SESSION II

INTRODUCTION TO THE WALTER/PALYA EXPERIMENT CONTROLLER AND ITS NETWORKING SOFTWARE; WITH USERS' GROUP MEETING

William L. Palya, *Presider*

An introduction to the Walter/Palya controller and ECBASIC

WILLIAM L. PALYA

Jacksonville State University, Jacksonville, Alabama

An overview of the Walter/Palya ECBASIC experiment controller is provided. The hardware, firmware, and underlying design considerations are presented. The controller is a simple printed circuit board microprocessor that interfaces directly to 8 inputs and 40 outputs and directly to any general-purpose computer. It can be programmed in a specially written BASIC (ECBASIC) that provides high-level commands for most experimental manipulations. It can function as either a stand-alone computer or a networkable preperipheral processor. It costs about \$150.

It has been 20 years since Uttal's (1968) article about computer interfaces for psychological research appeared in *Behavior Research Methods & Instrumentation*. Since then, the use of computers in the psychological laboratory has developed to the point that the computer is no longer simply an infertile tool for self-gratification (Sidowski, 1976). Nevertheless, a tour through the 20 volumes of this journal leaves me with the same feeling as does a war memorial: admiration for the hard won and often brilliant victories, but a parallel sense of sadness at the apparent waste. Innumerable good systems have been proposed, and hundreds of man years have been spent on their development, yet few systems are implemented outside their original site or endure for more than a few years (Castellan, 1976).

Many of these systems are developed with what appears to be the most inescapable of deductive logic but fail to become popular. Time will demonstrate whether the system presented here will be any more successful, but at present the rationale and solution appear reasonable. Currently, there are almost 100 Walter/Palya experiment controllers in service, and users' group discussions are scheduled at several professional meetings.

Initially, I tried to conceptualize an ideal lab computer. It became apparent that the following assets were critical: (1) low purchase price; (2) a simple package with no interfacing required; (3) versatility to be used in a variety of situations; (4) an easy to learn and easy to use programming language; (5) high reliability; (6) a low maintenance cost and a short mean time to repair; (7) the ability to resist obsolescence; (8) the very best operating system environment and application tools; and (9) transportability of control programs, database, and application software to other laboratories.

The solution strategy was to (1) develop a microprocessor-based experiment controller as a high-quality printed circuit board, (2) implement in firmware a specially written BASIC optimized for experiment control (ECBASIC), and (3) design the board to function as a peripheral processor on a network supervised by whatever general-purpose computer the researcher wishes to use. As a result, the board has the following assets:

1. It is inexpensive. A complete controller costs about \$150.¹
2. One board does everything. The board attaches directly to the experimental apparatus with no additional interface; it even contains its own white noise generator.
3. It is versatile within the constraints of providing only digital input and output. By providing up

---

The author gratefully acknowledges the contributions of Elizabeth Palya and the support provided by John Gibbon. Requests for reprints should be sent to the author, Department of Psychology, Jacksonville State University, Jacksonville, AL 36265.

to 8 inputs and 40 outputs, experiments with three 12-stimulus projectors can be controlled. The board can be partially populated if only a few inputs or outputs are required. Additionally, it can function as either a stand-alone computer or as a networkable controller. The board can also be used as a totally dedicated embedded controller by burning the program onto PROM.

4. Research can be easily implemented. BASIC is simple, easy to learn, and easy to use. Many people can program in BASIC, and many how-to books are available. Also, we designed into our ECBASIC many high-level process control commands and detailed error messages. It may not be structured, but it is straightforward and generally very easy to follow. This is especially true within the niche of ECBASIC (i.e., a control program for a single subject).

5. It is reliable. We have experienced *no* in-house post-start-up failures (10 boards have run for 3 years). Of the boards in the field, the only failures were associated with high levels of microwave radiation or with failure to install a jumper.

6. It can be repaired quickly and has a low maintenance cost. Immediate maintenance entails replacing a malfunctioning board with a spare. Chip failures can usually be identified by swapping chips with those from a working board. It is very unlikely that swapping chips would fail to identify the problem. The board can be discarded if maintenance costs escalate beyond \$50 or \$100. As a result, repair can never cost more than \$150.

7. It resists obsolescence. Often the pressure to upgrade a lab computer has been because of the availability of faster computers that provide better operating system utilities and application software, not because of the weakness of the process control component. Separating the experiment control subsystem from the general computer allows the researcher to upgrade or change either one whenever it is appropriate without losing the other.

