Constructive interaction between computer science and physics programs

Kyle Forinash and Ray Wisman
Natural Science Division, Indiana University Southeast, New Albany, Indiana 47150

(Received 9 August 1988; accepted 7 November 1988)

This paper reports on the culmination of a year-long collaborative effort between physics and computer science programs at a small branch campus. The project involved students in a computer science course who wrote generalized software capable of timing and voltage measurements in interfaced physics experiments. The software and experiments were used the following semester in a junior level physics laboratory course.

INTRODUCTION

On most large campuses, computer science courses dealing with interfacing and physics courses using interfaced experiments are taught independently of each other. On smaller campuses, since resources are scarce, classes from several disciplines often share the same equipment. For several years our physics students borrowed Apple IIe computers from the computer science program. Recently, IBM-AT's equipped with hard disks and Metra-Byte model UC-DASH-16 analog to digital interfacing boards have become available for use by physics and computer science. Because we were sharing this equipment, we began to think that to make the best use of these computers we might somehow combine the efforts of the computer science and physics programs. Computer interfacing is of use to both physicists and computer scientists and therefore we felt we shared some common ground.

Because of the development of inexpensive methods for taking data using computers, it has become the practice to introduce students to techniques of computer interfacing even in lower level undergraduate physics laboratory courses.[1] Since the goal in this kind of course is usually not to teach interfacing techniques, students are provided with easy-to-use software that is specific to the experiment, yet as flexible as is reasonably possible. By contrast, the primary goal of a computer science course in interfacing is to expose students to the details of both the software and hardware components of computer interfacing, generally without too much concern for realistic applications. Through careful planning we were able to combine these goals into a joint project involving a computer science course and a physics course taught in a subsequent semester. We feel the methods, equipment, and results of the project are certainly of interest to others facing the same problems and are especially applicable to small campuses with limited resources.

I. THE COURSES

The students who enrolled in the new computer science course had a fairly homogeneous background of computer science, having had computer structures (assembly language) and organization (digital design) courses in addition to the standard introductory courses. They came into the course with a good understanding of the basic hardware involved in a minimum mode 8088/8086 processor system such as CPU, memory, bus control, and simple I/O devices. Their concepts of the software methods for polling, interrupts, and direct memory access were rudimentary in comparison to their structural and organizational skills since computer science courses seldom provide experience in real time applications.

This new computer science course consisted of classroom lectures with regular in-class tests. In addition, the students worked in groups of two on six programming exercises and then did a final oral report.

The physics course offered in collaboration with the computer science course was a standard physics laboratory course at the junior level,taught in conjunction with a modern physics lecture course. We introduced laboratory techniques by having students repeat some of the classic experiments in modern physics. The exact choice of experiments has ranged from year to year, depending on availability of equipment and inclination of the instructor. Possible choices of experiments that can be interfaced to computers include the following: nuclear decay, photoelectric effect, Frank–Hertz experiment, optical diffraction, Faraday's law, and the Hall effect. Many other suggestions for interfaced experiments appropriate to a laboratory class of this kind can be found in the literature.[2–5]

The physics students worked in groups of two and rotated among the various experiments so that only one setup of each apparatus was needed. Apparatus was not setup in advance; instead, we gave the students parts lists, wiring diagrams, and minimal instructions or warnings and set them to work. It was the students' responsibility to go to the library and uncover the theory on their own. The students kept laboratory notebooks in the same fashion as industrial research notebooks.

II. THE COMPUTER ASSIGNMENTS AND THEIR RELEVANCY TO PHYSICS EXPERIMENTS

A. Timing

Timing is one of the most fundamental measurements taken in physics laboratory courses. We gave the computer science students three computer assignments (completed in Turbo Pascal) in order to lead up to the concepts they

needed to write useful timing software. The first programming assignment was a planned failure, one that would show the students a need for more sophisticated techniques. The students wrote a simple dumb-terminal emulator program to alternately poll the keyboard for data to send and to poll the serial interface for received characters. The design was purposely too slow to poll the keyboard and serial port while displaying data to the screen without losing characters now and then. The failure of this first implementation of a dumb-terminal routine to keep up with all program operations caused students to rethink their belief in an infinitely fast computer. The loss of characters became more pronounced at higher baud rates, thus providing an impetus for class discussion concerning more efficient methods, such as interrupts.

