Behavior Research Methods, Instruments, & Computers
1986, 18 (2), 214-221

SESSION VII
DATA ACQUISITION AND TRANSFER

Paper Session
Jonathan Vaughan, *Presider*

An inexpensive real-time microcomputer-based cognitive laboratory system

ROGER RATCLIFF
Northwestern University, Evanston, Illinois

CHRISTOPH PINO
Yale University, New Haven, Connecticut

and

W. TODD BURNS
Northwestern University, Evanston, Illinois

A computer system, based on the Radio Shack Color Computer, for running experiments in cognitive psychology is described. The system was designed according to the following principles: first, all of the equipment should consist of inexpensive, off-the-shelf components; second, the language used to implement experiments should have real-time commands embedded within the character strings to be displayed, and these real-time commands should be interpreted at run time; third, the system should serve multiple subjects, yet one host should be able to run display terminals for several subjects on independent experiments; and fourth, the system should be able to interface to other display devices and other response-recording devices. Two examples of other devices are discussed: an oscilloscope for rapid visual display and an Apple Macintosh for display of pictures.

We designed a real-time cognitive laboratory system, keeping several constraints in mind. First, the system was based on inexpensive off-the-shelf components, because it is easier to obtain grant money in psychology to support salaries (e.g., programmers) than to buy equipment; thus, it is more advantageous to design a system that requires significant programmer effort than one that requires expensive hardware. Furthermore, if the equipment is inexpensive and off-the-shelf, then failed components will be relatively easy to replace. (In addition, by using off-the-shelf components, one is not dependent on the electronics technician who designed the hardware. This can be particularly disastrous if the technician resigns without leaving adequate circuit diagrams or documentation.) Second, the system was written in assembly language, because some of our applications require millisecond accuracy in timing and because some of the operations required in experiments are quite complex (e.g., instructions that branch on accuracy or reaction time of the response). Timing loops written in BASIC would be sufficient for some applications, but the speed of assembly language is needed for other applications. Third, the language used to perform real-time operations was designed to be interpretive, with real-time commands embedded within the stimulus materials (e.g., text or other characters). With previous experience, we have found that an interpretive language is easiest to use because the generation of stimulus lists is decoupled from debugging the real-time component of the experiment. This language has proved extremely simple for graduate and undergraduate students to learn and use.

Support for this project was provided by Grant BNS 8203061 from the National Science Foundation, Grant HD18812 from the National Institute of Health, and internal funds from Yale University and Northwestern University. We thank Craig Murphy and Jeff Blaschak for programming assistance and Gail McKoon for comments on this manuscript and for help in testing the system. Reprint requests can be obtained from R. Ratcliff, Psychology Department, Northwestern University, Kresge Hall, Evanston, IL 60201.

The programs for the Color Computer and Apple II+ are available on floppy disk (at nominal cost to cover materials). We caution that except for someone familiar with organizing such a system, it is easy to make mistakes and fail to get the system running.

HARDWARE AND COMMUNICATION

The Radio Shack Color Computer was chosen because it offered a low price, a powerful CPU (6809), and a serious operating system for assembly language development, FLEX, and useful peripheral controllers. Reed (1979, 1982) described other advantages, for example, access to a horizontal-scan video signal (an interrupt) that allows accurate timing.

The configuration of the experimental system was designed as a network of Color Computers serviced by a single host. We did not want to use one stand-alone machine with one floppy disk drive per subject station because it would double both the cost and the hardware complexity (a multipak interface box and a disk drive would be required). In addition, an experimenter running four or five subject stations would be continually swapping floppy disks, a process prone to error when the laboratory has several experiments running at once.

We decided on a network, shown in Figure 1, in which one host computer (an Apple II+ with a hard disk) stores the stimulus lists and receives the data. This host serves several Color Computers, each using a terminal (Datamedia Elites and ADDS Viewpoints) for stimulus presentation, and each having the ability to run an independent experiment. The key to this network scheme is a Giltronix Autoscan unit, which allows one host to connect to several slaves through RS-232 serial lines running at 19.2 Kbaud. Radio Shack markets an RS-232 cartridge pack for the Color Computer with the hardware UART (i.e., the character-to-bit string and reverse conversions are carried out in hardware) that we use for communication. The built-in RS-232 port in the Color Computer converts characters to bit strings and vice versa in software, and drives the display terminal.

