Expanding MEL response box to accommodate up to 16 external buttons

YVES LACOUTURE

Université Laval, Quebec, Quebec, Canada

Micro Experimental Laboratory (MEL) is a widely used software package developed for the automation of psychological experiments. This report presents a simple way to implement and use an expanded MEL response box that accommodates up to 16 external switches. Hardware and software considerations are reviewed, and relevant MEL code is provided. This includes a simple algorithm that permits the reading of multiple responses within a single trial.

Micro Experimental Laboratory (MEL), marketed by Psychology Software Tools, Inc. (PST; Schneider, 1988), is widely used in psychological experiments. The package provides an interface that allows the user to assemble different types of forms to permit the presentation of stimuli on a computer screen and the collection of subject responses. Of particular interest are MEL event tables and timing routines which allow precise measurement of response latencies. Response collection can be performed easily through the computer keyboard or with the PST response box, which provides four response buttons and a voice-key interface. Unfortunately, the MEL system does not provide an easy way to interface the software with other external devices. It is therefore difficult to run experiments where a special key arrangement is required or where the experimenter wishes to record and time the closing of several switches activated by the subject.

This paper presents a simple way to expand the standard PST response box to accommodate up to 16 external buttons. The hardware of the response box was designed to be expandable, but the basic MEL software does not support this expansion and custom-made MEL code is needed. Both the required software and hardware implementation are described in this paper.

Hardware Consideration

The MEL response box is connected to the parallel port of an MS-DOS personal computer. In normal mode, the box uses 3 bits to encode responses from any of the four

response buttons or the voice key. Bits 1 and 2 are used to encode the closure of one of four response buttons of the box, Bit 3 indicates triggering of the voice key, and Bit 4 generates an interrupt signal whenever a response (voice key or button) is detected. Two jumpers on the printed circuit board of the box allow for the modification of the encoding method. One jumper is located between points "A" and "2" on the PC board; the other is located between points "1" and "P." As indicated on Figure 1a, these two jumpers may be removed and replaced by two jumpers placed, respectively, between points "2" and "C" and between points "1" and "P." When modified in this fashion, the box uses 4 bits to encode the closure of up to 16 switches, while a fifth bit is used to generate the interrupt signal that marks the occurrence of a response. It may be convenient to install a switch to replace the jumpers and easily change the box mode back and forth between normal (4 buttons) and expanded (16 buttons) mode.

The box has 4 input bits. In normal mode, each one corresponds to one of the input buttons. When the jumpers are changed, the response box accommodates up to 16 switches connected in a 4×4 matrix, as described in Figure 1b. This allows for the encoding of the closure of any of the 16 switches using the 4 input bits.

Software Consideration

From a software point of view, reading a response box in normal mode is a simple matter. The frame form, which is generally used to present stimuli, has an "input mode" slot. This slot is filled whenever the presentation of a stimulus or text on the screen is followed by the collection of a response. For example, if the input mode is set to "key," the program waits for a press on the computer keyboard. The response box is read when the input mode is set to "rbox." The software will then wait for a keypress on one of the four buttons of the response box or for a trigger of the voice key. Responses detected by the box are labeled 1-5. Responses 1-4 correspond to buttons, and Response 5 is the voice key.

To read the responses collected by an expanded box, it is necessary to program in MEL code. This is done by fill-

This work was supported by a grant from the Natural Sciences and Engineering Research Council of Canada, an FCAR (Quebec) New Researcher Grant, and an FCAR (Quebec) team grant. The author is a user of MEL but is in no other way involved in Psychology Software Tools, Inc. The author is grateful to Michel Vinette, electronic technician, who contributed greatly to his understanding of the response-box hardware and provided drafts of the wiring diagrams, and to Addie Dutta and an anonymous reviewer for their careful reviews and constructive criticism of an earlier version of this report. The author's mailing address is: Laboratoire de Recherche en Psychologie Cognitive, École de Psychologie, Faculté des Sciences Sociales, Université Laval, Quebec, PQ Canada G1K 7P4 (e-mail: yves@psy.ulaval.ca).

Figure 1. Wiring diagram. (a) A bipolar switch replacing the two jumpers on the PC board. Numbers and letters on the diagram correspond to connecting points of the box's PC board. (b) Sixteen switches (S1-S16) connected to the 4 input bits of the response box. Numbers on the diagram correspond to connecting points of the box's flat connector.

ing the "input-mode" slot of the frame form with the keyword "code." The "port#/length" slot indicates which form code is to be executed to collect the response. The specified form code, analogous to a subroutine, is called each time the box is to be read.

To understand how to read an expanded box, it is necessary to describe how the response box processes a keypress. This takes place in two steps. First the box signals the occurrence of the response by raising the interrupt bit (Bit 6 on the parallel port). In the second processing step, the box determines which key was pressed and encodes the response in a suitable pattern of bits. The box needs a short delay to process and encode the keypress. Using MEL code, the interrupt can be detected by a `PORT_IN` statement checking Bit 6 on the printer port. The `PORT_IN`

command allows for the exact measurement of latencies. After a brief delay, a second `PORT_IN` statement is needed to get the content of the port, which is then available using the `PORT_VALUE` command.

