

A PC parallel port button box provides millisecond response time accuracy under Linux

NEIL STEWART

University of Warwick, Coventry, England

For psychologists, it is sometimes necessary to measure people's reaction times to the nearest millisecond. This article describes how to use the PC parallel port to receive signals from a button box to achieve millisecond response time accuracy. The workings of the parallel port, the corresponding port addresses, and a simple Linux program for controlling the port are described. A test of the speed and reliability of button box signal detection is reported. If the reader is moderately familiar with Linux, this article should provide sufficient instruction for him or her to build and test his or her own parallel port button box. This article also describes how the parallel port could be used to control an external apparatus.

In many psychology experiments, a stimulus is presented, and the latency of a participant's response is recorded. There are two key pieces of information that are required to determine a participant's reaction time: the time at which the stimulus was presented and the time at which the participant responded. I have described how to present stimuli with millisecond accuracy under Linux in Stewart (2006). The present article is concerned with recording the time at which a response is made to the nearest millisecond.¹

Keyboards and mice are often not suitable for recording responses with millisecond accuracy for two reasons. First, these devices introduce a variable delay when responding to a buttonpress. Segalowitz and Graves (1990) reported that normal keyboards are scanned only every 10 msec, although some USB keyboards reportedly have a 1-msec scan period. Plant, Hammond, and Whitehouse (2003) tested 8 mice (including serial, PS/2, and USB mice), a keyboard, and a serial port button box from Psychology Software Tools. They used a signal generator to simulate a buttonpress and recorded the response latency of the device, using an oscilloscope. Out of all of the devices tested, the range of the response latency data was less than 1 msec for only 1 of the mice, with some devices exhibiting a range as high as 30 msec. Plant et al. (2003) also highlighted the fact that some mice that appear to be identical and share a model number have quite different response time characteristics, because they, in fact, have different internal chips and microcodes. The second reason is that the physical layout of the keyboard keys or mouse

buttons is not always appropriate for collecting responses, especially if there are more than two alternatives.

In this article, I will describe how to build and test a simple button box that can be connected to the parallel port and used to register response times with millisecond accuracy. Although I concentrate on the input function of the parallel port, the methods and source code presented here could also be used for output functions, such as controlling a feed hopper or advancing a slide projector (see Sorokin, 2002).

The PC Parallel Port

Nearly all modern PCs have a parallel port, which was originally designed to communicate with a printer. If your PC is not equipped with a parallel port (some modern PCs ship without them), you can purchase a parallel port PCI expansion card quite cheaply. Indeed, it is a good idea to do this, to avoid damaging a port that is integrated into the (much more expensive) main motherboard. The port is normally a 25-pin DB socket. Table 1 shows the assignment of signal names to pin numbers. Some of the pins receive inputs to the PC and can be used to detect the states of buttons. Other pins can be set as outputs.

Modern parallel ports come in a number of types: the original, standard, output-only parallel port; the extended, bidirectional parallel port; the extended capabilities parallel port (ECP); and the enhanced parallel port (EPP). In this article, I shall assume an original parallel port, although I will describe a solution for the bidirectional parallel port at the end of the article. Many ECPs and EPPs can be made to emulate standard and bidirectional ports.² Axelson (1997) has provided a complete description of the parallel port.

The Port Addresses

The port registers are bytes of memory that set and reflect the status of the voltages on the pins of the parallel port. Standard and bidirectional parallel ports have three

I thank Martin Aldred and Dave Sleight for their help in constructing leads and button boxes and Matthew Roberts and James Adelman for many Linux discussions. Correspondence concerning this article should be addressed to N. Stewart, Department of Psychology, University of Warwick, Coventry CV4 7AL, England (e-mail: neil.stewart@warwick.ac.uk).

Table 1
Pin Names for a 25-Pin Parallel Port

Pin Number	Name	Direction from PC
1	Strobe	output
2	Data 0	output
3	Data 1	output
4	Data 2	output
5	Data 3	output
6	Data 4	output
7	Data 5	output
8	Data 6	output
9	Data 7	output
10	ACK	input
11	Busy	input
12	Paper Empty	input
13	Select	input
14	Auto Feed	output
15	Error	input
16	Initialize Printer	output
17	Select Input	output
18–25	Ground	

1-byte port addresses (the data port, the status port, and the control port). They are arranged sequentially in the computer's memory. Table 2 shows the assignment of the bits of each port to signal names. Some of the bits use inverted logic (i.e., 0 = high and 1 = low) to prevent unconnected printers from initializing and spooling and unconnected PCs from trying to send data.

