ECE532: Digital System Design

Audio Visualization Device

Individual Report

Gary Pong 992339144

April 2, 2007

Table of Contents

1. Introduction

This report details the contributions that I have made to the Audio Visualization project described in the Group Report. The goal of this project was to implement an audio visualization device that takes an input sound signal and outputs a visualization of that signal onto a VGA monitor at a resolution of 640x480 in 24-bit colour.

It is similar to the visualization feature found in popular audio players such as Windows Media Player and WinAmp. In the end, we successfully implemented three different visualizations that can be switched at run-time through the use of the multimedia board's push buttons.

In the course of achieving our final product, I have overcome many obstacles and learned a lot about hardware system design.

2. Project Partitioning

The project was roughly partitioned according to the blocks set out in Figure 1 of the Group Report. The different blocks came together quite nicely because of the planning we did ahead of time before we actually started implementing the individual blocks. Even when writing the project proposal, significant amounts of discussion about our design architecture have already taken place. A testament to this is the fact that our final block diagram looks almost identical to the initial system block diagram put forth in our proposal.

We also used the OPB bus, the FSL bus, and FIFOs extensively in our design. The use of these common interfaces to connect our different modules simplified the process of integrating different parts of our project. There was never any ambiguity on how an interfaced functioned. We only had to worry about the data that needed to be sent.

However, I think that one of the keys to our project's success was the incremental approach taken. No one person worked exclusively on one part. Our design methodology was for a person to add a few features and then pass on the multimedia board to the next group member. Constantly adding incremental features avoided the problem of having one big mess that resembled trying to fit pieces from different jig-saw puzzles together. Furthermore, this allowed us to perform something akin to a peer review process as we always needed to look at the newly added features before implementing our own additions.

This process also applied to debugging. When we came across the mysterious problems in the VGA and ZBT RAM modules, Jason and I took turns trying out various techniques in an attempt to fix and isolate the problem. When one of us ran out of ideas to try, we would pass on the multimedia board to the other. It was through this cooperation that we

were able to exhaustively test and narrow down the cause of the problems to some timing issues.

Areas that were exclusively in my domain include integrating the AC97 controller, FFT, and push-button components into the project. Our work-sharing strategy also meant that I contributed to the development of the VGA controller, the integration of ZBT RAMs into the system, and the coding of the software component in our project.

3. Design Flow and Methodology

We used the Subversion version control system for source control. This source control system was an invaluable part of our success. As mentioned before, we added features incrementally and took turns adding features. The source control system allowed us to effectively review and track changes to the source files.

Whenever I received the multimedia board back from a fellow group member, I would make sure to review all changes made since the last time I modified the code. This allowed me to understand the system as it evolved and allowed me to review the work of my teammates. Next, I would rebuild the system from the source files and retest it to make sure that all of the correct files have been checked in by my partners. I would only start my work once I have confirmed that my current copy of the project is up to date.

Our design flow consisted of setting a milestone for the week and then breaking it down into subtasks that must be completed. For each feature, I would start by looking at how it would fit into the rest of the project. After understanding all the parts that the new feature must interact with, I would start to draw out a design diagram such as a state transition diagram or a diagram of the data path.

With the design completed, I would implement the design in Verilog and then perform basic functional simulation using ModelSim. For simple modules, I found it sufficient to use a simple do script to exercise all the states in a state machine. With more complex modules such as the VGA controller, a Verilog test bench was used for simulation. After simulation, I would compile the design and test it on the FPGA.

If the new code did not work, I would try to duplicate the observed erroneous behaviour on the simulation. Once the problem has been duplicated, I would proceed to debug it by looking at additional signals in the simulation. When the problem has been found and fixed, I would repeat my simulations to verify this and then test it on the FPGA again. I would iteratively apply this process until all the features have been implemented and tested.

For my tasks, I also like to apply an incremental approach to adding features. For example, to add the ability to take the FFT of an input audio signal, I took the following steps:

1. Take AC97 controller from a past project and test it as-is. Verify that the audio samples exhibit larger magnitudes as the input sound gets louder.
2. Modify AC97 controller to save audio samples in a FIFO and then add in a test interface to read data from FIFO.
3. Add in an empty FFT controller pcore and test that the AC97 controller pcore's audio sample FIFO can be read from the FFT controller pcore.
4. Add in Xilinx FFT core to FFT controller and add in finite state machine to control the transfer of data from audio sample FIFO to FFT core. Save the FFT results in BRAM and try to read it back via the OPB bus.
5. Add code to use printfs to textually display the FFT result. Use an input sinusoid of a known frequency and check the printed magnitudes for the tell-tale spike near the corresponding frequency bin.

