Teaching Robotics and Computer Science with Pinball Machines*

Daniel Wong Darren Earl Fred Zyda Sven Koenig
Computer Science Department
University of Southern California
Los Angeles, CA 90089-0781, USA
wongdani@usc.edu earl.darren@gmail.com fzyda@usc.edu skoenig@usc.edu

Abstract

Roboticists need to have a solid understanding of hardware and software. The standard computer science education in the United States, however, tends to teach students only about software. To remedy this situation, we explore new ways of teaching them about hardware in a playful way. Realizing that pinball machines are simple robots, we have developed a pinball machine interface between a PC and a recent Lord of the Rings pinball machine, which enables students to implement pinball games and gain knowledge of hardware and interface programming in the process. This paper describes both our pinball machine interface and our experience developing it. As far as we know, this is the first time that anyone has managed to control an existing pinball machine completely.

Figure 1: LOTR Pinball Machine

Introduction

Roboticists need to have a solid understanding of both basic robot hardware components and how to interface to them, for example to be able to write efficient software that runs on the hardware or controls it. The standard computer science education in the United States, however, tends to teach students only about high-level software. To remedy this situation, we want to teach computer science students the hardware basics of robotics, including signal generation, embedded systems, communication protocols, interface programming and real-time programming. We decided to explore a new way of teaching the students these topics in a playful way without overwhelming them. We build on our insight that solid-state pinball machines interface to the physical world and basically consist of a computer that reads the physical and optical switches and controls the lights, motors, solenoids, speakers and the dot-matrix display (Rossignoli 1999). Pinball machines are thus simple robots that contain actuators (in the form of motors and solenoids), sensors (in the form switches) and visual outputs (in the form of lights and the dot-matrix display). Their game is determined by the input-output behavior of the computer, that is, what outputs the computer activates and when it activates them in response to its input-output history. We can therefore use pinball machines to introduce students to basic robot hardware components. Pinball machines have two advantages over traditional robots: First, pinball machines are cheaper than traditional robots and much easier to maintain. Their low level control is simple and the resulting behavior thus robust, resulting in a motivational experience. Second, students are often highly motivated to learn how to implement computer games (Cliburn 2007). We do not expect pinball machines to be an exception to that rule since pinball machines are fun to play. Students are thus motivated to learn more about how they work and how to program them both because they are curious and because they can play their pinball games themselves.

We imagine a pinball class to be a semester-long capstone class for students that have already taken some traditional computer science or engineering classes and now have to use a variety of knowledge and skills that they have acquired in these classes to solve a larger problem, namely to develop a way of programming a pinball machine easily and to create a pinball game that is playable, engaging and entertaining. The students need to let lights blink, that is, switch them on and off at the correct times. They need to form complex light patterns of light with the many available lights. They need to detect patterns of switch closures and determine the

*Part of this material is based upon work supported by NSF under grant number 0413196 and by the Fund for Innovative Undergraduate Teaching at the University of Southern California. Daniel Wong was supported by scholarships from the USC Undergraduate Merit Research Program and the Rose Hill Foundation. We thank the Orange County Pinball League for their hospitality. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors.

mode based on them. They need to synchronize the lights, sounds, speech, music and information displayed on the dot-matrix display, which is important because the various output modalities take different amounts of time and compete for the available output resources. For example, a sound effect and an animation on the dot-matrix display take different amounts of time and are long enough that the situation might change, which might require a different sound effect and a different animation. On one hand, it is not satisfactory to stop the old sound effect and animation before they are finished. On the other hand, it is not satisfactory either to delay the new sound effect and animation long beyond the event that prompted them. The students need to experiment with different ways of implementing the necessary hierarchical control structure to achieve real-time control, assign scarce resources, manage the complexity of the software and make it easily adaptable to changing the specifications of the pinball game (including how to best represent the rules of the pinball game), as will frequently arise during game development. These tasks require understanding of concepts from operating systems, concurrent processes and rule-based expert systems (such as black-board systems) as well as software engineering. Furthermore, many computer science or engineering majors might not have practical experience with playing sounds, speech and music. The students also need to have a good understanding of what makes pinball games engaging. For example, pinball players look for variability in game play via different modes but also the opportunity to make multiple shots and work on multiple objectives, switching among them as opportunities present themselves. Thus, the rules of pinball games need to get designed carefully, including the shots, modes and scoring. The user interface also needs to get designed carefully, including the lights, sounds, speech, music and information displayed on the dot-matrix display, both to communicate the mode of the pinball game to the user (which requires user-interface skills that some students will be familiar with) and set an appropriate mood for the current mode (which requires artistic skills typically not taught to computer science or engineering students).

