs (PDL(0) and PDL(1) in 5 K BASIC) and three switch inputs. The actual box which I created is shown in photo 5. The joystick, which was purchased from James Electronics, has four 100 K potentiometers, of which only two are used, one on each axis. The switches were pushbuttons removed from an archaic surplus digital test jig (circa 1960 technology). The entire device was mounted in the bottom of a gourmet food container which I purchased retail at a shopping mall for this purpose. (I actually had to buy a matched set of three food containers for about $10, of which one was the right size.) The large hole for the lever of the joystick was drilled first, after which holes for the four #4-40 mounting screws were carefully located and drilled. (Use of a transparent container helped immensely here.) Wiring according to figure 1 was done point to point after mounting the push button switches.

Making a Program. . .

The wiring of the actual control box was derived directly from the documentation which came with the Apple II (although it required some knowledge of the way a 555 style timer is used to measure a resistance by controlling the width of a pulse (see May 1977 BYTE, page 42, figure 2)). In a similar way, the hook for use of the resistance measurements as controls of a program is built into the 5 K integer BASIC which is part of the Apple II. This hook is the built in function PDL(x) where x is an integer from 0 to 3 corresponding to the four possible paddles which may be used. In the case of the paddle box of figure 1, only PDL(0) and PDL(1) will give any externally variable value when referenced. It is one thing to read the cursor value for a display from these inputs, but it is quite another to use it, as I found.

My first attempt was to use the integer value from 0 to 255 returned by PDL(n) as a direct cursor control for the position of an action being performed on the screen. The only problem here was that when I normalized the values to a range of 0 to 39 appropriate for the 40 by 40 matrix of points of the Apple II, the characteristics of my potentiometers prevented fine control of which point on the screen was addressed (certain points proved totally unadddressable). My second attempt was to use a tabular transfer function to convert observed value to a 0 to 39 coordinate value. After this did not work well, as a final expedient I then reduced the joystick inputs to the logical equivalent of a set of four single pole single throw switches which would input an effective "velocity" value of −1, 0 or +1 for each axis of motion of the joystick. Once I had an effective way of input for the cursor motion commands defined by the joystick, I could begin to design a program to allow definition of color values, and depositing of colors under control of one of the switch inputs of the control box.

The final program, a result of several iterations, is given in listing 1, photographically reproduced from the screen of my television display. The program begins with setting of the "graphics" mode with the GR

