Games as Motivation in Computer Design Courses: I/O is the Key

Erik Brunvand
School of Computing
University of Utah
elb@cs.utah.edu

ABSTRACT

The design of computer games can be a powerful motivator as students learn about computer architecture and design. Students in classes where computer designs are developed and implemented (usually on Field Programmable Gate Arrays (FPGAs)) seem much more highly motivated if their computer design can be used for something visual and interactive when the project is complete. However, ensuring that the student teams can have a working game by the end of a semester requires careful planning of how their computer designs will interact with the world. Keyboard inputs and VGA outputs are a relatively simple set of I/O interfaces that open up significant new potential for development of game applications on the student's own computer designs. In this paper we describe the curriculum of a computer design course that uses game design as a "carrot" to encourage active student exploration and deeper understanding of computer architecture, I/O subsystems, and computer implementation.

Categories and Subject Descriptors:
K.3.2 [**Computer and Information Science Eduction**], C.5.3 [**Microcomputers**]

General Terms: Design

Keywords: Games, Computer Design, FPGA

1. INTRODUCTION

Good motivation for learning is a powerful part of any successful curriculum. Especially in project courses where students design and build an artifact, the time spent working on the project, and the depth of knowledge that comes out of the project, are dramatically increased if there is some strong internal motivation to go beyond the basics and delve into the project details [3, 7]. In particular we focus on a course where students learn about computer design by designing and building a computer from scratch on a Field Programmable Gate Array (FPGA). We believe, as do others [8], that the best way to learn about computer design is to design a computer. For many students the opportunity to develop interactive games on a computer they designed is a strong motivator. Without some form of graphical I/O, however, computer design projects are often limited to an interface where programs are loaded into memory, programs are run, and memory is uploaded to see if the correct answer is in the correct memory location. While computing Fibonacci numbers into a correct memory location may be interesting to some students, interactive games with colorful graphics are a much more exciting demonstration application. Exotic I/O devices are not necessary, but something beyond switches and seven segment LEDs is definitely required. Relatively simple interfaces such as PS/2 keyboard input and VGA output are standard I/O systems that dramatically increase the potential for students to design visually interesting games. However, success in using these I/O systems with student-designed processors requires that the details of these systems and the infrastructure to support them be carefully planned.

Designing a computer processor on an FPGA in a computer design course is certainly not novel. A number of papers have described interesting variations on this curriculum (e. g. [9, 4, 1, 11, 12]). Likewise, using games as motivators in CS software classes is also a well-studied area (e.g. [14, 15, 2, 10, 16]). What we argue in this paper is that games can also be a strong motivator in a hardware-centric computer design course, but that to facilitate games the curriculum should include specific information about I/O interfaces that support game development.

A focus on interactive games as a driving application for computer design courses has a significant positive side effect of forcing the students to pay careful attention to details of the I/O system that are too frequently overlooked in system design projects. Designing a computer that can use VGA outputs to display an interactive game requires that students understand not only the VGA controller itself, but also how game information relates to screen (frame buffer) locations, how to organize the frame buffer into a reasonable set of memory requirements, how to use pre-designed glyphs which may be stored in a separate memory, how time, clock cycles, and memory are shared between the processor and the VGA controller, and other issues that contribute to a much deeper understanding of the computer system as a whole.

It's also interesting that the increased complexity of the computer design projects to support games is largely student-driven. That is, the goal of the class has not changed (build a computer and demonstrate it), but there has been a significant jump in complexity of the projects when games become the preferred demonstration application. The teams that target games also tend to increase their own software support more than other teams by developing assemblers and other support systems for their own code development. The results are reasonably complex and realistic 16-bit computer systems demonstrating interactive games though standard keyboard and VGA interfaces. These are all the more impressive as they are the result of only the second course in digital design in our curriculum.

In this paper we discuss the curriculum of the junior-level Computer Design Lab course at the University of Utah (CS/ECE 3710) that uses game design as a motivator for computer design and implementation projects. We also highlight the projects from three student teams. The games that have been implemented on these student-designed 16-bit computers will not put Playstation or Xbox out of business any time soon, but they are visually interesting, run on completely student-designed hardware, and motivate the students to have a much richer experience in their computer design course.