Normally the first parallel port (lp1) has a base address of `BASEPORT = 0x378`. (The prefix 0x denotes hexadecimal numbers.) If there are any additional parallel ports, they normally have `BASEPORT` addresses of `0x278` and `0x3bc`. Some BIOSs allow the user to select which `BASEPORT` their parallel port uses. By accessing the addresses `BASEPORT`, `BASEPORT + 1`, and `BASEPORT + 2` in software, one can set and read the state of the parallel port.

When a bit of either the control or the data port is set (1), the corresponding output pin is set to a high voltage of more than 2.4 V. When the bit is unset (0), the corresponding output pin in the parallel port is set to a low voltage of less than 0.4 V. The pins remain latched in this pattern until the bit is next altered. Bits in the status port reflect the voltage applied to the status pins and can be used for input purposes. Bits will be set if the voltage is above 2.4 V.

Testing Control of the Parallel Port

To test the functioning of the parallel port, I made a lead to connect the parallel ports of two PCs together. I wired the first five data pins on one machine to the five status pins on the other (i.e., 2 to 15, 3 to 13, 4 to 12, 5 to 10, and 6 to 11) and vice versa (15 to 2, 13 to 3, 12 to 4, 10 to 5, and 11 to 6). I also connected a ground pin (18 to 18). I then wrote a simple C program (`test.c`)³ for Linux⁴ to write a 5-bit number to the data pins and read a 5-bit number from the status pins. The program first requests access to the port registers using the `ioperm` system call:

```
ioperm (BASEPORT, 3, 1);
```

requests access to the first 3 bytes from the address `BASEPORT`. Failure to do this will result in a segmentation fault at run time. At the end of the program, access is relinquished with

```
ioperm (BASEPORT, 3, 0)).
```

Next, a 5-bit number is sent to the output pins, using the `outb` call. Then

```
outb (send, BASEPORT);
```

sends the byte `send` to the port at address `BASEPORT`. Because only Pins 2–6 are connected (i.e., Data0–Data4), only the first 5 bits of the byte `send` are actually transmitted. Finally, the five connected status pins are read using the call `inb`:

```
printf ("Received %d\n",
(inb (BASEPORT + 1) >> 3) ^ 0x10).
```

The signal must be bit shifted left (`>> 3`) to delete the three lowest order bits from the status port (which are not connected) and then XORed with `0x10` to uninvert the busy bit signal, which is inverted in the parallel port hardware. If the ports on the two machines are functioning as expected, running `./test X` on one machine should make a subsequent running of `./test Y` on the other machine report a value of `X`.

Testing the Latency of the Parallel Port

To test the latency of the parallel port, I sent 100,000 signals from one machine to the other, and back again. By this method, the time from a signal's appearing on the parallel port to its registration in software can be established. The connecting cable was that used in the above test. Machine A (running `send_receive.c`) was set to send a signal and then wait for it to be returned before sending the next signal. Machine B (running `bounce.c`) was set to bounce the signal by waiting for it and then immediately sending it back again. Thus, Machine A would send a signal to Machine B (by calling `outb`) and record the time (using `gettimeofday`). Machine B would detect the signal by repeatedly sampling its status port:

```
while (((inb (BASEPORT + 1) >> 3) ^ 0x10) != sig).
```

The while loop causes the machine to *busy wait*. In general, this is bad programming practice, since this completely monopolizes the system. However, as long as the PC is not being used for another task, this is perfectly acceptable.⁵ When the signal `sig` is detected, it is immediately sent back to Machine A via Machine B's data port by calling `outb`. Machine A detects the signal by busy waiting and then immediately records the time. Machine A would then send a different signal to Machine B, beginning the next round trip. A different signal was used to prevent the return signal from Machine B on the previous round trip from being mistaken for the return signal for the current round trip. The times were stored in memory and dumped to a text file when the program finished. The latency of a round trip was calculated by subtracting the first recorded time from the second recorded time.