Listing 1: The color sketchpad program, implemented in 5 K Apple BASIC. Showing complete lack of form, this program is written by its author without REM statements to explain what is going on. See the text of the article for detailed comments.

```
  5 DIM T$(100)
100 GR : COLOR=0
110 FOR I=0 TO 39: HLIN 0,39 AT
    I: NEXT I
120 X=20:Y=19
130 ODEP=0:DEPOSIT=0
140 SCOL=0
160 LNG=2
170 BLINK=LNG
179 PRINT "WHAT ARE JOYSTICK "
180 INPUT "DEAD ZONE LIMITS=",DZ
190 ZD=255-DZ
1000 KEY= PEEK (-16384)
1003 GOSUB 8700
1005 GOSUB 3000
1010 IF KEY<128 THEN GOTO 1000
1015 POKE -16368,0
1030 IF KEY= ASC("F") THEN 2000
1051 IF KEY= ASC("Q") THEN 8000
1052 IF KEY= ASC("C") THEN 8500
1053 IF KEY= ASC("T") THEN 2500
1054 IF KEY= ASC("J") THEN 179
1070 GOTO 1000
2000 PRINT : PRINT : PRINT
2010 INPUT "FILL SCREEN WITH COLOR="
    ;CC
2020 IF CC<0 OR CC>15 THEN 1000
2030 COLOR=CC
2040 FOR I=0 TO 39
2050 HLIN 0,39 AT I: NEXT I
2055 SCOL=CC
2057 GOSUB 2600
2060 GOTO 1000
2500 PRINT : PRINT : PRINT
2510 PRINT "ENTER TEXT TAG FOR DRAWIN
    GS"
2520 INPUT T$
2530 GOSUB 2600
2540 GOTO 1000
2600 PRINT : PRINT : PRINT : PRINT
    T$
2610 RETURN
3000 BLINK=BLINK-1
3010 IF BLINK=0 THEN 3100
3020 IF BLINK=LNG/2 THEN 3200
3030 RETURN
3100 COLOR=0
3105 IF SCOL=0 THEN COLOR=6
3110 PLOT Y,X
3120 BLINK=LNG
3125 GOTO 3300
3200 COLOR=SCOL
3210 PLOT Y,X
3220 RETURN
3300 XX= PDL (0):YY= PDL (1)
3310 IF XX<DZ THEN GOSUB 6000
3320 IF YY<DZ THEN GOSUB 7000
3330 IF XX>ZD THEN GOSUB 6500
3340 IF YY>ZD THEN GOSUB 7500
3350 RETURN
6000 COLOR=SCOL: PLOT Y,X:X=X-1
6010 GOTO 9900
6500 COLOR=SCOL: PLOT Y,X:X=X+1
6510 GOTO 9900
7000 COLOR=SCOL: PLOT Y,X:Y=Y-1
7010 GOTO 9900
7500 COLOR=SCOL: PLOT Y,X:Y=Y+1
7510 GOTO 9900
8500 PRINT : PRINT : PRINT : INPUT
    "WHAT NEW COLOR",SCOL
8510 SCOL=SCOL MOD 16
8519 GOSUB 2600
8530 GOTO 1000
8700 DEPOSIT= PEEK (-16285)
8710 IF DEPOSIT<128 THEN 8740
8720 DEPOSIT=-1
8730 GOTO 8750
8740 DEPOSIT=0
8750 IF DEPOSIT=ODEP THEN RETURN

8760 IF DEPOSIT=0 THEN 8800
8770 QQ=SCOL
8780 ODEP=DEPOSIT
8790 RETURN
8800 SCOL=QQ
8810 ODEP=DEPOSIT
8820 RETURN
9900 IF X<0 THEN X=0
9902 IF X>39 THEN X=39
9904 IF Y<0 THEN Y=0
9906 IF Y>39 THEN Y=39
9908 IF DEPOSIT THEN 9950
9910 SCOL= SCRN(Y,X):BLNK=LNG
9920 RETURN
9950 COLOR=SCOL
9960 PLOT Y,X
9999 RETURN
```

statement, after which a loop contained in line 110 clears the screen to black color with 40 HLIN function calls to draw horizontal lines across the whole screen with the last color value (set by the statement COLOR = 0 in line 100). Lines 120 to 190 then initialize several variables. X and Y are used by the program as the current cursor values, ranging from 0 to 39, initially set approximately to the center of the screen. DEPOSIT and ODEP are flag values used to coordinate whether or not the cursor leaves a trail of color. SCOL is used temporarily to store color values. LNG is the length of the delay loop which controls how fast the cursor blinks, and must be an even number so that it can be divided by 2. BLINK is the running counter for this delay loop. The joystick dead zone limits are defined with a request to the user, and are the number of states between 0 and 255 at either end of the range which will be considered equivalent to the nonzero velocity values for the cursor. With entry of the value 100, the calculation of the dead zone gives nonzero velocity if the input from the joystick measurement is 0 to 100, or 155 to 255 in that direction. With the hardware I built for the external control box, a dead zone of 100 is a typical useful value; nonlinearities in the potentiometers make lower values impossible to use, and higher values make it very difficult to set the joystick to dead center and stop motion of the cursor. The variables ZD and DZ contain the dead zone limits.

The main routine of the color sketchpad program is found in the region from lines 1000 to 1070 of the listing. This "executive" loop begins with a small loop that scans for key input from the Apple II's keyboard. The current output of the keyboard is obtained by the magical incantation on line 1000. Two subroutines are called in this scanning loop at lines 1003 and 1005. The keyboard scanning loop waits until a valid key code (greater than 127) is returned from the keyboard before decoding a keyboard command. The subroutine at lines 3000 to 3350 blinks the cursor and reads the paddle, moving the cursor according to the values XX and YY which are input when the BLINK count has reached zero. The subroutines which implement motion are found at lines 6000, 6500, 7000, and 7500.

