18-545: Advanced Digital Design Project

Fall 2008

Team Vertigo

Final Report

Joseph Greco, Travis Iwanaga, and Michael Rozyczko

jfgreco@andrew.cmu.edu,

tni@andrew.cmu.edu, mrozyczk@andrew.cmu.edu

Submission Logistics

Accompanying this report should be our submission, titled inferno.zip. At the root level of the archive should be a single file entitled "system.ace". This is a System ACE file which can be placed onto a CompactFlash card for easy bootstrapping of a XUPV2P development board.

Also at the root level of the archive should be a folder entitled "inferno". This is a Xilinx Platform Studio project folder. Note that to work with this project, you may need to edit the "ModuleSearchPath" line of system.xmp to reflect the location of the XUPV2P peripheral libraries on your system. If you are running a Linux system with the peripheral libraries at /opt/Xilinx91i/EDK/lib, this file will not require any editing.

Our source code is provided in inferno/SDK_projects/inferno (relative to the root of the archive). If you wish to build it, you should use the XPS SDK.

Note that there is a bug in the framebuffer in some versions of these peripheral libraries. The issue causes the VGA image to be shifted vertically off of center. To ensure that this bug does not affect you, please inspect the README in the "patch" folder in the root of our submission.

Contents

Introduction

Our project's aim was to recreate Parallax Software's *Descent* on the Xilinx Virtex-II platform. Our goal was to start with a straight software port, complete with functioning input, video, sound, and network multiplayer. Once we had reached this stage, we would examine what areas of the project were lagging in performance, and write hardware to accomplish accelerate these functions.

The Mission

On the first day of the course, we were tasked with turning the Xilinx Virtex-II FPGA and the XUPV2P platform built around it into a gaming system. How we accomplished that was up to us. Examples from past projects included an NES written from the ground up in Verilog and implemented on the FPGA, a standalone graphics accelerator that communicated with a host PC via Ethernet, and a game utilizing a virtual reality helmet.

However, the project that stood out most to our group was XilQuake. XilQuake consisted of a port of the source code of ID Software's *Quake* to the Virtex-II's PowerPC platform, and a hardware Floating-Point Unit to accelerate the graphics engine. Of the projects described by the professors, this was among the most impressive, and easily the most accessible – they brought a XUPV2P and a keyboard into the conference room, booted it with a CompactFlash card containing a System ACE file, and demonstrated the project on the projector.

Our group sought to replicate their success. The first order of business, then, was choosing a suitable game to port.

The Game

Even with the hardware FPU, XilQuake was pushing the limits of playability. Their project taxed the performance of the onboard PowerPC CPU to its limits. If we wanted our project to be successful, we needed a game from the same era of Quake or earlier, as later games relied more and more on complex, dedicated graphics processing hardware.

One of the first games to spring to mind was X-Wing, released by LucasArts in 1993. It was one of the first simulations to use 3-D polygonal models instead of bitmaps and sprites, and the cause of many pleasant childhood memories. This game, and its sequel, Tie Fighter, seemed to be at about the right complexity for what we needed.

These games and Quake, however, have one large difference, besides their genres and level of graphics complexity. Quake's source code has been released, while X-Wing's has not. Classics like X-Wing are difficult enough to get running on modern x86 platforms – porting it to another CPU architecture without the source code would be next to impossible.

Our next and ultimately final idea was Descent, developed by Parallax Software and published by Interplay in 1994. It also used a fully 3D graphics engine, included sound, and even included

multiplayer (unlike X-Wing or Tie Fighter). More importantly, its source code had been released more than a decade earlier, and there was still an active multi-platform project around the source code. Furthermore, all three of us had played the game in our childhood and enjoyed it immensely. The decision was made.

The Platform

The platform provided to us is a Xilinx University Program Virtex-II Pro, referred to in the rest of this document as an XUPV2P. It comes with a plethora of input/output ports, but the ones most interesting and useful to us are the dual PS/2 ports, the 15-pin VGA connector, the 3.5mm audio minijack connectors, the 8-pin serial port, the DDR SDRAM slot (complete with 512 MB of RAM), and the Ethernet port. These are backed up in the FPGA by reference implementations of a PS/2 controller, a VGA framebuffer, an AC'97 controller, a serial controller, and a 10/100 Ethernet controller, respectively. These, in turn, are connected to the one or both of the PowerPC 405 CPUs through the use of reference interconnect bus designs.

