Putting the Apple II Work

Part 1: The Hardware

A high-speed system for the acquisition and analysis of data

The world we live in is anything but static. We are constantly exposed to a changing environment that our central nervous system samples, analyzes, and, when necessary, responds to. In many ways, computer systems are a lot like the human body, which is equipped with a number of specialized sensors that convert complex, time-dependent information into a form that can be sampled by the nervous system. The nervous system processes the incoming information and makes decisions that cause the system to respond appropriately. Computers, when equipped with specialized sensors, also can sample the surrounding environment, process this incoming information according to some predetermined algorithm, and effect an appropriate response.

Many commercially available transducers can be used to convert physical-energy variations into time-varying electrical voltages. For example, thermistors can be used to measure temperature, dimensional changes can be measured by resistive strain gauges, and PIN diodes can be used to measure changes in light intensity. If a physical parameter changes very slowly and you have an abundance of time, you can use a digital voltmeter, a digital clock, and a pencil to record data and enter it into the computer by hand at some later time. However, if you desire an automated system or if the transducer output voltage changes at a rate that makes manual sampling impractical, you will need a computer-based data-collection scheme that will reduce the amount of operator interaction and still allow the collection of large amounts of data.

In part 1 of this article, I'll introduce the hardware required for such a system, discuss its operation and construction, and go through preliminary checkout and testing. In part 2, I'll provide the Applesoft and machine-language listings and discuss their development and use.

While most computers are quite proficient when it comes to handling binary (on/off) voltages, they usually are not capable of directly handling the analog voltages from the output of most transducers. Placing an analog-to-digital (A/D) converter between a transducer and the computer enables the computer to monitor the changing physical parameter as well as to automate the sampling process.

An Apple II was selected several years ago for use in our laboratory. While many new computers have since been introduced into the marketplace, the Apple II continues to be my first choice for the following reasons:

1. The Apple II features eight built-in connectors that make adding external interface circuitry a relatively easy task.
2. An abundance of commercial software is available.
3. The multicolor, high-resolution graphics software enables several channels of data to be displayed simultaneously.
4. The logical structure of the 6502 and the existence of a miniassembler within the Apple firmware make machine-language programming relatively easy.
5. It is extremely reliable. When it has needed repair, service was easy to find and the repairs quickly completed.

Design Criteria

I designed the circuitry discussed in this article to perform the specific task of digitizing the complex voltage waveforms produced by a muscle being exercised. I needed to simultaneously sample three channels of this electromyographic (EMG) information so the data would be synchronized at specific points in time. Because I was preprocessing the EMG by taking the absolute value and then passing those signals through a low-pass filter, I knew that the input voltage to the A/D converter would always be positive, that it would never exceed a maximum value of 5 volts (V), and that the highest frequency component would be no greater than 100 Hz. With this information in mind, I determined the design specifications for the A/D converter.

Essentially, three factors create major limitations to the accuracy and usefulness of data collected through an A/D converter: loss of significance, resolution, and sampling rate.

Loss of significance is what occurs when the maximum magnitude of the input signal is much less than the A/D converter's maximum input range. As an example, suppose that you are using an 8-bit A/D converter that has a maximum input of 10 V. The input range of 10 V is then spanned by the 256 (2^8) possible voltage levels that the A/D converter can quantize. When the input voltage is equal to 10 V, the entire number of possible voltage levels is used. The signal-to-noise ratio then can be expressed as 20 times the logarithm of the ratio of the input voltage to the smallest quantized voltage level, or 20 $\log_{10}$ (255/1) = 48 decibels (dB). Suppose the input voltage had been only 2 V. The A/D converter would have then used only 50 of the 255 possible voltage levels, reducing the signal-to-noise ratio to 20 $\log_{10}$ (50/1) = 34 dB. Consequently, it is important to match the maximum input voltage to the maximum input range of the A/D converter whenever possible.

Resolution is related to the ability to distinguish between two voltage levels that are nearly equal. The smallest magnitude difference that can be detected defines the resolution of the system. For an A/D converter that represents a 10-V analog input voltage by an 8-bit binary number, resolution is equal to 10/256 = 0.039 V. Under ideal conditions, the system should be able to distinguish between two signals with a voltage difference of 0.039 V.

The sampling rate determines the computer's ability to detect time-dependent changes in the input voltage. As an example, consider an input signal that is changing at a rate equal to 50 V/second. Suppose that you wanted to resolve the input signal with a 3 percent accuracy (within ±0.3 V for an input equal to 10 V) at any point in time. An input voltage that is changing at a rate equal to 50 V/second changes by 0.3 V in 6 milliseconds (ms). Consequently, to achieve 3 percent accuracy, you must sample the input signal at least once every 6 ms.

If you don't have an intuitive feel for the accuracy you need, a good rule of thumb is to set the sampling rate to twice the maximum frequency component of the input signal. When you sample too slowly, you can have problems with *aliasing*, which results when a high-frequency signal impersonates a low-frequency signal (see figure 1). For applications in which you will be sampling at rates that are less than twice the highest frequency component, you must insert a low-pass filter at the input of the A/D converter to limit the frequency content and to ensure faithful reproduction of the input signal. When you sample too quickly, you will quickly expend the available memory in the computer. However, it is generally better to have too much data than not enough.

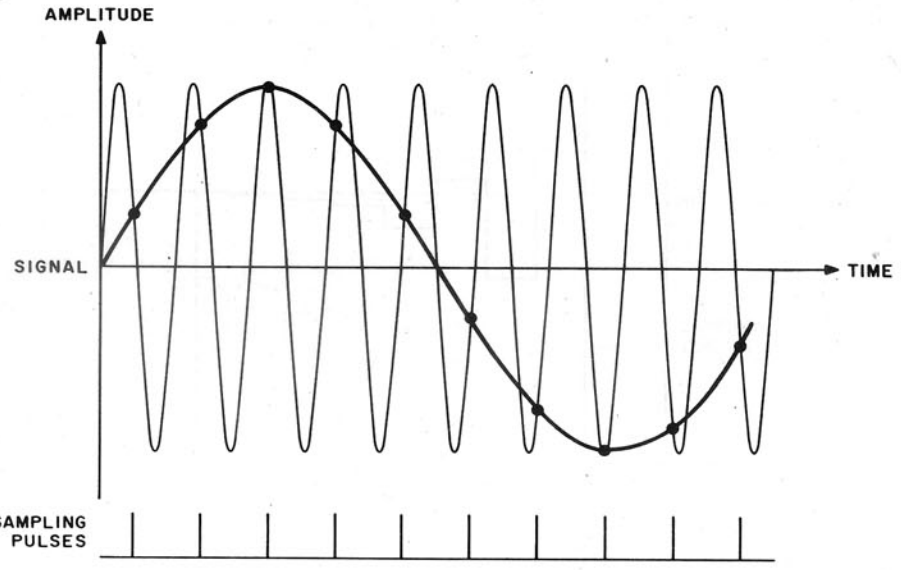


Figure 1: *An example of aliasing caused by a sampling rate that's too low.*

Loss of significance, resolution, and sampling rate are three factors involved in data collection through an A/D converter.

System Hardware

The AD7570 from Analog Devices Inc. (Two Technology Way, Norwood, MA 02062, (617) 329-4700) is a successive-approximation-type A/D converter that requires only an external reference and a comparator to provide either an 8- or 10-bit output representation of the input signal. A three-state output register is used to buffer the digital output signals, enabling several AD7570s to be connected in parallel to a single data bus. This feature permits you to use a separate A/D converter for each input channel, thus providing increased system throughput rate.

The AD7570 uses a conversion scheme known as successive approximation to achieve the high resolution and conversion speed necessary for

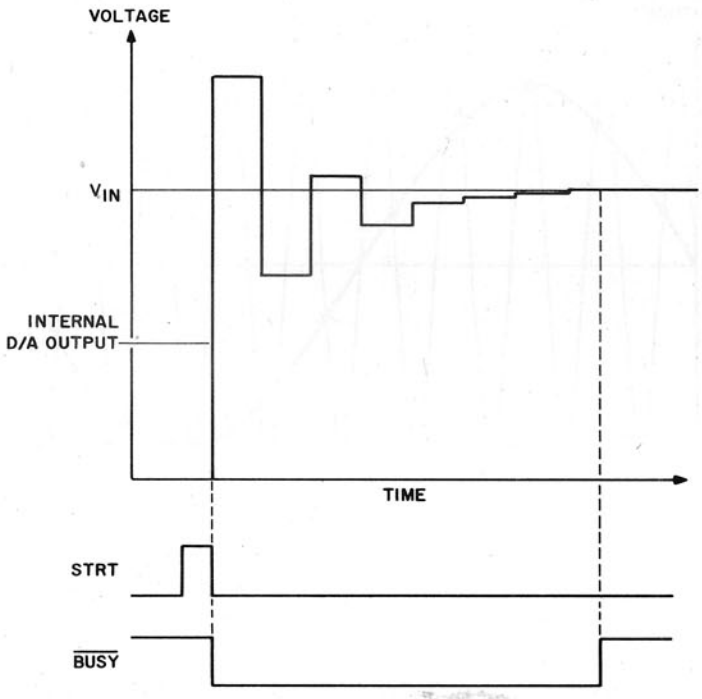


Figure 2: *Analog-to-digital (A/D) conversion example using the successive-approximation technique. The A/D converter output makes several steps before matching the input voltage.*

some computer applications. Successive approximation involves comparing the unknown input voltage with a preset series of voltage increments that are binary fractions of the maximum input range that the A/D converter can handle.

Initialization of the conversion sequence begins when the convert start (STRT) input goes to a logical 1 (see figure 2). At this time, the most significant bit (MSB) of the data latch is set to a logical 1, and the remaining bits of the data latch are set to a logical 0. When the STRT line is returned to a logical 0, the actual conversion process begins. The output of the internal digital-to-analog (D/A) converter is sequenced bit by bit from the MSB to the least significant bit (LSB).