Each slave Color Computer can request access to the host by raising the voltage on the RTS handshake line. (Essentially, handshake lines are lines separate from the transmitting and receiving lines and have the function of signaling between the two communicating systems.) If the host is not already connected to another Color Computer,

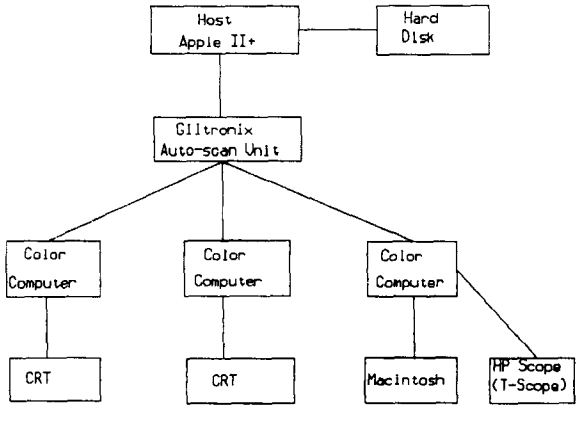

Figure 1. The networking system for one host and several Color Computers.

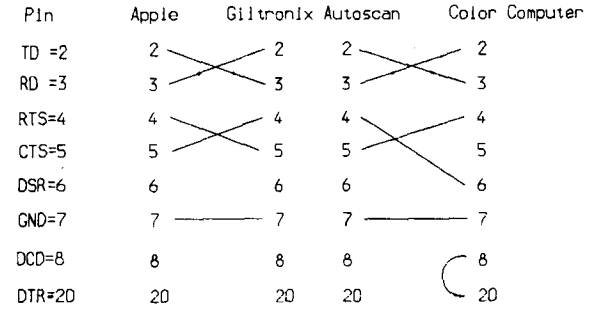

Figure 2. Pin connections for the RS-232 communication scheme.

the autoscan unit provides a transparent connection between the host and the Color Computer that is requesting service. The logic of the system is to have the Color Computers request service, either for downloading of more stimulus materials or for uploading of responses, by setting the handshake line. If a connection is made, then the handshake line of the host (CTS on the host and DSR on the Color Computer) is inspected, and if this signals availability, then a block of information is transferred and the line is released for the next request.

The program running on the host opens up to five input files for five experiments (either the same or different) and an output file for the output from all 5 subjects. When a Color Computer requests service, either the responses are added to the output file or new input information is transferred down. The host signals its availability by setting the CTS line on the serial port (the Apple II+ uses a California Computer System 7710A serial card for communication). The host program we use is written in UCSD Pascal and uses UNITREADs and UNITWRITEs for communication and BLOCKREADs and BLOCKWRITEs for disk file access. After the data is collected, the single output file is split using a simple program that puts output data into individual subject files based on the leading subject number.

One large obstacle to the usefulness of such a system for the average cognitive psychologist is, believe it or not, cabling. An experienced technician would have little difficulty in wiring the RS-232 cables with solder joints of high quality and correct connections among pins. However, a cognitive psychologist with little technical support can easily make mistakes in the connections and fail to debug the cables to find such mistakes. In fact, for someone having no experience in these matters, it can be difficult to discover whether a failure is an error in assembling the system or a failure in a component. The correct cable connections are shown in Figure 2. The connections are dictated by which handshake lines can be controlled by the Color Computer and which by the host computer. For example, the Color Computer RS-232 pack allows control of the RTS handshake line, which is then connected to the Giltronix Autoscan CTS handshake line (which is used to request a line to the host). Likewise, the CTS handshake line on the host is used to signal that it is ready to communicate with a Color Computer. The

CTS line is connected from the Autoscan unit (RTS line) to the DSR handshake line on the Color Computer RS-232 pack because it is one of the two handshake lines that can be detected by the Color Computer.

Another problem with the network system concerns loading the operating system (we will call the program Cogsys from here on) into the Color Computer (this is a problem because we decided that our Color Computer system should have no disk drive). In the current version, the binary code for the Color Computer program is contained in a file on the host and it is downloaded into each Color Computer. The sequence begins by loading a simple BASIC program from cassette into each Color Computer, which comes with a built-in BASIC interpreter in ROM. The program initiates a download of Cogsys from the host. The program consists of a series of POKEs that sets up the serial port, reads a binary file into memory from the serial port, and initiates execution of that binary file. (Essentially, the POKEs into memory set up a simple assembly language routine.) When the system is debugged, the kernel system is burned into ROM, and then simply turning on the Color Computer loads Cogsys and begins execution.