8. The best operating system and application software is always available. The task of maintaining state-of-the-art and optimal general-purpose software is left to commercial vendors who can provide the resources to develop and support it. The researcher may choose whatever hardware and software is appropriate, because virtually every general-purpose computer made provides the interface necessary to connect an experiment controller.

9. Because all boards are the same, all control programs are the same. ECBASIC is therefore transportable. However, data storage and analysis standards have not yet been established.

Our solution strategy has avoided the major stumbling block that typically restricts other labs from benefiting from someone else's system. In the past, implementing

someone else's computer-controlled laboratory system was difficult. Published experiment control systems often evolved over several years, and potential users were required to buy into the whole package in one budget year. Additionally, the system was often idiosyncratic in that some of the components or the computer itself were suboptimal, obsolete, or largely undocumented local constructions. Obtaining most components of a control system was usually not adequate. Either the identical hardware was necessary or the services of an in-house expert were required. Our experiment controller apparently has avoided these pitfalls by being an inexpensive, completely self-contained printed circuit board computer with on-board ECBASIC in firmware. The only task facing the researcher is to plug it into an existing computer and attach it to the experimental apparatus.

The solution strategy also avoided the major difficulty in developing a behavioral control system. Writing and debugging a timesharing operating system is a lot of work and requires meticulous attention to detail if it is to be done well. Additionally, there is a great deal of overhead in trying to implement several processes on the same CPU. Timesharing several experiments or experiment control and general-purpose computing dramatically slows down the response time to any one task. A multiprocessor solution was the obvious choice since individual controllers are so inexpensive.

The arguments against the use of timesharing computers for laboratory control are as applicable today as they were when computers first arrived in the laboratory (Markowitz, 1969; Uttal, 1972). The only change has been that dedicated lab computers have imperceptibly evolved into large timesharing computers. The 4-MB 32-bit timesharing computer with 100 MB of disk storage is now on the desk rather than across campus.

I repeat here Walter and Palya's (1984) argument for a network of independent experiment controllers. Each cycle that the computer uses to perform the simple task of arranging the contingencies of an experiment diminishes its ability to provide expensive resources to users. As a result of the diverging niches of laboratory control and general-purpose computing, two different computers are appropriate: a computer optimized for running experiments that is relatively simple, is inexpensive, and has few resources, and a machine optimized for general-purpose use that has many resources and that provides a high degree of functionality.

Fortunately, the two distinct machines can be integrated into a single system that fulfills both needs economically. A network can be formed in which a general-purpose computer passes experiment control tasks down to inexpensive peripheral processors and provides for the orderly collection and storage of the obtained data. In this type of system, experiment control is relegated to an inexpensive machine, whereas a computer with many resources provides for general services, such as desktop publishing, software development, interactive analysis, and modeling.

THE EXPERIMENT CONTROLLER

The experiment controller is a commercially available printed circuit board about 1 ft square.² It is designed to be physically attached directly to an experimental apparatus along with a small power supply. As a result, each experimental apparatus is a self-contained unit. The only connections required are a 110-V power line and the three wires for RS-232 communication.

The experiment controller can function as a stand-alone computer or as an interface between virtually any general-purpose computer and a digital experimental apparatus. It requires no special interface for connection to either the computer or to the apparatus.

Hardware

A controller can support up to eight normally closed switches. For example, wires can be attached directly to eight Gerbrands pigeon keys or any other kind of switch. The firmware provides a 40-msec debounce delay, but the delay is adjustable. A controller can also support up to 40 outputs. A controller can be populated with 40 2-A relays, 40 300-mA integrated circuit drivers, or 4 relays and 36 drivers. These outputs can be attached directly to indicator lamps, including 110 V ac in the case of relays and any bulbs using less than 30 V dc in the case of drivers. The controller can be partially populated if only a few inputs and outputs are needed.

The firmware provides on-board diagnostics. On power-up, the general functioning of the board is certified with a memory test. Additionally, an apparatus test mode is available. When the board is in diagnostic mode, any input causes the next consecutive output to be turned on. In this mode an apparatus can be quickly tested by depressing each operandum several times and cycling through all indicator lights and other outputs.

Figure 1. A schematic of the experiment controller architecture. The typical network usage is illustrated with a connection to a main general-purpose computer. The stand-alone mode with terminal, printer, and audiotape is illustrated with dotted lines.