A cornerstone of a pinball class is a pinball machine interface. Our pinball machine interface consists of a hardware interface, a software interface to drive the hardware interface and provide low and high level real-time control of a recent Lord of the Rings pinball machine, libraries that provide abstractions of this software interface and a program that uses these libraries to implement a simple pinball game. A standard desktop PC controls the pinball machine via a Digital I/O Board that gives the PC a number of general purpose I/O ports. The PC thus controls the pinball machine similar to a microcontroller and prepares the students to control robots or parts of them via microcontrollers. PCs are good gateways towards the use of microcontrollers since computer science or engineering students are already familiar with PCs and C/C++ programming and debugging is simple. In both cases, programs need to run in real-time. A serial interface is used to send images from the pinball machine interface to the dot matrix display and prepares students to control global positioning systems, inertial measurement units and compasses on robots via serial interfaces. Pulse-width modulation is used to drive the solenoids and motors of the pinball machine and prepares students to drive servo motors on robots. We developed the pinball machine interface in approximately 200 hours in Summer 2008 and 75 hours in Fall 2008, split roughly half and half between the hardware and software interface. This paper describes both our pinball machine interface and our experience developing it. As far as we know, this is the first time that anyone has managed to control an existing pinball machine completely. The pinball machine interface can be adapted by other universities and colleges for the cost of a PC, a small amount of hardware and a used pinball machine.

Figure 2: Playfield and Underside

Pinball Machine

We decided to work with a used solid-state Lord of the Rings (LOTR) pinball machine, see Figure 1, from Stern Pinball of Chicago, currently the only manufacturer of pinball machines. Stern Pinball produced about 5,100 of these pinball machines, starting in 2003. Used LOTR pinball machines cost about \$3000-\$3500. The layout of their playfield is flexible and not particularly theme-specific, see Figure 2 (left). LOTR pinball machines are highly rated for playability and need to be used in creative ways to create engaging pinball games since we do not want to alter the playfield physically. We control the pinball machine via its I/O Power Driver Board, which was used by all Sega and Stern pinball machines with a WhiteStar or WhiteStar II board system (roughly from 1995 to 2004). Thus, we expect the pinball machine interface to be usable with a variety of pinball machines from Stern. The LOTR pinball machine consists of the following input and output devices, see Figure 2 (right):

- The **input devices** of the LOTR pinball machine consist of the 58 playfield switches and the 7 dedicated switches, which correspond to switches that humans interact with (such as the left and right flipper buttons). The LOTR pinball machine supports up to 64 playfield switches, arranged in an 8x8 matrix. The controller reads the playfield switches by software polling. It strobes each column and then reads the row signal after it has gone through RC filters (for noise filtering) and a comparator (for signal buffering) to create a stronger steady signal. The LOTR pinball machine supports up to 8 dedicated switches. The controller reads the dedicated switches directly.

- The **output devices** of a LOTR pinball machine consist of the lights (namely 80 lamps, 9 flash lamps and 19 LEDs) and 23 low and high current solenoids (including 2 slingshots, 3 vertical upkickers, 3 pop bumpers, 2 flippers, 1 loop diverter, 1 ring magnet and 1 Balrog motor). They also consist of a pair of speakers and the dot-matrix display. The lamps are arranged in a 10x8 matrix and need to be strobed, similar to the switches, at approximately 1 ms to minimize flickering.

