Behavior Research Methods, Instruments, & Computers
1991, 23 (4), 513-522

— COMPUTER TECHNOLOGY —

A 6809 single-board computer system for the control of behavioral experiments

JOHN M. HORNER
Colorado College, Colorado Springs, Colorado

A computer system consisting of a 6809 single-board computer in conjunction with an IBM-compatible Personal Computer (PC) is described for the control of behavioral experiments. The single-board computer uses the C programming language to program experimental events. Each component of the system (the single-board computer, a digital interface, the PC, and the software) is outlined with its capabilities and drawbacks noted.

With the advance of digital computers came the promise of powerful experimental designs and more thorough data collection and analysis (Uttal, 1968). Sadly, this promise has not borne much fruit, partly because the amount of technical expertise required to build a computer system is extensive, and partly because the cost in time and money to build a computer system from scratch is enormous. This paper attempts to redress these problems by providing a blueprint for a relatively inexpensive yet powerful computer system for the control of behavioral experiments.

The system outlined herein attempts to maximize a researcher's control and analysis capabilities. The present system is capable of controlling, timing, and recording 19,000+ events to within 1/2400 sec. Also, because the system is based on the high-level language "C" and software that does not assume any particular experimental approach, the programming of experiments and the analysis of data are left up to the experimenter's capabilities and imagination. Hence, the present system is a powerful and flexible tool for behavioral research, yet it costs less than other comparable systems.

The system is based on a 6809 Control Module (6809 CM) single-board computer that runs the experiment (see Figure 1). The 6809 CM controls and senses experimental events, such as standard behavioral electromechanical devices and manipulanda, through a digital interface. An IBM-compatible Personal Computer (PC) serves as the "host" for the 6809 CM. The PC performs the tasks of compiling and transferring programs (that run the experiments) to the 6809 CM (i.e., downloading) and storing and analyzing data from the 6809 CM (i.e., uploading). The PC and the 6809 CM communicate through a serial (i.e., RS-232) line. The C programming language serves for both the programming of experiments and the analysis of the data from each experiment.

This design follows the philosophy that each 6809 CM serves as a stand-alone system for running a single experiment. Thus, the system is simplified because one computer (6809 CM) runs one experiment. Hence, each 6809 CM is isolated from one another in such a way that failures in one do not produce failures in another. This design has the added benefit that the PC is left free to perform other tasks (e.g., word processing, data analysis, etc.) while the 6809 CM is running experiments. This one-computer, one-experiment design is feasible because the cost of each 6809 CM is relatively inexpensive compared with other methods of controlling experiments.

Although the system is simple as a whole, it is a complex collection of parts that requires some assembly. The following sections of this paper describe each component and explain how it functions as part of the system. Additional information will be supplied by the author on request.

THE SINGLE-BOARD COMPUTER

The 6809 CM used by this system is based on the Motorola 6800 family of chips and is commercially produced. The central processing unit (CPU) is the hybrid 8/16-bit 6809 chip, which is a powerful microprocessor that supports a number of higher level programming techniques, yet is straightforward in operation. The 6809 represents

The author would like to thank Alan Campbell for his help in the design and construction of the system described here—without his help this system would never have been developed. The author would also like to thank Ken Steele and other anonymous reviewers for their comments on earlier drafts of this paper. All three output circuits described herein were designed by Alan Campbell. The input circuit is based on a hybrid circuit that incorporates the design features of Alan Campbell, John Hinson, Alliston Reid, Ken Steele, and myself. The data-storage technique described herein is based on a technique first devised by John Hinson. Any design errors in this article are the responsibility of the author. The project was supported by a generous grant from Colorado College for faculty research and development. Reprint requests and other correspondence should be addressed to John M. Horner, Psychology Department, Colorado College, Colorado Springs, CO 80903 (e-mail: JHORNER%CCNODE@VAXF.COLORADO.EDU).

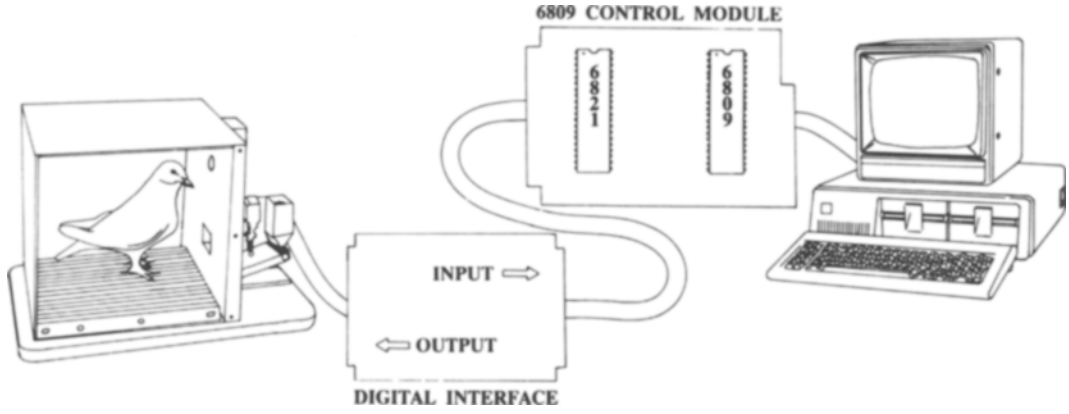

Figure 1. A stylized representation of the present system.

the acme in 8/16-bit design—to improve CPU performance by a significant amount, one would need to expand to a larger bit structure. Because behavioral events (i.e., keypeck, hopper operations) are fairly imprecise by computer standards, an 8/16-bit microprocessor is all that is necessary to run most behavioral experiments. These facts, in conjunction with the 6809's use in numerous industrial and commercial applications (e.g., GM Engine CM), its proven performance in many research applications (e.g., Walter/Palya [1984] system), and its extensive support by other chips in the 6800 family of microprocessors and peripherals, make it the ideal CPU for running behavioral experiments.

