AC 2012-5377: FPGARCADE: MOTIVATING THE STUDY OF DIGITAL HARDWARE

Dr. Danial J. Neebel, Loras College

Danial Neebel is as Associate Professor of engineering and computer science at Loras College. His research interests include digital system design and testing, computer architecture, and computer science education.

Mr. Nicholas J. Burek, Loras College
Thomas Griebel, Loras College

FPGArcade: Motivating the Study of Digital Hardware

Abstract

This paper presents the FPGArcade system that makes game development simple while still providing insightful details into low-level concepts. Fledgling programmers can test the waters of the hardware/software interface without having to go through several courses in digital logic, computer architecture, and electric circuits. Using an inexpensive FPGA-based platform from Digilent Inc. and software from the Xilinx Corporation, a programmer with some basic skills in C or C++ programming is able to create simple computer games. Since the system is FPGA-based the student can dig deeper and learn about hardware development by modifying the VHDL to provide new features.

Introduction

Liberal arts colleges across the country offer engineering and computer science programs that are sometimes limited in the number of courses available to students. Large schools are able to offer a sequence of courses in a variety of areas, whereas a small school may only offer one course in some of the major areas of study in computing and engineering. The authors' institution is one such place. There is one computer organization and architecture course that needs to provide a good architecture background for students in computer science and exposure to embedded systems for students in the engineering program. One group is fearful of any hardware and the other group thinks hardware means resistors, capacitors and light bulbs or beams, bolts and weldments. Digital logic would be the ideal starting point for each major. Later, students would branch off into either computer architecture or microprocessor applications. However, at small institutions there are not enough resources to offer multiple classes and the programs must make one course fit for two majors. The authors have provided a way for the interdisciplinary students to work on one system together while concentrating on their own expertise. The computer science students can address programming issues and the engineering students can address timing and interface issues in the hardware. This also addresses the skills required to work on multi-disciplinary teams—a required student outcome required for ABET accredited programs.

Engineering students might feel right at home with hardware, but it tends to intimidate many computer science students. Most are much more comfortable with a mouse, a keyboard, and a monitor than with an embedded system that only provide buttons and lights for I/O. Often embedded systems offer limited debugging tools and students are forced to analyze their code more closely than when using a full IDE. The advent of VHDL and today's readily available hardware development tools are making hardware development available to those with a rudimentary knowledge in software and digital logic. But, the fear of hardware still exists when students get into the lab.

Declining enrollments in engineering and computer science fields have led educators to look for new ways to motivate students. Video and computer games are often used to excite students about math and science. Institutions such as DigiPen[1] are developing projects to motivate

middle and high school students to learn more about game development and 3D animation. Using game development and easy to use graphics programming systems is not a new technique for motivating learning in computer science and engineering. Carnegie Melon's Alice[2,3] project has students in middle and high school developing 3D animations. Several schools now offer a track or an entire major in game development as a way to entice more students in the computer science field. ACM has hosted the annual conference on Game Development in Computer Science Education since 2006. Development of games on an FPGA platform has also been explored.[4] However; such courses require a digital logic course and a computer architecture course as prerequisites for the FPGA-based course.

Using game development as an education medium can prove difficult in early courses since game development requires knowledge of a variety of advanced topics such as digital design, I/O interfaces, real-time systems, graphics, and in some cases artificial intelligence. Tools that make game development easy for students tend to provide interfaces that abstract away many low-level details. The high level of abstraction implies that they are not as effective at helping students learn the underlying principles. Students hopefully will catch on and find themselves immersed in the joys of technical creativity. It is a gateway into the field and serves as a means for students to continue to grow intellectually.

Digilent Inc. is working to motivate students to study hardware development by providing low cost FPGA and microcontroller hardware to students and academic institutions. The mission is to put the hardware into the hands of the students. Digilent sponsors the Digilent Design Contest an annual world-wide contest to encourage students to be innovative.[5] The idea of the contest is to give the students a piece of hardware, some development tools, and turn them loose. When this opportunity came to two students at a small liberal arts school they put their heads together and decided to develop something that interested both of them rather than interested the faculty mentor; one student being a computer science major with side interests in mathematics and philosophy and the other an engineering student with a side interest in computer science. Starting with the idea to recreate a retro arcade game, the project proposal soon grew to a programmable arcade system that would provide the backbone for creating any arcade game. The FPGArcade[A] a system that could be programmed in C or C++ and use their own design for a Picture Processing Unit similar to the original Nintendo Entertainment System. The following describes the system put together by these two students.

A. What's in a name? A quick Internet search will lead you to http://www.fpgaarcade.com[8], a commercial web site for a board developed to emulate a variety of older arcade systems. Despite the similar name, the goals of the FPGA Arcade system and the FPGArcade system are quite different. Our system, FPGArcade was developed as a learning experience and a learning tool.