2. CURRICULUM

The course described in this paper is an undergraduate semester-long project-based computer design course. It is typically taken in the Fall semester of a student's junior year. It is essentially a second semester of digital system design, but with a computer design as the focus. The prerequisites are:

Digital System Design I: This course is a semester-long lab-based course that covers the fundamental concepts of digital system theory and design. The course covers digital logic, Boolean algebra, binary number representation, design of combinational and sequential circuits, arithmetic circuits, finite state machines, the Verilog hardware description language, and includes labs where students build, test, and demonstrate circuits on an FPGA board.

Computer Organization: This course is a semester-long course in computer architecture where students examine the design of modern computers with an emphasis on understanding computer performance. Topics include RISC instruction sets, number systems, CPU design, pipelining, memory and caches, and multicore issues. Labs in this course currently involve assembly language programming and the MARS simulator [18].

Based on these prerequisites we can assume that the students have an understanding of computer architecture, specifically as it relates to executing an instruction set architecture. They also have a working knowledge of a hardware description language such as Verilog. Students who have their earlier class in VHDL find that although it seems daunting, it takes only a short time to make the switch to a different HDL. The main activity in the computer design class described in this paper is to take the architecture knowledge and the circuit knowledge from previous classes, and combine them together in a semester-long project to design and implement (on an FPGA) a fully-functioning computer processor. Computer Design Lab is an elective course for Computer Science and for Electrical Engineering majors, and a required course for Computer Engineering majors.

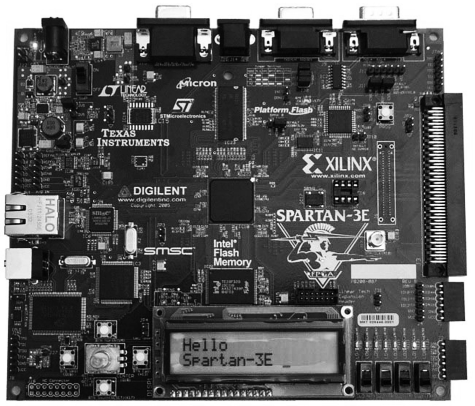

Figure 1: Spartan3e FPGA Prototyping Board

The course proceeds with lectures and individual assignments for the first five weeks of the semester. Teams of 3-4 students are then formed to work on designing and building their project computer for the rest of the semester. The class culminates with an open-house where the students demonstrate their processors and applications.

Currently we use FPGA boards from Xilinx [20] that use a Xilinx Spartan3 FPGA, and a variety of ancillary circuits including SDRAM, flash memory, LEDs, a variety of switches, PS/2 and VGA connectors, and a 2-line 16-character LCD display. The board is shown in Figure 1. It is important to note that the I/O connectors on the board are just that, connectors. Any circuits that interface to those connectors must be designed by the students and mapped onto the FPGA. We use the free ISEWebpack tools, also from Xilinx, for system design, simulation, and mapping to the FPGA board.

The main lecture topics in the first part of the course are:

Digital design review	Finite state machines Arithmetic/logic circuits RTL descriptions
Verilog language review	FSM examples Arithmetic code examples
Processor design overview	Simple MIPS architecture Class baseline ISA and memory map
I/O systems	Memory mapped I/O PS/2 and VGA Serial communications: RS232, SPI
Programming support	Assembler basics

This curriculum is a fairly standard lead-in to a computer design project, but includes a focus on I/O that is specifically intended to lead to a games-capable computer design. The labs done by individuals prior to forming teams are:

Lab1 - Finite State Machine: In this lab the students design a simple finite state machine that takes inputs from the buttons on the FPGA board and produces output on the LEDs on the FPGA board. This lab refreshes the students' understanding about FSMs and Verilog, and introduces them to the design tools and the specifics of the FPGA board we use.

Lab2 - MIPS architecture: In this lab students are given a complete Verilog description of a small MIPS-inspired processor. We use the 8-bit mini-MIPS from the VLSI text of Harris and Weste [19]. The task is to add a new instruction to this machine, ADDI (Add Immediate), include memory mapped I/O to the switches and LEDs on the FPGA board, and then write the dreaded Fibonacci program using that new instruction. The program is required to compute the first 14 Fibonacci numbers and store them in consecutive memory locations starting at location 128 (the mini-MIPS has only an 8-bit address space so this is the high half of the accessible memory). It should then enter a loop where it reads the value on the switches on the FPGA board, and uses that as an address to the memory. The program should display the value stored in that memory location. That is, by changing the switches once the program has entered the loop phase, the corresponding Fibonacci number will be displayed on the LEDs.