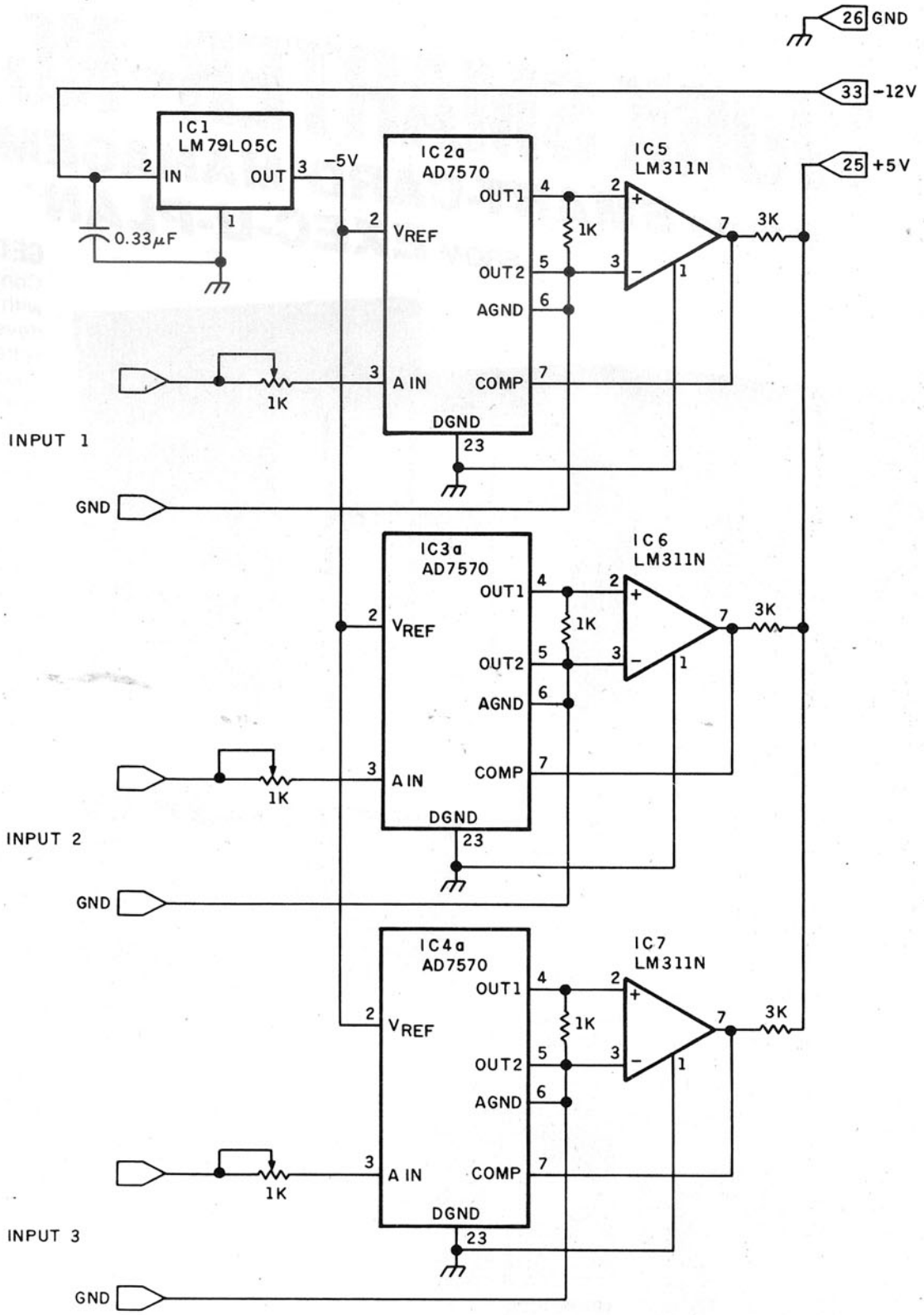
The external comparator determines whether the addition of each successively weighted bit creates a voltage that is greater than or less than the input voltage. When the voltage is greater, the bit is turned off (set to a logical 0); when the voltage is less, the bit is left on (set to a logical 1). After this comparison is made between all bit combinations, the conversion is complete and the internal successive-approximation register contains the binary code that represents the converted input signal. Thus, for a converter circuit that can measure an input voltage varying between 0 and 10 V (10 V is full scale), the comparisons would be made between voltage levels that varied in 0.039-V increments (10 V divided by 256 discrete levels).

When an unknown input voltage is to be converted, first, the MSB of the internal D/A converter's output (one-half full scale) is turned on, comparing the input voltage to 5 V. If the input voltage is less than 5 V, the MSB is turned off, the next bit (one-fourth full scale) is turned on, and the input voltage is compared to 2.5 V. If the unknown input voltage is greater than 2.5 V, the second bit is left on, the next bit (one-eighth full scale) is turned on, and the input voltage is compared to 3.75 V. (2.5 + 1.25). If the unknown input voltage is less than 3.75 V, the third bit is turned off, the next bit (one-sixteenth full scale) is turned on, and the input is compared to 3.125 V (2.5 + 0.625). This process continues in order of descending bit weight until all bits have been tried. The conversion process is thus completed, and the 8-bit binary number representing the unknown input voltage is ready to be read by the computer.

I have divided the circuitry associated with the A/D converter into two classifications: circuitry that deals primarily with analog signals and circuitry that deals primarily with digital signals. Figure 3 shows the circuitry dealing with analog signals. IC1 is a three-terminal voltage regulator that provides −5 V to the reference voltage terminal (pin 2) of the AD7570s. I used a 5-V reference because the signals that I am digitizing do not exceed 5 V. The AD7570 is capable of accepting voltages from 0 to +10 V at the input terminal. Because the three input sections are identical to each other, I will describe only the circuitry associated with Input 1. The A/D converter (IC2a) works in conjunction with the comparator (IC5) to determine the binary representation of the input signal. As the internal successive-approximation register changes the weighted bit pattern, IC5 compares the output of the internal D/A converter with the input signal. The results of the comparison are fed back to pin 7 of the AD7570, and the successive-approximation register makes appropriate adjustments to the weighted bit pattern. The 1k-ohm resistor is connected across the comparator input terminals to reduce the settling time of the comparator, which ultimately reduces the conversion time.

Figure 4 shows the digital circuitry. As in the description of the analog circuitry, only one input channel will be discussed because the other two channels are identical. The AD7570 has a provision for what is called a short-cycle conversion. This is accomplished by connecting the $\overline{SC8}$ (pin 26) control line to a logical 0, forcing the converter to stop the conversion cycle after 8 bits, and reducing the conversion by two clock cycles. Even more important than achieving the time savings of two clock cycles is the time saved by having to read in only 8 data bits per input channel. For my applications, the increase in sampling rate that could be achieved was considered to be worth the resolution that was lost.

Operating under the short-cycle format, the conversion process still starts with the MSB and works down



IC	+5 V	+12 V	−12 V	Ground
1	—	—	2	1
2, 3, 4	22	1	—	6, 23
5, 6, 7	—	8	4	—

Figure 3: *A/D converter analog input circuitry.*

to DB2. The 8 bits of digitized data are then contained on data lines DB2 through DB9. The high-byte enable (HBEN) control line is a three-state enable for DB9 (MSB) and DB8. When HBEN is a logical 1, digital data from the internal latches appears on the data lines. The low-byte enable (LBEN) control line is a three-state enable for DB0 (LSB) through DB7. When LBEN is a logical 1, digital data from the internal latches appears on the data lines. Because the short-cycle mode uses only data lines DB2 through DB9, HBEN and LBEN are connected together so that a logical 1 causes the digital data representing the converted input signal to appear on the data lines.