Before presenting the results of this test, a few comments about the Linux operating system are in order. The Linux

Table 2
The Port Addresses

Bit	Data Port (BASEPORT+0)	Status Port (BASEPORT+1)	Control Port (BASEPORT+2)
0	Data 0	Reserved	Strobe*
1	Data 1	Reserved	Auto feed*
2	Data 2	IRQ status	Initialize printer*
3	Data 3	Error	Select input
4	Data 4	Select	IRQ enable
5	Data 5	Paper empty	Direction
6	Data 6	ACK	Reserved
7	Data 7	Busy*	Reserved

*Inverted logic (i.e., 0 = high and 1 = low).

operating system, like many modern systems, is a multiuser multitasking operating system. In order to ensure accurate timing, three things are necessary. First, there must be an accurate timer. Second, one must be able to prevent the program collecting the reaction times from being interrupted by other programs. Third, one must make sure that the program is not “swapped out” of main memory into swap memory on the hard disk. Finney (2001) has shown that all of these requirements can be met by Linux. The source code examples accompanying this article include a simple code that sets these circumstances.

The requirements of timing, interruption, and memory were also met by the old MS DOS operating system (Myors, 1999a). Although Myors (1999b) found that the MS Windows 3.1, 95, and NT4.0 operating systems do not meet these requirements, this conclusion may be premature. When the experimental program is promoted to the real-time priority class to prevent interruption, success in testing a millisecond accuracy timer in MS Windows 95, 98, and 2000 has been obtained by MacInnes and Taylor (2001). Millisecond accuracy in registering a response, using the parallel port in MS Windows 95 and 98, has been reported by Chambers and Brown (2003). Plant et al. (2003) measured the error in the response time reported by the E-Prime software running on MS Windows 98. They found that although there was a constant error of the order of tens of milliseconds, the variability in this error was as low as 2 msec for some serial and PS/2 mice. Their key finding was that the variability depended on the particular hardware used, rather than on the operating system. For a complete MS Windows tachistoscope solution requiring additional hardware, see McKinney, MacCormac, and Welsh-Bohmer (1999). Plant, Hammond, and Turner’s (2004) commercial apparatus for testing timing accuracy in experimental setups is controlled via the parallel port. The testing program runs under the real-time scheduling priority in MS Windows NT, 2000, or XP and provides submillisecond accuracy. The DMDX application (Forster & Forster, 2003) for Windows 95, 98, 98SE, ME, XP, and 2000 has also been reported to obtain close to millisecond accuracy for measuring response times. The source code for these commercial programs has not been made available by the authors.

The results of the present test were as follows. Using two dual-processor⁶ AMD 1.53 GHz machines with par-

allel ports built in to the Tyan Thunder K7 motherboards produced times (to the nearest microsecond) between 3 and 20 μ sec for a single round trip. Of the 100,000 signals sent, 32.43% of the round-trip latencies were 3 μ sec, and 67.17% of the latencies were 4 μ sec. The remaining 0.40% of the latencies were between 5 and 20 μ sec. Since each latency is the time to send a signal from one machine, receive it on the other, send a signal from the other, and then receive it back on the original machine again, the time from a buttonpress to software detection will be, at most, half of this (not including the physical characteristics of the buttons). That is, the latency from the close of the button contacts to software detection of the press where the time can be logged is about 2 μ sec. (The system calls `gettimeofday`, `inb`, and `outb` take only a few microseconds.) In conclusion, a Linux system and parallel port button box will allow response times to be measured with more than millisecond accuracy and almost with microsecond accuracy.

A Button Box for the Standard Parallel Port

An exceedingly simple button box can be constructed by taking advantage of the fact that open (i.e., unconnected) pins are read as logic 1 and grounded input pins are read as logic 0. Thus, if a push-to-make button is connected from an input pin to a ground pin, pressing the button changes the logic from 1 to 0. Up to five buttons may be connected using the five status pins and five ground pins (i.e., between Pins 10 and 18, 11 and 19, 12 and 20, 13 and 21, and 15 and 22).

Some switches can produce a “bounce” and open and close the connection many times when they are pressed. Thus, a single buttonpress may register many times. This problem can be overcome by using a different kind of switch (e.g., a mercury-tipped switch) or by introducing into the software a refractory period after the detection of a first contact, during which subsequent contacts are ignored.

A Button Box for the Bidirectional Parallel Port

There are two reasons why it might be necessary to use a bidirectional parallel port. First, more than five buttons might be required. On a bidirectional parallel port, if Bit 5 of the control port is set, the data port (Pins 2–9) can be used for input, instead of output, allowing the use of a further eight buttons. Second, some standard parallel ports do not follow the convention that unconnected input pins should be read as logic 1. In this case, it is necessary to somehow set the input pin high when the switch is open, since the pin will not default to this state. This can be done by using one (or more) of the data port pins for input as follows: (1) unset Bit 5 of the control port to set the data port to output mode, (2) set the required data port pins to high, (3) set Bit 5 of the control port to set the data port pins to input mode. When a data port pin is grounded by closing a switch, its logic will change to 0 from 1. There is an advantage to having a parallel port that behaves in this way: switch “bounce” will not be registered, since the first close of the contacts will ground the pin and it will remain

grounded for subsequent opening and closing, until the pin is set high again in the software.