Once the port value is obtained, it is necessary to decode the bit pattern and set MEL's internal variable that indicates which response was provided. When a response is collected from a standard box or computer keyboard, MEL automatically sets one of two internal variables to represent it. Variable `FORM_KEY` is used when the "input-mode" slot of the frame form is set to "key" or "rbox." Variable `FORM_INPUT_STRING` is used when the input-mode slot is set to "string." However, when the input mode slot of a frame form is set to "code," MEL uses either internal variable, depending on *the last frame form generated*


```

|                                     CODE SPECIFICATIONS   #255 c:\mel\box1
| COMMENT
| Enter MEL code exactly as you want it written to the experiment file.
|
| port_out(890,256)    ! output to printer port and reset the Box
| port_out(890,0)      ! 890 is the address of the printer port (out)
|                     ! on most MS-DOS computers
|
| PORT_IN( FORM_KEY[5],"6", "W",889) ! wait for the interrupt line (bit 6)
| WAIT( 30000 )        ! wait up to 30 sec. or until response
| EXECUTE              ! 890 is the address of the printer port (in)
|                     ! on most MS-DOS computers
|
| tone(500)           ! makes a click when key press
| wait(20)            ! gives time to set bit 3457
| tone(0)             !
|
| port_in(form_key[5],"", "e",889) ! set port value
| execute              ! flush event table
| util=port_value      ! get port value
|
| pat=" "
| if(util=79) pat="0"           !test which button was pressed
| if(util=95) pat="9"
| if(util=87) pat="8"
| if(util=71) pat="7"
| if(util=207) pat="6"
| if(util=223) pat="5"
| if(util=215) pat="4"
| if(util=199) pat="3"
| if(util=111) pat="2"
| if(util=127) pat="1"
| if(util=119) pat="A"
| if(util=113) pat="B"
| if(util=239) pat="C"
| if(util=255) pat="D"
| if(util=247) pat="E"
| if(util=231) pat="F"
|
| form_key[5]=pat           ! set internal variable
| form_input_string=pat     ! index is number of form frame calling this code

```

Figure 3. Code form #255, which reads the expanded response box. The code is called as a subroutine to measure latency, decode the output of the response box, and set MEL's internal variables.

In this example, it is assumed that the maximum number of responses allowed within a single trial is 10. The variable "nresp" will be used to record the number of keypresses performed by the subject during the current trial; "duration" represents the total duration of the trial—this is the time during which keypresses are recorded; "rrt" is an array of integers that holds RTs, and "resu" is a string that holds the concatenated responses.

Code form #254, presented in Figure 4, is a modified version of code form #255 that has a main loop and allows for the reading of several keypresses within a single trial.

The responses are concatenated in the string "resu," and individual RTs are stored in the array of integer "rrt." The rrt[1] value represents the elapsed time between the beginning of the trial and the first keypress, rrt[2] represents the time elapsed between the first and the second keypresses, and so on. The time needed by the computer to decode the response is not included in the latency measurement. Moreover, on a 386 or 486 computer, the time needed to perform the main loop is a negligible quantity. Note that additional code specific to the experiment will be needed to save the RT values.

```

|_____ CODE SPECIFICATIONS #254_c:\mel\box2 _____|
|COMMENT INPUT DU CLAVIER 16|
|Enter MEL code exactly as you want it written to the experiment file.|
|
| ! code to read multiple key presses within a single trial
|
|resu=**                ! clear previous result
|for i=1 to 10 do rrt[i]=0    ! clear previous RTs
|
|duration=30000          ! total duration of trial
|nresp=1                ! reset number of within trial key press
|while (duration>0) do
|  begin
|    port_out(890,256)    ! reset the response box
|    port_out(890,0)
|
|    latency(form_rt)      !reset rt counter
|    PORT_IN( FORM_KEY[5],"6", "W",889) ! wait for the interrupt line (bit 6)
|    WAIT( 30000 )        ! wait up to 30 sec. or until response
|    EXECUTE              ! 890 is the address of the printer port (in)
|                        ! on most MS-DOS computers
|
|    tone(500)    ! makes a click when key press
|    wait(20)    ! gives time to set bits 3457
|    tone(0)    !
|    port_in(form_key[5],"", "e",889) ! set port value
|    execute                ! flush event table
|    util=port_value        ! get port value
|    pat=" "
|    if(util=79) pat="0"          !test which button was pressed
|    if(util=95) pat="9"
|    if(util=87) pat="8"
|    if(util=71) pat="7"
|    if(util=207) pat="6"
|    if(util=223) pat="5"
|    if(util=215) pat="4"
|    if(util=199) pat="3"
|    if(util=111) pat="2"
|    if(util=127) pat="1"
|    if(util=119) pat="A"
|    if(util=113) pat="B"
|    if(util=239) pat="C"
|    if(util=255) pat="D"
|    if(util=247) pat="E"
|    if(util=231) pat="F"
|
|    if (form_rt=0) then duration = 0    ! no response
|    else
|      begin
|        rrt[nresp]=form_rt    ! set array variable
|        resu=resu+pat        ! concatenate the key press value in the string
|        duration=duration-form_rt ! set the remaining within trial time
|        nresp=nresp+1        ! increment the counter
|        if(nresp>10) do duration=0 ! maximum number of key presses reached: EXIT
|      end
|    end
|    form_input_string=resu    ! set internal variable according
|                            ! according to resu string

```

Figure 4. Code form #254, which reads the expanded response box and allows for multiple keypresses within a single trial.

Reading an Expanded Response Box with MEL Version 2.0

It is expected that MEL Version 2.0 will be ready for shipment soon and, according to PST, the new version will handle the current response box, although, as in MEL Version 1.0, there will be no provision to read an expanded "old" box. MEL's Version 2.0 control language will be an improved version of the current one, but its basic characteristics will remain the same. The code reported in this paper is based on the PORT_IN and PORT_OUT control-language statements, which will be available in the next ver-

sion of MEL, and should adequately run under Version 2.0 of the software.

REFERENCE

- SCHNEIDER, W. (1988). Micro Experimental Laboratory: An integrated system for IBM PC compatibles. *Behavior Research Methods, Instruments, & Computers*, **20**, 206-217.

(Manuscript received June 9, 1994;
revision accepted for publication October 6, 1994.)