Returning to the main keyboard scanning loop, the second subroutine called within this loop is found at lines 8700 to 8820, and is responsible for reading the "DEPOSIT" switch, switch #2, and setting appropriate flags to perform the action of depositing a color. Eventually, at line 1010 a KEY value of an ASCII character code

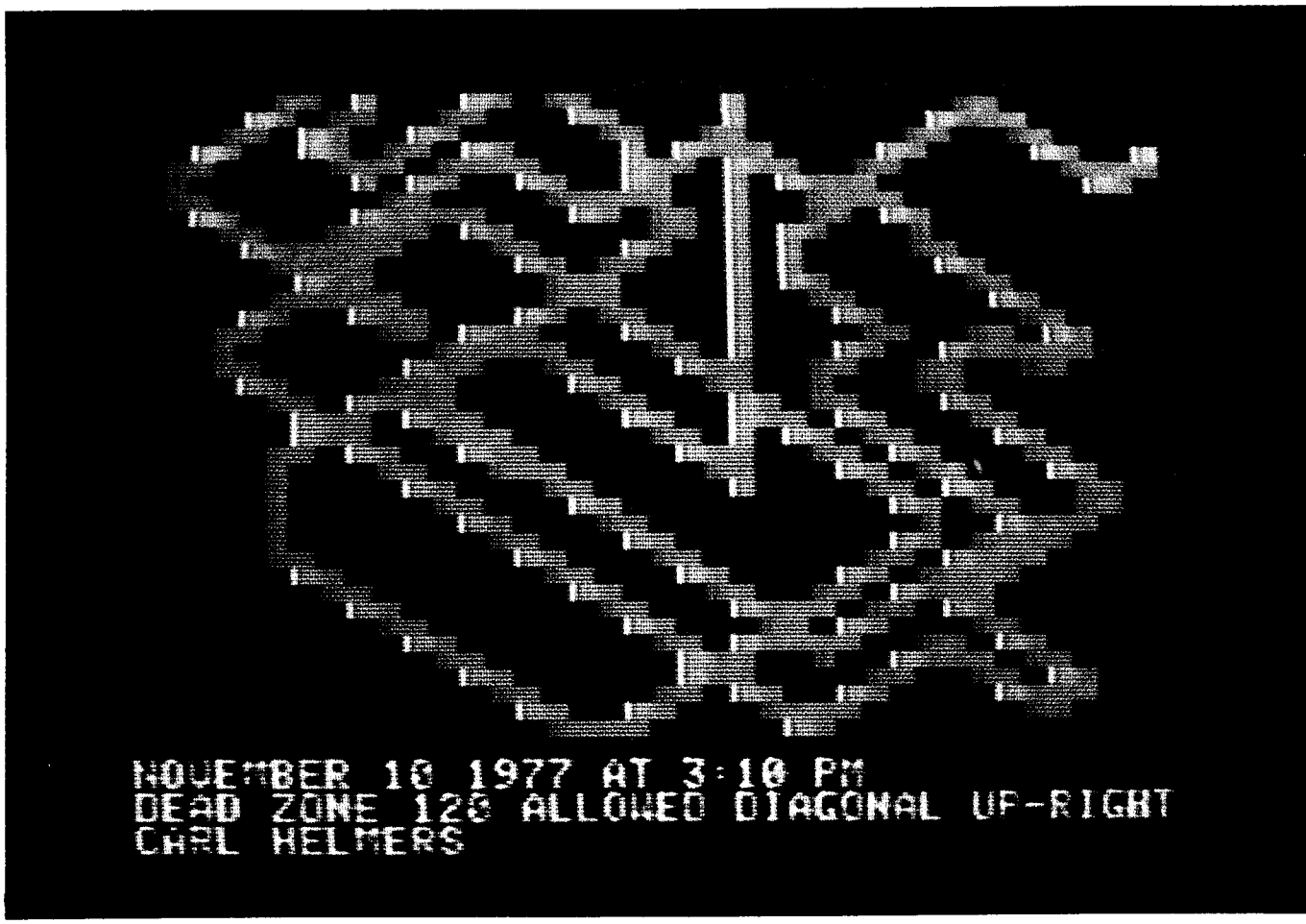


Photo 6: An example of another product of the color sketchpad program, in this case illustrating a text tag entered by the program's "T" command to identify when and who did the illustration. The comments on the second line of the text at the bottom refer to an experiment with the program's joystick dead zone parameter.

with a high order bit of "1" is detected in the range 128 to 255, and the inner keyboard scanning loop temporarily ends. Input of the KEY is acknowledged to the keyboard by the magical incantation to the hardware of POKE −16368,0 at line 1015. (All this hardware specific information was elicited from the documentation which came with the system.) After the KEY has been acknowledged, its value is checked and one of five command routines is chosen with a series of IF statements. If none of the valid commands is entered, the GOTO 1000 at line 1070 continues execution of the keyboard scanning loop looking for a valid command.

Turning to the implemented keyboard commands of this color sketchpad program, the "F" command is used to fill the screen with an arbitrary color. The details, found at lines 2000 to 2060, print a message to the user "FILL SCREEN WITH COLOR=" after which the user enters a value from 0 to 15 which is range limited by statements at 2020. (Thus if "F" is hit by mistake, typing an invalid color value outside the range 0 to 15 gets the user back to the main loop without erasing the current picture.) If a valid color is indicated, the screen is filled with that color, erasing all previous work.