The 6809 CM used by the present system is commercially produced by the Wintek Corporation and consists of the 6809 CPU, 56K of memory (RAM), a 1-MHz clock, two RS-232 serial ports, and two peripheral interface adaptor (PIA) chips (6821), which handle up to 32 TTL compatible parallel input and output lines in addition to their controlling logic. The 6809 CM has its own on-board monitor/debugger (Fantom 9), which aids in downloading, running, and debugging programs. This particular 6809 CM was chosen over a number of other systems and manufacturers for the following reasons: (1) Each 6809 CM is inexpensive; (2) each has a number of standard hardware features, such as serial and parallel ports and memory; (3) this product has software support, which includes a C cross-compiler/assembler, an on-board monitor/debugger (Fantom 9), and a communication package (Terminal Emulator); (4) the 6809 CMs come already assembled on an industry-standard 4.5 × 6.5 in. circuit board with a standard 44-pin connector; (5) there is extensive hardware support for the 6809 CM, which allows the user to easily expand beyond the basic system; and (6) it is based on the Motorola 6800 family of 8-bit microprocessors.

The 6809 CMs can be equipped with an erasable, programmable read-only memory (EPROM) monitor/debugger called Fantom 9, which allows the 6809 CM to act as a stand-alone computer—performing interactive tasks with the operator, such as running programs, debugging software, and transferring data. The 6809 CM does not need Fantom 9 to function, because an experiment (i.e., computer program) can be permanently stored (i.e., "etched" into EPROM) in memory and physically fixed directly onto the 6809 CM; however, since Fantom 9 facilitates the operator's interaction with the 6809 CM, this approach limits the flexibility of the system and is therefore not advised.

The 6809 CM is sufficient to run the system described herein; however, the manufacturer does supply a number of additional input/output (I/O) boards and memory modules, which allows the user the flexibility to easily expand the system. Memory can be expanded to 512K, and I/O can be expanded to 96 lines. The basic system for running experiments is based on a minimal configuration of only 8 outputs, 8 inputs, and 24K of memory. Expansion of the basic system would require additional purchases and/or a minimum of programming changes.

The 6809 CPU chip, although it is the "thing" that runs the experiment, is not the component that makes the 6809 CM useful. The 6809 CM has two PIAs (6821) that enable the CPU to sense and control experimental events. The two PIAs allow for up to 32 separate events to be handled concurrently. This is far more than the standard number of events used in virtually any behavioral experiment. Because of voltage differences between computers and most standard behavioral apparatuses, it is necessary to provide a digital interface between the PIAs and the apparatus. The next section describes such an interface.

THE DIGITAL INTERFACE

The 6809 CM senses and controls experimental events through a researcher-manufactured digital interface. The control of events in the behavioral chamber (outputs) is performed via a digital interface by turning on and off one of the digital switches in one of the PIA chips. This in turn switches the power on and off to any electrical or electromechanical device. The technology for the control of these devices is well established and too numerous to reference (see Ratzlaff, 1987, for an excellent overview of computer-assisted experimentation; see Carr, 1984, or Wolfe, 1982, for more cookbook approaches);