It is possible to alter several features of the initialized system before an experiment is run using input from the Color Computer keyboard. (We use the terminal keyboard for experiments because only one character is transmitted for each keypress; the Color Computer keyboard provides continuous signals as long as a key is depressed, so that software delays are needed.) It is possible to change from taking responses from the terminal keyboard to taking responses from the Color Computer keyboard by hitting the K key. When Cogsys is burned into EPROMs, a subject number will be burned into each EPROM. To allow the subject number to be changed, before the experiment begins, the command N followed by a digit alters the subject number. The system checks for the @ key to be pressed on the Color Computer at each %B. If the @ key is detected, the program terminates and restarts (this is used for debugging purposes). The : key on the Color Computer begins the experiment, minimizing the chance of a subject's beginning the experiment while fooling around with keys.

THE COMMAND LANGUAGE

Overview and Advantages

The command language is the key to ease of use of this system (see Ratcliff & Layton, 1981). The logic is simple; a real-time command is prefixed by one of these characters: $, #, %, or @. The stimulus file is read from the host computer into an input (ring) buffer in the Color Computer. As the experiment proceeds, characters are read out of the input buffer and printed on the CRT screen unless a prefix is encountered. When a prefix is encountered, the next character is used to search a table for the code that will execute the command. The command consists of a prefix followed by a single letter followed optionally by a series of parameters (depending on the command). (A prefix can be printed by preceding it with a "\" character.) Once a real-time command is decoded, the operation is carried out.

Because the language has a "what you see is what you get" structure, it is extremely simple to use and debug. In fact, students can master it within a day. For example, if the word *toad* is to be presented for 500 msec, the operation is coded as "toad#W500." Since *toad* does not contain a prefix character, it is printed on the CRT. The prefix "#" signals a command, and a table is searched for "#W." This command expects as a parameter a set of digits representing a wait in milliseconds. The system waits for 500 msec with the word *toad* displayed, and then the next character is processed. In using this language, the only cost in terms of efficiency is that experimental files for text experiments can get large (e.g., 50 Kbytes). However, the direct representation of real-time commands in the language outweighs this size inconvenience.

If an error is made in the stimulus list, the system does one of two things. If "3W" is typed instead of "#W," "3W" is simply printed on the screen. If "##" is entered by mistake, and it is not a command, the system exits with an error message and, after a keypress, resets itself to the start of the Cogsys program. Debugging is easy because the experimenter knows where the error has been made by noting the last text successfully presented. Although a number of mistakes produce error messages, others will cause the system to crash. However, these are easy to debug since the command language is simple.

Stimulus lists are usually generated using a high-level language. To generate a list, one reads the set of stimulus materials into an array, and assigns the materials to stimulus conditions according to the randomization required by the experimental design (Latin square, completely random, etc.). The test items are also read into an array or selected from the study material array and randomized with the set of constraints from the design. Once the lists are generated and errors are removed, the real-time commands are added into the format statements to provide the presentation commands. For a simple study-test procedure (Ratcliff & Murdock, 1976), the program would be about one page in length. For more complex designs, programs are rarely longer than four pages.

An advantage of the system is that students can use an editor to construct stimulus lists with real-time commands embedded. They do not have to learn a programming language or learn to call timing subroutines.

Available Commands

Table 1 contains a list of real-time commands and their effects. Using these commands, we can implement almost all of the experimental paradigms in cognitive psychology that involve presentation of alphanumeric characters. To indicate how the different commands work, we present several sample programs and describe their operation.

Table 2 shows examples of experiments, designed to show features of the language. Example 1 shows a sim-