The "C" command is used to access the routine starting at line 8500 which prompts the user for a new color value which will be used for drawing. The prompting message from an INPUT statement is "WHAT NEW COLOR?". As in most BASIC interpreters, the question mark comes from the INPUT statement's operation and the actual string found in the program has no question mark. The user response to this INPUT statement at line 8500 is a value limited to the range 0 to 15 by the assignment statement at line 8510 using the MOD function.

The "T" command is used to input a text tag contained in the string T$. This tag is typically output by a reference to the subroutine at line 2600 prior to resumption of the scanning loop after a command is executed. In the example of photo 6 the text tag was used to identify the date and time at which the picture was composed, and the "artist" responsible for it.

Finally, the "J" command is used to reset the joystick dead zone limits in the

event that the first value entered during initialization was inappropriate.

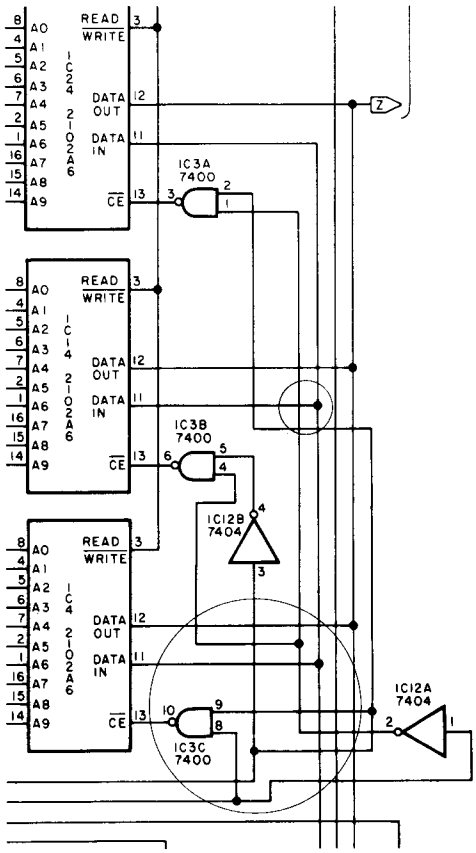
Summary of the Apple II

As noted at the beginning of this review, I was very enthusiastic about the prospect of this machine from the first word of its existence. To be sure, there are minor annoyances with this product in its present state, like the mechanical problems with respect to its case and keyboard mounting, and the persistent problems of radio frequency interference which I found using my particular color television set as the primary display. But such problems are minor compared to the achievements of this design.