This lab is essentially a tiny version of the project in a nutshell. The students will see a complete (albeit tiny) processor, understand it well enough to modify the code to add new features, learn how to use the internal memory on the FPGA, how to map that memory into the processor's address space, and to write and run a small assembly language program on that processor.

Lab3 - VGA: In this lab every student in the class implements a simple VGA controller. Students implement a control and timing circuit for a 640x480 VGA output circuit. They demonstrate that they can paint the entire screen in a different color based on reading the switches on the FPGA board (our board allows only one bit each of R, G, and B so there are eight colors possible without designing a new VGA output circuit). They also take the state machine from Lab1 and drive regions of the VGA screen to correspond to the LEDs that lit up in Lab 1. This lab uses the same state machine from Lab 1, but maps its output to the VGA screen instead of to the LEDs on the FPGA board.

This lab is a response to experience in the course related to the VGA controller. We have found that teams that get VGA working earlier in the semester have a much higher chance of finishing and polishing their project. With this lab, every student in the class will have designed and built a simple VGA interface before the team project starts which will give all students a running start on the main project's I/O requirements.

The project portion of this course is defined rather loosely as "design and implement a computer system and demonstrate it doing something interesting." Unless the students have a specific alternative project in mind, the "something interesting" is defined to be an interactive game running on the computer and displaying on a VGA display. Students are given a baseline instruction set architecture (ISA) for a 16-bit computer architecture. To this baseline ISA the students add extra instructions as desired, and I/O interfaces. The teams usually also develop assemblers for their customized versions of the baseline ISA and write their game applications in assembly language. Some groups have gone even further and written simple compilers for their machines but this is certainly not required.

ADD, ADDI	Addition
SUB, SUBI	Subtraction
CMP, CMPI	Compare Subtract with no writeback to RF
AND, ANDI	Logical AND
OR, ORI	Logical OR
XOR, XORI	Logical XOR
MOV, MOVI	Move data
LSH, LSHI	Logical shift by one Direction is in extended opcode
LUI	Load and 8-bit left shift
LOAD, STOR	Memory load and store
Bcond	Branch to PC-relative signed displacement (16 conditions)
Jcond	Jump to address in Rtarget (16 conditions)
JAL	Jump and save PC in link register

Figure 2: Baseline ISA

In the project phase of the course there are no further lectures unless specific topics come up that the whole class would benefit from hearing. Instead the teams each meet with the instructor and the TA once a week to discuss the project, describe their current state, and make goals for the following week. The general plan is to demonstrate a series of checkpoints (that may not occur in the same order for each team). The checkpoints are:

– Functioning register file and ALU
– Complete datapath including a connection to the block RAM on the FPGA
– Memory map including the I/O mappings
– Instruction decoding and interactions with the datapath
– I/O system demonstration
– Full system demonstration

The final report consists of written documentation in the form of a conference-style paper, a collection design documents including Verilog code, schematics, etc., and a user guide to the group's processor and programming environment.

2.1 Baseline Architecture

For our baseline we have developed an ISA based on the National Semiconductor CR-16 microcontroller [6]. Our version of this ISA has a subset of the full CR-16 instruction set, and simplified instruction encoding, but it retains the flavor of a "real" microcontroller rather than an overly-simplified pedagogical architecture. The instruction set is a basic two-address load-store format with the result being written to the same register as one of the arguments. That is, `ADD R1 R2` performs the function `R1` ← `R1 + R2`. Instructions are encoded in single 16-bit words, and memory is accessed in a word-aligned way. ALU operations (ADD, SUB, CMP) set the conditions in the condition code register (FLCNZ flags). Instructions ending with "I" are immediate and use the eight least-significant bits of the instruction as data (sign-extended or zero-extended to 16 bits). The baseline instruction set is shown in Figure 2.