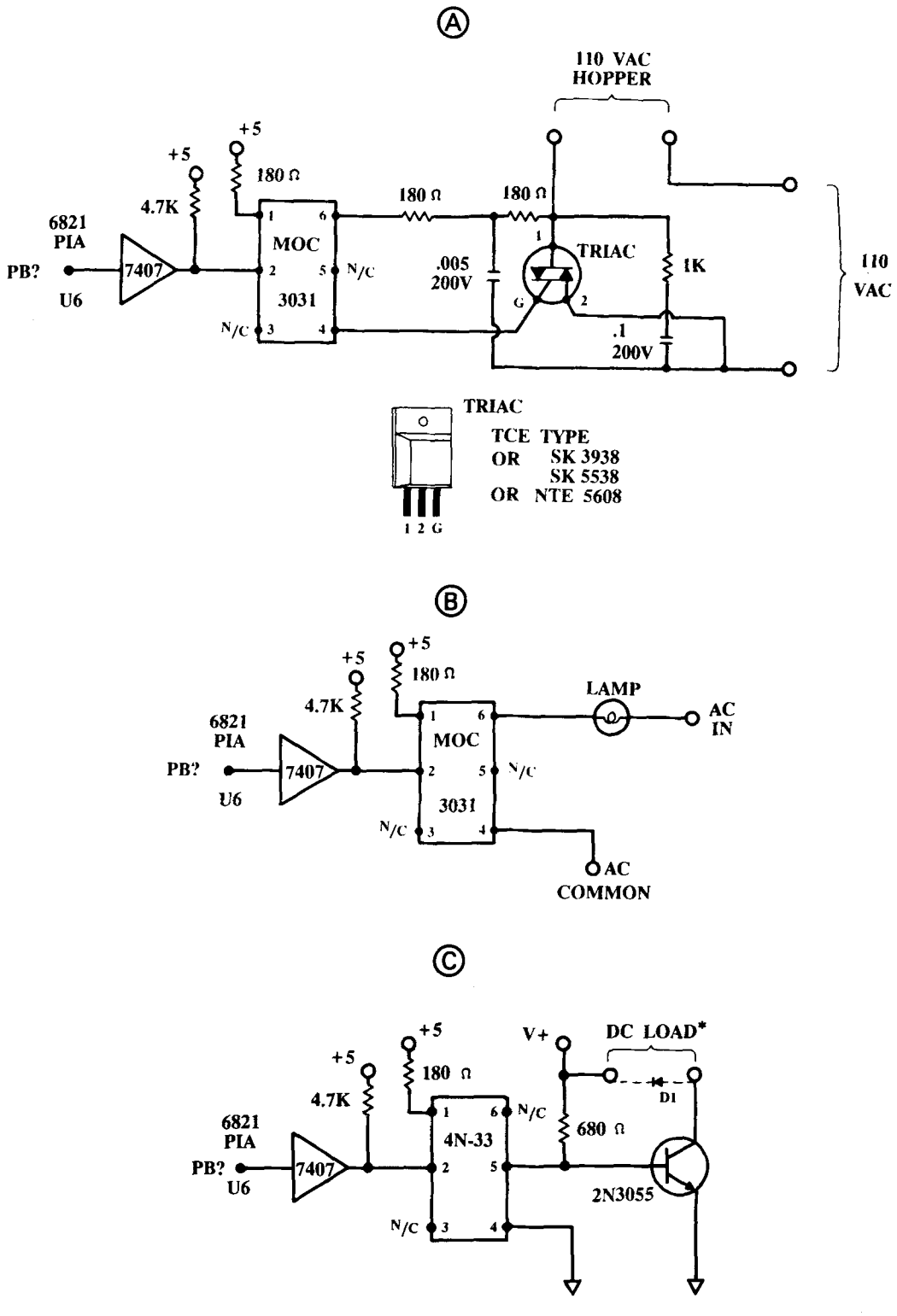

Figure 2. Output circuits for the digital interface. The 7407 is a standard TTL device and runs on a +5 V dc power source. PB? refers to any output port on the 6821 PIA of the CM 6809 (chip U6). Figure 2A shows a high-current ac circuit, whereas Figure 2B is a low-current ac circuit. (The high-current device in Figure 2A uses a Quadrac, usually found under Triacs in most catalogs.) Figure 2C shows a comparable dc high-current circuit. (*If the dc load is inductive, a diode [D1] should be placed in parallel with the load [D1 = 1N4004 THR].) The optical isolator, 4N-33, is rated at 30 V, and the NPN transistor, 2N3055, is rated at 15 A, although lower amp ratings would do for most uses.

suffice it to say that anything that can be controlled by an electrical switch can be controlled by a computer with the appropriate interface. To be appropriate, an interface must convert the voltage signal used by the computer to a voltage signal used by the electrical device, while providing a buffer between the two voltages.

Although a number of circuits would suffice to perform this function, Figures 2A, 2B, and 2C show schematics for output circuits that have been used with this or similar systems. Figures 2A and 2B show schematics for ac-output digital devices. Both ac circuits use two simple chips, a TTL hex inverter (7407), and an optical isolator (MOC 3031). The optical isolator allows various types of currents and devices to be used by the digital switches without mixing the current and thus damaging the digital interface.

Figure 2A illustrates the kind of circuit used to power ac electromechanical devices that draw a lot of current, such as an ac grain hopper. A Quadrac is used to switch power on and off to the device, as opposed to switching directly through the optical isolators. Figure 2B shows how the optical isolator can be used to switch a low-current device directly. Figure 2C illustrates a circuit used to power any low-voltage dc device, and is based on the same philosophy of design as the ac circuits.

The number and kind of outputs used are left to the experimenter. In my experiments, I rarely use more than eight outputs; thus, I have never had to use more than one of the 6809 CM's two PIA chips. Outputs are added to the system by duplicating the appropriate circuits and connecting the circuits to the appropriate PIA line and electromechanical device.

The registering of experimental events (inputs) is performed by the same PIA chip. Experimental event (i.e., keypecks, barpresses) produces changes in the state of the PIA—via the digital interface—which then signals the CPU chip that an event has occurred. The input circuit incorporates a straightforward digital pulse former and signal conditioner (see Figure 3), which allows a relatively clean and consistent signal to be accepted by the PIA. Any other noise is filtered out by the software and is transparent to the user.

The input circuit shown in Figure 3 is based on an "interrupt" design philosophy. An interrupt is a signal from a PIA to the 6809 CPU that an event has occurred. This causes the CPU to stop what it is doing and take care of (i.e., service) the interrupt. An interrupt allows for events in the world to be detected much more reliably and accurately than other approaches. Other methods, such as polling, are less efficient with CPU time.

Additional inputs can be added by duplicating the signal conditioning part of the circuit and chaining the Q outputs from the 74121 into the interrupt circuit's 7486 (see Q_2 and Q_3 insert in Figure 3). In this manner, a number of input circuits can be added to the hardware with a minimum amount of digital circuitry.

Putting together a board with eight input and eight output circuits involves an expense of under $100 and a day's worth of soldering. This job is well within the skills of any accomplished solderer. An electronics technician is