The busy enable (BSEN) control

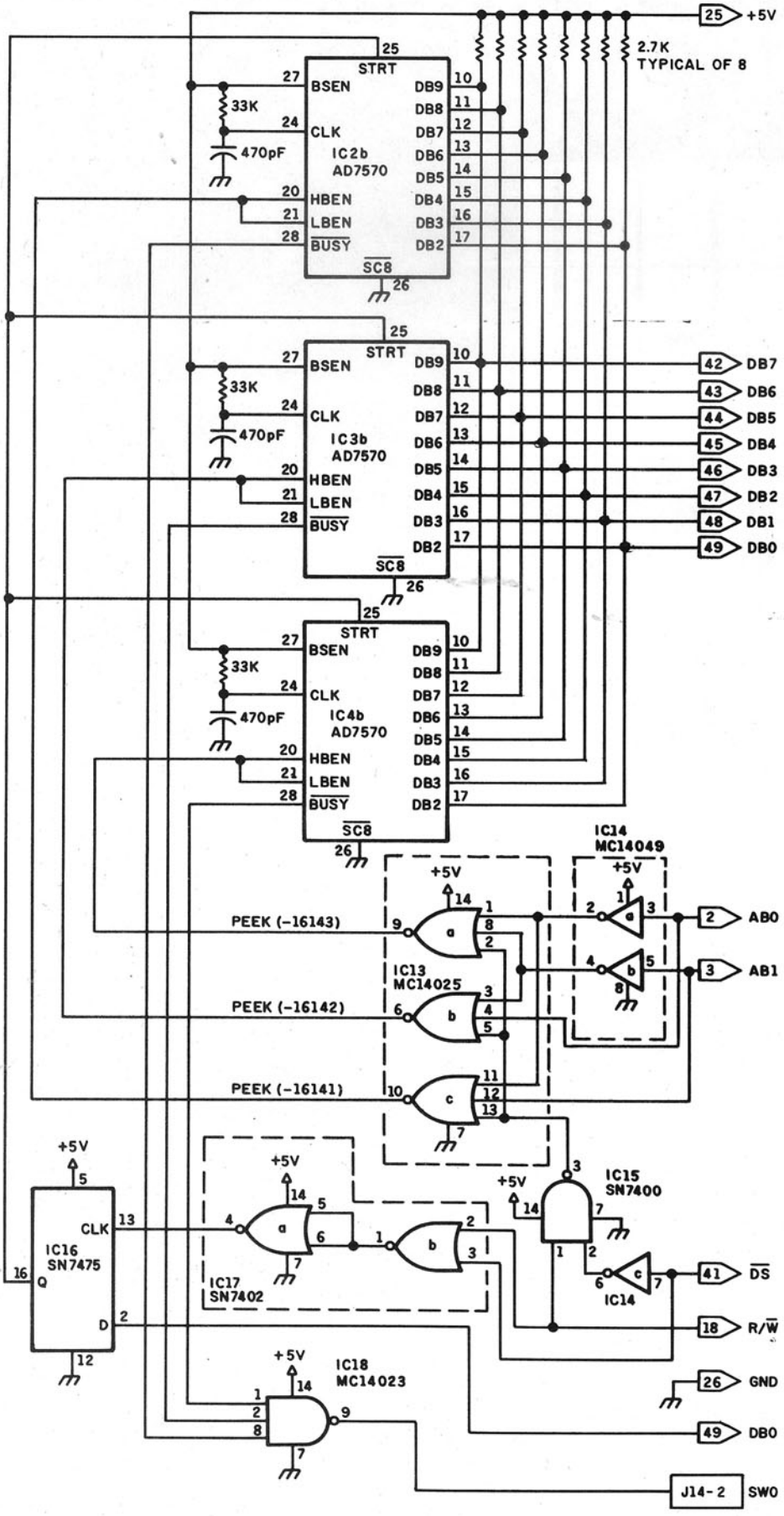


Figure 4: *A/D converter digital circuitry.*

line (pin 27) is used to determine if the converter status line (BUSY) is enabled or if it is floating. In this application, I connected the BSEN line to a logical 1. Thus, the BUSY line always reflects the status of the converter. During the time that the conversion is being performed, the BUSY line is set to a logical 0. Upon completion of the conversion, BUSY is set to a logical 1.

The 33k-ohm resistor and the 470-picofarad (pF) capacitor are used to determine the internal clock frequency. With these values, the clock frequency is approximately 100 kHz. Clock activity begins upon receipt of a conversion start pulse and ceases upon completion of the conversion.

The remaining circuitry in figure 4 shows the control logic necessary to initiate the conversion cycles for all three converters, to sense when the conversion cycles have been completed, and to coordinate the transfer of data into the computer. The circuit is designed so that the peripheral card resides in I/O (input/output) slot 7 on the Apple II motherboard. The device-select signal goes to a logical 0 whenever memory locations (hexadecimal) C0F0 through C0FF are addressed. [**Editor's Note:** *All addresses and number values are hexadecimal unless otherwise specified.*] The least significant 2 bits of the address are decoded by IC13 and are used to transfer data from one of the three converters by enabling the three-state buffer of the appropriate converter. A conversion cycle is initiated by performing an LDA #01, STA C0F0 followed by an LDA #00, STA C0F0. This causes the output of the D-type flip-flop (IC16) to go from logical 0 to logical 1 and back to logical 0. This pulse is connected to each of the AD7570s, causing the three unknown input signals to be converted simultaneously. IC18 is used to indicate to the Apple II that all three converters have completed their conversion cycles. The output of IC18 is connected to one of the inputs on the game connector. Performing an LDA C061 loads the status of the game input into the 6502's accumulator; rotating the accumulator to the left and testing the carry bit enables the

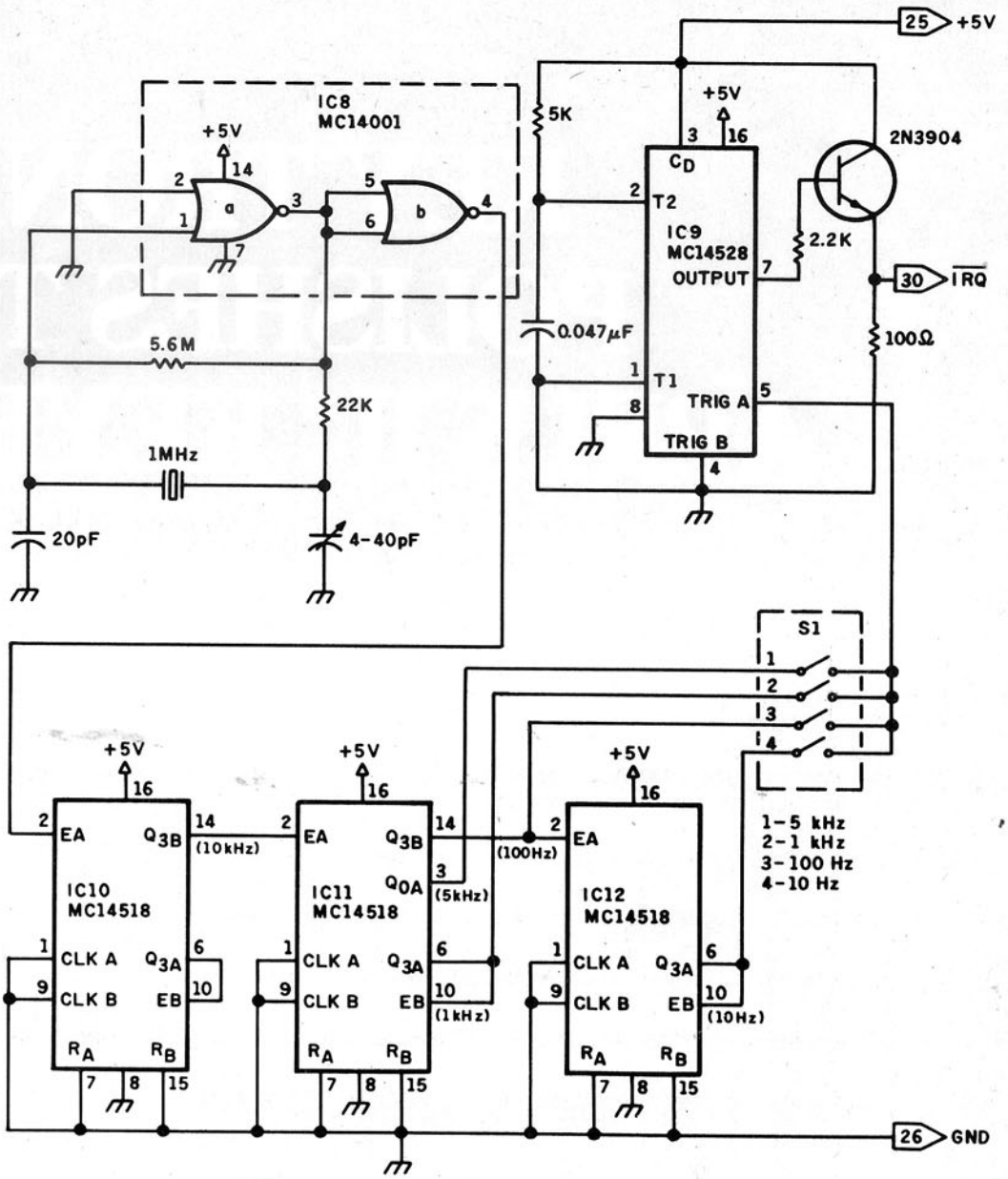


Figure 5: *Crystal-controlled time base used in the A/D converter.*

program to determine whether the conversions are complete.

It is desirable to take periodic data samples and to know the relationship between the magnitude of the data and time. This enables displaying the data as a function of time and permits the analysis of the data with respect to time. Some analysis techniques (such as fast Fourier analysis) require the data to be sampled periodically and the time between samples known. I used the interrupt-request line ($\overline{\text{IRQ}}$) going to the 6502 microprocessor to control when a sample was to be taken. The $\overline{\text{IRQ}}$ control line is called a maskable interrupt because the system will jump to a given memory location if an interrupt request is received and if the interrupt system has been enabled. The CLI (clear interrupt-disable bit) command is used to arm the 6502 so that it will respond to the next interrupt request it receives.

Once a request is received, the 6502 first executes an indirect jump using the address contained in memory locations FFFE (LSBs) and FFFF (MSBs). The 6502 then executes a short subroutine that serves to handle the interrupt request. Ultimately, the 6502 is forced to jump to the memory location contained in memory locations 3FE (LSBs) and 3FF (MSBs). The Hello program, which is executed when the computer is first turned on, uses POKEs to place the desired interrupt entry point into addresses 3FE and 3FF. Hello also disables the interrupt system so that an interrupt will not be prematurely executed. Once execution of the interrupt routine has been

Listing 1: *A/D converter system initialization routine.*

```
10 REM HELLO
20 HOME: REM CLEAR SCREEN
30 PRINT "****************************************"
32 PRINT "*          A/D CONVERTER               *"
34 PRINT "****************************************"
50 REM DEFINE INTERRUPT JUMP ADDRESS
52 POKE 1022,64:POKE 1023,144
54 REM INTERRUPT DISABLE SUBROUTINE
56 POKE 1016,72:POKE 1017,8:POKE 1018,120:POKE 1019,40:
   POKE 1020,104:POKE 1021,96
58 CALL 1016: REM EXECUTE SUBROUTINE TO DISABLE INTERRUPT
   REQUEST LINE
80 D$="": REM DOS CONTROL CHARACTER
82 PRINT D$;"RUN TEST,D1": REM LOAD AND EXECUTE MAIN APPLESOFT
   PROGRAM
99 END
```

performed, an RTI (return from interrupt) command is executed to force the processor to return to the instruction that was being executed when the interrupt request was first received.

Figure 5 shows the circuit that controls when a sample is taken. IC8 and its associated resistors, capacitors, and 1-MHz crystal form a stable, accurate, square-wave oscillator time base. IC10, IC11, and IC12 divide the output frequency of the oscillator so that several different sampling frequencies can be obtained. IC9 is a monostable multivibrator that provides a fixed width pulse that is synchronized to the sampling frequency. The 2N3904 transistor is used to provide a low-impedance output to the $\overline{\text{IRQ}}$ going to the 6502. I designed the clock circuitry so that sampling rates of 5 kHz, 1 kHz, 100 Hz, and 10 Hz can be obtained by closing the appropriate contacts on S1.