Conclusion

In conclusion, a PC Linux system with a parallel port button box is easily capable of measuring the time of a response with millisecond accuracy.

REFERENCES

- AXELSON, J. (1997). *Parallel port complete: Programming, interfacing and using the PC's parallel printer port*. Madison, WI: Lakeview Research.
- CHAMBERS, C. D., & BROWN, M. (2003). Timing accuracy under Microsoft Windows revealed through external chronometry. *Behavior Research Methods, Instruments, & Computers*, **35**, 96-108.
- FINNEY, S. A. (2001). Real-time data collection in Linux: A case study. *Behavior Research Methods, Instruments, & Computers*, **33**, 167-173.
- FORSTER, K. I., & FORSTER, J. C. (2003). DMDX: A Windows display program with millisecond accuracy. *Behavior Research Methods, Instruments, & Computers*, **35**, 116-124.
- MACINNES, W. J., & TAYLOR, T. L. (2001). Millisecond timing on PCs and Macs. *Behavior Research Methods, Instruments, & Computers*, **33**, 174-178.
- McKINNEY, C. J., MACCORMAC, E. R., & WELSH-BOHMER, K. A. (1999). Hardware and software for tachistoscopy: How to make accurate measurements on any PC utilizing the Microsoft Windows operating system. *Behavior Research Methods, Instruments, & Computers*, **31**, 129-136.
- MYORS, B. (1999a). The PC tachistoscope has 240 pages. *Behavior Research Methods, Instruments, & Computers*, **31**, 329-333.
- MYORS, B. (1999b). Timing accuracy of PC programs running under DOS and Windows. *Behavior Research Methods, Instruments, & Computers*, **31**, 322-328.
- PLANT, R. R., HAMMOND, N., & TURNER, G. (2004). Self-validating presentation and response timing in cognitive paradigms: How and why? *Behavior Research Methods, Instruments, & Computers*, **36**, 291-303.
- PLANT, R. R., HAMMOND, N., & WHITEHOUSE, T. (2003). How choice of mouse may affect response timing in psychological studies. *Behavior Research Methods, Instruments, & Computers*, **35**, 276-284.
- 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.
- SOROKIN, A. V. (2002). Instrument-to-PC interfacing using an enhanced parallel port. *Instruments & Experimental Techniques*, **45**, 516-520.
- STEWART, N. (2006). Millisecond accuracy video display using OpenGL under Linux. *Behavior Research Methods*, **38**, 142-145.
- ULRICH, R., & GIRAY, M. (1989). Time resolution of clocks: Effects on reaction time measurement—good news for bad clocks. *British Journal of Mathematical & Statistical Psychology*, **42**, 1-12.

NOTES

1. It is often not necessary to measure response times with millisecond accuracy (Ulrich & Giray, 1989).
2. An ECP port can be made to implement a standard parallel port by setting bits 5, 6, and 7 of BASEPORT + 0x402h to 0, 0, and 0, respectively. If the bits are set to 1, 0, and 0, a bidirectional port will be emulated. Alternatively, the emulation can often be set at boot time in the BIOS setup. See Axelson's (1997) parallel port Web pages (www.lvr.com/parport.htm) for more details.
3. All of the source code is available from the author's home page (www.warwick.ac.uk/staff/Neil.Stewart/), together with a simple library of example control functions.
4. Axelson's (1997) parallel port Web pages (www.lvr.com/parport.htm) describe how to access the parallel port under MS Windows operating systems for a variety of languages.
5. This busy wait method is used here to provide a proof of concept. In a real application, it is, of course, possible to complete other tasks in this loop, and even to give control back to the kernel for OS tasks to be performed. Providing that the time of each parallel port poll is recorded, the time of the first occasion on which a buttonpress was detected can be compared with the time of the last occasion on which a buttonpress was not detected, to provide, with certainty, the interval of time during which the button must have been pressed. I have found, in typical applications, that the maximum error is normally only a few microseconds.
6. Single-processor machines are perfectly adequate though: The testing programs are not multithreaded.

(Manuscript received July 16, 2003;
revision accepted for publication February 8, 2005.)