Table 1
Cogsys Commands and Their Operation

#W	One argument: either a number or variable (e.g., V11). Waits for the specified number of milliseconds. If time is greater than 250 msec, there will be an attempt to transfer information from the host to the input buffer (if there is room in the buffer) or from the output buffer to the host (if there are enough characters in the output buffer).
#R	No arguments. Collects a response time plus key pressed and puts them, preceded by the subject number, in the output buffer.
#N	No arguments. Used to signal the end of the experiment from the host. Used only in the communication program on the host.
#I	Three arguments: the first argument is a Boolean expression in parentheses, the second is an action (a string of Cogsys commands) taken if the expression is TRUE, and the third is an action taken if the expression is FALSE. The Boolean expression can be composed of several operations or variables: A, N, O, >, >=, =, <>, <, <=. These represent, respectively, logical AND, negation, logical OR, greater than, greater than or equal, equal, not equal, less than, and less than or equal. In addition, the expression K=&/ returns "true" if the last key pressed was a / and "false" otherwise. R refers to the last response time (see the examples in Table 2).
#S	One argument: example, #S/abc/ The action is to send the string between the / characters back to the host prefixed by the subject number. This command is used to transmit codes that identify particular conditions of the experiment so that each response has the code associated with it in the data file.
$$	Syntax is: $$4 commands $$, where "commands" represents a string of commands or text to be printed to the screen. The string "commands" replaces the string $4 wherever it appears in the experiment test list. To allow some simple programming functions, a macro can be nested in a macro (only one deep).
$	One argument: a one-digit number or lowercase letter up to j. Denotes a macro and executes the commands stored in the macro.
$V	One argument. Allows a variable to be assigned a character. Example: $V V11=Z sets variable V11 to the letter Z.
$A	One argument. Assigns a numeric value to a variable. Example: $A V12=2000 sets variable V12 to the value 2000.
$M	One argument. Performs arithmetic operations with variables or between a variable and a constant. Syntax is: $MV11= identifier operator identifier. An identifier can either be a variable or a numeric constant. The operator is +, −, *, /, or \ . The \ is used as modulus.
$B	Blocks input/output operations during waits (#W). This allows systems with buffered input/output to control Cogsys (e.g., UNIX systems). The one limitation is that all transfers take place at %B commands, thus introducing delays into the experiment between blocks of trials.
$S	One argument, a variable: example, $SV11 displays the value of the numeric variable V11 onto the screen.
$L	No argument. Collects a line of characters up to a carriage return and puts them, preceded by the subject number, into the output buffer.
$R	No argument. Displays the last response time on the CRT screen.
%B	No argument. Denotes blocks of the test list such that everything must be run consecutively. The experiment will not continue to execute until the next %B has been downloaded into the input buffer.
%T	No argument. Allows a switch of terminal type. The Datamedia Elite 1520 and the ADDS Viewpoint are supported.
@	Two arguments, two-digit integers: cursor-control command. Syntax is @xxyy where the xs are single digits and represent row number and the ys represent column number.
@C	No arguments. Clears the screen.
@D	No arguments. Cursors down one line.
%X	No arguments. Branches out of the current macro. Also stores the user number, the character "#," and the number zero into the output buffer. This is used to signal that the macro was exited at the %X as opposed to a normal exit or a %Y exit.
%Y	No arguments. Branches out of the current macro.
%Z	No arguments. Jumps to the starting position of the current macro.

Table 1
(Continued)

External Subroutines

#P	Two arguments. Takes a number or variable representing a number followed by a string in braces (curly brackets). Syntax: #P500 {abcde}. Waits 500 msec, presents the string abcde on the next line down and picks up any response made before, during, or after presentation of the string and puts the key pressed and reaction time, preceded by the subject number, into the output buffer. If the response is made before the string is to be presented, the string is not presented.
#C	One argument. Collects a time-limited response. Example: #C5000 If a key is pressed before 5 sec elapse, then the key pressed and response time, preceded by the subject number, are sent to the output buffer. After 5 sec expire, the subject number, the character @, and the time of 5000 are stored in the output buffer.
#A	Sets attributes for the HP scope. Syntax: #A[command1 command2...] Commands are: T—Time to refresh screen: 0-60 msec X—Set the X coordinate: 0-2047 Y—Set the Y coordinate: 0-2047 I—Line intensity: 0-3, where 0 is blank and 3 is full L—Line types: 0-3, where 0 is solid and 3 is short dashes O—On/Off beam, where 1 is on and 0 is off. Used for line vectors. S—Character size: 0-3, smallest to largest. R—Character rotation: 0-3, for 0°, 90°, 180°, 270°. Example: #A[X=0 Y=1000 T=1 I=1 S=0]
#V	One argument: a string. Syntax: #V[string] Displays a string on the scope and collects a response time from the keyboard. The subject number, key pressed, and response time are stored in the output buffer.
#U	Two arguments: an integer or variable representing an integer and a string. Syntax: #U500[string] Displays the string on the scope for the time given by the integer.
#T	Two arguments: an integer and a string. Syntax: #T500[string] Displays the string on the screen and either waits for the time to expire or collects a response time. The subject number, key pressed, and response time are stored in the output buffer.
$G	One argument: a variable name. Syntax: $GV11 Sets a variable from the terminal keyboard. Requires a double entry of the number for verification. This is used, for example, for setting timing intervals as a function of prior performance in the experiment.

ple study-test procedure. Five words are presented for 1 sec each, followed by a row of asterisks presented for 1,000 msec. Then a test list of *old* and *new* words is presented. For a *new* word, if the subject responds correctly (the Z key), the reaction time is printed on the screen for 500 msec. If an error is made, the word "ERROR" is printed for 2,000 msec. For an *old* word, the correct response is the / key. The #S command sends back to the host a "0" if the trial is negative and a "1" if the trial is positive. The %B at the end signals that the trial cannot begin until the whole study-test list is in the input buffer of the Color Computer.