There are unused opcodes (up to 28 depending on how instructions are grouped) that allow the student groups to

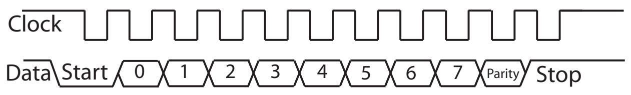

Figure 3: PS/2 keyboard interface serial protocol

add additional instructions to this baseline instruction set. Typical additions included arithmetic shift, unsigned arithmetic, multiple precision arithmetic (add with carry, for example), multiplication, multiply-accumulate, and instructions to support interrupt processing. The baseline architecture uses a 16-bit PC which allows an address space of 128k bytes (word-aligned), and 16 16-bit registers. The memory map is defined by the students but a starting point is to use the top two address bits to define I/O space so that 1/4 of the address space is usable for memory mapped I/O such as a frame buffer. This ISA serves as a good generic baseline from which the students can branch out and explore enhancements to the basic architecture.

3. I/O INTERFACES

The main distinction in this course's curriculum is a specific focus on I/O in the computer architecture. Our argument is that by spending time talking about I/O we create an environment where the students pursue the project with extra enthusiasm because the final result is visually interesting. We spend time making sure that the students have been exposed not only to the I/O protocols themselves, but also how those I/O devices are typically mapped into the computer's memory space. The two main I/O systems that we cover are PS/2 (keyboards) and VGA (graphic output).

3.1 PS/2 Interface

A PS/2 interface allows the students to use a standard keyboard as an input device. There is a PS/2 socket on the FPGA board, but no built-in controller so the interface circuit is designed by students. The low-level PS/2 protocol is a relatively simple serial interface: Codes are sent serially using two wires: a *Clock* and a *Data* wire as seen in Figure 3 (see [13] for more information). Data changes when the clock is high and is valid when the clock is low. Frames are always 11 bits long and begin with a start bit followed by eight data bits, an odd parity bit, and one stop bit. The key-mapping protocol is more complex than simple ASCII with one or two bytes being sent then the key is pressed (called the *make code*), and two or three bytes sent when the key is released (called the *break code*). Teams typically implement this interface by mapping the PS/2 data registers to a memory location which can be read by their processor for decoding.

3.2 VGA Interface

Game outputs are all about graphics. Augmenting the processor with a VGA output interface is critical to making the processor usable for games. A VGA interface can seem daunting to a student at first, but once understood it can be fairly easily added to the processor design. The main point is that the monitor is always moving the electron beam across and down the screen at some rate. The VGA interface keeps track of where the beam is on the screen and provides intensity data on the R, G, and B signals that correspond to the pixel color at that spot on the screen (see [17, 5] for details). What the VGA interface really controls is when the beam stops tracing and moves back (*retraces*) to the starting point. To cause the beam to stop tracing a row and retrace back to start tracing the next row the VGA interface sends a low-going pulse on the *Hsync* wire. To cause the beam to stop moving down the screen and move back to the top the VGA interface sends a low-going pulse on the *Vsync* wire.

The VGA timing interface is essentially just a pair of counters that keep track of the horizontal and vertical position of the beam and generate the required retrace timing signals. The timing for these signals is straightforward, and as seen in Figure 4, there are "official" timings, and there are relaxed timings that students can use with fewer pixels on the screen. The on-screen graphics happen between the retrace signals when the circuit sends signals to the display to make colors on the screen. This leads to essentially three different modes of generating an image on the screen:

Bitmap mode: In this mode a color value is stored for every pixel on the screen. The VGA controller reads the colors from that memory (the frame buffer) as it traverses the screen. This is the most flexible approach, and the most memory-intensive.

Glyph mode: In this mode the screen is broken up into multiple pixel chunks (8x8 pixels for example), and for each chunk a pointer is stored instead of a color. The pointer references a glyph which is stored in a separate memory. If there are (for example) 256 glyphs, then an 8-bit pointer is stored for every 8x8 chunk on the screen. Character maps are often stored as glyphs.

Direct mode: In direct mode the drawing circuit checks the horizontal and vertical counters directly and decides when to change the beam color. This mode is useful for drawing backgrounds or other static structures on the screen. It is also the mode most likely to be used in the VGA lab described in Section 2.