The software platform included a compiler for the FPGA's PowerPC, drivers that allow interaction the peripherals, and useful libraries such as a filesystem library.

The Tools

All of our development was done on Ubuntu Linux 8.04, using version 9.1 of Xilinx's Embedded Development Kit (EDK). Specifically, we used the Xilinx Platform Studio (XPS) to set up the hardware, and used the XPS SDK (amusingly, a slightly modified version of Eclipse) for our software development.

Design Flow

The progress of the course was not entirely free-form. Groups were given a series of five labs they were required to complete and demonstrate in order to familiarize themselves with the platform.

Infinite Optimism

One of our first assignments in 18-545 was a reading assignment: The Pentium Chronicles, by Robert P. Cowell. In it, he imparts a myriad of lessons learned during his time at Intel developing the Pentium Pro, a CPU whose architecture would serve as the foundation for many of Intel's later chips. Foremost among the lessons that made an impression on us was one of Cowell's most simple mantras: "Under-promise, over-deliver."

In our first meetings, we took these words to heart, perhaps even a little too much. Alongside the Gantt chart we drew up and presented at our project proposal and design reviews, we drew up a second, more aggressive Gantt chart. We kept this second chart secret and internal. While our public schedule had us finishing the networking components after DR3, our private schedule had networking complete well before DR3 and left the time after DR3 entirely as slack. We briefly discussed what to fill this slack with – perhaps further optimizing the graphics code to the point where we could run at HD resolutions, or perhaps porting the code of Descent 2.

We examined the various versions of the Descent source code available to us. We could choose from the original PC source code (complete with DOS dependence), the original Macintosh port, a version from 1999 which boasted Linux compatibility, and a recent version complete with updated resources. We elected to go with the recent version as our foundation. It is available from http://www.dxx-rebirth.de.

Infrastructure

When we attempted to dive in and begin work on the labs and porting the code, we were met with an unexpected surprise. Our workstations had been updated to the newest version of the Xilinx tools, version 10. These tools regarded many of the reference peripherals for the XUPV2P as deprecated. The results were crippled hardware projects with little ability to interact with the outside world.

This necessitated a move back to the version 9 tools, which the course staff was able to acquire after less than a week. Meanwhile, Mike migrated the workstations from Windows XP to Linux, as the source we had acquired was easily compiled on that platform. He also set up the source control system and began the work of migrating the code to the Xilinx platform, as Joe and Travis began tackling the labs.

Setbacks

Slowly but surely, our schedule began to creep back, until by the end of the project our actual progress was in line with our public schedule. A good number of things contributed to this, but chief among them was real life. Mike had a sizable commitment to his Digital Circuits class,

Travis had a significant amount of work from his Computer Graphics course, and Joe was forced to spend more and more time in the robotics laboratories for his class from that department.

Design Review 1

By Design Review 1, the team was able to get the entire project to compile with the Xilinx toolchain. Even this has grown to be a more weighty task than anticipated: the more recent versions of Descent required the PhysicsFS and SDL (Simple DirectMedia Layer) libraries, which has been added to the source for better cross-platform compatibility. To us, they were two additional projects which had to be stripped of references to libraries not provided by Xilinx and massaged into working on a new toolchain.

SDL is a library designed to simplify communicating with hardware in cross-platform projects. A platform with a full SDL implementation has access to audio, CDROM, timer, joystick, and video hardware through a standardized interface. Fortunately for us, they provide a set of null drivers, essentially stubs that fulfill the requirements of the library yet offer no functionality. Because of this, we were able to get SDL to compile relatively quickly.

PhysicsFS was another matter. It provided no null filesystem driver. The closest match to our platform was the Unix/Posix implementation, however Xilinx, no matter which filesystem library is chosen, has no concept of a directory entry (or a dirent). This led to an interesting game in which header files like dirent.h were copied verbatim from the Linux host platform, which would of course lead to more missing header files and unsatisfied requirements. Eventually, the library was made to compile in time for the design review, though it certainly would not have been able to function at that point.