Example 2 shows how variables can be used to score accuracy and reaction time. The task requires the subject to press the / key for a digit and the Z key for a letter. Variable V12 contains the total number of trials, and V11

Table 2
Sample Stimulus Lists Using the Cogsys Language

```
1. $$1#R#S/1/@C#I(K=&/ O K=&?){$R#W500@C}{ERROR#W2000@C}$$
   $$2#R#S/0/@C#I(K=&Z O K=&z){$R#W500@C}{ERROR#W2000@C}$$
   $$3#W1000@C$$
   gallant  $3
   legend   $3
   robust   $3
   chair    $3
   glue     $3
   *******$3
   blue     $2
   robust   $1
   sky      $2
   glue     $1
   legend   $1
   %B
```

If the subject responded "old," "old," "new," "old," "old," to the five test items and the subject were tested on Color Computer Number 2, the response file would contain:

```
2/552
20
2/783
21
2Z831
20
2/759
21
2/537
21
```

where the first digit refers to the subject number, the character is the key pressed, and the last digits are the reaction in milliseconds.

```
2. $$1#R#I(K=&/ A R<1000){$MV11=V11+1 $MV13=V13+V5}{}$$
   $$2#R#I(K=&Z A R<1000){$MV11=V11+1 $MV13=V13+V5}{}$$
   $AV11=0 $AV12=0 $AV13=0
   8$1  $MV12=V12+1
   J$2  $MV12=V12+1
   4$1  $MV12=V12+1
   G$2 $MV12=V12+1
   R$2 $MV12=V12+1
   7$1  $MV12=V12+1
   2$1  $MV12=V12+1
   Q$2 $MV12=V12+1
   You got $SV11 correct out of $SV12. Average reaction time for correct responses was
   $MV13=V13/V11 $V13
   %B

3. $$1$2#WV11@C********#R#I(K=&/){%X}{$MV11=V11+17 %Z}$$
   $AV11=17 $$2CAT     $$
   $1
   $AV11=17 $$2HOUSE $$
   $1
   %B
```

the number of correct responses faster than 1,000 msec. Variable V5 contains the last response time and is added to V13 to keep a running sum of reaction times. At the end, the average correct reaction time is given by V13/V11. The $S displays the values of the variables on the screen with text to indicate what the numbers mean.

Programming Within Cogsys

One apparent disadvantage of the command language is that once a stimulus list is created, the sequence of events in the experiment is fixed. There are two levels at which we may want to adjust the experiment as a function of the performance of the subject. The first adjustment is timing (e.g., presentation time) as a function of performance. Such adjustments can be made using variables and arithmetic operations. For example, suppose that we use three presentation times in a sequence of threshold-setting trials. If we want to set performance so that subjects are at 75% accuracy, variables are used to accumulate the number of correct responses for each presentation time. These can be used to set another variable with simple interpolation (arithmetic operations between variables), or the results can be printed out on the screen and the experimenter can enter the presentation time into a variable using the $G command (Table 1).

A second, more complicated adjustment is to allow the number of presentations to be a variable, which can be accomplished by repeating blocks of code. However, since the input buffer is a ring buffer, once code is executed, there is no guarantee that it will be in the buffer at a later point in time (it may be overwritten by a new download of information). But macros contain instructions that are kept throughout an experiment, unless they are redefined within the list, and so it is possible with the %Y and %Z commands to branch as a result of previous performance within a macro. To make it possible to alter text presented within a macro, a macro can be embedded within a macro. Example 3 in Table 2 shows this: The macro $2 contains the test word, and the variable V11 contains the presentation time. The test word is presented for V11 msec followed by a mask of asterisks, and the subject is required to press a key to indicate whether the string is a word or not. (This example is not part of an actual experiment but illustrates the use of the commands.) The "/" key indicates a "word" response. If this response is not made, the presentation time is incremented by 17 msec and the process is repeated. When a "word" response is made, the macro is exited using the %X, and 1#0 is entered in the output buffer (if this were Subject 1). The scoring program reads responses (from the #R) until the character "#" is found as a response which indicates an exit from the macro.