4. EXAMPLE SYSTEMS

The following examples demonstrate what the students teams can achieve in the semester-long class. These are admittedly among the better projects that students have done in the recent offerings of the course, but most groups achieve similar, if slightly less polished, results. These examples all use the same baseline 16-bit ISA described in Section 2.1. Each group has extended the baseline ISA in some way, and integrated memory-mapped I/O suitable for their project. Screen shots and pictures of these example projects are seen in Figure 5.

4.1 Slalom/Chicken

This group targeted a 512x480 VGA screen with 64 colors using a 25MHz VGA clock. They created an 8x16 pixel character set which was stored in the flash memory on the FPGA board. This allowed them to fit 64x30 characters on the screen which could include text characters as well as glyphs used for their game applications. By shrinking the logical frame buffer to only 64x30 locations where each location contains a pointer to an 8x16 character glyph they were easily able to map that frame buffer (under 2k bytes) into their main memory map.

Horizontal Timing

	640x480	128x256
A = Scanline time	31.77 us	30 us
B = Hsync pulse	3.77 us	2.0 us
C = Back porch	1.89 us	10.7 us
D = Bright time	25.17 us	12.8 us
E = Front porch	0.94 us	4.5 us
Pixel clock frequency	24.4 MHz	10 MHz

Vertical Timing

	640x480	128x256
A = Frame time	16.68 ms	16.68 ms
B = Vsync pulse	0.06 ms	0.09 ms
C = Back porch	1.02 ms	4.86 ms
D = Bright time	15.25 ms	7.65 ms
E = Front porch	0.35 ms	4.08 ms
Vertical Frequency	60 Hz	60 Hz

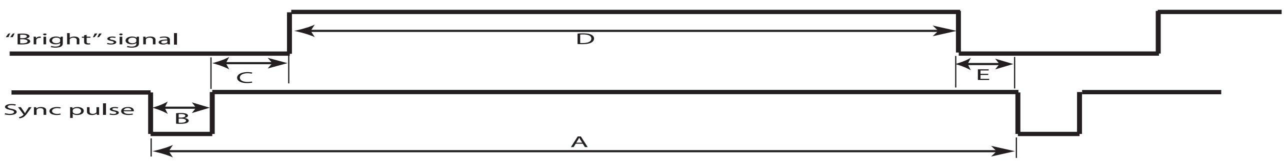

Figure 4: VGA interface signal timing

In addition to an assembler for their ISA, this group also built a character editor web application so that the task of designing characters and game glyphs could be shared by all group members. Because they focused on getting their VGA controller working early in the project, they actually used their VGA output to help debug their code as the project progressed by writing diagnostic data to the screen in their assembly programs. This group wrote three different games for their system which involved approximately 2000 lines of assembly code:

Giant Slalom: The skier tries to make it down a course without hitting trees or other obstacles.

Chicken: a two-player game based on the arcade classic Frogger. Each player tries to get their chicken across the road without being hit by any of the cars.

Tron: based on the classic arcade classic Tron. Players move around a playing field leaving trails.

4.2 Pong/Breakout

This group emulated classic video games Pong and Breakout by taking a different approach to their VGA output. They used 128x96 pixel resolution where these pixels are expanded on the screen such that the 128x96 pixels took up the entire screen space. In order to save frame buffer space each 16-bit location in their frame buffer memory held four 4-bit pixel color values. They implemented a library of assembly language graphics functions including painting single pixels, drawing a line from point to point using Bresenham's line drawing algorithm, drawing a box, and drawing characters on a screen using a simple font map stored in a separate area of memory.

4.3 Dance Dance Revolution

This group used dance pads from a home version of this popular arcade game as inputs, and used a combination of splash screens stored in flash memory and glyph-based VGA to control the moving arrows during game play. They synchronized the movement of the arrows to a song playing on an attached mp3 player. The mp3 player's controls were mapped into the processor's I/O space so that the start and stop of the music could be controlled by the program along with the graphics. This game was by far the most popular during the class open house at the end of the semester.

5. CONCLUSIONS

We have presented some details of a semester-long project-based course on computer design from the University of Utah. In this course the ability to design interactive games on student-designed computer hardware has greatly increased student excitement about their projects, and also increased the depth of student learning related to computer design and implementation. In order to target games as an application area important, but often overlooked, project features such as I/O must be dealt with in a strong way. While we did not do detailed assessment, it was clear to us from the final project reports that the game teams ended with a deeper understanding of many core computer design issues than non-game teams. The infrastructure required to add this dimension to project courses is not overwhelming, and the payoff is large. Not only do the students learn more about computer design, architecture, and implementation, but they have more fun doing it and they are much more likely to keep their projects to show off to friends, family, and prospective employers.