I find that by proceeding with small steps at a time, problems are easier to pinpoint and solve.

4. Project Contributions

This section describes the modules that I have worked on and my responsibilities for that module. For a description of each module, please refer to the Group Report.

4.1 AC97 Controller pcore

I added the AC97 controller from the 2005 AudioToMIDI project and modified it to save audio samples into a FIFO to be accessed by the FFT block.

4.2 FFT Controller pcore

I wrote a new pcore that interfaces with the OPB bus and the AC97 controller. The OPB interface is used by the Microblaze processor when it needs to start an FFT operation and when it needs to know if the FFT operation has finished. It is also used by the processor to read the real and imaginary components of the FFT result. The AC97 interface is used to transfer audio samples to the Xilinx FFT core.

4.3 VGA Controller pcore

I modified the VGA controller from Jason's initial check-in to add support for drawing from the ZBT RAM back buffer to the screen. I also added in the OPB interface for the processor to instruct the controller to flip the back and front buffers. Lastly, I added in a FIFO that acts as a buffer to prefetch pixel data from ZBT RAM in order to hide the memory latency when pixels are drawn to the screen.

4.4 Push-button controller pcore

I modified the Xilinx-provided push-button controller and added an OPB interface so that the Microblaze processor could query the states of the individual push buttons.

4.5 Software

In the area of software, I wrote the code required to utilize the components listed above. When I first added in the FFT module, I updated the software to initialize the AC97 codec, capture 1024 audio samples, and perform the FFT on it. The resulting FFT components were then displayed onto the console using a series of printfs.

When I enhanced the VGA controller to start drawing from the ZBT RAMs, I also changed the software to draw test patterns consisting of alternating rows and columns of different colours to see if the result was as expected. With the VGA working, I replaced the FFT printfs with our first graphical visualization mode: software rendering. For each FFT bin, a column of pixels would be drawn with a height proportional to the magnitude squared of that FFT bin.

Later, when the push-button pcore was added, I updated the software to be able to dynamically switch between our three rendering modes.

4.6 ZBT RAM Controller

For the ZBT RAM controller, I took part in debugging the issues that arose when more than one RAM controller was used in our system. These problems are mentioned in the next section.

5. Problems encountered

As is expected in hardware design, we encountered many problems along the way. However, it was surprising that we had spent over half of our total project time debugging rather than in adding new features. Part of this could be attributed to our unfamiliarity with the Xilinx tools.

The first problem I encountered was relatively minor. When I tried to initialize the AC97 codec using the driver, the system would hang. When I stepped through the code, I saw that the processor would get stuck in a while loop waiting for a response from the codec chip. I reasoned that the software and HDL must have been correct since I had copied it from the AudioToMIDI project and it had worked for them. Further reading of the AC97

datasheet (http://www.national.com/pf/LM/LM4549A.html) and Multimedia Board user guide (http://www.eecg.toronto.edu/~pc/courses/edk/doc/Multimedia_UserGuide.pdf) led me to the separate reset pin that must be driven in addition to the system reset pin.

Other minor problems include our slight change in design when I discovered that the push-buttons were only accessible indirectly through the CPLD.

The bulk of our debugging time was spent on timing issues. When Jason first added the ZBT RAM controllers, he discovered that while using only one RAM chip in our system worked, adding in two RAM controllers would cause BOTH controllers to not work. When values were written to the RAM, different values would be read back from it. After he tried many different things to no avail, the multimedia board was passed to me to see if I would have any better luck with it.

We had determined early on that the problem was most likely a timing issue. It would appear that writes were being performed incorrectly while reads seemed to work. I hypothesized that the reads were semi-working because the data read from different RAM addresses were different and a read to the same RAM address always yielded the same data. This suggested to me that there was some sort of timing violation occurring on writes.

With no clear idea on how to solve this problem, what happened next was pure trial and error. I opened up Timing Analyzer tool to check the timing paths to and from the RAM pins and then I tried adding timing and I/O constraints to the .UCF file in order to improve timing. I referred to Xilinx's Constraints Guide to see which constraints would be of help to our problem. This guide is available on the Xilinx webpage at: http://toolbox.xilinx.com/docsan/xilinx8/books/docs/cgd/cgd.pdf.