COMMUNICATION PROTOCOL

Communication between a Color Computer and the host takes place (1) at a #W command when there is longer than 250 msec to wait; (2) at a %B when there is no other %B in the input buffer; and (3) at the end of an experiment to empty the output buffer and sign off. The first of these is the most complicated. The routine that performs the wait uses the horizontal sync interrupt (that occurs every 63.5 μsec) for timing. To use this interrupt, the code has to be written in blocks that take less time than 63.5 μsec. Each block is followed by an instruction that halts processing until an interrupt is detected, and then continues (called the SYNC instruction in 6809 assembly). Thus the code consists of blocks of about 20–25 assembly language instructions (2–4 machine cycles per instruction, machine clock at .89 MHz), followed by the SYNC instruction, followed by code to increment the time counter.

The code for communication with the host requests the host by setting a handshake line; if it successfully gains access to the host through the autoscan unit then it checks the host's handshake line. If the line is available, then either an upload of responses or a download of a text file occurs. When there are more than 128 characters in the output buffer, an upload is initiated and 128 characters are transferred up. When there are not enough characters for an upload, a download is initiated and 256 characters are transferred down to the Color Computer. The transmission rate is 19.2 Kbaud (i.e., a little less than 2,000 characters per second). The protocol for transfer at a %B is similar. If there is a %B in the input buffer, the program continues; otherwise, first the output buffer is emptied to less than 128 characters (in case there are no transfers at #W commands) and the input buffer is filled, in 256 character blocks, until another %B is transmitted down.

COLOR COMPUTER MEMORY ORGANIZATION

The memory organization in the Color Computer reflects several constraints. For debugging purposes, it is useful to have the real-time Cogsys program, buffers, temporary work areas, and so forth loaded into a part of memory that does not contain the development operating system or the debugger. This limits the available space but actually has no practical consequences for the user. Table 3 shows the memory map of Cogsys. There are several things to note: First, the regions denoted "interrupts" and "hardware" are reserved by the physical system. Second, the input buffer and output buffer, which are organized as ring buffers, are actually very large. Most experiments in our laboratory take 10–50 Kbytes of stimulus information for a 1-h session. To see how large 50 Kbytes or 50,000 characters is, suppose a subject can read 16 characters per second (i.e., four four-letter words). Multiply this by 3,600 per hour, and we get 57,600 characters per hour. This places a rough upper limit on the size of stimulus lists in a text experiment. Thus an 11-Kbyte input buffer will store about 20% of the largest experiment. The output buffer is also large, 2 Kbtyes. Most reaction time experiments that we run produce an output buffer of about 12 Kbytes, so again the output buffer is near 20%.

The Cogsys program takes about 8 Kbytes of storage, and the variable space takes about 6 Kbytes. The variable space is used for the macro buffer (20×100 character macros), variable space for the variables, intermediate products of operations on variables, and other variables used by Cogsys. The S and U stacks are user-defined but are required for the system. There are 8 Kbytes reserved for new commands that can be developed independently of Cogsys and then loaded into the system at system startup. This allows us to freeze the core of Cogsys and at the same time allow for future growth. The debugger and the FLEX operating system reside above this area so that if many more routines are developed, they can be debugged individually and then loaded in with all the others. If all the routines are larger than 8 Kbytes, then

Table 3
Color Computer Memory Map

$0100-$010A	Interrupts
$0200-$2FFF	Input buffer
$3000-$37FF	Output buffer
$3800-$46C9	Variable space
$46CA-$4FFF	S,U stacks
$5000-$6FFF	Cogsys kernel program
$7000-$8FFF	Subroutine space (new downloaded routines)
$9000-$BFFF	Debug (only present in development)
$C000-$FFFF	FLEX operating system plus memory mapped I/O locations

when they are combined into one file, they will overwrite the space in which the debugger resides during development.

INSERTING NEW COMMANDS

One important feature of this type of system is that it is expandable. As a new feature is needed, it can be added to the system. Since we have now placed the core system in an 8K EPROM, which replaces the Extended BASIC ROM in the Color Computer, there has to be a method of adding new routines. The way we accomplished this is by the following rules. The code for each new command consists of a header that specifies the command name, followed by the code. The code can use some of the more useful Cogsys routines by accessing a table at the end of Cogsys that stores the starting addresses of the routines (routines such as converting numbers, sending a character to the terminal, placing a response plus reaction time in the output buffer, etc.). The routines for new commands are stored in a file on the host and when the system is first run, Cogsys initiates a request to load that file down if it has not already been downloaded.