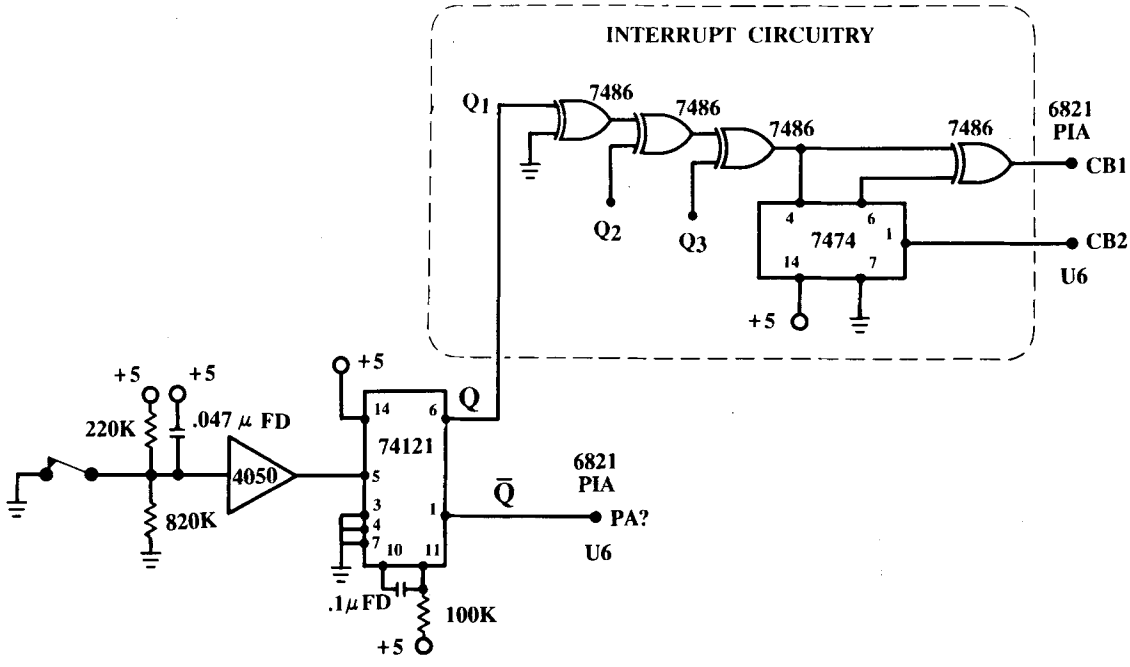

Figure 3. Input circuit for the digital interface. Both the CMOS 4050 and the TTL 7486 use standard +5 volts dc power sources. PA? refers to any input port on the 6821 PIA of the 6809 CM (chip U6). The 6821 ports CB1 and CB2 make up the interrupt hardware of the 6821 PIA and work in conjunction with the interrupt circuitry outline in this figure and the software described in the text. The interrupt logic shown in Figure 3 assumes that no two events occur simultaneously. If simultaneous events are possible within an experimental framework, then a single multiple NOR gate (7430) may be used in place of the 7486 chain. All unused 4050 inputs should be tied to V_{cc} (5+).

very handy but is not necessary for the construction of such an interface. For those who may find this task too daunting, the author will, on request, supply the schematics and specifications for building the interface, and the job may be contracted out. Presently, others are working on a commercially produced I/O board that would be suitable for this system.

The Wintek corporation does supply a number of I/O boards (for under $150) that are acceptable for behavioral research. However, they are based on a "polling" philosophy of input handling, which, although adequate for the imprecision of behavioral work, conflicts with the design of the present system. Thus, the author suggests that users of this system follow the digital interface design provided here.

THE PC

The 6809 CM uses a standard desktop IBM PC XT or PC AT with 320K RAM, two 5.25-in. double-sided, double-density disk drives, and DOS version 2.0 (or later) as its host. The PC performs a number of tasks, the most significant of which are cross-compiling programs, downloading them to the 6809 CM, transferring data from the 6809 CM, storing these data, and analyzing them. The PC is the biggest expense in the system, but it can serve a broad spectrum of functions and thus is useful when independent of the system. Presently, the manufacturer supplies only PC-based software for programming and controlling the 6809 CM; hence, a PC is essential for running the system.

The PC communicates with the 6809 CM through a standard RS-232 serial line. A PC program called the Terminal Emulator (TE), provided in the C cross-compiler/assembler package by the manufacturer, allows direct communication between the two computers. A single PC can serve a number of 6809 CMs concurrently. Thus, any number of 6809 CMs may be connected to the PC's RS-232 serial port, or ports, to create a distributed network. This network may be as simple, inexpensive, and crude as a multiple-pole switch to change between 6809 CMs, it may utilize multiple serial ports on the PC, or a network may be established by an intelligent serial controller. Any of these options are available within the present system's framework. One practical drawback is that the PC can only communicate with one 6809 CM at a time; however, because each 6809 CM may be running a different experiment, this approach is logical and not necessarily restrictive.

The PC also serves for transferring, storing, and analyzing data from an experiment. To do this, a higher level language, preferably C, is required to write programs for performing these tasks. Any commercially produced C programming language will probably work. The author will make available programs and source codes to perform some of these simple tasks.

THE C PROGRAMMING LANGUAGE AND OVER_CS

Although the 6809 CM is the heart of this system, a firm understanding of its operations is not necessary in order to use it. The ability to write the software that programs the 6809 CM is essential. The manufacturer supplies an extensive and well-written cross-compiler/assembler package for programming the 6809 CM in C on the PC. Once programs are written and compiled on the PC, they are then downloaded to the 6809 CM. Previous single-board computers (e.g., Sym, Kim) required that programs be written in machine language, which required days of work to write and test. A higher level language such as C ensures that an experiment can be devised and implemented on the same day.

The reason that the author chose C as the programming language for this system is fourfold. First, it is the only higher level language supplied by the manufacturer for the 6809 CM. Second, C is the best formal language for serving the diverse functions—from handling hardware to analyzing data—required by this system. Thus, one need only learn one language to run experiments and analyze data. Third, C is a consistent and structured language and can be made simple enough so that college undergraduates and high school students can program experiments and analyze data using C. Lastly, C was designed to be easily portable from one computer to the next. Therefore, experiments written on one computer can be transported to another computer with very little rewriting. This ensures that even if the present hardware becomes obsolete at some point in the future, the software can endure. Since the real time and energy with this system is in the software development, not all effort will be lost when changing to a new hardware system.