The LOTR pinball machine is controlled by the following components, see Figure 3:

- The **CPU/Sound Board** processes the input signals and generates the control signals for the speakers, the Display Controller Board and the I/O Power Driver Board. It includes the 8-bit 68B09E microprocessor, the CPU Game ROM and the Sound/Voice ROM.
- The **I/O Power Driver Board** drives the output devices. It includes registers that hold the status of each output device. The CPU/Sound Board sets the data of these registers. The I/O Power Driver Board then activates the output devices accordingly.
- The **Display Controller Board** controls the 128x32 plasma dot-matrix display (DMD). It includes the Image ROM. The CPU/Sound Board determines which image from the Image ROM to display. The Display Controller Board then retrieves the image and generates it on the DMD.

Hardware Interface

One extreme of programming the LOTR pinball machine is to reprogram its ROMs. However, a reverse engineering effort is very difficult without any available documentation for the proprietary boards other than the LOTR manual. The other extreme of programming the LOTR pinball machine is to replace all boards. However, this is costly and time-consuming since there are more than 100 input and output devices. We therefore decided to replace the CPU/Sound Board with a PC. We decided to interface the PC to the pinball machine via a small number of existing connectors rather than soldering wires onto the existing boards, which would make it difficult to both make additional pinball machines programmable and transform them back into their original state (which means that they would lose retail value). We also decided to interface the PC to our own Display Controller Board because we wanted to be able to generate images on the DMD on the fly rather than having to store them in the Image ROM. Finally, we decided to use the PC speakers for sounds, speech and music. Thus, the pinball game can play arbitrary sound files with the high-level commands provided by SDL. The hardware interface consists of the following components, see Figures 4 and 5:

- The **PC** is used for programming and running the game code. We use a Dell Outlet Inspiron 530 PC with an Intel Core 2 Quad Q6600 Kentsfield 2.4GHz processor and the default installation of OpenSuse 10.3 Linux, except that we recompiled the kernel with support for a high-resolution timer and installed libraries for the Digital I/O Board (NIDAQmx driver) and serial communication (libserial). The PC costs $469, and Linux costs $0.

