Monitoring running-wheel movement using a serial mouse and an IBM-compatible system

ALLEN D. SZALDA-PETREE, ANDREA M. KARKOWSKI,
LISA R. BROOKS, and NABIL F. HADDAD
University of Montana, Missoula, Montana

A running-wheel-movement-detection system is described for use with an IBM-compatible system and a serial mouse. It is an adaptation of a system developed for use on a Commodore 64. The IBM-compatible system provides obvious advantages in speed, memory, data storage, and programming ease over the earlier Commodore 64 system.

The increased reliance on IBM-compatible computers for behavioral research has resulted in the use of various response devices, such as the keyboard, mouse, and joystick, to record behaviors (Crosbie, 1990; Morris, 1992; Segalowitz & Graves, 1990; Washburn, Hopkins, & Rumbaugh, 1991). For example, various methods have been described for the use of the mouse button as a reaction time response device (Beringer, 1992; Morris, 1992; Segalowitz & Graves, 1990). Although the mouse has become a useful tool for behavioral research, it seems that another unique quality of the mouse has been overlooked; the mouse is an analog-to-digital converter that allows discrete measurement of continuous behaviors, such as wheel-running.

We recently presented a paper describing a wheel-movement-detection system, which utilized a modified Commodore 1350 mouse and a Commodore 64 computer, capable of detecting partial revolutions and changes in direction, and requiring no interface equipment (Petree, Haddad, & Berger, 1992). Although the Commodore 64 has been a useful laboratory computer, the limitations in CPU speed, memory, and availability have necessitated a change in many laboratories to IBM-compatible machines, which are vastly more powerful. Moreover, most AT-class IBM-compatibles support up to four serial ports, which would allow monitoring of four wheels per computer.

The pulley reduction system presented for the Commodore 64 can be used for IBM-class machines, but reading the mouse and serial port on the IBM is a more complicated procedure than it is on the Commodore 64. Specifically, the photo-interrupt signals for each axis and direction of movement from the Commodore 1350 mouse are sent along single conductors, and movement is determined by assessing the state of the appropriate input pin of the serial port. Because the serial ports on IBM-compatibles are used by a variety of input/output devices, each serial

port must be configured to match the individual piece of hardware—in this case, a mouse. In addition, the series of bytes conveying information about button and movement states, which is sent along a single conductor, must be decoded. The series of bytes is called a "byte pack" and varies in number depending on the individual mouse specifications. For most applications, the port configuration and decoding of the byte pack are performed by the mouse-driver program that accompanies the mouse.

The simplest system design would be one computer monitoring one running wheel. Implementation of this system would require the installation of the mouse driver, connection of the mouse to the appropriate serial port (usually COM 1), and generation of a program, in any of a variety of languages, to read the 0×33 interrupt registers (see Listing 1). With the mouse driver installed, the serial port and mouse baud rates are synchronized, and any change in mouse state, either a buttonpress or movement of the mouse, results in the generation of a 0×33 interrupt. Upon interrupt, the mouse driver updates internal variables—for example, buttonpress and movement information (see Hogan, 1991, for a detailed list of mouse functions), and returns.

A more efficient system design would consist of only one computer monitoring multiple running wheels. However, with a multiple-wheel system, the mouse driver program cannot be used because of two design limitations in IBM-compatible machines. First, although IBM-compatibles are capable of using up to four serial ports (in some cases more), there are only two hardware interrupts available for serial ports. Second, only a single software interrupt has been designed for mouse information for IBM-compatible machines. To overcome this obstacle, it is necessary to generate programs that configure each serial port used and to decode the byte pack of information sent by each mouse without the use of interrupts.

Configuration of the serial port requires synchronization of the baud rates for the serial port and the input device, which is accomplished by setting the line control register to access the baud rate divisor latch register

Correspondence regarding this article should be sent to A. Szalda-Petree, Department of Psychology, University of Montana, Missoula, MT 59812 (e-mail: py_adp@lewis.umt.edu).

(DLR). The DLR LO and HI bytes are then set to define the baud rate for the serial port. In the present apparatus, the mouse baud rate is 1200 bps, so the divisor latch LO and HI bytes are set to 96 and 0, respectively. The line control register is then returned to a parity-enabled state, releasing the baud rate divisor latch.

Reading and decoding the mouse input requires a polling routine equipped with directional accumulators (see Listing 2). The mouse used in the present apparatus is a JDR-3-Button mouse (Model JDR-MOUSE-3) that sends state information in a single byte pack. Because mouse byte-pack size and coding are not standardized, the program example presented will only function for the JDR mouse, but most mice should work after simple modifications in the program.

Direction of movement is bit-coded: Clockwise movement returns only the first bit on, and counterclockwise movement returns all bits on. Directional states are stored in an array and then totaled in a directional accumulator by using a 4-bit shift to decrease the size of the numbers to a manageable resolution. The parameters of the present apparatus generate 250 photo interrupts per wheel rotation (0.45 cm/interrupt).