There is a substantial amount of "software overhead" that must be written in order to program an experiment. Fortunately, most of the software needed to run any behavioral experiment has already been written by the author. This software overhead, known as OVER_CS, is the same regardless of what experiment is programmed. Appendix A shows a C program that runs a simple FR schedule using OVER_CS; Appendix B gives an example of what the user must do to run the program. The C code listed in Appendix A are the only lines of code that need to be written by the experimenter—the rest is handled by OVER_CS. Only a few lines need to be changed in order to change the experiment from an FR- to a VR-, FI-, or VI-schedule experiment.

Because of the extensive nature of OVER_CS and its supporting PC-based software, and because OVER_CS is written in C and is therefore potentially portable to other machines, OVER_CS is a software laboratory system that is potentially independent of the 6809 CM hardware system outlined here. Thus, the present paper will only out-

line OVER_CS design philosophy and some of its capabilities. A future paper (Horner, in preparation) will outline OVER_CS functions in greater detail.

OVER_CS handles all necessary hardware functions for the 6809 CM and most of the functions necessary for running behavioral experiments. OVER_CS provides software routines for (1) testing the 6809 CM's hardware inputs and outputs, (2) storage and transfer of data between the 6809 CM and the PC, (3) formatted input of variables and text from the PC, (4) random-number generation, (5) low-level serial and parallel I/O, (6) interrupt handling of keyboard inputs, response inputs, and hardware clocks, (7) count-up and count-down clocks, and (8) some simple math functions. What are not included in OVER_CS routines are prepackaged functions for running specific kinds of schedules or experimental paradigms. This allows the experimenter maximum flexibility and control over the experimental paradigm, while providing the necessary functions for sensing, controlling, and recording experimental events.

The C programming language provides all the control-flow constructions (i.e., if-else, switch, while, for, do, and break statements) necessary for well-structured programs. C forces the programmer to produce well-structured code. OVER_CS takes advantage of this programming philosophy by providing general functions for tasks, as well as an easy-to-understand set of hierarchical programming levels. For instance, the software for the running of an experiment (see Appendix A) is really just a subroutine (i.e., do_experiment()) that is accessed from a "main" program that provides access to a number of different functions, such as: (1) testing the 6809 CM hardware inputs and outputs, (2) transferring data between the 6809 CM and the PC, and (3) setting program variables (i.e., init_var()). Each of these functions appears as a choice in a menu within the main routine.

Students find the structure of OVER_CS, with respect to both operation and programming, easy to understand. OVER_CS has undergone two years of rigorous student testing and author rewriting to make it a robust environment for behavioral experiments. In recent student use, there were no failures as a result of computer hardware or software problems. Thus, I feel confident that OVER_CS represents a reliable and convenient programming environment for behavioral experiments.

One major advantage of this system is that programming it to perform complex experiments is relatively simple. For instance, the author has programmed an experiment in which the probability of reward on a key changes over time in a sinusoidal fashion—not a simple task for a relay rack-based experiment, but a relatively simple one for a program running in C.

Another major advantage is that each experimental event (i.e., responses, food deliveries, stimulus changes, etc.) can be recorded with its time of occurrence to within 1/50 sec by using clock functions provided by OVER_CS. (Finer temporal discriminations are possible with OVER_CS but are unnecessary for the kind of work that the author does.)

OVER_CS uses a data-storage technique that codes both time and events into a single 16-bit word. The highest 4 bits code for the event, while the lowest 12 bits code for the time of the event since the beginning of the session. In a typical experiment, with a resolution of 1/50 sec, this means that the system must mark the passage of a minute in the data in order to keep track of the time in the session. This operation is transparent to the user. To calculate the amount of 8-bit memory (bytes) necessary to run an experiment, simply estimate the number of events in the session (e.g., reinforcement, responses, stimulus changes) and the total number of minutes in the session. Adding the two quantities together and multiplying by 2 (to account for 16 bits per event rather than 8 bits of memory) give an estimate of the number of bytes required. For instance, in the simple experiment in Appendix A, if one were only going to deliver 60 rewards on an FR-15 reinforcement schedule and planned to terminate the session after 30 min, 1,980 bytes of storage would be needed (60 reinforcers + 900 responses + 30 min, all multiplied by 2, = 1,980; half of that, 990 or more, is the number of events that must be specified in the text of the program; see Appendix A). One could increase the amount of storage available by using a number of other data-storage techniques or by reducing the temporal resolution or the number of events coded for; however, this technique seems to work for most behavior experiments. Because the source code for running the experiment (OVER_CS) is available to the user, the programmer is not restricted to the data-storage technique described here.

Because so much data can be stored by using the system (19,000+ individual events with the basic system; this storage capacity can be expanded to almost 430,000+ events with additional hardware), the question of data management becomes an issue. Once an experiment is finished, the data stored on the 6809 CM can be transferred and stored on the PC with the use of utilities provided by OVER_CS and supporting PC software. Once stored in files on the PC, the data may then be analyzed. Each file is given a unique name that encodes the date, experiment type, and animal number. Software for the PC, written and supplied by the author, handles data storage, data retrieval, and some cursory analysis. All data storage and transfer is transparent to the user. All the user must decide is what is to be stored; OVER_CS and its supporting PC software handles the storage and transfer. Further analysis is facilitated by PC-based routines that store, transfer, retrieve, and decode data into event and time arrays for easy access by higher level analysis routines.