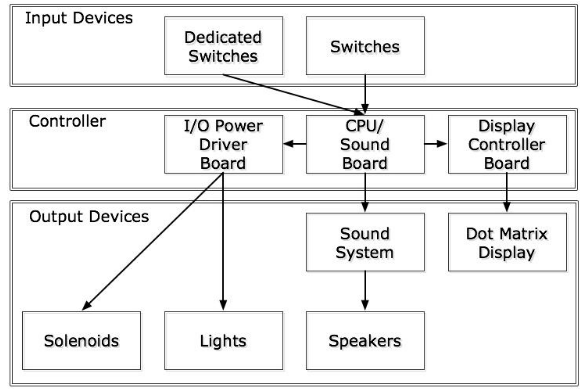

Figure 3: Original Hardware Architecture

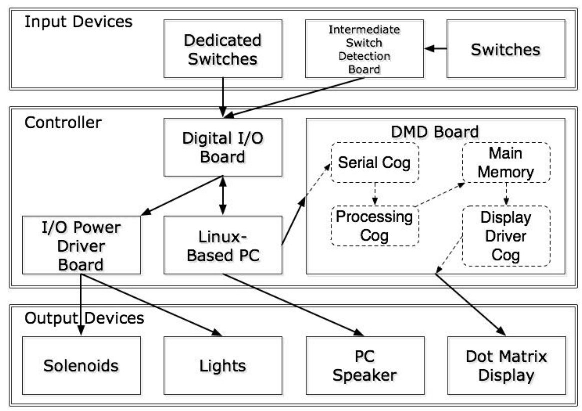

Figure 4: New Hardware Architecture

- The **DMD Board** interfaces the PC to the DMD. We use a Parallax Propeller, an 8-core 32-bit microprocessor. Each of its cores (called cogs) is capable of running independently of the others. We use one cog to drive the DMD at about 70Hz, the recommended refresh rate. We use one cog to handle the serial connection with the PC at about 115.2k Baud. Finally, we use one cog to process the data received from the PC. When data is received from the PC, the Serial Cog hands it over to the Processing Cog, which writes the data to memory, forming an image. The Display Driver Cog then reads the image and drives the DMD to display it. The images are double buffered to create a steady image on the DMD. The Parallax Propeller Demo board costs about $80.
- The **Intermediate Switch Detection Board** filters and buffers the incoming switch signals similar to what the CPU/Sound Board did before. The parts of this custom-made board cost about $30.

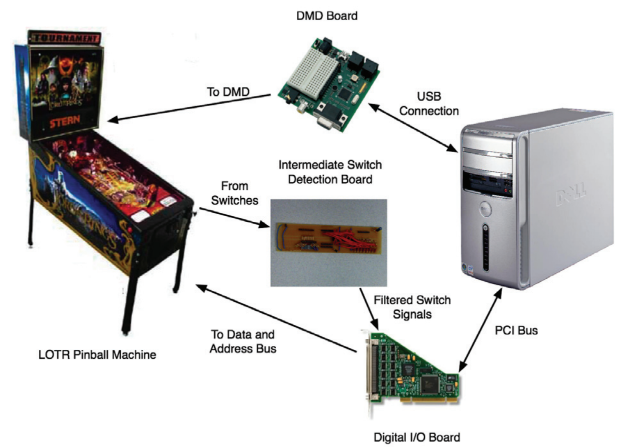

Figure 5: Hardware Interface

- The **Digital I/O Board** interfaces the PC via the PCI bus to the I/O Power Driver Board and the Intermediate Switch Detection Board. We use a National Instruments PCI-6509, a high-current 96-channel 5V TTL/CMOS Digital I/O card. We chose the PCI-6509 for its ample ports, its support for 5V TTL/CMOS levels which makes it compatible with components on the I/O Power Driver Board, and its ability to source or sink up to 24mA of current which enables it to drive the data bus of the I/O Power Driver Board. Its 96 channels are organized into 12 ports of 8 channels each. To read the playfield switches from the Intermediate Switch Detection Board, we use one port as output to strobe the column and one port as input to read the row signal. To read the dedicated switches from the Intermediate Switch Detection Board, we use one port as input. To set the data of the registers of the I/O Power Driver Board, we use two ports as outputs, one for the address of a register and one for the data to be written into that register. The PCI-6509 costs $299, and its cabling kit costs $259.

The total cost of all additions is approximately $708 plus the cost of the PC. We expect to be able to reduce this cost to approximately $300 plus the cost of the PC with an embedded system currently in development.

Software Interface

The software interface allows a pinball game to control all aspects of the pinball machine via function calls. It is written in C/C++ and provides the interface between the pinball game and the Digital I/O Board via the NIDAQmx driver and the pinball game and the DMD Board via serial communication (a USB connector as a virtual COM port). It allows students to implement pinball games easily by providing an almost complete abstraction of the hardware. The standard Application Programming Interface (API) of SDL allows for the creation of graphics and audio, and a custom API provides all remaining functionality. The software interface consists of the following components, see Figure 6:

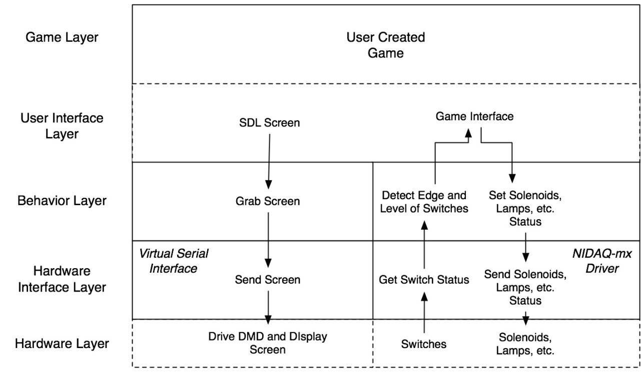

Figure 6: Software Architecture

- The **Display Controller Interface** interfaces the pinball game to the DMD Board and can display approximately 15 frames per second on the DMD, which is sufficient for full motion animation. It consists of two layers:

  The **Hardware Interface Layer** communicates with the DMD using serial communication.