Once the stage was set for an improved dumb-terminal routine, the discussion of interrupts went very smoothly. A second assignment required the use of a common queue to buffer characters as they were received, placed in, and removed from the queue by an interrupt-driven process. Ordering projects in this way, we were able to introduce the concept of mutually exclusive access to data common to several asynchronous processes. Again, students could readily see the consequences when mutual exclusion was unenforced, leading to characters not being counted in the queue.

The third assignment was cast in terms of measuring the acceleration of a free falling object. The input status lines of the computer system printer port were specified to serve as signal inputs from two photogates incorporated with a voltage comparator.[6] These photogates can be monitored very easily by a simple polling routine. The students employed interrupts in programming the time-of-day counter to access a service routine which counted the number of times the interrupts occurred. In this way the students could determine the duration an object is within each gate and the time taken between gates.

We found several benefits in these assignments. The students were forced to recognize physical parameters, the limitations of computer hardware, and the corresponding concerns of efficiency in the development of real-time software. The simplicity of the physical interface, using photogates and the printer port, allowed students to comprehend easily the interfacing operations. Perhaps the largest benefit was the students' enthusiasm when they saw the immediate results of their hardware and software efforts. A healthy competition developed as to who could determine the timing data with the greatest degree of accuracy.

B. Voltage

Along with timing, a commonly measured quantity in the physics lab is voltage. The most efficient means for rendering voltages into a form acceptable to a computer is the analog to digital conversion board. The fourth programming exercise explored the use and control of a standard A/D interfacing device. A commercial interfacing board was used to provide direct memory access (DMA) and other interfacing resources. The assignment involved determining the frequency of an audio input expressed as a voltage signal to the A/D board. The students could find the frequency in a number of ways, one of the most obvious being to count the number of peaks in the signal over some fixed time of observation. Since no filtering devices were applied to the audio signal before conversion to digital signal, software "filtering" was required before peak counting could be performed.

To build upon past exercises and extend into new methods, we used a combination of polled and interrupt methods in conjunction with DMA. Since DMA occurs as a background process to the CPU operation, the students had to take into account completion synchronization between the DMA device and the CPU. This synchronization is most easily provided by the CPU initiating the DMA process and entering a busy-wait loop that polls a variable. That variable becomes true when the CPU services an interrupt generated by completion of the DMA. The audio signal voltage for a set number of samples or a set sampling time is thus sent directly to memory.

The students tested the interface using tuning forks and a calibrated frequency generator. The upper limit of frequency that can be measured is controlled by the rate at which the A/D converter can be sampled by a polling routine. The students viewed the audio signal on an oscilloscope while the data were sampled by the computer so that they could observe the background noise and identify the need for smoothing data. This exercise was useful for introducing the use of the interfacing board, A/D conversions, analog data manipulation methods, and application of several software techniques simultaneously (polling, interrupt, and DMA).

The fifth exercise was intended to provide software design practice of several real time concurrent processes using a method termed triple-buffering. The processes the students used were interrupt driven, polled, and DMA controlled. These processes performed data output, processing, and input, respectively, to one of three memory buffers. All buffers were accessed by all processes but in a controlled sequence while maintaining exclusive access by each process to a buffer.

To help the students accomplish the objective of the exercise, we required a program that would take audio data via a microphone from an A/D converter, process it in some fashion such as signal averaging, then output the processed data to a D/A converter and a speaker. Our hope was that students could have some fun with speech being digitally manipulated to produce echoes, or other phenomena. Because of the crudeness of the setup, however, only simple sounds, such as tuning forks, could be discerned by a listener.