Data-storage, -retrieval, and cursory-analysis software routines manipulate data in a manner that does not presume any particular kind of analysis. All that is assumed is that events occur in time; therefore, the experimenter may choose any method of analysis after the fact. The

experimenter may graphically present the data as a cumulative plot, analyze it for IRT, or even process events as a waveform in a Fast Fourier Transform by using the present storage and retrieval techniques. The experimenter is not limited to rates of responding as a dependent variable or to particular decisions made about how the data are to be analyzed prior to the experiment.

Beyond the software routines provided by the author, the experimenter is on his/her own for writing higher level analysis programs. Summary statistics (i.e., rate of responding) are obtainable from the program running on the 6809 CM, but higher levels of analysis require programming on the PC. To remain consistent with the 6809 C cross-compiler/assembler, the author uses the C programming language for higher level analysis. Other, more flexible, data manipulation can be obtained by using commercially produced spreadsheets and data bases. However, data must be converted from their raw form by programs written in a higher level language before they can be used by these software packages.

DISCUSSION

Unfortunately, no computer system presently available for the control of behavioral experiments is without cost in time and effort. The system presented here is not an exception. To use this system effectively, the C programming language must be mastered. To build this system, a digital interface for connecting the 6809 CM to the behavioral chamber must be constructed. Although simple in design, such an arrangement is at first time-consuming to learn, make, and test. An electrical engineer is invaluable in this endeavor but not indispensable.

The time it takes to get the first experiment running with this system ranges from six months to a year, depending on the user's original level of expertise and the amount of help available from local sources. Once learned, the system is relatively easy to duplicate for more experiments.

Six months to a year is a long time out of a career to devote to technology, but the benefits are worth it for researchers who wish to extend their experimental reach. Another advantage of this system is that it is inexpensive. In a period of budget austerity, a single first experiment can be put together for under $2,900 (see Table 1). This price includes a PC, the CM 6809, the digital interface, a C compiler for the PC, and the C cross-compiler/assembler package. This price does not include the cost of the behavioral chamber or any other software. Subsequent experiments can be added for a mere $600 per behavioral chamber. (The total cost may be reduced further by using a PC XT with two floppy disk drives and 320K RAM rather than a PC AT with a 40-MB hard disk, one floppy disk, and 640K RAM. For general usability, however, a more powerful PC is essential. The initial start-up cost of the system can be reduced by almost another $1,000 by purchasing a less demanding C language for the PC and by taking advantage of special manufacturer discounts.)

Table 1
Equipment Costs for a 6809 Single-Board Computer System

IBM PC 286 clone (with 40-MB HD)	$900
Wintek 6809 CM (constructed)	350
Digital interface and miscelllaneous	250
C cross-compiler/assembler, linker, and TE	950
C language for IBM	450
Total	$2,900

The present system reflects optimization among a number of design variables—cost, ease of construction, usability, power, flexibility, and life expectancy were all considered in its development. However, the design philosophy dictated that when user-friendliness conflicted with power and flexibility, the latter two won out. The basic system allows the user considerable latitude for expansion and idiosyncratic design changes (e.g., memory, I/O, serial port adaptations, bus interfaces, digital interfaces, etc.). This, combined with the C programming language, makes the system extremely flexible. For instance, by means of the RS-232 serial line, this system can be used to drive an Amiga computer, which can serve as the stimulus projector in perceptual experiments, allowing for more varied and complex visual stimuli than the usual stimulus projector or slide projector. The basic system is capable of greater data storage than is any other comparably priced system (Wynne, 1990), and its speed and timing capacity make it a rival among more expensive systems (Edgell & Hertel, 1989).

A design philosophy that stresses power and flexibility over user-friendliness does not mean that the system is user-hostile. With minimal instruction, college undergraduates or high school students can operate it without supervision. However, the system requires a sophisticated user to assemble, design experimental applications, and perform data analysis. Those seeking computer systems that stress user-friendliness should consider other applications.

The most natural comparison for this system is the Walter/Palya (W/P) computer system (Walter & Palya, 1984). The W/P system has proved itself in a number of experimental applications as an inexpensive, user-friendly, and powerful system for running behavioral experiments. Although the system described here uses much of the same technology and does many of the same things as the W/P system, the present system is not a reinvention of the W/P system. The W/P system excels over this system in its user-friendly nature, its ease of construction, and its relative cost of adding additional modules; however, the W/P system is less flexible and powerful than the present system. Which system is better depends on the circumstances of the individual researcher, the nature of the experimentation, and the laboratory. The present system provides a powerful, yet inexpensive, alternative for the control of behavioral experiments. Its design attempts to extend one's experimental reach by providing a system that allows for more complex experimental designs and more thorough data collection and analysis.

AVAILABILITY

All components except the PC and supporting PC-based software (not otherwise listed), standard behavioral equipment, and the digital interface are manufactured by the Wintek Corporation, 1801 South St., Lafayette, IN 47904-2993. The 6809 CM used in this system is a Wintek 6809 Control Module (MCH88) with a Fantom 9 on-board monitor/debugger EPROM (RRF09). The 6809 C cross-compiler/assembler, macro assembler, linker, and Terminal Emulator for the IBM PC (PCC09) were used in the host PC.

A disk containing the C source code and assembly-language source code for OVER_CS and other supporting programs for transferring, storage, and simple analysis of data can be obtained from the author for $10 to cover the cost of materials and handling.

REFERENCES

CARR, J. J. (1984). *Interfacing your microcomputer to virtually anything*. Blue Ridge Summit, PA: Tab Books.

EDGELL, S. E., & HERTEL, S. A. (1989). Running laboratory experiments using the RSX operating system and FORTRAN on the PDP-11. *Behavior Research Methods, Instruments, & Computers*, **21**, 303-306.

RATZLAFF, K. L. (1987). *Introduction to computer-assisted experimentation*. New York: Wiley.