By this point, Joseph and Travis had made excellent progress with the labs – labs 0, 1, and 2 were down, with some progress on lab 4. Progress on lab 3 was stymied by the fact that, at this point, the licensing issue with the Ethernet hardware had not yet been resolved. Our private schedule, incidentally, had assumed all labs would be completed by DR1.

Travis had also made excellent progress with an FPU which he'd salvaged from his days in 18-340. We assumed we'd need an FPU because most modern graphics engines rely heavily on floating point math, and because inclusion of an FPU was, according to their report, the only thing that had made the XilQuake project playable.

Design Review 2

For this design review, the most significant progress was made implementing drivers for SDL. While the code for the game had been compiling successfully on the Xilinx toolchain since the previous design review, the filesystem support still was not complete. Joeseph and Travis, the ones largely responsible for this progress, tested their implementations by using a separate, simpler software project which had no dependence on filesystem resources. As a result, our demo for this design review consisted of a small program drawing a series of simple shapes onto the screen.

XMD, the Xilinx command-line debugger, proved infinitely valuable. Changes to the code could be made and compiled, and the new ELF file could be instantly downloaded to the board and run. Stdout was redirected to the serial port, which allowed debugging messages to appear on the connected host machine.

Progress on the filesystem was stymied by the directory entry problem. Support for basic file operations had been written, but PhysicsFS required the ability to list a directory's contents in its initialization. The previous work done providing the definition of the dirent structure to PhysicsFS proved fruitless, because neither of Xilinx's filesystem implementations either allowed directory information to be read into a dirent, nor did they provide it in a fashion which could be easily shoehorned into one. Eventually, a method for providing this information to PhysicsFS was written, but for some reason file related calls were still failing in the game.

Additionally, Travis was able to get his FPU synthesized and running on the board by this checkpoint.

Design Review 3

Debugging the filesystem problems in Descent proved difficult and extremely frustrating. Each of the file operations appeared to work in isolation. Furthermore, the point at which the game would crash moved about as one inserted and removed calls to xil_printf().

Eventually, it became apparent that when the game was crashing, it was becoming stuck in malloc() code. This resulted in increased scrutiny to each malloc() call, which eventually brought the problem to light. In haste, certain functions in the PhysicsFS code had been modified to return constant strings to the rest of the library, where the library was expecting a malloc()-created buffer containing the desired string. Thus, when the library eventually tried to free these strings, massive corruption to the memory image was caused even if the call did not immediately fail.

For the first time, the team was able to see Descent progress to the opening menu. The PS/2 input drivers, written in isolation in the separate SDL software project, required a bit of massaging to work with what Descent was expecting, but followed quickly afterwards. Disappointingly, the game proper did not function properly, but instead failed an assertion relating to the player file just before the start of the mission. Sound support had also been written, but without a working game there was no way to test it.

For design review 3, Team Vertigo was able to demo a mostly working menu system, and also the briefing system, which drew onto the screen the same textured models that would be used during gameplay, using the same graphics function calls.

The unsettling thing about this progress, however, was how slowly the hardware was doing everything. Drawing one of the title screen bitmaps to the screen took just over a second. Even if we managed to get the game working, how were we going to get an acceptable framerate? Closer inspection of the code revealed that the texture mapper used by descent dealt entirely

with fixed-point math, so an FPU would not be able to rescue us the way it did the XilQuake project.

The Final Demo

The two weeks before the demo were a time of rapid progress, despite the intervening Thanksgiving holiday. It was discovered that (yet another) simple bug in the filesystem was causing the problem stopping us from reaching gameplay. Several more minor bugs were squashed, and gameplay was finally accessible. However, it ran at 2 frames per second at most. Mike, using XilQuake as a base, switched to a floating point texture mapper and using the FPU. Performance was dismal with the FPU, so we switched back (not to mention that the floating point texture mapper was buggy and showed evident signs of display corruption). XilKernel, which we had previously migrated our project to because we believed it necessary for access to the timer, was dropped with no performance improvement. We considered implementing the entire texture mapping function in hardware, but the extreme number of parameters we would have needed to pass, even over the OCM bus, likely outweighed any speedup we might receive. We were stumped, because this game had been playable on a 486 with a quarter of the clock speed back in 1994. This was when we realized we hadn't enabled caching for the CPU.