  The **Behavior Layer** repeatedly grabs the content of the screen and uses the Hardware Interface Layer to transfer it to the DMD Board. Thus, the pinball game can create arbitrary graphics on the fly with the high-level commands provided by SDL.

- The **Digital I/O Board Interface** interfaces the pinball game to the Digital I/O Board. It consists of two layers:

  The **Hardware Interface Layer** communicates with the Digital I/O Board via the NIDAQmx driver to read the switches and drive the lights and solenoids. For example, it converts the desired status of the lights and solenoids into byte representation, places the address of the register onto the address bus, the data of the register onto the data bus and finally clocks the register to save the data.

  The **Behavior Layer** uses the Hardware Interface Layer to implement behaviors for every output device. Pinball games need to contain a main loop that repeatedly calls the Main Loop Function of the Behavior Layer to read the current status of the switches and set the current status of the lights and solenoids. The Main Loop Function detects both levels and edges of switches by monitoring the history of the switch status. It activates the lights and solenoids according to behaviors that the pinball game can set via function calls. The Behavior Layer uses counters to generate the pulse-width modulated signals that drive the lights and solenoids.

The Behavior Layer of the Digital I/O Board Interface implements three different behaviors:

- The standard behavior for lights and solenoids is to turn on (= activate) immediately and then turn off (= deactivate) after a set period of time.
- A special behavior is needed for those solenoids that need to stay on for extended periods of time and thus are typically individually fused, namely the flippers, the ring

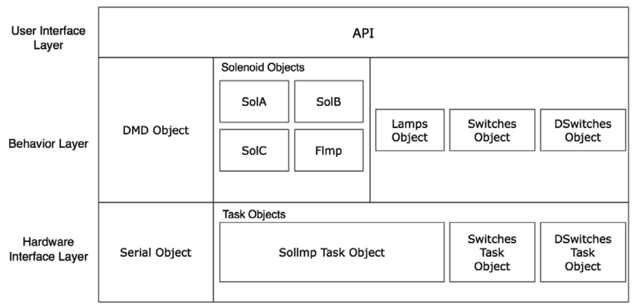

Figure 7: Implementation of the Software Interface

magnet, the loop diverter and the Balrog motor. Their behavior is identical to the standard behavior but they require pulse-width modulation to limit the amount of current going through them, which reduces the amount of heat and allows them to stay on longer. Thus, the behavior needs to activate and deactivate them repeated while they are turned on.

- Another special behavior is needed for the Balrog, which acts as a door that opens and closes. Its behavior needs to set the relay to the desired turning direction and then turn on the Balrog motor.

The Behavior Layer of the Digital I/O Board Interface also performs safety checks to prevent fuses and transistors from blowing. These safety checks are the result of trial and error. They implement conservative time-outs for all solenoids driven by pulse-width modulation. For example, the loop diverter automatically turns off after one minute, and the Balrog motor turns off immediately when Balrog is completely open or closed. The safety checks implement timeouts of half a second for the lights and all other solenoids since they need to be active only for fractions of a second.