C. Packaging

The final assignment had as a goal the merging of the concepts learned previously into a user-friendly software package. By this point the computer science students could readily see what was required by a physics laboratory in the way of data handling capabilities. We suggested that the software would be the most flexible and easiest to use if the data were presented in a spreadsheet.[7] In this way the data, once collected, could be manipulated by the user as needed. Students worked in small groups on various portions of the project and met several times all together to coordinate the finished product. As part of their final evaluation, each student made a presentation concerning his role in the implementation.

The software that resulted was imminently practical, most likely because the computer science students who wrote and tested it were aware of the limitations inherent in the hardware and software techniques which they had used and because they had a specific goal before them.

The real importance of an understanding of interfacing techniques at the most sophisticated level was readily apparent to these students.

In the following semester, physics students used the software in two out of seven experiments which were required for the course: the photoelectric effect[3] and a demonstration of Faraday's law.[2] In the Faraday's law experiment where 1000 data points are taken at 200 μs intervals, students could clearly understand the power of using interfacing tools. The consensus among physics students was that interfacing provides a way to measure quantities that would have been difficult or impossible to measure before the advent of computers.

III. WHAT WE LEARNED FROM THE PROJECT

Our giving students specific interfacing projects taught us a new respect for each other's disciplines and also gave us new ideas about teaching interfacing techniques to both computer science students and physics students. We found that computer science students are capable of learning very sophisticated techniques if they could see a purpose or use for that technique. The students who were most successful in the computer course were the ones who had taken or were taking introductory physics courses. In these cases, the student could more easily perceive how the constraints implied by the physical data dictated choices of programming strategies.

We also found that the end result for the physics course was much more useful than we had initially imagined. It is not too difficult for a student or instructor to write a simple program in BASIC with one or two specific goals or capabilities. The software our students produced was far more powerful: a data to spreadsheet software package capable of any timing or voltage measurement. The flexibility of the software the students created is such that the same package can be used for many different experiments by customizing the spreadsheet to the experiment by building in as much or as little help in the data analysis as is appropriate to the level of student using the program.

Finally, we found that the finished product can be improved each time we offer the computer science course. Presently the software only monitors one A/D input line and has only rudimentary graphing capabilities. A future group of computer science students could easily rectify these and other problems. It would also be possible to involve advanced physics students in the planning stages of incorporating the software into new experiments, thus expanding the number of physics experiments that are interfaced. So far we have not had a student take both courses, but we hope to encourage this in the future.

We feel the project has been extremely successful. The computer science students appreciated having tangible goals linked to real programming problems with visible results. Physics students who used the software were exposed to introductory interfacing techniques and saw the value of using and learning about interfacing capabilities. As instructors, we learned a great deal about each other's fields and resolved some of the communications problems that inevitably arise between separate disciplines. It is doubtful that either of us working separately would have been nearly as successful in accomplishing our respective goals. We recommend this type of joint effort to anyone interested in teaching computer interfacing, either at an advanced or introductory level in physics or computer science courses.

ACKNOWLEDGMENTS

This project was completed as part of a pilot study for a National Science Foundation Instrumentation and Laboratory Improvement Program grant proposal. We would like to acknowledge the NSF for the awarding of Grant No. USE-8851189.

REFERENCES

1. R. G. Fuller, Am. J. Phys. **54**, 782 (1986).
2. R. C. Nicklin, Am. J. Phys. **54**, 422 (1986).
3. R. L. Bobst and E. A. Karlow, Am. J. Phys. **53**, 911 (1985).
4. J. Priest and J. Snider, Phys. Teach. **25**, 303 (1987).
5. D. L. Vernier, *How to Build a Better Mousetrap* (Vernier Software, Portland, OR, 1986).
6. R. Wisman and K. Forinash, to appear in Am. J. Phys.