Immediately, every part of our program received a twenty-fold performance boost. The game now ran at an extremely playable framerate. Six hours later, sound was working. Soon after, some bugs leading to sticking input were fixed.

Significant progress was made on networking and music, however the results were not useful in time for the final public demo. What we were able to demonstrate was a game with completely working input, sound, graphics, and saved games.

Subsystems

Once the header files were adapted to work on the Xilinx platform, the core code of Descent proved remarkably robust. The areas where we were needed to make the project function was primarily where the software expected to interact with the hardware. This interaction can be divided into five categories: filesystem, networking, sound, graphics, and input.

Filesystem

As previously mentioned, the filesystem support was one of the most frustrating parts of this project. The absence of directory entries, and, by extension, functions for opening and reading directories, complicated the implementation of certain aspects of the driver. Subtle ways that calls to xilmfs differed from standard Posix calls led to inexplicable bugs which took hours to track down.

These problems were exacerbated by how Descent uses PhysFS. Descent was originally written with its own filesystem abstraction, titled simply "cfile". When the community attempted to make Descent cross-platform, this abstraction was re-engineered to point to PhysicsFS, another layer of abstraction. However, later coders would bypass cfile altogether and make calls directly to PhysFS. This inconsistency in coding style would also lead to many headaches.

As far as the final product is concerned, PhysicsFS implements a robust wrapper for xilmfs. Reading and writing of files in the memory file system is done flawlessly as far as the game requires.

Xilmfs was chosen because Descent required non-sequential access to its archive files. For example, after consulting the header at the start of the archive, Descent would then attempt to seek directly to the file in question, rather than having to read all the intermediate files on the way. While xilfat does not support this, xilmfs does, and thus it was chosen for our filesystem.

Video

Double-buffered video support was one of the easiest things to get working for this project. SDL's framebuffer format is easily convertible to the format Xilinx expects to send out, meaning that conversion is inexpensive and can be done on-the-fly as the framebuffer is being copied to the hardware.

A method was written to have Xilinx flip the framebuffers whenever Descent would tell SDL to flip the framebuffer, converting the SDL framebuffer in its pixel format to the framebuffer format for Xilinx and to the appropriate framebuffer.

One of the things we chose to do to improve video performance is to actually render the framebuffer at 320x200 (Descent's original, native resolution), and quad each pixel as it is copied into the Xilinx framebuffer. This allows the image to fill the screen, even if what's being rendered is not a full 640x480.

Input

Getting input from the PS/2 keyboard, to SDL, and finally to Descent was somewhat more complicated. Due to prompting from the XilQuake report, we did not even attempt to get polled input working, as Descent, like Quake, polls for input between frames. This would have led to missed keyboard scancodes and dropped user input.

Instead, we used an interrupt-based architecture. Our interrupt handler takes keyboard scancodes and places them in a FIFO buffer. Our polling function creates an SDL keyboard event from the scancode, and passes it on to Descent's key handler. This required a key look-up table, as SDL's scancode definitions do not line up perfectly with PS/2 scancode definitions.

Audio

Audio also uses an interrupt-based architecture. When the AC'97 controllers FIFO buffer is half empty, it fires an interrupt. This interrupt is serviced by a modified version of Descent's mixer, which mixes several samples together and puts them into the AC'97's FIFO.

Adding music to the game proved more difficult. All of Descent's music was in MIDI format. Sadly, we did not have time to implement a MIDI synthesizer on the FPU, so instead, we found recorded samples of each track of game music.

However, playback of the music made apparent distortion that was barely audible with the rest of the game audio. This distortion was eventually discovered to have two sources.

The first is that it is somehow possible to "overfeed" the AC'97 FIFO structure, despite using a while() loop on a test to see whether the FIFO is full. One possibility is that this is because the test function was designed with a 44100 Hz sample rate in mind, but Descent uses a 11025 Hz sample rate. Regardless, limiting the number of samples the audio interrupt handler was allowed to write to the FIFO at any given time made the distortion in the music go away.