Construction Hints

If you have built electronic circuits before, either from scratch or from a commercially available kit, you should consider building the high-speed A/D converter. If you are careful, the chances of damaging your Apple are low and the chances of the circuit working are high. I will try to increase your probability of success by providing some advice and some specific points to check as you finish building each section.

I recommend that you buy the hobby/prototype board for the Apple II and use wire-wrap construction. This type of construction goes together fast and lends itself to easy correction of wiring errors. The cost of the wire-wrapping tools is a little high, but it is doubtful that you will ever wear them out. You can order the A/D converter ICs directly from the manufacturer; the rest of the components can be purchased from Jameco Electronics (1355 Shoreway Rd., Belmont, CA 94002, (415) 592-8097).

Start by building the crystal-controlled time-base oscillator shown in figure 5. Beg or borrow an oscilloscope and perform the following tests:

1. Initially, do not connect the $\overline{\text{IRQ}}$ line from the 2N3904 transistor to pin 30 on the hobby/prototype board.
2. With the computer turned off, plug the hobby/prototype board into peripheral I/O slot 7.
3. Turn the computer on. It should function normally. If the computer does not function normally, turn it off and pull out the hobby/prototype board. Turn the computer back on to see if normal operation has been restored. If so, you have made an error in wiring or you probably have inserted one of the ICs into a socket backward.
4. Once you get the Apple to work with the board plugged in, connect the oscilloscope to pin 2 of IC10, where you should see a distorted square wave having a frequency approximately equal to 1 MHz. Adjust the 4-40-pF trimmer capacitor until this frequency is equal to 1 MHz.
5. Measure the pulse width of the

$\overline{IRQ}$ output at the 2N3904 transistor. It should be approximately equal to 0.1 ms. If there is no output pulse at this point, work your way back toward pin 2 of IC10 until you find the square wave again. Once you find the square wave, you can be pretty sure that you have made a wiring error somewhere between that point and the $\overline{IRQ}$ line.

6. The frequency of the $\overline{IRQ}$ pulse train should change as you open and close the various switches on S1. If it does not, you should check the wiring at this point in the circuit.
7. Initialize a new disk using the Hello routine shown in listing 1. Connect the $\overline{IRQ}$ line to pin 30 on the hobby/prototype board and turn the computer on. If it does not function normally, you probably have made an error in entering Hello.

Next, wire the logic circuitry shown in figure 4 and perform the following measurements:

1. Execute the following BASIC statements; you should see the Start Conversion pulse (pin 5 of IC16) periodically go from 0 to +5 V.

```
100 POKE -16143,0
110 FOR I=0 TO 100:NEXT I
120 POKE -16143,1
130 FOR I=0 TO 100:NEXT I
140 GOTO 100
```

2. Execute the following BASIC statements; you should see the Data Strobe pulse for Input 1 (pin 9 of IC13) periodically go from 0 to +5 V.

```
100 X=PEEK(-16143)
110 FOR I=0 TO 100:NEXT I
120 GOTO 100
```

3. Execute the following BASIC statements; you should see the Data Strobe pulse for Input 2 (pin 6 of IC13) periodically go from 0 to +5 V.

```
100 X=PEEK(-16142)
110 FOR I=0 TO 100:NEXT I
120 GOTO 100
```

4. Execute the following BASIC statements; you should see the Data Strobe pulse for Input 3 (pin 10 of IC13) periodically go from 0 to +5 V.

```
100 X=PEEK(-16141)
110 FOR I=0 TO 100:NEXT I
120 GOTO 100
```

If you have made it this far, congratulations. The next phase is the most difficult to test, so be especially careful when you wire it up. For now, you should wire up the AD7570 associated with Input 1. Keep the leads between IC2 and IC5 short to minimize the tendency for the circuit to oscillate. Once you have finished building the circuit, perform the following tests to make sure it is working correctly:

1. The voltage at pin 2 of IC2 should be equal to −5 V.
2. Execute the following BASIC statements; you should see the $\overline{BUSY}$ line periodically go from 0 to +5 V.

```
100 POKE -16143,0
110 POKE -16143,1
120 POKE -16143,0
130 FOR I=0 TO 100:NEXT I
140 GOTO 100
```

If your circuit passed all these tests, there is a high probability that it is wired correctly. You will now need to test your hardware with the software routines I'll provide next month in part 2.■

Acknowledgment

This project was supported by the Human Capability Corporation of Southfield, Michigan.

Richard C. Hallgren is an associate professor in the Department of Biomechanics, Michigan State University, East Lansing, MI 48824. He works on applications of microprocessor-based systems to scientific research.

Putting the Apple II Work

Part 2: The Software

A high-speed system for the acquisition and analysis of data

Last month, I described the overall system approach and provided you with construction details and preliminary testing. In this concluding part, I'll discuss the software I've developed that makes the system operational.

System Software

The software that enables the computer to collect and display the data can best be visualized by breaking down the total program set into a number of subroutines:

1. A main routine written in Applesoft BASIC is responsible for calling all machine-language subroutines, displaying the data on the high-resolution graphics screen, and storing the data on disk.
2. A machine-language routine that controls the digital section of the analog-to-digital (A/D) converter and provides high-speed transfer of the binary data into the Apple II.
3. A machine-language routine that scrolls the displayed data horizontally across the video display.
4. A machine-language routine that enables you to mix text with the data displayed on the high-resolution graphics screen.

The Applesoft program expects the machine-language routines to be stored on disk drive 1 and to have the following names:

A/D — routine that controls the digital section of the A/D converter
Shift — routine that scrolls the data
Hires — routine that writes text onto the high-resolution graphics screen
Table — graphics character look-up table

After you have loaded these programs and stored them onto a disk initialized with the Hello routine, execute the Applesoft routine. If the program jumps to the A/D routine but never returns, you probably have one of two problems:

1. The program did not enter the A/D routine correctly. Usually, you will get strange characters appearing on the screen, and/or the keyboard will not respond without turning the power off and then back on.
2. Absolutely nothing happens. Make sure that the $\overline{\text{IRQ}}$ signal is getting to pin 30 on the interface connector.

Once you get the program to go to the A/D routine and to return, the end is in sight. If the data does not plot correctly, check the section in the Applesoft routine that supports this. For example, if you try to scroll the data and the computer does strange things, take a close look for mistakes in the scroll subroutines.

Applesoft Routine

Listing 1 gives the program with comments. This BASIC routine first loads all the machine-language routines and then loops until the operator is ready to digitize data. Once the operator indicates that data is to be taken, the program jumps to the machine-language A/D routine that proceeds to digitize and store a predetermined quantity of data. Program control then returns to the Applesoft routine. The data is then plotted on the high-resolution graphics screen, and text is added to the plots. You then have the option of reviewing the data by scrolling it back and forth across the video display. If the data is "good," you can store the data on disk. If the data is not good, you can initiate the acquisition of a new block of data.

A/D Machine-Language Routine

The machine-language A/D converter subroutine is called from the BASIC program by executing CALL −28656. This forces the computer to execute the subroutine stored at memory location 9010 hexadecimal.

(Unless otherwise indicated, all addresses are hexadecimal.) Listing 2 gives the program with comments. Upon entering this subroutine, the contents of the accumulator, the contents of the X and Y registers, and the processor status are saved. The subroutine then clears the Y register and loads the X register with the 8 most significant bits (MSBs) of the memory address defining the upper limit of the block of memory reserved for data storage. The memory address for the lower limit of the block reserved for data storage is loaded into memory locations 0A (least significant bits or LSBs) and 0B (MSBs). These two memory locations serve as a pointer to the current location in memory in which a byte of data is to be stored.

The system interrupt logic is disabled while the 8 MSBs of the current data-storage address (the contents of memory location 0B) are compared with the 8 MSBs of the maximum allowable address (the contents of the X register). If the maximum limit has not been reached, the program jumps to memory location 9038. If the maximum limit has been reached, the subroutine restores the contents of the accumulator, the contents of the X and Y registers, and the processor status. After that, the return from subroutine (RTS) command forces the computer to return to the BASIC calling routine.

At memory location 9038, the subroutine enables the system interrupt logic and waits a few machine cycles to see if it is time to take another sample. The sampling rate is determined by connecting the output of the crystal-controlled oscillator and frequency-divider logic to the interrupt request line ($\overline{IRQ}$) going to the 6502. If it is not time to take another sample, the subroutine returns to memory location 9026, where the interrupt logic is disabled. If it is time to take another sample, the interrupt logic forces the computer to jump to memory location 9040. This address was determined by the Hello program, which was executed when the DOS (disk operating system) was initially booted.