FPGArcade Overview

An interest in hardware may be catalyzed if students were presented with an attractive, fun and visual way to learn engineering. This is the main aim of the FPGArcade. The system uses video games to entice programmers to understand hardware. The student may wonder just how pixels are streamed across the screen, how does the game know what the joystick is doing, and how does the instruction code command those invisible circuits? All of these questions might be raised with any video game console, but the FPGArcade lays that all bare and exposed. The

system is simple and easy to use. A developer can quickly peel back the layers and unravel the components and truly understand how the system works from the C libraries to the HDL specifications.

The FPGArcade is no foreign device for any gamer. In fact, it's inspired by older gaming consoles. The model for the FPGArcade system is based upon the Nintendo Entertainment System's Picture Processing Unit (PPU). The PPU used tiles to take advantage of graphical redundancy thus reducing the amount of memory needed. The FPGArcade uses a tile-based design and defines a set of instruction codes used to interact with the video memory. These instruction codes are available as memory-mapped I/O. Programmers can access the instructions directly or call easy-to-use wrapper functions which can set the pixels and location of individual tiles. Tiles can also be set to fixed or movable, transparent or opaque, and solid or not. The system is intuitive and allows for a quick visual way to interact with hardware from high-level C code. The FPGArcade system also provides a joystick interface as part of the C library to provide ease of use for the student developer. The modular design of the system allows for the development of additional interface devices. These features create a fun, interactive, and educational system that can be used to encourage students in engineering and computer science to further their understanding of hardware.

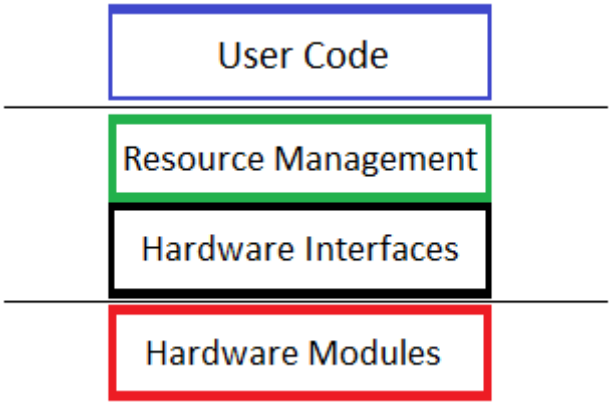

Figure 1: Layered Architecture of the FPGArcade.

The FPGArcade is designed to be used from a top down perspective. Users can dive right into game development and experience the reward of seeing their game running on a board. The real gem here is the transparency. The endless opaque code in contemporary machines abstracts the hardware making it difficult if not impossible for a novice to untangle. It is hard to understand what is truly going on under the hood of a machine when Windows, glut, and other API's obscure one's vision. The FPGArcade is modeled with three simple layers, shown in Figure 1, which can be peeled back. Students are encouraged to delve deeper to optimize the system for their needs and extend their creativity to its maximum end.

Graphical API and Hardware Design

The FPGArcade is implemented using a tile-based architecture similar to many classic video game console architectures. A video RAM stores tile data. Tiles are 8x8 pixel graphical fragments; there is storage for 128 tiles in total. For the display a buffer of 80 by 60 tiles is used

yielding a typical 640 by 480 pixel VGA display . Tiles displayed on the screen grid are called blocks. This system reduces memory usage through the use of redundancy. Storage for 16 movable tiles is allocated as well. Moveable tiles refer to tiles in main storage and also maintain an x and y coordinate, and transparency.

To change the state of the system, a series of operational codes are defined. These are low-level codes consisting of a series of 32 bits. GPIO functions are used to create memory-mapped I/O. These GPIO operations are wrapped in the systems hardware interface API. Using a series of "set_" functions, a variety of graphical instructions can be conveniently called by the user. An API exists for both the graphical functions as well as the joystick control functions. Another layer exists on top on the interface API. This is called the Tile Management System and is used to make intelligent use of the hardware through resource management. The Tile Management System (TMS) also provides a higher level interface for a programmer to use; making the system even easier to use. Figure 2 shows the architecture of the Tile Management System.