However, the game audio was still somewhat distorted. The reason for this is that Descent's sound resources are presented in 8-bit unsigned raw PCM format, while the AC'97 controller expects a 16-bit signed value for each channel. The method we had been using up to this point to make the sound audible was simply to shift it left by eight bits.

A much more elegant solution was found at http://www.gamasutra.com/features/sound_and_music/19981218/surround_02.htm. Among the things discussed is a method for computing a mixing table, which, given an 8-bit unsigned sample and a desired volume, produces an appropriate 16-bit signed sample. This table was adapted to our needs and included in our code, and led to a distortion-free mixer.

Networking

Network was not finished in time for submission. Descent's networking code, like PhysFS's code, was heavily entrenched in Unix's networking libraries and system calls, many of which were not provided by xilnet. Many hours of testing were rendered useless after we realized that we'd

been testing on a board that had been booted for weeks, when the networking IP was designed to stop working after just seven hours due to a licensing issue.

Personal Notes

Joseph

Video

After Mike got SDL set up compiling with the null drivers, I was able to look through the Descent code to see the SDL video calls that were used by Descent. I observed that few were actually needed, as Descent only used SDL to set up the video driver and kept its own frame buffer. I wrote a short test program to test the functions Descent would be using without the game actually running at that time. Travis helped me find the pixel format used by Descent's frame buffer and I was able to write a little driver using bitwise operators to translate the frame buffer to what Xilinx wanted. The labs helped me get familiar with the Xilinx calls. One of the minor problems that I ended up spending too much time on was the VGA Frame Buffer IP's vertical sync. I was able to find a fix online (documented in the code) that was a modification to the hardware description which stopped the screen from moving around with each run. I was happy with the video after it was all complete.

Sound

I wrote a preliminary driver before we got the game actually working. Once that was complete, I wrote a short driver which turned on the AC'97, then rewrote and directly tied the mixer callback Descent used to the interrupts and exceptions framework provided by the XilQuake code. Mike later went and added music to the game, substantially rewriting the code.

Input

Input was one of the first subsystems written. Travis did some work to integrate some of my input code from lab in as synchronous with the polling, and then I turned it into an interrupt handler. I had no previous experience writing interrupt handlers, and it turned out I put too much code in the handler, which probably contributed to it losing key presses. Mike replaced some of my code with a FIFO queue to attempt to fix the problem, but it did not. I eventually fixed the problem by modifying something, but I am unable to say what I did that fixed it exactly because I was dealing with a bug somewhere else which, after I fixed it, fixed input. I am not quite as happy with input as the other two subsystems. Mike's testing suggested that if the audio handler were called a lot more, or there was some other load on the board, that the input would begin dropping key presses again. While in testing with the "final" code this didn't seem to happen, I could easily see it happening if we had tried to get more performance out some way or another. This system, probably more than any of the others, was a learning experience for me about how hardware/software should interact.

Summary

I really enjoyed working on this project, and it was very fortunate that Mike had OS experience and Travis hardware experience. I think that not having working tools and IP at some points was a problem. We spent a bit too long on getting a working platform to develop on, but once we did we were able to progress a good amount. The labs were very useful, but some more on

other topics would have been helpful (for us, at least, some other projects may not have needed them.)

Travis

Most of my contribution to the project was in the form of working on some initial drivers for input and sound, researching and implementing hardware and networking.

Code Study and Compilation/Runtime errors

During our initial research of the project, I looked through the source code and tried to determine how the source used graphics and where we may be able to accelerate the game. At this point, I was certain floating point would be a large portion of the computation, after tracing the code and finding many floating point functions. However, I did not take into account the amount of times each function was called which was later found through profiling. I also worked with the entire team to get the code to begin compiling with the Xilinx. After fixing a couple of bugs the code was compiling, and I left it to Mike to fix linking errors. I also spent a good deal of time attempting to trace some of the errors we were getting during runtime. Joseph and I eventually traced far enough to find that our file system was the culprit and Mike resolved those issues.

Video

I assisted Joseph in writing a video driver based on his work on Lab 1. I researched how Descent actually saved bitmap data, and the pixel (color) formats of that data, which Joseph implemented in a small test program. I also worked with Joseph to debug issues within the video driver.