At memory location 9040, the three

Listing 1: *A/D converter main routine written in Applesoft BASIC.*

```
10  REM  HIGH SPEED A/D CONVERTER
20 D$ = ""
22  PRINT D$;"BLOAD A/D,D1"
24  PRINT D$;"BLOAD HIRES,D1"
25  PRINT D$;"BLOAD TABLE,D1"
26  PRINT D$;"BLOAD SHIFT,D1"
32  VTAB 10: PRINT "PRESS THE SPACE BAR WHEN YOU ARE": PRINT "READY TO DIG
    ITIZE DATA."
40  GET K$
42  IF K$ <  >  CHR$ (32) THEN  GOTO 40
44  GOTO 2100
100  REM  SCROLL DATA TO THE LEFT
102  IF K1 > 28600 THEN  RETURN
112  POKE  - 30875,230: POKE  - 30869,227: POKE  - 30751,0: POKE  - 30744,
    232: POKE  - 30742,28: POKE  - 30865,26: CALL  - 30976
130  HCOLOR= 1: FOR I = 1 TO 14
132 Y = ( PEEK (K1 + DI * I)) / 1.5
134  HPLOT 195 + I,175 - Y: NEXT I
136 K1 = K1 + DI * 14
140  HCOLOR= 2: FOR I = 1 TO 14
144 Y = ( PEEK (K2 + DI * I)) / 1.5
146  HPLOT 195 + I,175 - Y: NEXT I
148 K2 = K2 + DI * 14
150  HCOLOR= 3: FOR I = 1 TO 14
154 Y = ( PEEK (K3 + DI * I)) / 1.5
156  HPLOT 195 + I,175 - Y: NEXT I
158 K3 = K3 + DI * 14:SL = 1
199  RETURN
200  REM  SCROLL DATA TO THE RIGHT
202  IF K1 < 25230 THEN  RETURN
212  POKE  - 30875,227: POKE  - 30869,230: POKE  - 30751,27: POKE  - 30744
    ,202: POKE  - 30742,255: POKE  - 30865,254: CALL  - 30976
221  IF SL = 0 THEN  GOTO 230
222 K4 = K1 - 210 * DI:K5 = K2 - 210 * DI:K6 = K3 - 210 * DI
230  HCOLOR= 1: FOR I = 14 TO 1 STEP  - 1
234 Y = ( PEEK (K4 - DI * I)) / 1.5
236  HPLOT 14 - I,175 - Y: NEXT I
238 K4 = K4 - DI * 14:K1 = K4 + 210 * DI
240  HCOLOR= 2: FOR I = 14 TO 1 STEP  - 1
244 Y = ( PEEK (K5 - DI * I)) / 1.5
246  HPLOT 14 - I,175 - Y: NEXT I
248 K5 = K5 - DI * 14:K2 = K5 + 210 * DI
250  HCOLOR= 3: FOR I = 14 TO 1 STEP  - 1
254 Y = ( PEEK (K6 - DI * I)) / 1.5
256  HPLOT 14 - I,175 - Y: NEXT I
258 K6 = K6 - DI * 14:K3 = K6 + 210 * DI
299  RETURN
2100  REM  DIGITIZE DATA
2102  HOME : TEXT : VTAB 10: PRINT "DATA IS BEING DIGITIZED."
2132  POKE  - 28643,112: POKE  - 16143,0: CALL  - 28656
2200 K1 = 24576:K2 = 24577:K3 = 24578:DI = 3: GOSUB 3000: GOSUB 10000
2250  GET K$
2254  IF K$ =  CHR$ (8) THEN  GOSUB 100
2256  IF K$ =  CHR$ (21) THEN  GOSUB 200
2258  IF K$ =  CHR$ (32) THEN  GOTO 2100
2260  IF K$ =  CHR$ (27) THEN  GOTO 4000
2299  GOTO 2250
3000  REM  PLOT DATA
3010  HCOLOR= 1: HGR2
3030  FOR I = 0 TO 209:Y = ( PEEK (K1 + DI * I)) / 1.5
3032  HPLOT I,175 - Y: NEXT I
3034 K4 = K1:K1 = K1 + 210 * DI
3036  HCOLOR= 2
3038  FOR I = 0 TO 209:Y = ( PEEK (K2 + DI * I)) / 1.5
3040  HPLOT I,175 - Y: NEXT I
3041 K5 = K2:K2 = K2 + 210 * DI
3042  HCOLOR= 3
3044  FOR I = 0 TO 209:Y = ( PEEK (K3 + DI * I)) / 1.5
3046  HPLOT I,175 - Y: NEXT I
3048 K6 = K3:K3 = K3 + 210 * DI
3049  RETURN
4000  REM  ESCAPE SUBROUTINE
4002  TEXT : HOME
4010  VTAB 4: PRINT "PRESS THE KEY CORRESPONDING TO YOUR": PRINT "CHOICE:"
4014  VTAB 10: PRINT "R  TO RETURN TO CURRENT DATA"
4016  VTAB 12: PRINT "S  TO SAVE CURRENT DATA ON DISK"
4018  VTAB 14: PRINT "D  TO DIGITIZE NEW DATA"
4019  VTAB 16: PRINT "H  TO STOP"
4020  VTAB 20: GET K$
4022  IF K$ = "D" THEN  GOTO 2100
4023  IF K$ = "R" THEN  POKE  - 16304,0: POKE  - 16299,0: POKE  - 16297,0:
    GOTO 2250
4024  IF K$ = "R" THEN  POKE  - 16304,0: POKE  - 16299,0: POKE  - 16297,0:
    GOTO 2250
4026  IF K$ = "H" THEN  END
4028  IF K$ = "S" THEN  GOTO 4050
4029  GOTO 4020
4050  HOME
```

Listing 1 continued on page 384

AD7570 A/D converters are simultaneously instructed to begin the conversion of their respective input signals. The subroutine then loops until all three units have finished their conversion cycles. The subroutine then proceeds to load the digitized signal from the first AD7570 into the accumulator. The contents of the accumulator are then transferred into the memory location determined by the contents of memory locations 0A (containing the 8 LSBs) and 0B (containing the 8 MSBs) and the contents of register Y (which are added to the contents of memory location 0A).

After the data has been stored, the Y register is incremented. The subroutine tests the Y register to see if the increment caused the register to be equal to zero (a transition from # FF to #00). Such a transition indicates that memory location 0B then needs to be incremented. The subroutine then proceeds to load and store data into successive memory locations until all three converters have been serviced. A return from interrupt (RTI) command then forces the computer to return to the point in the program where the interrupt request was detected. The subroutine ultimately ends up back at memory location 9026, where the interrupt logic is again disabled and a test is made to see if the maximum allocated data-storage address has been exceeded.

Once the data has been digitized and stored, program control returns to the BASIC routine. The first 209 data samples from each input channel are displayed on the high-resolution graphics screen. Differentiation of the data is achieved by using a unique color for each input channel. The full width of the graphics display is not utilized for data so that reference text can be added on the right-hand side of the screen.

High-Resolution Text Generator

The text-generator software is used to write textual information on the high-resolution graphics screen. This capability lets you identify data points and display the magnitude of selected data points along with the data. The character set for the graphics generator was purposely limited

Listing 1 continued:

```
4060  VTAB 10: PRINT "ENTER THE NAME OF THE DATA FILE"
4064  VTAB 14: INPUT K$
4070 D$ = ""
4072  PRINT D$;"BSAVE ";K$;",A$6000,L$1000,D1"
4099  GOTO 4000
10000  REM  IDENTIFY PLOTS AND ADD TEXT
10002  POKE 54,0: POKE 55,143: POKE  - 16299,0
10010  VTAB 23: HTAB 1: PRINT "PRESS <-- OR --> TO SCROLL THE DATA."
10050  VTAB 24: HTAB 1: PRINT "PRESS SPACE BAR TO DIGITIZE MORE DATA."
10052  VTAB 14: HTAB 32: PRINT "PRESS ESC"
10054  VTAB 15: HTAB 32: PRINT "TO EXIT."
10060  HCOLOR= 1
10062  HPLOT 215,12 TO 219,12: HPLOT 215,20 TO 219,20: HPLOT 217,12 TO 217
     ,20: HPLOT 223,20 TO 223,12 TO 227,20 TO 227,12: HPLOT 231,20 TO 231,
     12 TO 235,12 TO 235,16 TO 231,16
10064  HPLOT 239,12 TO 239,20 TO 243,20 TO 243,12: HPLOT 249,20 TO 249,12 TO
     247,12 TO 251,12
10066  HPLOT 257,20 TO 261,20 TO 259,20 TO 259,12 TO 257,14
10070  HCOLOR= 2
10072  HPLOT 216,32 TO 220,32: HPLOT 216,40 TO 220,40: HPLOT 218,32 TO 218
     ,40: HPLOT 224,40 TO 224,32 TO 228,40 TO 228,32: HPLOT 232,40 TO 232,
     32 TO 236,32 TO 236,36 TO 232,36
10074  HPLOT 240,32 TO 240,40 TO 244,40 TO 244,32: HPLOT 250,40 TO 250,32 TO
     248,32 TO 252,32
10076  HPLOT 258,32 TO 262,32 TO 262,36 TO 258,36 TO 258,40 TO 262,40
10080  HCOLOR= 3
10082  HPLOT 216,52 TO 220,52: HPLOT 216,60 TO 220,60: HPLOT 218,52 TO 218
     ,60: HPLOT 224,60 TO 224,52 TO 228,60 TO 228,52: HPLOT 232,60 TO 232,
     52 TO 236,52 TO 236,56 TO 232,56
10084  HPLOT 240,52 TO 240,60 TO 244,60 TO 244,52: HPLOT 250,60 TO 250,52 TO
     248,52 TO 252,52
10086  HPLOT 258,52 TO 262,52 TO 262,56 TO 258,56 TO 262,56 TO 262,60 TO 2
     58,60
10099  RETURN
```

Listing 2: *This routine provides high-speed data transfer from the A/D converter to the Apple II.*

```
9010    8D 00 90    STA   $9000     Save accumulator
9013    8E 01 90    STX   $9001     Save X register
9016    8C 02 90    STY   $9002     Save Y register
9019    08          PHP             Save processor status
901A    A0 00       LDY   #$00
901C    A2 63       LDX   #$70      Load X register with maximum
                                    data storage address
901E    A9 00       LDA   #$00
9020    85 0A       STA   $0A       Memory locations $0A and $0B
9022    A9 60       LDA   #$60      contain the start address for
9024    85 0B       STA   $0B       data storage
9026    78          SEI             Disable interrupt
9027    E4 0B       CPX   $0B       Compare current data storage
9029    D0 0D       BNE   $9038     address with maximum address
902B    AD 00 90    LDA   $9000     Restore accumulator
902E    AE 01 90    LDX   $9001     Restore X register
9031    AC 02 90    LDY   $9002     Restore Y register
9034    28          PLP             Restore processor status
9035    60          RTS             Return to calling routine
9036    EA          NOP
9037    EA          NOP
9038    58          CLI             Enable interrupt
9039    EA          NOP
903A    4C 26 90    JMP   $9026
903D    00          BRK
903E    00          BRK
903F    00          BRK
9040    A9 01       LDA   #$01      Start A/D conversion
9042    8D F0 C0    STA   $C0F0
9045    A9 00       LDA   #$00
9047    8D F0 C0    STA   $C0F0
904A    AD 61 C0    LDA   $C061     Check and see if all conversions
904D    2A          ROL             are complete
904E    B0 FA       BCS
9050    AD F1 C0    LDA   $C0F1     Load data from input #1
9053    91 0A       STA   ($0A),Y   Store data
9055    C8          INY             Increment LSD of data storage
                                    address
9056    D0 02       BNE   $905A     Branch on result not zero
9058    E6 0B       INC   $0B       Increment MSD of data storage
                                    address
905A    AD F2 C0    LDA   $C0F2     Load data from input #2
905D    91 0A       STA   ($0A),Y   Store data
905F    C8          INY             Increment LSD
9060    D0 02       BNE   $9064     Branch on result not zero
9062    E6 0B       INC   $0B       Increment MSD
```

Listing 2 continued on page 386

to numbers and uppercase letters to conserve memory. Listing 3 gives the high-resolution graphics, text-generator program, and table 1 is the graphics character look-up table. The program takes the textual character that is to be displayed on the graphics screen and matches it to a corresponding graphics character contained in the look-up table. This graphics character is then displayed on the screen by loading it into the correct memory location in page 2 of the high-resolution-graphics memory block. By using this subroutine, you avoid having to "draw" text on the graphics screen using the PLOT commands. The routine is initialized by using POKEs to insert the subroutine entry address into memory locations (decimal) 54 and 55. Any PRINT statements that follow will force the text that was to be printed to be displayed on the graphics screen.