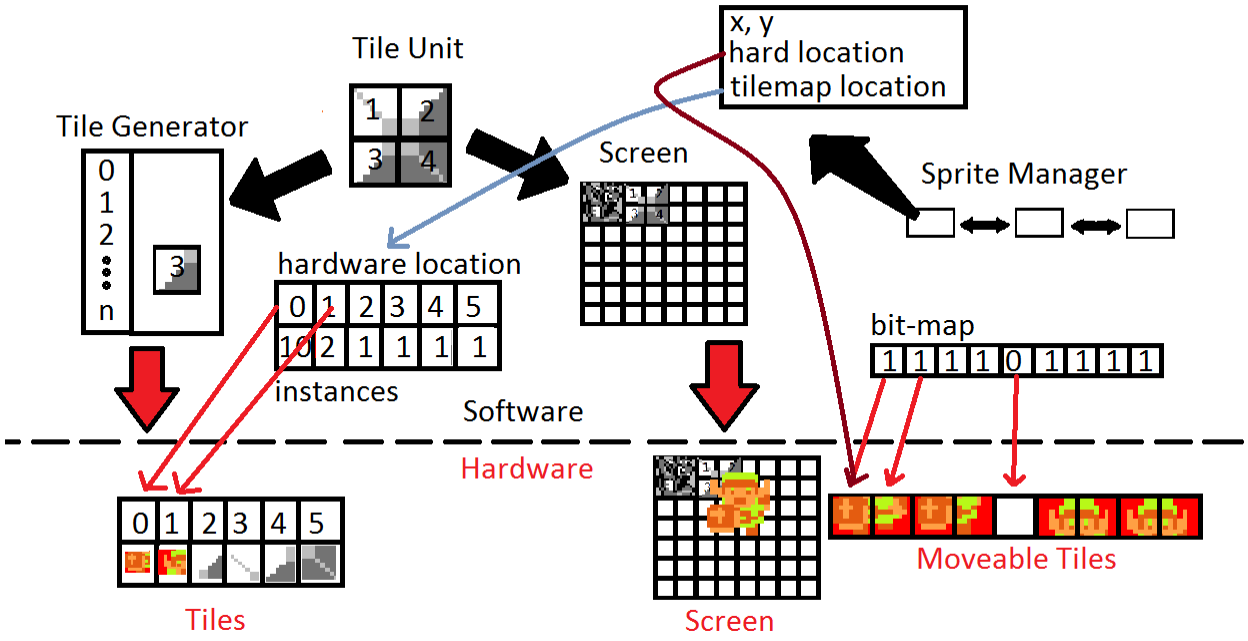

Figure 2: Tile Management System Overview.

A user can create games with this system with either the hardware API or the TMS (which is built off the hardware API). With the TMS a student can use the board and make games quickly. However, to make efficient use of the system the lower functions will need to be incorporated. For instance low-level functions exist to flip and rotate tiles. The API provides a quick way to display graphical data on the screen and can offer the student several insights into how the memory mapped IO can be utilized in useful ways and peals away the mystery of how intangible programs command the physical hardware.

Next, the design goal of the TMS will be explained. This system is designed for students who would be comfortable on a systems level. As a student creates their game they may desire to tweak and fine tune the memory management systems at work in the TMS. For example, the

sprites (moveable tiles) are stored in hardware using a least-recently-used replacement algorithm. The game designer may find it advantageous for their game to use an algorithm based on priority. Their goal may be to have the main game character should display at all times. Other memory related tweaks might also prove useful. At present, instructions to write animations to the screen must be executed per game-loop if the sprites are to be guaranteed to display on the screen otherwise the animation might be replaced by newer animations. The student might automate the write-out process by extending the software queue and making a few other system modifications. The TMS offers the student a fun way to familiarize themselves with low-level memory management. Best of all the results are visuals. The concept of thrashing, for instance, can be visually represented as character animations flicker and flash while the system struggles to swap out images in time.

Since the TMS is designed for general purpose use, a student will eventually reach a point where they need a more optimized way of performing an operation. This is where the system API comes into play. The Tile Management System presents ways in which software *manages* hardware information and utility, but the system API demonstrates ways in which software *interacts* with hardware. With the system API the student is able to directly read and write to the memory mapped I/O locations, allowing them to optimize how their code works to eliminate any unneeded overhead. It is also the software side of the boundary between the computer code and the hardware.

Since the all the hardware is implemented in VHDL code it enables students to modify the hardware system to either optimize it for their specific application or to add new modules. So if a student decided that they wanted to add a keyboard to get input for their game they would be able to create their own driver in VHDL and then add it to the memory mapped I/O modules. This gives them experience not only creating hardware components, but also developing new low level system API functions to interact with those new components.

Future Plans

The students have built and tested the FPGArcade system; the instructor can now take the new tool and go back into the classroom and lab to further the learning of the next group of students. There is a range of possibilities given the layered approach to the architecture. The earliest that FPGArcade could be used in the curriculum is after at least one course in programming such that the students have some understanding of complex data types, pointers, and bit manipulation code, such as bitwise AND, OR, and XOR. Here are two exercises that could be divided up into one or two hour lab exercises. These exercises will require the use of Xilinx's EDK software that is made available through the Xilinx University Program[6].

With the given background above FPGArcade could be used for several exercises in a freshman or sophomore computer engineering course in embedded systems. A set of exercises could be developed for use in the laboratory or if the students have their own boards, they could take them home. Here are two such possible exercises for early in a computer engineering curriculum.