Recently, much discussion has been focused on the accuracy and reliability of serial port interfaces and various response devices (keyboard, serial mouse, and bus mouse) to record behaviors (Beringer, 1992; Morris, 1992; Segalowitz & Graves, 1990). Specifically, it has been noted for reaction time studies that although accurate millisecond-timing programs have been generated, there is little information on response registration times for individual response devices. For example, Segalowitz and Graves determined that response registration delay for the Microsoft serial mouse was 31 msec. Thus, for accurate response times, 31 msec must be subtracted from the original response time. Furthermore, activation of the photo-interrupter modules increases the response registration delay to between 30 and 60 msec. Since the wheel-

movement-detection system described above uses the photo-interrupter modules, it is expected that the registration delays are reasonably similar to those found by Segalowitz and Graves for the Microsoft mouse. Therefore, if this system were used in a study requiring precise timing (e.g., in milliseconds), correction algorithms based on individual hardware specifications would have to be used to provide accurate timing.

The adaptation of the Commodore 64 running-wheel-movement-detection system to an IBM-compatible computer provides clear advantages in speed, memory, and data storage. Additionally, the decrease in new programming languages available for the Commodore 64 and a rapid increase in the sophistication and ease of use for IBM-compatible languages such as Turbo C++ and Turbo Pascal make IBM-compatible designs more appealing.

REFERENCES

- BERINGER, J. (1992). Timing accuracy of mouse response registration on the IBM microcomputer family. *Behavior Research Methods, Instruments, & Computers*, **24**, 486-490.
- CROSBIE, J. (1990). The Microsoft mouse as a multipurpose response device for the IBM PC/XT/AT. *Behavior Research Methods, Instruments, & Computers*, **22**, 305-316.
- HOGAN, T. (1991). *The programmer's PC sourcebook*. Redmond, WA: Microsoft Press.
- MORRIS, C. C. (1992). Using the IBM-compatible microcomputer's serial port as an input-output interface. *Behavior Research Methods, Instruments, & Computers*, **24**, 456-460.
- PETREE, A. D., HADDAD, N. F., & BERGER, L. H. (1992). A simple and sensitive method for monitoring running-wheel movement. *Behavior Research Methods, Instruments, & Computers*, **24**, 412-413.
- SEGALOWITZ, S. J., & GRAVES, R. E. (1990). Suitability of the IBM XT, AT, and PS/2 keyboard, mouse, and game port as response devices in reaction time paradigms. *Behavior Research Methods, Instruments, & Computers*, **22**, 283-289.
- WASHBURN, D., HOPKINS, W., & RUMBAUGH, D. (1991). Perceived control in rhesus monkeys (*Macaca mulatta*): Enhanced video-task performance. *Journal of Experimental Psychology: Animal Behavior Processes*, **17**, 123-129.

LISTING 1

C++ Routine for Scanning Running Wheel Movement for One Computer and One Running Wheel

[illegible]

LISTING 2
C + + Routines for Scanning Running Wheel
Movement for One Computer and Two Running Wheels

```
// SERIAL PORT 1 DEFINITIONS
#define p1__LO 0x03F8
#define p1__HI 0x03F9
#define p1__line__control 0x03FB

// SERIAL PORT 2 DEFINITIONS
#define p2__LO 0x02F8
#define p2__HI 0x02F9
#define p2__line__control 0x02FB

void set__port__states(void)
{
    output(p1__line__control, 0x80);           // Set line control to access baud
                                              // rate divisor latch register (DLR)

    output(p1__LO, 0x60);                     // Set DLR (LO byte) for 1200 baud
    output(p1__HI, 0x00);                     // Set DLR (HI byte) for 1200 baud
    output(p1__line__control, 0x03);          // Return line control

    output(p2__line__control, 0x80);
    output(p2__LO, 0x60);
    output(p2__HI, 0x00);
    output(p2__line__control, 0x03);
}

int movement__p1[256] = {0};                 // Mouse register array - Port 1
int movement__p2[256] = {0};                 // Mouse register array - Port 2

void read__wheels (void)
{
    for (i=0; i<300; i++)
    {
        movement__p1[inport(p1__LO)]++;      // Read byte values from Port 1
        movement__p2[inport(p2__LO)]++;      // Read byte values from Port 2
    }

    /* Accumulators for clockwise & counterclockwise movements */
    cw__p1 += movement__p1[1] >> 4; ccw__p1 += movement__p1[255] >> 4;
    cw__p2 += movement__p2[1] >> 4; ccw__p2 += movement__p2[255] >> 4;

    /* Reset directional registers to 0 */
    movement__p1[1] = movement__p1[255] = movement__p2[1] = movement__p2[255] = 0;
}

```

(Manuscript received June 1, 1993;
revision accepted for publication October 9, 1993.)