Data-Scroll Routine

The information that is routed to the video display when the Apple is in the high-resolution-graphics mode comes from an 8192-byte block of memory that is defined (for the secondary picture-page buffer) between memory locations 4000 and 5FFF (see figure 1). The rationale that determines the relationship between a dot's position on the screen and the dot's position in the picture-page buffer is not all that obvious to me. The best that I have been able to do is to map out the relationship between a dot's position on the screen and a memory-address location in the picture-page buffer.

Seven of the 8 bits in each byte contained in the picture-page buffer are displayed as dots; the eighth bit determines the color of the other 7 dots. A total of 40 bytes is displayed on each horizontal line of the video display. The LSB of the first byte in a line is displayed on the left-hand edge of the screen, followed by the second bit, the third bit, etc. A total of 280 dots (40 bytes × 7 dots) is displayed on each of the 192 lines (24 lines × 8 dots) that can be displayed on the screen.

In order to help myself understand the picture-page memory map, I con-

Listing 2 continued:

```
9064    AD F3 C0    LDA    $C0F3      Load data from input #3
9067    91 0A       STA    ($0A),Y    Store data
9069    C8          INY               Increment LSD
906A    D0 02       BNE    $906E      Branch on result not zero
906C    E6 0B       INC    $0B        Increment MSD
906E    40          RTI               Return from interrupt
```

Listing 3: *High-resolution text-generator routine.*

```
8F00    08          PHP               Save processor status
8F01    48          PHA               Save contents of accumulator
8F02    84 4E       STY    $4E        Save contents of Y register
8F04    C9 8D       CMP    #$8D       Test for carriage return
8F06    F0 07       BEQ    $8F0F
8F08    C9 8C       CMP    #$8C       Test for line feed
8F0A    D0 05       BNE    $8F11
8F0C    18          CLC
8F0D    90 5C       BCC    $8F6B
8F0F    F0 5C       BEQ    $8F6D
8F11    A5 25       LDA    $25        Relate cursor position to HGR2
                                      screen position
8F13    4A          LSR
8F14    29 03       AND    #$03
8F16    09 40       ORA    #$40       Define HGR page #2
8F18    85 2B       STA    $2B
8F1A    A5 25       LDA    $25
8F1C    6A          ROR
8F1D    08          PHP
8F1E    0A          ASL
8F1F    29 18       AND    #$18
8F21    85 2A       STA    $2A
8F23    0A          ASL
8F24    0A          ASL
8F25    05 2A       ORA    $2A
8F27    0A          ASL
8F28    28          PLP
8F29    6A          ROR
8F2A    18          CLC
8F2B    65 24       ADC    $24
8F2D    85 2A       STA    $2A
8F2F    68          PLA
8F30    29 7F       AND    #$7F
8F32    48          PHA
8F33    A9 88       LDA    #$88       MSB of graphics character
                                      look-up table
8F35    4A          LSR
8F36    4A          LSR
8F37    4A          LSR
8F38    85 27       STA    $27
8F3A    68          PLA                Match text character to graphics
                                      character position in look-up
                                      table
8F3B    48          PHA
8F3C    2A          ROL
8F3D    26 27       ROL    $27
8F3F    2A          ROL
8F40    26 27       ROL    $27
8F42    2A          ROL
8F43    26 27       ROL    $27
8F45    29 F8       AND    #$F8
8F47    85 26       STA    $26
8F49    A0 00       LDY    #$00
8F4B    B1 26       LDA    ($26),Y    Get first row of graphics design
                                      from look-up table
8F4D    84 4F       STY    $4F
8F4F    A0 00       LDY    #$00
8F51    48          PHA
8F52    68          PLA
8F53    51 2A       EOR    ($2A),Y
8F55    91 2A       STA    ($2A),Y    Store graphics design in screen
                                      memory block
8F57    A4 4F       LDY    $4F
8F59    A5 2B       LDA    $2B
8F5B    18          CLC
8F5C    69 04       ADC    #$04
8F5E    85 2B       STA    $2B
8F60    C8          INY
8F61    C0 08       CPY    #$08
8F63    D0 E6       BNE    $8F4B      Jump if all rows not finished
8F65    E6 24       INC    $24        Increment LSD of cursor position
8F67    A5 24       LDA    $24
8F69    C5 21       CMP    $21
8F6B    90 10       BCC    $8F7D
```

Listing 3 continued on page 388

Listing 3 continued:

```
8F6D    A5 20       LDA    $20
8F6F    85 24       STA    $24
8F71    E6 25       INC    $25       Increment MSD of cursor position
8F73    A5 25       LDA    $25
8F75    C5 23       CMP    $23
8F77    90 04       BCC    $8F7D
8F79    A5 22       LDA    $22
8F7B    85 25       STA    $25
8F7D    A4 4E       LDY    $4E       Restore Y register
8F7F    68          PLA              Restore accumulator
8F80    28          PLP              Restore processor status
8F81    60          RTS              Return to calling routine
```

8900-	00	00	00	00	00	00	00	00	Space
8908-	10	10	10	10	00	00	10	00	!
8910-	24	24	24	00	00	00	00	00	"
8918-	24	24	7E	24	7E	24	24	00	#
8920-	10	78	14	38	50	3C	10	00	$
8928-	00	46	26	10	08	64	62	00	%
8930-	0C	12	12	0C	52	22	5C	00	&
8938-	20	10	08	00	00	00	00	00	'
8940-	20	10	08	08	08	10	20	00	(
8948-	04	08	10	10	10	08	04	00	)
8950-	10	54	38	7C	38	54	10	00	*
8958-	00	10	10	7C	10	10	00	00	+
8960-	00	00	00	00	00	18	18	0C	'
8968-	00	00	00	7E	00	00	00	00	-
8970-	00	00	00	00	00	18	18	00	.
8978-	00	40	20	10	08	04	02	00	/
8980-	3C	42	62	5A	46	42	3C	00	0
8988-	10	18	14	10	10	10	7C	00	1
8990-	3C	42	40	30	0C	02	7E	00	2
8998-	3C	42	40	38	40	42	3C	00	3
89A0-	20	30	28	24	7E	20	20	00	4
89A8-	7E	02	1E	20	40	22	1C	00	5
89B0-	38	04	02	3E	42	42	3C	00	6
89B8-	7E	42	20	10	08	08	08	00	7
89C0-	3C	42	42	3C	42	42	3C	00	8
89C8-	3C	42	42	7C	40	20	1C	00	9
89D0-	00	00	18	18	00	18	18	00	:
89D8-	00	00	18	18	00	18	18	0C	;
89E0-	20	10	08	04	08	10	20	00	<
89E8-	00	00	3E	00	3E	00	00	00	=
89F0-	04	08	10	20	10	08	04	00	>
89F8-	3C	42	40	30	08	00	08	00	?
8A00-	38	44	52	6A	32	04	78	00	
8A08-	18	24	42	7E	42	42	42	00	A
8A10-	3E	44	44	3C	44	44	3E	00	B
8A18-	3C	42	02	02	02	42	3C	00	C
8A20-	3E	44	44	44	44	44	3E	00	D
8A28-	7E	02	02	1E	02	02	7E	00	E
8A30-	7E	02	02	1E	02	02	02	00	F
8A38-	3C	42	02	72	42	42	3C	00	G
8A40-	42	42	42	7E	42	42	42	00	H
8A48-	38	10	10	10	10	10	38	00	I
8A50-	70	20	20	20	20	22	1C	00	J
8A58-	42	22	12	0E	12	22	42	00	K
8A60-	02	02	02	02	02	02	7E	00	L
8A68-	42	66	5A	5A	42	42	42	00	M
8A70-	42	46	4A	52	62	42	42	00	N
8A78-	3C	42	42	42	42	42	3C	00	O
8A80-	3E	42	42	3E	02	02	02	00	P
8A88-	3C	42	42	42	52	22	5C	00	Q
8A90-	3E	42	42	3E	12	22	42	00	R

Table 1: *Graphics character look-up table.*

Table 1 continued:

8A98-	3C	42	02	3C	40	42	3C	00	S
8AA0-	7C	10	10	10	10	10	10	00	T
8AA8-	42	42	42	42	42	42	3C	00	U
8AB0-	42	42	42	24	24	18	18	00	V
8AB8-	42	42	42	5A	5A	66	42	00	W
8AC0-	42	42	24	18	24	42	42	00	X
8AC8-	44	44	44	38	10	10	10	00	Y
8AD0-	7E	40	20	18	04	02	7E	00	Z