Apple II is a well executed example of the way a personal "appliance" computer should be made circa 1977. It is suited for the relative beginner who knows BASIC but does not know hardware design, as well as for the advanced hacker who feels no qualms about using the system's documentation to create custom peripherals for the system. For the user who wants color graphics, the Apple II is the only practical choice available in the "appliance" computer class.■

BYTE's Bugs

A Flop in the Floppy



The drafting bug has bitten BYTE. The accompanying figure detail (taken from figure 1a of David Allen's article, "A Floppy Disk Interface," January 1978 BYTE, page 61, right-hand side) contains a number of drafting errors inadvertently included during the production of the article. The corrections are circled. In addition, IC18c, a three input AND gate, was incorrectly drawn as a NAND gate, and the A1 and A2 inputs of IC22 should be shown connected together. Our thanks to the readers who spotted these errors, and our apologies to David Allen.■

Clock Stops

I enjoyed M F Smith's article in the November 1977 BYTE on "Using Interrupts for Real Time Clocks." He did a great job in describing how to develop and use a real time clock. So I implemented his program on my AMI PROTO 6800 System, only to find it not functioning properly in updating the clock. After examing the program listing closer, I found the bug at line 29 of your listing on page 53. The index compare was only index by 7 into the constant table, it should be index by 8. After I made this change everything worked.■

William W Barncord
Burroughs Corporation
2473 S Memphis Way
Aurora CO 80013

Joystick Gets Stuck

This letter is in reference to "An Inexpensive Joystick Interface," which appeared in the March 1977 BYTE on page 88. I'd like to congratulate Tom Buschbach on a fine article, but there seem to be several mistakes in the schematic.

On the MC1408L-8, A1 is the most significant bit. The schematic shows this connected to the least significant bit of the counter. The eight digital inputs to the digital to analog converter should be reversed in order to obtain a proper sawtooth output.

Also, on IC2, 74193, the count up input is shown grounded. This must be held high to enable the count down input.

I have constructed this circuit using a different register chip (8212) and the above changes, and it works very nicely.■

William Lemiszki
424 Cambridge St
Allston MA 02134

Fractured Factors and Walsh Function Bugs

William Jackson's letter on page 172 of the October 1977 BYTE contains two errors which can be corrected by replacing the fourth equation with:

$$=(y+z) + j(x-z)$$

and the fifth equation with:

Where $x=(a+b)c$.

Regarding the September 1977 article "Walsh Functions: A Digital Fourier Series": in the third paragraph on page 196, one reads "Sin (11.25°) = 0.09802." The proper value is 0.19509. The value given is that appropriate for 11.25/2°. The sums should be, I think, 5.13842, −2.13578, −0.40974 and −1.03218 in table 4. The coefficients would then be 1, −0.4156, −0.0797 and −0.2009. Also the signs for SAL(7) at the bottom of the table should be PNPNNPNP. What was printed is CAL(7).

If I have not made a mistake, the resistors in table 5 would be unchanged, but shouldn't there be some differences in the circuit diagram?

J S Lefson
6609 Cote St Luc Rd, Apt 203
Montreal, Quebec
CANADA H4V 169

Dr Jacoby replies:

Mr Lefson is correct that SAL(7) in table 4 is misprinted. This can be easily seen by comparison to table 3 or by knowing that the SAL functions themselves are odd about their centers and even about their 1/4 and 3/4 period points, and thus should give identical values to the left and right of center in table 4.

In addition, the values for Sin 78.75° and Sin 101.25° are incorrect and should read 0.98078. I believe that the values for the sums are correct as given as well as the circuit connections [compare to figure 3 for SAL(7)]. One point of confusion here might be the signs of the coefficients, since they take into account the inverting *op amp configuration and thus appear reversed.*■