Figure 2. A diagram of a typical network. Several controllers are connected to a general-purpose computer through a single RS-232 line connected to a single serial port.

The controller can be used as a stand-alone computer or as a peripheral processor in a network. Stand-alone or network mode is selected by an on-board mode switch. Figure 1 illustrates how the board provides for the two modes. Normally, the controller would be connected to a general-purpose computer through its serial port. However, if desired, the board can function as a stand-alone computer by populating the printer and audiocassette portion of the board and connecting a terminal, printer, and audiocassette as illustrated in Figure 1.

Stand-Alone Mode

When the controller is in stand-alone mode, programs can be loaded either from the keyboard following line numbers or from the audiocassette with the LOAD command. Programs can be LISTED to the terminal, LLISTED to the printer, or SAVED on the audiocassette. Programs are executed with the RUN command. Data can be output to the terminal with the PRINT command or output to the printer with the LPRINT command.

Network Mode

In the network mode, the experiment controller connects to an RS-232 port (the modem port of a general-purpose computer, such as a Macintosh or IBM PC). Up to 16 controllers can be attached in parallel to the same three wires. The controllers provide for all the necessary communication protocol in firmware. The controllers ignore data on the line unless they have been specifically addressed. It is similar to a party-line telephone system. Figure 2 illustrates a typical network configuration with several controllers attached to a general-purpose computer.

Programs are generated on the main computer with an editor or word processor. The conventions of ECBASIC are followed when creating the file (line number followed by a space followed by a command). The program is then saved on disk as a text file. This is illustrated in Figure 3. A communications module, such as modem control software or one of the packages presented in the following

Figure 3. A schematic illustration of program generation. A user writes the experiment control program with the editor or word processor on the host computer and stores it on the disk of the host computer.

Figure 4. A schematic illustration of program loading. A user invokes a communication program that "wakes up" a particular controller, gets a file off the disk, and sends the file out the serial line. The attending controller loads the program.

Figure 5. A schematic illustration of the state of the host computer while experiments are running. Any user program can be running in the host computer. There is no interaction between the experiment controllers and the host computer.

Figure 6. A schematic illustration of data acquisition. A user invokes the communication program, which "wakes up" particular controllers, which then upload the data. The communication program then stores the received data on the disk.

papers, is then invoked. These communications programs wake up a particular controller and then pass the program to that controller (see Figure 4). While the experiment controllers are running, anything can be done with the host computer, including turning it off (see Figure 5). When the experiments are completed, the communications program authorizes each controller to upload data and then stores it (see Figure 6).

THE ECBASIC LANGUAGE

The experiment controller supports ECBASIC in firmware; as a result, when power is applied, the system comes up running ECBASIC.³ ECBASIC is very much like all other BASICS. Assignment of variables, program flow, and data handling are the same. Any textbook on BASIC will provide examples of how to accomplish various tasks. ECBASIC provides extensive error checking and prints appropriate error messages. Table 1 illustrates the control commands, arithmetic operators, and functions available in ECBASIC. Programs can be SAVED to or LOADED from audiotape. The program space can be initialized with NEW, programs can be executed with RUN, and programs can be LISTED to the terminal or LLISTED to the printer. Additionally, special control commands (unnecessary for the average researcher) have been added to ECBASIC. ECHO and NOECHO are provided for efficient network communication. OFF directs a controller to disconnect from the network, and OFFRUN directs a controller to disconnect and then begin running the control program. MON invokes a monitor for hackers wishing to write machine language code. It provides a wide variety of utilities.

ECBASIC was not intended as a number cruncher; therefore only add, subtract, multiply, and divide are provided. Variables are one letter. Two-dimensional arrays with up to 255 elements each are permitted. Variables will count to 2 billion (e.g., 8 months of 10-msec ticks) before rollover.