Listing 4: *Right-to-left scroll routine.*

```
8700    A9 00       LDA    #$00      Initialize base address
8702    8D FE 87    STA    $87FE
8705    A9 40       LDA    #$40
8707    8D FF 87    STA    $87FF
870A    A9 02       LDA    #$02      Initialize block counter
870C    8D FD 87    STA    $87FD
870F    A0 08       LDA    #$08      Initialize box counter
8711    8D F7 87    STA    $87F7
8714    20 50 87    JSR    $8750     Jump to main routine
8717    18          CLC
8718    A9 28       LDA    #$28      Set up for second block
871A    6D FE 87    ADC    $87FE
871D    8D FE 87    STA    $87FE
8720    CE FD 87    DEC    $87FD
8723    D0 EA       BNE    $870F     Jump if second block not
                                        complete
8725    A9 06       LDA    #$06      Number of boxes remaining (two
                                        boxes reserved for text)
8727    8D F7 87    STA    $87F7
872A    20 50 87    JSR    $8750     Jump to main routine
872D    60          RTS              Return to calling routine
8750    AD FE 87    LDA    $87FE     Save base address
8753    8D FB 87    STA    $87FB
8756    AD FF 87    LDA    $87FF
8759    8D FC 87    STA    $87FC
875C    A9 08       LDA    #$08      Initialize row counter
875E    8D FA 87    STA    $87FA
8761    AD FB 87    LDA    $87FB     Set up LSB of left hand side of
                                        screen
8764    8D E6 87    STA    $87E6
8767    18          CLC
8768    69 02       ADC    #$02      Set up shift distance
876A    8D E3 87    STA    $87E3
876D    18          CLC
876E    69 1A       ADC    #$1A
8770    8D F0 87    STA    $87F0     Set up LSB of right hand side of
                                        screen
8773    8D F3 87    STA    $87F3
8776    EE F3 87    INC    $87F3     Next byte
8779    AD FC 87    LDA    $87FC     Set up MSB of
877C    8D E4 87    STA    $87E4        left hand side of screen
877F    8D E7 87    STA    $87E7
8782    8D F1 87    STA    $87F1        right hand side of screen
8785    8D F4 87    STA    $87F4
8788    20 E0 87    JSR    $87E0     Jump to shift routine
878B    18          CLC
878C    A9 04       LDA    #$04      Add 4 to MSB of
878E    6D E4 87    ADC    $87E4
8791    8D E4 87    STA    $87E4        left hand side of screen
8794    8D E7 87    STA    $87E7
8797    8D F1 87    STA    $87F1        right hand side of screen
879A    8D F4 87    STA    $87F4
879D    CE FA 87    DEC    $87FA     Decrement row counter
87A0    D0 E6       BNE    $8788     Jump if box not complete
87A2    18          CLC
87A3    A9 80       LDA    #$80      Set up next box address
87A5    6D FB 87    ADC    $87FB
87A8    8D FB 87    STA    $87FB
87AB    A9 00       LDA    #$00
87AD    6D FC 87    ADC    $87FC
87B0    8D FC 87    STA    $87FC
87B3    CE F7 87    DEC    $87F7     Decrement box counter
87B6    D0 A4       BNE    $875C     Jump if block not complete
87B8    60          RTS              Return to calling routine
```

Listing 4

Listing 4 continued:

```
87E0    A2 00       LDX   #$00       Set up byte counter
87E2    BD 02 40    LDA   $4002,X    Shift 2 bytes (14 points) left
87E5    9D 00 40    STA   $4000,X
87E8    E8          INX               Increment counter
87E9    E0 1C       CPX   #$1C
87EB    D0 F5       BNE   $87E2       Jump if shift not complete
87ED    A9 00       LDA   #$00        Clear right most 14 points
87EF    8D 1C 40    STA   $401C
87F2    8D 1D 40    STA   $401D
87F5    60          RTS               Return to calling routine
```

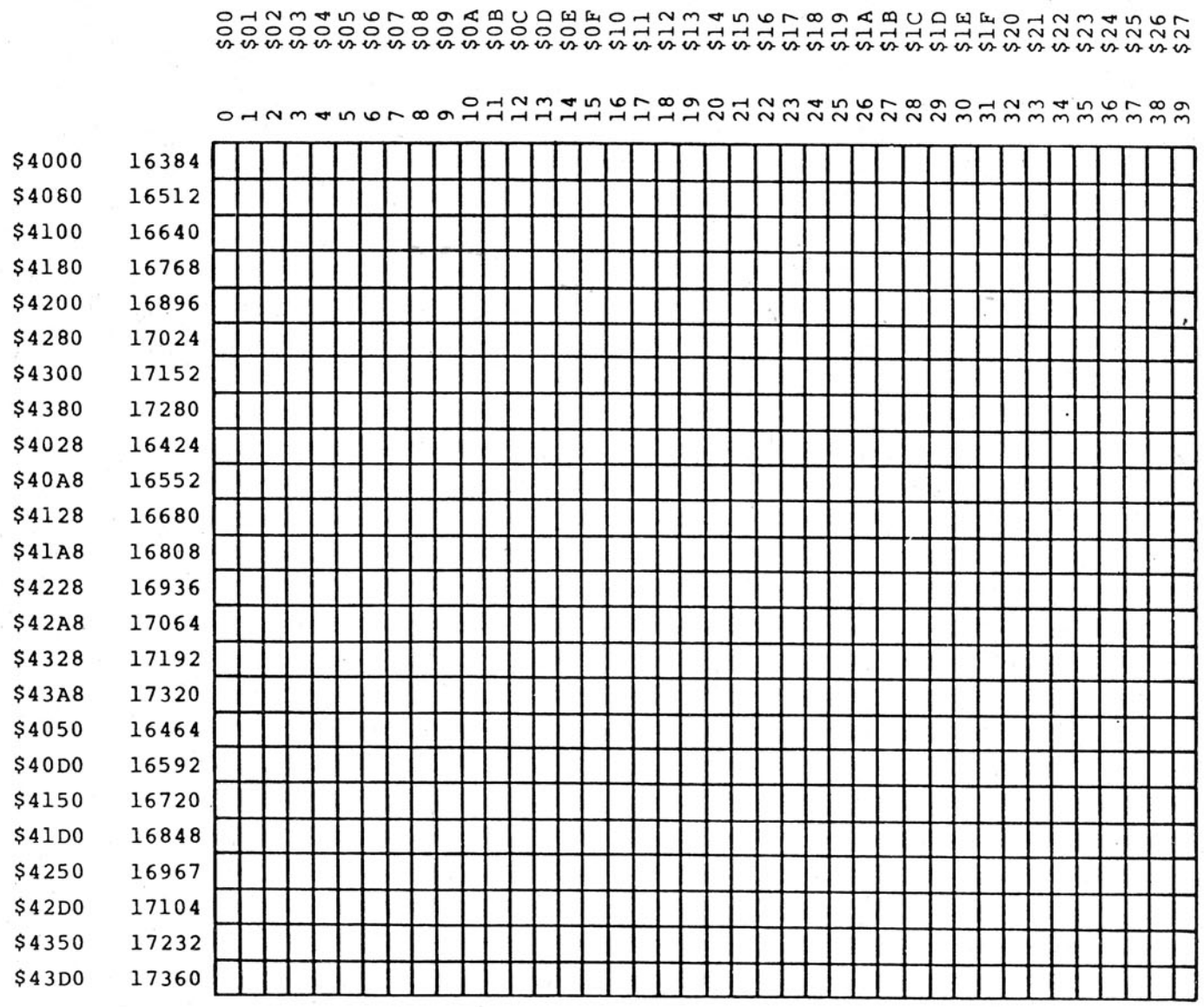


Figure 1: *A map of the Apple II's high-resolution graphics screen.*

		Row 1	$4000			Row 1	$4028			Row 1	$4050
		Row 2	$4400			Row 2	$4428			Row 2	$4450
		Row 3	$4800			Row 3	$4828			Row 3	$4850
	Box 1	Row 4	$4C00		Box 1	Row 4	$4C28		Box 1	Row 4	$4C50
		Row 5	$5000			Row 5	$5028			Row 5	$5050
		Row 6	$5400			Row 6	$5428			Row 6	$5450
		Row 7	$5800			Row 7	$5828			Row 7	$5850
		Row 8	$5C00			Row 8	$5C28			Row 8	$5C50
		Row 1	$4080			Row 1	$40A8			Row 1	$40D0
		Row 2	$4480			Row 2	$44A8			Row 2	$44D0
		Row 3	$4880			Row 3	$48A8			Row 3	$48D0
	Box 2	Row 4	$4C80		Box 2	Row 4	$4CA8		Box 2	Row 4	$4CD0
		Row 5	$5080			Row 5	$50A8			Row 5	$50D0
		Row 6	$5480			Row 6	$54A8			Row 6	$54D0
		Row 7	$5880			Row 7	$58A8			Row 7	$58D0
		Row 8	$5C80			Row 8	$5CA8			Row 8	$5CD0
		Row 1	$4100			Row 1	$4128			Row 1	$4150
		Row 2	$4500			Row 2	$4528			Row 2	$4550
		Row 3	$4900			Row 3	$4928			Row 3	$4950
	Box 3	Row 4	$4D00		Box 3	Row 4	$4D28		Box 3	Row 4	$4D50
		Row 5	$5100			Row 5	$5128			Row 5	$5150
		Row 6	$5500			Row 6	$5528			Row 6	$5550
		Row 7	$5900			Row 7	$5928			Row 7	$5950
		Row 8	$5D00			Row 8	$5D28			Row 8	$5D50
		Row 1	$4180			Row 1	$41A8			Row 1	$41D0
		Row 2	$4580			Row 2	$45A8			Row 2	$45D0
		Row 3	$4980			Row 3	$49A8			Row 3	$49D0
Block 1	Box 4	Row 4	$4D80	Block 2	Box 4	Row 4	$4DA8	Block 3	Box 4	Row 4	$4DD0
		Row 5	$5180			Row 5	$51A8			Row 5	$51D0
		Row 6	$5580			Row 6	$55A8			Row 6	$55D0
		Row 7	$5980			Row 7	$59A8			Row 7	$59D0
		Row 8	$5D80			Row 8	$5DA8			Row 8	$5DD0
		Row 1	$4200			Row 1	$4228			Row 1	$4250
		Row 2	$4600			Row 2	$4628			Row 2	$4650
		Row 3	$4A00			Row 3	$4A28			Row 3	$4A50
	Box 5	Row 4	$4E00		Box 5	Row 4	$4E28		Box 5	Row 4	$4E50
		Row 5	$5200			Row 5	$5228			Row 5	$5250
		Row 6	$5600			Row 6	$5628			Row 6	$5650
		Row 7	$5A00			Row 7	$5A28			Row 7	$5A50
		Row 8	$5E00			Row 8	$5E28			Row 8	$5E50
		Row 1	$4280			Row 1	$42A8			Row 1	$42D0
		Row 2	$4680			Row 2	$46A8			Row 2	$46D0
		Row 3	$4A80			Row 3	$4AA8			Row 3	$4AD0
	Box 6	Row 4	$4E80		Box 6	Row 4	$4EA8		Box 6	Row 4	$4ED0
		Row 5	$5280			Row 5	$52A8			Row 5	$52D0
		Row 6	$5680			Row 6	$56A8			Row 6	$56D0
		Row 7	$5A80			Row 7	$5AA8			Row 7	$5AD0
		Row 8	$5E80			Row 8	$5EA8			Row 8	$5ED0
		Row 1	$4300			Row 1	$4328			Row 1	$4350
		Row 2	$4700			Row 2	$4728			Row 2	$4750
		Row 3	$4B00			Row 3	$4B28			Row 3	$4B50
	Box 7	Row 4	$4F00		Box 7	Row 4	$4F28		Box 7	Row 4	$4F50
		Row 5	$5300			Row 5	$5328			Row 5	$5350
		Row 6	$5700			Row 6	$5728			Row 6	$5750
		Row 7	$5B00			Row 7	$5B28			Row 7	$5B50
		Row 8	$5F00			Row 8	$5F28			Row 8	$5F50
		Row 1	$4380			Row 1	$43A8			Row 1	$43D0
		Row 2	$4780			Row 2	$47A8			Row 2	$47D0
		Row 3	$4B80			Row 3	$4BA8			Row 3	$4BD0
	Box 8	Row 4	$4F80		Box 8	Row 4	$4FA8		Box 8	Row 4	$4FD0
		Row 5	$5380			Row 5	$53A8			Row 5	$53D0
		Row 6	$5780			Row 6	$57A8			Row 6	$57D0
		Row 7	$5B80			Row 7	$5BA8			Row 7	$5BD0
		Row 8	$5F80			Row 8	$5FA8			Row 8	$5FD0