Sound

While we were still using XilKernel for pthreads, I created a sound thread which would load and play a background track. There was severe sound distortion in this first attempt at a sound driver. Joseph would later replace this with an interrupt for the triggered sound effects, and Mike would use part of Joseph's infrastructure to add background tracks, since XilKernel took more system resources than expected.

Input

I wrote an initial input driver that did constant polling of the keyboard, based on some existing infrastructure. Descent was already set up in such a way to use polling, and I added Xilinx calls to read from the PS/2 address and decode the scan codes of our particular keyboard into SDLKey structs, which I passed into an SDL_Event queue. Eventually, Joseph would replace this with an interrupt, keeping some of my decoding methods.

Hardware

I did most of the hardware research and implementation for this project. I first began, as stated, with some code research. Based on this research I wrote a floating point unit which I subsequently validated and added to the Xilinx board. In retrospect this was a terrible idea, because I had not profiled the code (which was much easier than expected), and made some

bad assumptions. After I had profiled the source, I realized that most of the time the program was in a fixed point texture mapper. The code had some floating point, but the massive size of an FPU would not have been the best use of FPGA fabric over texture mapping hardware. Eventually, I wrote up some hardware for the texture mapper focusing on the delay of integer divides. However, I knew that for the speed up to be significant, we would have to parallelize. At this time there were other pressing issues like drivers and runtime errors that prevented me from exploring this further. By the time we got the game running we made the choice to work on having all the subsystems complete over hardware, since the game ran at a playable rate without any.

I learned a great lesson here about making assumptions. It's far better to check and verify these key assumptions instead of haphazardly assuming one that seems probable, but later having to completely rework the design. The hour or two it took to profile would have saved me many hours of time with the FPU.

Networking

Joseph and I did a frantic last-minute rush for networking. I decided to use Xilnet, and attempted to replace all of the Unix/Linux socket calls with Xilnet calls. This proved to be a large problem, as not all of the calls could be replaced very easily. I was forced to rewrite some of the calls by wrapping different Xilnet calls, as well as removing some sections of the Descent code, which I believed we did not need. After these changes and a few more hacks, the code compiled and we began testing. Our network code used polling instead of an interrupt, but our polling check condition was based on an interrupt condition. Eventually, we could send packets to a specified client, however, there was some sort of error in the way we were handling checking if a packet is ready to be read after received. This is as far as we got networking, which was something I was really hoping to get working before our public demo.

Reflections on the class

During the class, I've learned a lot about group dynamics, and how to integrate parts of a large project together. There were also many lessons learned in the design process about decision making, such as getting as much data about the problem as possible is preferable to a "try and see" approach. We also spent a good amount of time realizing that we were often bottlenecked with getting a working file system. After we thought it was fixed, another bug relating to that occurred, and needed to be worked on in order to get any other subsystems working. This was a good lesson in how to prioritize issues as well as how we allocated time in order to account for bugs and issues that could gate the entire team.

Overall I found the class to be very enjoyable, though it wasn't quite what I had thought we would be doing. I think I may have preferred a project which was a lot more hardware intensive, instead of mostly being a lesson in software systems engineering and debugging. I also believe that a TA who is well-versed in Xilinx would be a huge help as far as progress is concerned. We spent a lot of time getting gated on a lot of things that could have been solved quickly, such as increasing the heap size explicitly. I would have really liked to progress

further than we had, but we ended up getting the game up and running very late during the project.

Mike

I really enjoyed this course. It may not have been quite what I expected when I signed up, but it was a great ride all the same.

~~18-545~~ 18-549

One of the biggest surprises for me was that we spent so much time writing (and debugging) software. In my mind, I had this course pegged as a spiritual successor for 18-447, a course in which we'd built a CPU in Verilog.

Part of this is our group's choice in projects. I know the NES group spent plenty of time writing Verilog (and debugging it with ModelSim). Our group chose a heavily software project hoping to write some interesting hardware before the end, but when all was said and done Descent ran well enough without that hardware. Which is lucky – we probably wouldn't have had time to write it.

Xilinx

Xilinx is one of the more controversial parts of this class, especially in the comments sections of final reports.

My opinion of Xilinx after this class is more neutral, leaning slightly towards negative. On one hand, the hardware was stable and reliable. In the end, none of our problems could be traced back to faulty hardware. Even the "soft" hardware, the peripheral IP provided for the board, was pretty good, excepting of course that small framebuffer bug Joseph mentioned.