Implementation of the Software Interface

The software interface reflects the organization of the I/O Power Driver Board, see Figure 7:

- In the **Hardware Interface Layer**, the board connections are represented by one task object each, which handles the communication with one or more I/O ports. There are three task objects, namely a switches task object, a dswitches task object (for dedicated switches) and a sollmp task object (for solenoids and lamps). The switches task object handles two ports to read the switch data, the dedicated switches task object handles one port to read the dedicated switch data, and the sollmp task object handles one port for the data bus and one port for the address bus. Finally, the serial communication with the DMD Board is represented by one serial object.
- In the **Behavior Layer**, all lamps are represented by one lamps object, that organizes the lamps in a two-dimensional array according to their matrix organization on the I/O Power Driver Board. Similarly, all switches are represented by one switches object, that organizes the switches in a two-dimensional array according to their matrix organization on the I/O Power Driver Board. All dedicated switches are represented by one dswitches object. All solenoids, motors and flash lamps are represented by one playfield object each. Each group of eight solenoids, motors and flash lamps on the I/O Power Driver Board corresponds to a group of eight playfield objects that is represented by a solenoid object. There are four solenoids objects, namely SolA, SolB, SolC and Flmp. Finally, the DMD is represented by one DMD object.
- In the **User Interface Layer**, the API interfaces to the objects in the Behavior Layer and abstracts the low-level functions in the Behavior Layer from the end user, allowing them to use the system without knowledge of the hardware or its implementation.

Status of the Pinball Machine Interface

We encountered the following issues during the development of the pinball machine interface:

- We had the LOTR manual available but not much further documentation. Thus, we had to develop parts of the hardware and software interface via trial and error, which was especially problematic for the I/O Power Driver Board, DMD and LEDs.
- We had to develop the safety checks via trial and error, blowing many fuses and some transistors in the process.
- The hardware and software interface were more complicated than necessary due to our design decision to interface the PC to the I/O Power Driver Board via a small number of existing connectors. For example, we needed to design the Intermediate Switch Detection Board to replicate part of the original CPU/Sound Board, and we needed to strobe the playfield switches and lamps. The playfield switches cannot be strobed too quickly (otherwise the data will not have time to propagate through the wires), and the lamps cannot be strobed too slowly (otherwise a watchdog timer on the I/O Power Driver Board resets the system).
- The PC needs to run fast both to strobe the lamps sufficiently fast (to prevent them from flickering and the watchdog timer from triggering) and to control the solenoids with precision.
- The software interface suffers from slight inconsistencies in timing due to our design decision not to use an operating system with a real-time kernel since the system load then determines how often the software interface gets called in the main loop, which determines the flickering of the lamps and the precision of control of the solenoids. We mitigated this effect by recompiling the Linux kernel with support for a high-resolution timer with an accuracy of 1 ns and gave the pinball game high priority so that other processes would interrupt it less.

The pinball machine interface is now fully functional, with only three minor issues:

- The software interface is currently able to display only two colors on the DMD but is already designed to use software modulation to produce three levels of grayscale.
- The software interface does not perform all necessary safety checks to guard against unreasonable game programmers. For example, the flow of current that results from activating all solenoids at once can blow fuses and potentially damage the I/O Power Board.
- The software interface still suffers from minor inconsistencies in timing: The lamps flicker subtly (although this is barely noticeable); the ring magnet is not always able to catch the pinball and throw it backwards; and the force applied by the vertical upkickers is not completely consistent.

The development of the pinball machine interface continues. The software interface is being restructured, and the hardware interface is undergoing a complete makeover to improve its performance. In particular, we are working on creating an embedded system to produce the control signals for the pinball machine, which will solve the inconsistencies in timing, lower the total cost by replacing the Digital I/O Board and reduce the number of cables required by handling all data transfer through serial communication.

Testing the Pinball Machine Interface

To test the pinball machine interface and demonstrate its possibilities, we offered a pilot class at the University of Southern California on Designing and Implementing Games on Pinball Machines in Fall 2008. The students in this class developed a simple pinball game called Pinhorse, see our 11-minute video at `http://idm-lab.org/pinball/`. Pinhorse is a proof of concept game that demonstrates what can be accomplished with the interface. It incorporates almost all of the playfield parts as well as the dot matrix display and sound but needs to be developed further to reach the sophistication of existing pinball games. The Pinhorse program consists of about 680 lines of code, the helper classes for shot recognition, shot matching and light patterns consist of about 1600 lines of code, and the software interface consists of about 2500 lines of code. Almost all of this code can be reused to speed up the development of future pinball games and to increase their complexity.