Table 1
Control Commands, Arithmetic Operators, and Functions
Available with ECBASIC

Control Commands	
SAVE/LOAD	ECHO/NOECHO
NEW	OFF/OFFRUN
RUN	MON
LIST/LLIST	
Arithmetic Operators	
+ - * /	
32-bit integer	
one-letter variables	
two-dimensional arrays	
maximum subscript 255	
Functions	
ABS (X)	absolute value
SGN (X)	sign
RANGE (X,Y)	RAND in range
PROB (X)	true $p=0$ to 1,000
RAND ( )	random number between 0 and 1,000
MOD (X,Y)	remainder X/Y
PEEK (X)	examine memory location X
FREE ( )	amount of free memory
INCH ( )	read a character from keyboard
TIME (X)	time since X
	e.g.,
	10 B=TIME (0)
	60 IF TIME (B) < 1,000 THEN GOTO 400

Functions are also provided. A function stands for a value and is therefore similar to a variable. A function can be used in any arithmetic expression as if it were simply a number or a variable. ABS (X) is evaluated as the absolute value of X. SGN(X) is evaluated as a -1, +1, or 0, depending on the value of its argument. Range (X,Y) is evaluated as a random number in the range X to Y. PROB (X) is evaluated as a 1 with the probability specified by X. RAND ( ) provides a random number. MOD (X,Y) provides the remainder of X divided by Y. PEEK(X) and FREE ( ) are provided for diehards who still wish to know what is in particular memory locations or what is the amount of free memory. INCH ( ) returns the value of a character read from the serial port (e.g., the keyboard). TIME(X) is an extremely powerful and therefore useful function. It is evaluated as the time since variable X was used to anchor time. If, at any point in the program, B is set equal to the time now or "time since 0" (B=TIME(0)), then (TIME(B)) can subsequently be used to provide the time since B. No clocks need ever be checked and no timers need ever be incremented by the researcher. ECBASIC does all of that itself.

Table 2 illustrates the program commands available with ECBASIC. The traditional BASIC commands are available. FOR and NEXT set up and delimit a loop. GOTO unconditionally redirects program flow. GOSUB and RETURN provide the ability to exit program flow from a variety of places, execute a common subroutine, and subsequently return to the exiting point in the program. ON GOTO and ON GOSUB allow for multiple path branching. IF THEN provides for branching based on a comparison. INPUT reads numerical values from the keyboard. The READ, DATA, and RESTORE commands provide the facility to use an array of predetermined values in a program. PRINT allows a program to print to the screen or send data up the network line, and LPRINT outputs data to the line printer. REM provides for comments in a program. DIM allocates space for variables. STOP terminates program execution, and END signifies the last line of a program.

A number of experiment control commands have been added. Most are self-explanatory. TURNON turns on and TURNOFF turns off one of the 40 outputs. PULSON pulses on and PULSOFF pulses off a particular output for some time and then returns that output to its previous state. SLEEP halts program execution for a time period. FREEZE, BLANK, and THAW are for storing the status of the outputs, turning off all outputs, and subsequently returning them to their previous state, respectively, as is often desired for a reinforcement cycle. Typically, STICK, HTICK, POKE, CALL, and CLRVAR are unnecessary; they are provided for the more serious programmers. COUNT initiates the counting of an input and, like TIME ( ), is an extremely useful aspect of ECBASIC. ECBASIC can automatically and transparently increment a designated variable every time a response occurs on a particular input. For example, following COUNT 1,R, all responses on Key 1 automatically increment variable R.

Table 2
Program Commands Available with ECBASIC

FOR/NEXT	TURNON X	turn on X
GOTO	TURNOFF X	turn off X
GOSUB/RETURN	PULSON X,Y	turn on X for duration Y
ON X GOTO Y,Z	PULSOFF X,Y	turn off X for duration Y
ON X GOSUB Y,Z	SLEEP X	halt for X time
IF X ☐ Y THEN command	FREEZE	store current condition of outputs
↑ >, <, =, >=, <=, < >	BLANK	turn off all outputs
INPUT	THAW	restore saved output states
READ/DATA/RESTORE	STICK X,A	start ticker X decrementing variable A
PRINT	HTICK X	halt ticker X
LPRINT	POKE X,Y	at location X deposit byte Y
REM	CALL	call machine language subroutine
DIM	CLRVAR	clear variables
STOP	COUNT X,R	start input X incrementing variable R
END		e.g.,
		10 COUNT 1,R
		50 S=R
		60 IF R < S+100 THEN GOTO 982

The researcher need only base decisions on the current value of that variable or on the difference between its current value and some previous value at an anchor point.

SAMPLE PROGRAMS

Table 3 illustrates the main program sections of four simple schedules. On Line 10 of the first example, variable *S* is used to anchor the current value of a peck counter, much as a yardage chain is used in a football game. Line 20 simply specifies that if the current value of the

peck counter is less than 100 past the anchor point, ECBASIC is to execute that comparison again. ECBASIC will "hang" at Line 20 until the total pecks exceed the anchor by 100 (counting and timing are automatically handled by ECBASIC; therefore, hanging at a particular line has no adverse consequences). When 100 responses since the anchor have occurred, the reinforcement relay is operated for 4 sec (400 10-msec ticks). After reinforcement, ECBASIC returns to Line 10, reanchors the peck counter, and waits again.