6. ACKNOWLEDGMENTS

I thank my students in the Computer Design Lab course over the years who have risen to the challenge and designed great projects. In particular I thank the students who designed the projects described in this paper. For Slalom / Chicken / Tron: Ryan Anderson, Whit Johnson, Brad Macdonald, and Adam Walker. For Pong/Breakout: Mike Green, Derik Jolley, Vic Lopez, Erik Nordstrom, and Vinh Pham. For DDR Colby Litnak, Paul Willoughby, Jacob Johns, and Jason Hansen.

7. REFERENCES

[1] J. Amaral, P. Berube, and P. Mehta. Teaching digital design to computing science students in a single academic term. *IEEE Transactions on Education*, 48(1), feb. 2005.

[2] T. Barnes, H. Richter, E. Powell, A. Chaffin, and A. Godwin. Game2learn: building CS1 learning games for retention. In *SIGSCE conference on ITi '07*, Dundee, Scotland, 2007.

[3] J. Brophy. *Motivating Students to Learn.* Lawrence Erlbaum Associates, 2nd edition, 2004.

[4] N. Calazans and F. Moraes. Integrating the teaching of computer organization and architecture with digital

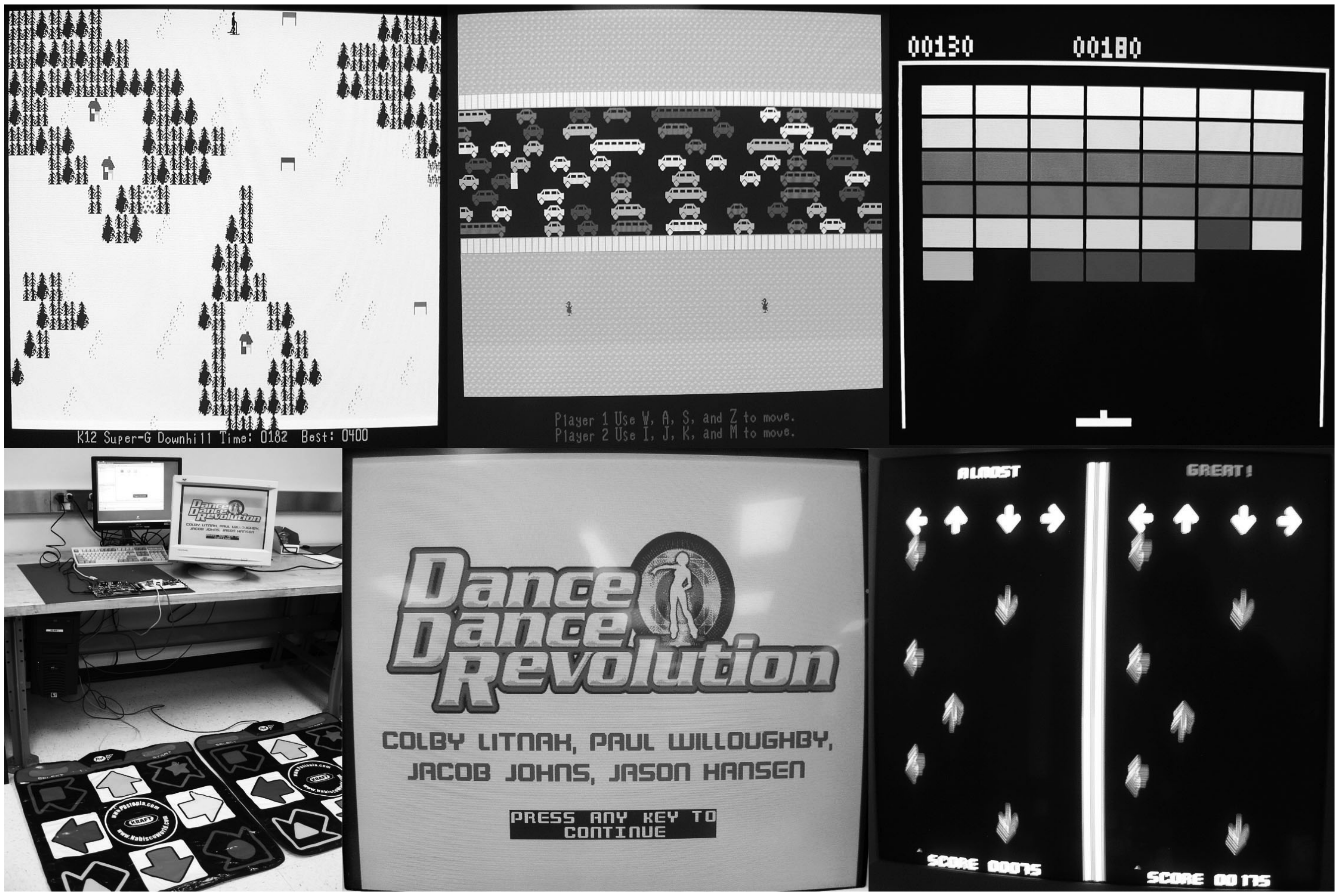
K12 Super-G Downhill Time: 0182 Best: 0400
Player 1 Use W, A, S, and Z to move.
Player 2 Use I, J, K, and M to move.
00130
00180
Dance Dance Revolution
COLBY LITNAK, PAUL WILLOUGHBY, JACOB JOHNS, JASON HANSEN
PRESS ANY KEY TO CONTINUE
ALMOST
GREAT!
SCORE 00075
SCORE 00175