Table 2: *Picture-page buffer/memory-address organization as discussed in the text.*

Listing 5: *Left-to-right scroll routine.*

```
8700    A9 00       LDA    #$00      Initialize base address
8702    8D FE 87    STA    $87FE
8705    A9 40       LDA    #$40
8707    8D FF 87    STA    $87FF
870A    A9 02       LDA    #$02      Initialize block counter
870C    8D FD 87    STA    $87FD
870F    A0 08       LDA    #$08      Initialize box counter
8711    8D F7 87    STA    $87F7
8714    20 50 87    JSR    $8750     Jump to main routine
8717    18          CLC
8718    A9 28       LDA    #$28      Set up for second block
871A    6D FE 87    ADC    $87FE
871D    8D FE 87    STA    $87FE
8720    CE FD 87    DEC    $87FD
8723    D0 EA       BNE    $870F     Jump if second block not
                                        complete
8725    A9 06       LDA    #$06      Number of boxes remaining (two
                                        boxes reserved for text)
8727    8D F7 87    STA    $87F7
872A    20 50 87    JSR    $8750     Jump to main routine
872D    60          RTS              Return to calling routine
8750    AD FE 87    LDA    $87FE     Save base address
8753    8D FB 87    STA    $87FB
8756    AD FF 87    LDA    $87FF
8759    8D FC 87    STA    $87FC
875C    A9 08       LDA    #$08      Initialize row counter
875E    8D FA 87    STA    $87FA
8761    AD FB 87    LDA    $87FB     Set up LSB of right hand side of
                                        screen
8764    8D E3 87    STA    $87E3
8767    18          CLC
8768    69 02       ADC    #$02      Set up shift distance
876A    8D E6 87    STA    $87E6
876D    18          CLC
876E    69 FE       ADC    #$FE
8770    8D F0 87    STA    $87F0     Set up LSB of left hand side of
                                        screen
8773    8D F3 87    STA    $87F3
8776    EE F3 87    INC    $87F3     Next byte
8779    AD FC 87    LDA    $87FC     Set up MSB of
877C    8D E4 87    STA    $87E4        right hand side of screen
877F    8D E7 87    STA    $87E7
8782    8D F1 87    STA    $87F1        left hand side of screen
8785    8D F4 87    STA    $87F4
8788    20 E0 87    JSR    $87E0     Jump to shift routine
878B    18          CLC
878C    A9 04       LDA    #$04      Add 4 to MSB of
878E    6D E4 87    ADC    $87E4
8791    8D E4 87    STA    $87E4        right hand side of screen
8794    8D E7 87    STA    $87E7
8797    8D F1 87    STA    $87F1        left hand side of screen
879A    8D F4 87    STA    $87F4
879D    CE FA 87    DEC    $87FA     Decrement row counter
87A0    D0 E6       BNE    $8788     Jump if box not complete
87A2    18          CLC
87A3    A9 80       LDA    #$80      Set up next box address
87A5    6D FB 87    ADC    $87FB
87A8    8D FB 87    STA    $87FB
87AB    A9 00       LDA    #$00
87AD    6D FC 87    ADC    $87FC
87B0    8D FC 87    STA    $87FC
87B3    CE F7 87    DEC    $87F7     Decrement box counter
87B6    D0 A4       BNE    $875C     Jump if block not complete
87B8    60          RTS              Return to calling routine
87E0    A2 1B       LDX    #$1B      Set up byte counter
87E2    BD 00 40    LDA    $4000,X   Shift 2 bytes (14 points) right
87E5    9D 02 40    STA    $4002,X
87E8    CA          DEX              Decrement counter
87E9    E0 FF       CPX    #$FF
87EB    D0 F5       BNE    $87E2     Jump if shift not complete
87ED    A9 00       LDA    #$00      Clear left most 14 points
87EF    8D 00 40    STA    $4000
87F2    8D 01 40    STA    $4001
87F5    60          RTS              Return to calling routine
```

Text continued from page 386:

sider the total display to be made up of three *blocks;* each block is made up of eight *boxes;* each box is made up of eight *rows*. Table 2 shows a breakdown of the picture buffer organized so that each row has a memory address associated with it that defines the leftmost 7 dots (plus the associ-

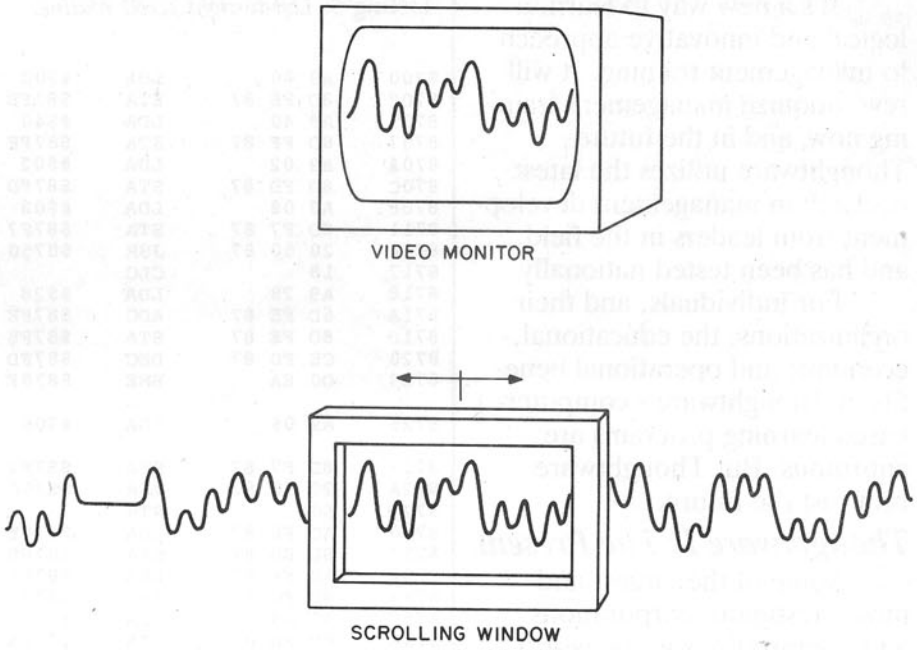


Figure 2: *A representation of how the scrolling-window software described in the text relates to the data displayed on the video monitor.*

ated color bit) for each horizontal line displayed on the screen. Notice that the memory address for each horizontal line on the display is not in sequential order with respect to magnitude, but that there is a repeating pattern.

The data-scroll routines let you control a window that permits examination of blocks of 209 adjacent samples of data. The position of this window is controlled by the left and right arrow keys (see figure 2). The data-scroll routines are broken up into two machine-language programs. Listing 4 gives the machine-language program that shifts data from right to left across the screen; listing 5 gives the routine that shifts data from left to right.

Without going into exhaustive detail, these routines move the contents of the picture-buffer memory so that the displayed data shifts either 14 data points to the left or the right on the screen. The rightmost (or leftmost) 14 data points are cleared so that new data can then be shifted in. The subroutines have to take into consideration the picture-buffer structure shown in table 2 (it would have been a lot easier if the picture buffer had been organized in a simple sequential manner). The shifting effect results in a window that can move back and forth across the memory block containing the digitized data.

Conclusion

I encourage those of you with modest data-acquisition and data-analysis requirements to consider the use of a system similar to the one described here. In our laboratory, we have found it to be a relatively inexpensive way to pursue research interests and have no doubt that it will continue to be a valued part of our laboratory in the years to come. The only items required are an Apple II and the circuitry and listings presented here.■

Richard C. Hallgren is an associate professor in the Department of Biomechanics, Michigan State University, East Lansing, MI 48824. He works on applications of microprocessor-based systems to scientific research.

Author's Note: *If you do not have either the time or capability to construct such a project, please write to me and I will direct you to a source for the hardware and the system software.*