PERIPHERAL DEVICES

One feature of the Color Computer that is useful is the availability of a limited number of peripheral boards. Radio Shack provides an RS-232 serial cartridge and a multipak expansion unit (which allows four cartridges to be plugged in), and Magnum Distributors provides a parallel port. Thus, any peripheral that runs off a parallel port or a serial port can be driven by the Color Computer. The serial port is used for communication between the host and the Color Computer. The serial port allows the use of three handshake lines, and there is hardware implementation of the bit-to-string conversion. There is one limitation: we have not been able to use more than one serial board with the multipak expansion unit. To drive another computer (e.g., the Macintosh), we use the built-in serial port and dispense with the display terminal, using the other computer as the display device. The parallel port uses a 6522 PIA chip which provides a 16-bit parallel port, several handshake lines, and several timers. To drive other parallel devices, accurate timing can be achieved using the built-in timer, and necessary handshaking from the peripheral can be handled using simple programming and the handshake lines.

Another peripheral that is of some importance is an EPROM burner for the Color Computer. This is available from Green Mountain Micro and runs on batteries. The software is contained in a BASIC cassette program and allows a binary file in Radio Shack DOS format to be burned into an EPROM. The FLEX operating system allows files to be converted to Radio Shack DOS format, and this transformation allows the binary file to be converted into a form ready to be burned into the EPROM to replace the Extended BASIC ROM in the Color Computer.

Accurate Visual Presentation

An HP 1345A display scope enables us to perform experiments with accurate visual timing on stimulus presentation. The scope has a built-in character set, and each character can be displayed in 16 μsec by the scope hardware. We interfaced this scope to the Color Computer through the parallel port and wrote routines that would allow presentation of ASCII characters.

The parallel board uses 16-bit transfers to the scope with a simple handshake sequence. Two of the 6522 handshake lines are used, one on output and one on input. When 16 bits of data are loaded into the PIA for transfer, a handshake line is set that indicates to the scope that it should accept the data. When the scope has accepted the data, it sets another handshake line and the Color Computer can begin loading the PIA again.

The time base for display is 1 msec; thus, each character is rewritten to the scope every 1 msec so that we can control the display presentation times to 1-msec increments. In practice, with software overhead on the Color Computer, it is possible to display 15 characters within the 1-msec refresh rate; however, more characters can be displayed by increasing the time base to 2 or 4 msec and adjusting intensity appropriately. Characteristics of the scope display can be set with a command, for example, character size, brightness, position, and rotation (see Table 1 for a complete list). The scope uses a P4 phosphor that decays to 1% intensity within 560 μsec and has the capability of displaying vectors within a 2048×2048 pixel universe. In addition to the text display capability, the system allows simple polygons to be displayed. By manipulating the time base, we can present a few vectors with short presentation times, or many vectors with relatively large steps between increments in presentation time.

Picture Experiments

We recently designed an interface procedure to allow us to display pictures on the Apple Macintosh computer. The idea is to store a set of pictures in files on the Macintosh and to use the Color Computer to initiate presentation of those pictures. The stimulus pictures are stored in MacPaint format, and the assembly language program running on the Macintosh expands them into a bitmap prior to presentation. Then presentation of a picture consists of a simple block move of the bitmap into the video memory of the Macintosh. The Color Computer communicates with the Macintosh through a serial port running at 9600 baud (about one character per millisecond; this is the maximum speed of the internal serial port and is the speed at which the display terminal is run), and the Macintosh replaces the terminal as the display device for the Color Computer. The command characters are designed to be little-used ASCII characters that the Color Computer treats simply as text, shipping them straight out to the Macintosh. The Macintosh writes any ASCII character to the screen, unless it is one of the special characters. The commands available on the Macintosh are shown in Table 4. Thus the Cogsys program did not have to be modified to control the Macintosh.

Table 4
Symbols Interpreted by the Macintosh for Picture Display

|#,#,#,...,#| where the # represent numbers. This indicates to the Macintosh which pictures to load into memory for presentation. The file names are stored in a file, and the numbers above represent positions in that file of names. A "file.s" refers to a picture containing a horizontal strip that is overlaid over the preceding picture. A carriage return following the last | initiates the loading of files. When loading is complete, a space is returned to the Color Computer. Thus, the syntax of this command is: |#,#,...#|"return"#R
The carriage return is necessary after the last vertical bar, and the #R picks up the space returned when the pictures have been loaded into memory.

|| (two vertical bars) ejects the Macintosh disk and requests a new list of file names.