UTTAL, W. R. (1968). "Basic Black" in computer interfaces for psychological research. *Behavior Research Methods & Instrumentation*, **1**, 35-40.

WALTER, D. E., & PALYA, W. L. (1984). An inexpensive experiment controller for stand-alone applications or distributed processing networks. *Behavior Research Methods, Instruments, & Computers*, **16**, 125-134.

WOLFE, G. W. (1982). *Computer peripherals that you can build*. Blue Ridge Summit, PA: Tab Books.

WYNNE, C. D. L. (1990). A Commodore 64-based interface system for the operant laboratory. *Behavior Research Methods, Instruments, & Computers*, **22**, 27-33.

APPENDIX A: C Program for Running an FR Schedule.

Below is a C program for running a simple FR schedule. The program incorporates many of the features of OVER_CS, such as formatted variable input and output, data storage and transfer, interrupt software control and clocks, and hardware input and output handling. Much of this program would be incorporated in any program for running behavioral experiments. The program is in the left hand column in LINE PRINT, comments are expressed on the right in **BOLD**.

```
#define     MINUTE              0x00
#define     KEY_PECK            0x02
#define     REWARD              0x09
#define     END                 0x0F

#define     EVERYTHING          0xFF
#define     FOOD                0x01
#define     KEY_LIGHT           0x02
#define     FINISH              0x80

#define     NUMBER_OF_EVENTS    3000

#define     PROG_NUM            6.0
#define     CC_CODE1            'F'
#define     CC_CODE2            'R'
#define     MESSAGE "\nFixed Ratio ---- ( %5.2f ) \n"

int         reward_received;
int         rsp_total,REWARDS,FR;
long        SESSION_LENGTH;


#define     TEST
#include    <OVER_CS.h>

/*------------ do_experiment() -------------*/
do_experiment()
{
    int result;
    clocks[2] = SESSION_LENGTH;
    rsp_total = 0;
    reward_received = 0;

    printf(" Begin Experiment - ");
    turn_off(EVERYTHING);
    turn_on(KEY_LIGHT);
```

⇐ These #define statements set certain program constants which remain the same throughout the experiment. Most of these would be the same regardless of the experiment. These #define constants determine what information is stored and how it is coded.

⇐ These #define constants determine the PIA's output for each computer controlled event.

⇐ This #define constant determines the number of events that can be stored.

⇐ These #define statements give a code for the experiment that is used by the program that transfers the data (TR) to the PC. They act to define what the experiment is and to provide a header for the experiment.

⇐ These global variables keep track of the number of rewards received during the experiment, the maximum number of rewards allowed during a session, the FR value, the maximum length of the session, and the total number of responses.

⇐ These two statements tell the compiler what software overhead to use and what part of the overhead to include in the program.

⇐ Subroutine do_experiment(). This is the subroutine that actually performs the experiment.

⇐ A dummy variable

⇐ Set the clock for length of the session.

⇐ Initialize the variables for response count and rewards received.

⇐ The experiment begins here.

⇐ These three statements inform the operator that an experiment has begun and then turns everything off and turns on the key light.

```
    setup_irq((RESPONSE|SYSTEM_TIME|CLOCK_2),0x01);   <= This statement sets up the interrupt routine to perform
                                                         3 functions: 1) to respond to a key peck (RESPONSE),
                                                         2) to keep track of the overall system time
                                                         (SYSTEM_TIME), and 3) to keep track of the length of
                                                         the session (CLOCK_2).

    while ( ( ( intflg & CLOCK_2 ) == 0 ) &&          <= This WHILE loop controls the duration of the
              ( reward_received < REWARDS ) ) {           experiment and says to continue the experiment until
                                                          either SESSION_LENGTH has passed or until the
                                                          maximum number of rewards have been delivered.

        if ( ( result = response_ck() ) != 0 ) {      <= This IF statement checks for a response and executes the
                                                         next section of code if a response has occurred.
                                                         Subroutine response_ck() returns a non-zero value if a
                                                         response has occurred. If a response has occurred then:
            turn_off(KEY_LIGHT);                      <= 1. Turn off the key light.
            remove_int(RESPONSE);                     <= 2. Don't respond to any more key pecks (for now).
            store_event(KEY_PECK);                    <= 3. Record the response.
            rsp_total++;

            if ( ( rsp_total % FR ) == 0 ) {          <= 4. Check to see if the number of responses is enough to
                                                         produce a reward. If enough then: REWARD !!
                store_event(REWARD);                  <=   4a. Record the reward.
                reward_received++;
                turn_on(FOOD);                        <=   4b. Turn on the food, wait 2.5 sec, then turn off the
                nap(2.5 SEC);                                food.
                turn_off(FOOD);
            }                                         <=   End of IF Reward statement
            turn_on(KEY_LIGHT);                       <= 5. Turn the key light back on.
            add_int(RESPONSE);                        <= 6. Reinitiate interrupt for responses.
            response_ck();                            <= 7. Check to see if a spurious response has occurred, and
                                                         if so throw it away.
        }                                             <= End of IF Response statement.
    }                                                 <= End of WHILE loop.
                                                      <= The experiment ends here.
    IRQ_off();                                        <= Turns off the interrupt routine.
    store_event(END);                                 <= Mark the end of the experiment.
    turn_off(EVERYTHING);                             <= Turn off all outputs.
    turn_on(FINISH);                                  <= Turn on the FINISH light, which informs the operator
    printf(" End of Experiment \n);                      that the experiment is over and print end of experiment
                                                         to the screen.
}                                                     <= End of subroutine do_experiment().
/*-------------- init_var() ------------------*/
init_var()                                            <= Subroutine init_var(). This subroutine allows the
{                                                        operator to enter session parameters.
    int temp_i;                                       <= Dummy variable.
    begin:                                            <= Begin statement.
    printf(" FR value ..... ");
                              scan_i(&FR);            <= Read in the value of the FR schedule.
    printf(" Maximum number of rewards ..... ");
                              scan_i(&REWARDS);       <= Read in the maximum number of rewards.
    printf(" Maximum length of session (in mins) . ");
                              scan_i(&temp_i);        <= Read in the maximum length of the session.
        SESSION_LENGTH = ( (long) temp_i ) * 3000L;
    printf(" FR value: %d Maximum rewards: %d         <= Print out variables for FR schedule, number of rewards
        Maximum time: %ld\n",                            and length of session.
        FR,REWARDS,SESSION_LENGTH / 3000L);
    printf(" Is this correct ? <y> <n> - ");          <= Operator check for validity of the parameters.
    if ( in_232() != 'y') goto begin;                 <= Return to begin statement if incorrect.
}                                                     <= End of subroutine init_var().
/*--------------- footer() ------------------*/
footer()                                              <= Subroutine footer(). This subroutine prints out
{                                                        summary statistics about the session at the end of the
    printf(" Number of rewards %d                        data file.
        Number of responses %d\n",
        reward_received,rsp_total);
    printf(" Total time in session: %d min %d sec\n",
        tot_min,(sys_time/50));
}                                                     <= End of subroutine footer().
```