Using the TIMESPEC key word, I added PADS TO FFS and FFS TO PADS constraints in a bid to reduce input and output delays to the I/O registers. Next, I tried the FAST constraint on the RAM pins to tell the tool to try and increase the speed of an IOB output. I also added the NODELAY constraint on input data pins to reduce the setup time on those pins at the expense of hold time. Lastly, I tried setting the output slew rate to fast by using the SLEW = FAST constraint. With these constraints in place, the RAM timings got slightly better. At this point, one of the RAMs would pass Jason's read/write test but the other RAM would still fail.

Unfortunately, by this time we only had a few hours until the milestone demonstration. With time short, I opted to go for the obvious quick-fix. I created a second DCM to generate a separate clock for the failing RAM module. Then, I tried using various phase shifts between the two RAM clocks until they both worked. While not very elegant, it helped us meet the milestone and bought us another week of time to fix it properly. As was consistent with our work-sharing strategy, I then passed the multimedia board back to Jason and so that he could try out additional techniques to fix the RAM issue more satisfactorily.

When I next received the multimedia board, I noticed that when drawing a colour gradient across the screen, some colours would be drawn incorrectly. Further testing showed that the exhibited problems seemed to depend on what was being drawn. Test patterns with specific colours seemed to turn out fine while on some test patterns there was a regular pattern at which columns of pixels would disappear. The most perplexing part of this was that the problems were only evident going horizontally across a row of pixels while going vertically down a row of pixels did not seem to show the same effects.

This was by far the most difficult problem encountered in our project. There were simply too many variables on what could be wrong. It could have been the VGA controller acting up or it could have been the RAM controller. The regularity of the symptoms seemed to suggest a functional problem but our simulations seemed fine and pointed to a timing problem. In the end, it happened to be a whole slew of small problems – both functional and timing – that caused the symptoms that we observed. This was extremely frustrating as we would find a small fix to the code and just when we thought we had the problem unraveled, testing on the FPGA would show a result that was slightly less incorrect, but still wrong. In the end, it was Jason that finally fixed the timing problem once and for all. Further details can be found in his individual report.

6. Tools Used

In addition to Xilinx Platform Studio and ModelSim, I also learned how to use some of the other tools included in the Xilinx ISE package. These other tools are the CORE Generator, the Timing Analyzer, and the FPGA editor.

The CORE Generator allowed me to instantiate Xilinx IP cores for our design. I instantiated blocks such the blk_mem_gen_v2_3 for dual-ported BRAM-based memories, the xfft_v3_1 for the radix-2 1024-FFT block, and the fifo_generator_v3_2 for the FIFOs.

The Timing Analyzer allowed me to see the entire critical path so that I could target my optimizations when the design failed to meet timing. It also helped me see the I/O timing delays for the RAM pins.

The FPGA editor allowed me to see how the CAD tool routed our clocks. This tool is instrumental in trying to understand what the tool did versus what we wanted to tool to do. A good example of when this helped was when we had a clock signal enter a module from the DCM and then leave the module to head for an output pin. I had assumed that the CAD tool would be able to figure out that the DCM clock signal would be able to directly drive the output pin. However, it turned out that the signal actually drove onto general routing and back out before reaching the pin. The FPGA editor allowed me to see this. Knowing what had happened, I was able to fix the problem by connecting the top level DCM clock signal directly to the output pin without first going through our module.

In the XPS package, I found the software debugger and the Xilinx Microprocessor Debugger (XMD) console to be extremely helpful in debugging many of the problems. With many of our modules on the OPB bus, I was able to perform OPB reads and writes to those modules using XMD to find out what internal states they were in. The OPB interface to the ZBT RAM controller was also instrumental in our debugging efforts on the RAM issues.

7. Community Contribution

I have not contributed to the bulletin board. However, I have helped another group who inquired about the FFT module and the AC97 module.

8. Feedback to Xilinx

The Xilinx Platform Studio tool was not able to detect changes made to the HDL files of pcores. Whenever the HDL source was modified, I needed to clean the hardware files before generating the bitstream again.

9. Course Feedback

I think the weekly milestones were very helpful in forcing us to do a little bit of work continuously. I have seen other groups doing projects such as the fourth year design project fall short of expectations due to a lack of motivation to work without incremental milestones.

I found the code from previous projects to be very helpful when trying to learn a complicated tool such as XPS. I think that examples are the best tool to learning and that the past projects have an example of all the basic modules that one would need to do for this course.