A variable-interval schedule (with prespecified values) is illustrated in the second example. Presuming an array "V" with 25 elements, Line 10 specifies that the program is to hang for a number of 10-msec ticks specified by a random element of the 25 elements in array V. Line 20 anchors the current number of pecks, and Line 30 waits for another peck. When a peck occurs, a 4-sec reinforcer occurs, after which the program is run through again. The next two program examples simply concatenate two simple schedules.

REFERENCES

- CASTELLAN, N. J., JR. (1976). Standards: Real-time extensions to programming systems. *Behavior Research Methods & Instrumentation*, 8, 207-210.
- MARKOWITZ, J. (1969). Can a small dedicated machine find happiness in a company that pioneered time-sharing? *Behavior Research Methods & Instrumentation*, 1, 269-272.
- SIDOWSKI, J. B. (1976). On-line instrumentation in psychology: Dildo or the real thing? *Behavior Research Methods & Instrumentation*, 8, 52-55.
- UTTAL, W. R. (1968). "Basic Black" in computer interfaces for psychological research. *Behavior Research Methods & Instrumentation*, 1, 35-40.
- UTTAL, W. R. (1972). Misuse, abuse, overuse, and unuse of on-line computer facilities by psychologists. *Behavior Research Methods & Instrumentation*, 4, 55-59.
- 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.
- WALTER, D. E., & PALYA, W. L. (1985). *Document set for experiment*

Table 3
Example Schedules

Schedule	Program
Fixed Ratio 100	10 S=P 20 IF P < (S+100) THEN GOTO 20 30 PULSON 3,400 40 GOTO 10
Variable Interval	10 SLEEP V (RANGE (1,25)) 20 S=P 30 IF P=S THEN GOTO 30 40 PULSON 3,400 50 GOTO 10
Second Order	10 FOR I=1 TO 5 20 S=P 30 IF P < (S+50) THEN GOTO 30 40 PULSON 2,25 50 NEXT I 60 PULSON 3,400 70 GOTO 10
Chain	10 TURNON 5 20 SLEEP V (RANGE (1,25)) 30 S=P 40 IF P=S THEN GOTO 40 50 TURNOFF 5 60 TURNON 6 70 S=P 80 IF P < (S+100) THEN GOTO 80 90 TURNOFF 6 100 PULSON 3,400 110 GOTO 10

controller. Jacksonville, AL: Jacksonville State University, Department of Psychology.

NOTES

1. This is the approximate cost of buying all the parts and constructing the board. The cost can be reduced by obtaining many of the parts from your university's electronics shop. It is strongly recommended that users build their own boards; however, if that is not feasible, working boards can be obtained through the author for \$500.

2. Detailed information (166 pages) on the hardware, ECBASIC, and ECL (a compiled, ALGOL-like language) is available in the manual *Document Set for Experiment Controller* (Walter & Palya, 1985). It is available from the author for the cost of production and mailing (\$5).

3. The experiment controller actually supports two languages, ECBASIC and ECL. ECL provides 1 msec timing and automatic recording of all events. To date, ECL can be used only if the network control computer is a PDP-11 or if you can modify a C program to run on your system. An overview of ECL is available in Walter and Palya (1984).

Notice

Publications Office Accepts Manuscripts on Disk

The Psychonomic Society Publications Office is now able to accept manuscripts on IBM PC or compatible floppy disks, and authors of accepted manuscripts are encouraged to make use of this new capability. The use of such disks, in conjunction with typewritten manuscripts, should both reduce journal production costs and shorten publication lag time.

To use this new system, send your disk containing the file to the journal's editor *with* the final typewritten revision of your accepted manuscript. Note that the electronic version should match exactly the hard-copy version of the manuscript.

Use an ASCII format on a 360-kilobyte, double-sided, double-density IBM PC or compatible disk, and be sure to indicate the name of the software used. Also, although not required, it would be helpful if the electronic version was adjusted to eliminate (1) hyphens at the ends of lines; (2) right-hand justifications; and (3) specialized word processor commands, such as those that create underlines, boldface, centering, tabs, and indents, or those that insert excessive space into the text, such as page commands.