1) Exercises for building the architecture of a game.
   a) Generate a sprite and place it on the screen in a fixed location. The Sprite can be the same tile rotated or four different tiles.
   b) Then add a four tile sprite that moves from left to right across the middle of the screen.

c) Next, change the program such that the sprite moves diagonally across the screen. The sprite must reappear on the opposite edge if it moves off the screen. The speed of the movement can be fixed and dependent upon delays placed in the program since the program need not perform other operations.
d) Make the sprite "bounce" at screen boundaries or for an added challenge use the collision detection features to make the sprite bounce off "solid" tiles.

2) Building an interactive game. Students will update their design from the previous exercise to add user input that controls the movement of sprite on the screen.
   a) Use the joystick to move the sprite horizontally and vertically.
   b) Using the movement ability from Parts A and B, create a "game" that requires the user to put the sprite in a "box" on the screen.
   c) For an added challenge, modify the current program so that the left button on the joystick causes the sprite teleport to a random location on the screen.

Future use for FPGArcade includes plans to incorporate the VGA module with hardware support for the tile-based graphics into an embedded VHDL MIPS core. The MIPS core currently in use at the authors' institution is a subset of the MIPS architecture presented in COD 4th Ed[7]. The current core implements the basic ALU, supports procedure calls, beq, bne, slt. No floating point support or support for more complex branch instructions. The MIPS core and tile-based VGA module will be used in a sophomore/junior level Computer Organization and Architecture course. Possible lab exercises include:

1) Write MIPS assembly code to perform memory mapped I/O that generates and displays background tiles.
2) Write MIPS assembly code to create a moveable sprite made of one tile that moves left to right across the screen, then add vertical motion to make tile move diagonally.
3) Next, expand the sprite to four or more tiles that move together to create a larger moving object.
4) Another challenge would be creating a set of character tiles and using them to write text to the screen.

Conclusions

What started out as an extra challenge for two students will hopefully become something useful for future students. The FPGArcade system will be classroom tested in a computer architecture course in the fall of 2012. The expectation is that the experience of working with joystick and video signals and a "game console" will be highly motivating. The hope is that after working with the FPGArcade this fall more students will decide to work on an FPGA or microcontroller project and compete in a contest or just further their learning. As of this writing, the success of the co-authors at the Digilent Design Contest 2011 has motivated to first year students to enter the contest for 2012.

Finally, the development of the FPGArcade was itself a good learning experience. The two students who designed and developed the FPGArcade wished to comment on their learning experience. Below are their individual comments.

Student 1: This project served as a mini-capstone of our education. We combined our collective knowledge from engineering, computer science, and mathematics to create a useful and fun

system. Actually seeing the bare circuits illuminate a monitor gave us a dizzying rush. In short, it was as addicting as it was educational.

Student 2: I think this project was a great culmination of different areas of both computer science and engineering. It not only combines computer software and hardware into a single project, but also covers a wide range of topics within each. There are ties to digital logic, computer architecture, software engineering principals, computer graphics, operating systems, and a range of other areas of study. One thing I distinctly remember was sitting in my Operating Systems class one day learning about memory paging when I realized that I had actually implemented a rudimentary memory paging system just the past week to manage storing and mapping the graphical pixel data to the screen. It is experiences like that, were a student is able to tie what they are learning in class to an actual project, that I think encourages students to keep learning and expanding their knowledge of different topics.

Those interested in seeing the FPGArcade system can find the documentation and code for FPGArcade is currently available on github at: https://github.com/nburek/FPGArcade and on the Digilent Inc, 2011 contest web site.[8] Updates to the package for use as courseware will be posted after formal testing in the classroom and lab in the fall of 2012.

Bibliography

1. (n.d.). Retrieved from DigiPen: http://www.digipen.edu
2. Randy Pausch, e. a. (1995, May). Alice: rapid prototyping for virtual reality. Computer Graphics and Applications, 15(3), 8-11.
3. Carnegie Melon University. (n.d.). Retrieved from Alice: http://www.alice.org
4. Brunvand, E. (2011). Games as motivation in computer design courses: I/O is the key. 42nd ACM technical symposium on Computer science education (SIGCSE '11) (pp. 33-38). New York, NY: ACM.
5. J., M. (2012). FPGA Arcade. Retrieved 1 2011, from http://www.fpgaarcade.com
6. Xilinx University Program. Retrieved 2 2012, from http://www.xilinx.com/university/index.htmDigilent Inc. (2011, 12).
7. Patterson, D. and Hennessy, J., (2008), Computer Organization and Design, Fourth Edition: The Hardware/Software Interface, Morgan Kaufman Publishers.
8. Digilent Design Contest 2011. Retrieved 1 9, 2012, from Digilent Inc. - Digital Design Engineer's Source: http://www.digilentinc.com/showcase/contests/designcontest.cfm?ContestID=8