APPENDIX B: Interactive Commands for Running an Experiment.

Below is the set of commands that an operator initiates in order to download and run the experimental program listed in Appendix A. The commands, on the left, also show how data is transferred from the 6809 CM to the PC. All command below are initiated on the PC's keyboard and seen on the PC's monitor. Letters in *ITALIC* are illustrative of commands given by the operator, those in LINE PRINT designate responses returned by either the 6809 CM or the PC. The back arrow sign (↙ indicates where the "Enter" key (i.e., carriage return) has been pressed. The phrase "DOS_PROMPT >" specifies where the PC is in normal MS-DOS command mode. Special key command, such as function keys, are surrounded by brackets []. Comments are expressed on the right in BOLD.

```
DOS_PROMPT > TE↙
Wintek Terminal Emulator V1.0r2
(c) Copyright 1985 Wintek Corp.
```
⇐ Once an RS-232 connection is made, the operator establishes contact with the 6809 CM by running the Terminal Emulator (TE). At this point, the PC is acting as a terminal for the 6809 CM.

```
[F9]
TE INTERRUPT!
f>[F2]
name of file to download []?a:fr.mik↙
loading address: XXXX
download complete
f>G↙
Choose a positive number ..... > 3798↙
Fixed Ratio ----  ( 6.00 )

[T] Test system
[R] Run exper.
[A] run Again
[V] View data
[S] Set rand()
[X] eXit
6.00 > R
```
⇐ Function key F9 causes TE to reset, & the 6809 CM returns the Fantom 9 prompt (f>).

⇐ Function key F2 instructs both TE and the 6809 CM to download a program, FR.MIK on the a: drive, from the PC to the 6809 CM. Once the program has been downloaded, the 6809 CM return the f>.

⇐ The command is given (G) to run the program.

⇐ First, the operator seeds the random number generator.

⇐ The operator is then given a choice from the programs main menu. The operator may either:
1) Test the system's Input/Outputs.
2) Run an experiment.
3) Run the experiment again using old parameters.
4) View the data from a session.
5) Reset the random number generator. Or....
6) Exit to Fantom 9.

⇐ The operator gives it the command to run (R) the experiment.

```
FR value ........................... > 15↙
Maximum number of rewards ......... > 60↙
Maximum length of session (in mins). > 30↙
FR value: 15  Maximum rewards: 60  Maximum time: 30
Is this correct? <y> <n> - y
Prepare experiment.  Then press any key to continue.
[Space Bar]

Begin Experiment -

                          End of Experiment
Fixed Ratio ----  ( 6.00 )
6.00 > [F1]
```
⇐ First, the session parameters are inputted. These are accepted in subroutine init_var(). The FR value, maximum number of reward, and the maximum time of the session are input, printed out, and checked for accuracy.

⇐ Then the program halts to allow for the experiment to be prepared. Once the experiment is ready, it can be started by pressing any key.

⇐ The subroutine do_experiment() is initiated and the experiment begins running.

⇐ When the experiment is finished, it returns to the main menu.

⇐ At this point the operator exits TE by the F1 function key and returns to MS-DOS.

```
DOS_PROMPT > TR FRX 99↙
Ready to receive - XXXX - got it.


Fixed Ratio ----  ( 6.00 )
Number of rewards 60   Number of responses 900
Total time in session: 24 min  13 sec
Comments: A test run.↙
File Name: a:\frx00921.99
System Time at Save: Fri Sep 21 10:33:13 1990
```
⇐ The operator runs a program (TR) that transfers the data from the 6809 CM to the PC, and places the data into a file which stores all 900 responses and the times they occurred.

⇐ The information given in the footer() subroutine is appended to the end of the file.

⇐ Operator comments may be appended to the file.

⇐ The file is then given a unique code that designates the experiment, the animal's number (both are given in the TR command line), and the date as determined from MS-DOS. The file is then stored on drive a: with information about the file, user comments and the date.

```
DOS_PROMPT > TE↙
Fixed Ratio ----  ( 6.00 )
6.00 >
```
⇐ Once TR has completed transferring the data, it returns to MS-DOS, where the operator can reinitiate TE and return to the main menu of the program.

(Manuscript received February 20, 1990;
revision accepted for publication June 4, 1991.)