^ commands the Macintosh to display the next picture. To display the picture for 500 msec, the syntax is: ^#W500

~ commands the Macintosh to accept a keypress and transmit the character back to the Color Computer. The syntax for displaying a picture and collecting a keypress and reaction time is: ^~#R where the #R collects the character and records the response time.

@C transmits a control-L to the Macintosh and clears the screen. Note that clearing the screen takes up to 22 msec because the whole bitmap has to be reset. If the cursor is only part way down the screen after writing text, the clear is much faster since the bitmap is only cleared down to the current cursor position.

There are several characters that the Macintosh interprets as commands, and there is one setup command that gives a list of file numbers to the Macintosh. A file on the Macintosh disk contains a list of picture file names. The command from the Color Computer instructs the Macintosh to read the files indicated by the numbers in the list of file names. So, for example, |5,21,8| causes the Macintosh to read the 5th, 21st, and 8th files in the list of names into memory for display. The commands to display the pictures are sent to the Macintosh using a special character (e.g., ^), which signals for the next picture to be displayed.

There are limitations on the speed with which a picture can be displayed. The full bitmap can be redrawn in 22 msec, which means that whole pictures can be displayed sequentially with a minimum SOA of about 50 msec. In order to do certain kinds of experiments (e.g., priming experiments), the researcher may wish to alter part of the picture to present a priming picture or part of a picture (e.g., a name on a map). To do this and to reduce the amount of storage required by a picture (e.g., a bitmap uses about 22 Kbytes of memory), part of a picture is stored and displayed by overwriting. For example, suppose we want to add a string of text (e.g., a word) to a picture. We enter Macpaint, read in the picture, delete all the picture above and below the horizontal strip containing the name, leaving the parts of the original picture in that strip. This strip is then saved in Macpaint format in a file with a ".s" extension. When the files are read-in, any file that is labeled ".s" is expanded into a memory bitmap with the strip before the first nonwhite pixel and the strip after the last nonwhite pixel removed. Thus, only a horizontal strip of the picture is stored (along with numbers indicating the starting and stopping rows). This results in faster display time (one half of a bitmap can be displayed within one raster operation, 16.7 msec) and in efficient storage in the Macintosh memory, since the size of a strip of a picture is much smaller than the full picture bitmap. Thus, larger numbers of pictures can be run in sequence before the next set has to be loaded into memory from disk. There are two characters that have special operations (see Table 4). The ^ requests display of the next picture and the ~ requests a keypress. The way these are implemented is shown in the example in Table 4: When a ~ is presented, it is followed by a #R in Cogsys. The ~ requests that a character be transmitted back to the Color Computer, and the #R collects that character from the serial port and obtains the reaction time. Characters are printed to the screen only after a carriage return because the process is too slow to print them as the serial port receives them.

In addition to these full-screen pictures, we will soon implement a command that allows the size and type of font to be chosen by software control, thus allowing control of the physical characteristics of the displayed text. Such control is beyond the capabilities of most CRT displays.

CONCLUSIONS

The aim of this project was to develop a real-time system that would be easy to use and would be composed of inexpensive off-the-shelf components (so that failure of components would require only a new purchase order and little money). The system fulfills these aims. In addition, it is highly expandable: new routines can be added without too much understanding of the kernel Cogsys program, and new devices can be added through the serial and parallel ports. Although the system is too slow to record speech, it is possible to hook up a voice key, drive a speech-production device, such as the Digital Equipment Corporation DECtalk, through the serial port, or drive another computer, such as an ATARI ST520 or Commodore Amiga, to allow the use of Color graphics (the Color Computer display is too slow and limited). The advantage of using other computers for special purposes (with communication, display, and timing signals controlled by the Color Computer), driven through the serial port, is that the program in the new computer requires only limited functions. For example, it has to accept ASCII characters over the serial line, and display text or pictures in color. No additional functions are required to make the new system run quite quickly.

REFERENCES

RATCLIFF, R., & LAYTON, W. M. (1981). A microcomputer interface for control of real-time experiments in cognitive psychology. *Behavior Research Methods & Instrumentation*, **13**, 216-220.

RATCLIFF, R., & MURDOCK, B. B., JR. (1976). Retrieval processes in recognition memory. *Psychological Review*, **83**, 190-214.

REED, A. V. (1979). On choosing an inexpensive microcomputer for the experimental psychology laboratory. *Behavior Research Methods & Instrumentation*, **12**, 607-613.

REED, A. V. (1982). The Radio Shack Color Computer in the experimental psychology laboratory: An evaluation. *Behavior Research Methods & Instrumentation*, **14**, 109-112.