What leaves a bad taste in my mouth is the documentation for this IP. Or, rather, the *complete lack* of documentation in many cases. XilKernel was well documented, and so was xilmfs (though these calls didn't always do exactly what the documentation said they would). But none was to be found for the AC'97 controller, or for the software interface to it. This made tracking down the little distortion bugs in the audio extremely frustrating, because I didn't even know what format the audio was supposed to be in. The fact that it expects a 16-bit signed little-endian quantity was revealed entirely by trial and error.

Most of learning how to use the peripheral library comes from reverse-engineering publically available demos. Sometimes, when you're trying to do something a bit more complicated, that's not quite enough.

Coding Style

I'd also like to point out that the coding style used by the D1X project is atrocious. It's bad enough that two of the original programmers from Parallax said in an interview, "This is a great example of how not to write code!" In the intervening years, the community behind this project, while adding more functionality, only made the coding style worse. The filesystem is hidden

behind no less than three layers of abstraction: one in cfile, the next in physfsx.h, the next PhysFS itself. Sometimes the first two are skipped in favor of the third, for no apparent reason.

This isn't to mention the indefensible number of inline functions in header files. Despite what whoever decided to put them there may have thought, they did need to be modified, and this required the recompilation of the entire project when they were changed.

Dead code dating back to the original release is also present, never called, and likely never to be used in future projects.

What I Did

In the first two weeks of the course, I migrated the two workstations under our control (Horizon and Liberty) to Linux, twice –first to Gentoo, then to Ubuntu when getting the cable driver to work proved difficult. Then I fetched the external libraries required by Descent, and got them to compile.

The next few weeks were spent wallowing through Descent's code, cleaning up references in their header files to make it compile for the Xilinx toolchain. After that, I tried to tackle the filesystem while Travis and Joseph used their experience from the labs to tackle SDL.

This would eventually spiral out of control into a protracted battle that would take weeks and span several design reviews. I was extremely grateful for the debugging skills OS had forced me to learn during this time, as some of the errors I was tracking down defied explanation.

That's not to say that the entire time was spent grappling with the filesystem. After PhysFS was implemented well enough that Descent could read its own resource archives, I would slowly debug things going on in Descent's main(), seeding it with xil_printf()s to see how far it would get, fixing a small bug, and then downloading again with XMD – until it would become apparent that yet another bug was being caused by the filesystem.

I also made myself useful in other ways. Keeping one of Xilinx's SDK_projects folders under version control was painful because of how often Eclipse would arbitrarily delete entire folders. I worked around this and wrote up a handy install guide for getting the Xilinx tools to work on any recent Ubuntu install (even 64-bit).

Once the filesystem was done (correctly), I assisted in debugging the input driver and cleaned up a few things there. My major contribution was moving malloc() traffic out of the interrupt handler, making it more efficient. That's where the FIFO Joe mentioned came into play – I pulled it from the keyboard handler from my OS days and used it to send scancodes from the interrupt handler to the polling function.

My next task was getting music to work. Because of the aforementioned difficulties with even understanding what format the controller was expecting, this task extended beyond the final demo. Including the music ballooned the filesystem image, but it adds an authentic touch (even if it isn't actually being played by a MIDI synthesizer).

About the class…

Overall my experience with the class is positive. Even though it's not what I expected, I did enjoy it. I loved the amount of freedom the professors gave us. We were free to both excel and make mistakes, and our team did a good deal of both. That made the class *extremely* instructive. It was great trying to meet our own goals and deadlines, instead of trying to fulfill an arbitrary class specification. This meant that the professors wouldn't necessarily have all the answers, because we might even be doing something that had never been done before.

There were the moments of exhilaration – standouts for me include seeing the Interplay logo for the first time, seeing the first level for the first time, and having a playable framerate for the first time – and there were the moments of frustration – watching the project die in mid-printf for no apparent reason, hearing pops and clicks in music that should be in the correct format, and writes failing for no reason rank up there.

My only complaint is that I didn't get to write any hardware, but that's entirely our own fault. Other than that, I had a blast! A+++, would take again.