Related Work

There are only a small number of teaching and research efforts that use actual pinball machines. Actual pinball machines have, in research, been used to study hybrid system control (Lichtenberg and Neidig 2003) and develop and evaluate machine learning algorithms by an undergraduate student of Sven Koenig (one of the authors of this paper) at Georgia Institute of Technology (USA) in 1999. In teaching, they have been used to teach real-time and embedded systems as part of "Introduction to Embedded and Real-Time Programming" (CS160) at Brown University (USA) in Spring 2007, "Smart Product Design Laboratory" (ME218a) at Stanford University (USA) in 2007, "Designing with Microcontrollers" (EE476) at Cornell University (USA) in Spring 2007 and "Special Topics in Electrical Engineering: Pinball Machine Project" (ENEE 488Q) at the University of Maryland at College Park (USA) in Spring 1997. A similar effort existed at Brooklyn College (USA) around 1997 (Clark 1997). Pinball machines have also been used to teach signal and image processing as part of "Project Course in Signal Processing and Digital Communication" (EQ2430/EQ2440) at Kungliga Tekniska Högskolan (Sweden) in Spring 2004 and "Project: Pinball" (EOH 2004) of the ACM Special Interest Group for Computer Architecture at the University of Illinois at Urbana Champaign (USA) in Spring 2004. One of these efforts attempted to build a pinball machine from scratch, which allows them to customize the hardware for easier control. Other efforts used old electro-mechanical pinball machines, which are easier to control than newer solid-state pinball machines. The state of the art in controlling a pinball machine was as follows: Some efforts controlled the flippers by modifying the hardware (Clark 1994). Some tracked the ball via input from the playfield switches (Lichtenberg and Neidig 2003) or an overhead camera (Gustafsson et al. 2004) (Cohen 1989). The most sophisticated project so far mimicked the microcomputer-based control unit to read the switches and control the solenoids (Bork 2005b) (Bork 2005a). We, on the other hand, provide a hardware and software interface that controls all aspects of an existing solid-state pinball machine (including the lights, solenoids, DMD and speaker) without needing to modify its hardware. Furthermore, we have used the hardware and software interface to program a complete (but simple) pinball game that makes use of all of these aspects.

References

Bork, J. 2005a. Controlling a pinball machine using Linux. *Linux Journal* 139.

Bork, J. 2005b. Reverse engineering a microcomputer-based control unit. Master's thesis, Industrial Technology, Bowling Green State University, Bowling Green (Ohio).

Clark, D. 1994. An inexpensive realtime testbed - the pinball player project. In *Proceedings of the IEEE Workshop on Real-Time Applications*, 86–88.

Clark, D. 1997. Progress toward an inexpensive real-time testbed: The pinball player project. In *Proceedings of the Real-Time Educational: Second Workshop*, 72–79.

Cliburn, D. 2007. Games across the curriculum: Can we quantify their effectiveness? (Birds of a Feather Session). In *ACM Technical Symposium on Computer Science Education*.

Cohen, R. 1989. Designing an experimental pinball wizard. *The Electronic System Design Magazine* 19.

Gustafsson, S.; Munoz, J.; Norell, S.; Real, D.; and Xiao, Y. 2004. Smart pinball project - final report. Technical report, Skolan för Elektro- och Systemteknik, Royal Institute of Technology, Stockholm (Sweden).

Lichtenberg, G., and Neidig, J. 2003. An example of hybrid systems control: The pinball machine. Technical Report 2003.13, Lehrstuhl für Automatisierungstechnik and Prozessinformatik, Ruhr-Universität Bochum, Bochum (Germany).

Rossignoli, M. 1999. *The Complete Pinball Book*. Schiffer.