Figure 5: Images from the interactive games developed by students. From left to right on top: Slalom Super-G, Chicken, and Breakout. On the bottom from left to right: Dance Dance revolution — dance pads, splash screen, and game.

hardware design early in undergraduate courses. *IEEE Transactions on Education*, 44(2), may. 2001.

[5] P. Chu. *FPGA Prototyping by Verilog Examples: Xilinx Spartan-3 Version*. Wiley Interscience, 2008.

[6] National semiconductor CR16 microcontroller http://www.national.com/appinfo/cr16.

[7] K. Eble. *The Craft of Teaching: A Guide to Mastering the Professor's Art*. Jossey-Bass, 2nd edition, 1994.

[8] D. Harris. The microprocessor as a microcosm: a hands-on approach to VLSI design education. In *Frontiers in Education Conf.*, Boston, MA, Nov 2002.

[9] M. Holland, J. Harris, and S. Hauck. Harnessing FPGAs for computer architecture education. In *Microelectronic Systems Education, 2003.*, jun. 2003.

[10] S. Leutenegger and J. Edgington. A games first approach to teaching introductory programming. *SIGCSE Bull.*, 39(1):115–118, 2007.

[11] R. Paharsingh and J. Skobla. A novel approach to teaching microprocessor design using FPGA and hierarchical structure. In *Microelectronic Systems Education, 2009.*, jul. 2009.

[12] M. Pearson, D. Armstrong, and T. McGregor. Design of a processor to support the teaching of computer systems. In *Electronic Design, Test and Applications, 2002.*, 2002.

[13] PS/2 protocol description. http://www.computer-engineering.org/ps2protocol/.

[14] S. Shabanah, J. Chen, H. Wechsler, D. Carr, and E. Wegman. Designing computer games to teach algorithms. In *Information Technology: New Generations (ITNG), 2010*, apr. 2010.

[15] K. Sung. Computer games and traditional CS courses. *Commun. ACM*, 52(12):74–78, 2009.

[16] E. Sweedyk, M. deLaet, M. C. Slattery, and J. Kuffner. Computer games and CS education: why and how. In *SIGCSE '05*, St. Louis, Missouri, USA, 2005.

[17] VGA timing description. http://www.epanorama.net/documents/pc/vga_timing.html.

[18] K. Vollmar and P. Sanderson. MARS: an education-oriented mips assembly language simulator. *SIGCSE Bull.*, 38(1):239–243, 2006.

[19] N. Weste and D. Harris. *CMOS VLSI Design: A Circuits and Systems Perspective*. Pearson Education, 3 edition, 2006.

[20] Xilinx corporation web page. http://www.xilinx.com.