ECE532: Digital System Design

Audio Visualization Device

Group Report

Jason Wong 992353421

April 2, 2007

Table of Contents

1 Introduction

The goal of this project is to implement an audio visualization device which 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.

Our hardware implementation of audio visualization uses a Microblaze CPU for processing and a rendering module to provide hardware acceleration for drawing to the VGA backbuffers. The AC97 audio codec samples the input sound signal and its result will be processed by the Xilinx-provided Fast Fourier Transform block. Software running on the Microblaze then analyzes the frequency components of the signal and generates an appropriate visualization for it. The ZBT RAM modules are used as frame buffers to store the pixels to be drawn to the monitor by the VGA controller. Please refer to the Group Report for more detailed discussion about the project design.

This report attempts a summary of my tasks, the problems encountered, and how they were solved in this project.

2 Tasks

The project was divided up into separate modules (see Group report), but each module was not really assigned to just one person to do. Instead, most modules were developed in an incremental manner, and each incremental step was mostly done by one person. Each week, the board circulated around to a different member of the group and that person would complete his/her incremental task on the board.

The tasks I worked on can be grouped into four categories: the VGA controller, ZBT RAM controller, render module parameterization, and software. None of those tasks were solely completed by me. The parts that I did work on are documented in the following sections.

2.1 Design Flow

Since each module was worked on by multiple people on the team, a crucial element to ensure orderly execution was the use of a source control system. The Subversion source control system was used to coordinate all of our source files, which included everything from our HDL code to our reports.

It was also important for us to have simulations for each of our components so that individual component testing can be performed. We also have test benches that integrate a number of components so that their combined behavior can be examined. More

detailed discussions about the different simulations used to ensure correctness will be presented later.

In general, the routine I followed for HDL development is as follows. First, HDL code is first written or modified in an incremental manner. This is followed by functional simulation on the one module. If the module is heavily dependent on another module, an integrated functional simulation is performed. Finally, after all simulations pass, a bit stream is generated with the HDL code and tested on the development board.

The changes may pass all functional simulations but not work on the development board. In such a scenario, I first checked the simulations and think about any possible cases not covered by the test vectors. Failing to find such a case, I proceeded as if the problem is timing related, which was solved on a case-by-case basis.

2.2 VGA Controller

I developed the basic VGA controller module. The basic controller only outputted hard-coded pixel colours instead of interfacing with the frame buffers. This basic controller was tested on-chip and verified by checking the output on a VGA monitor.

In the initial stages of the project, the VGA controller was modified by Gary to only use 16 colours. This was done because the original ZBT RAM controller took a large number of cycles to perform a single operation. After the ZBT RAM controller was improved to support overlapping operations and burst operations (see section 2.3), I modified the VGA controller to take advantage of those features. The new RAM controller greatly enhanced memory bandwidth utilization, allowing full 24-bit colour to be used by the VGA controller.

The increased complexity of the VGA controller after those modifications required a more thorough simulation to verify correctness. A test bench was created that connected the VGA controller with two ZBT RAM controllers (for front and back buffers) which then connected to two instances of a behavioral model of the ZBT RAM chip. Each word in the RAM was initialized to a different value and then the simulated pixel outputs of the VGA controller was checked by the test bench.

After the module was verified to be functionally correct, the on-chip testing revealed some anomalies in the pixel output: every 10^{th} or so pixel was black. This was a perplexing problem because it did not change over time (for example, no flickering), which is not a typical attribute of a timing problem. In the end, the problem was related timing and specifically to the way the VGA clock was being driven off-chip.

In the original design, the VGA DCM output was used to drive the internal VGA logic and the VGA clock output pin. XST had actually issued a warning that there may be excessive skew on the clock, but the warning was not seen until this problem appeared. It was suspected that using a clock that drives internal logic to also drive an output pin

(non-dedicated clock output pin) led to a clock that was routed using normal net routing instead of clock net routing. Depending on the location of the output pin in relation to the location of the logic, this setup could lead to the clock becoming unaligned with the data. Given the periodic nature of the problem, what probably happened was that every 10 or so pixels the clock edge was in a position such that a timing violation occurred at the monitor interface, hence the pixel colour was missed.

A solution was tried where one DCM generated the VGA clock used for the internal logic and another DCM was used to generate the VGA clock to be driven off-chip. As it turned out, this clocking arrangement rectified the black pixels. The primary lesson to take away from this is to watch out for clocks that are not actually routed on a clock network.

2.3 ZBT RAM controller

The first version of the ZBT RAM controller I developed was just a simple OPB wrapper around the ZBT RAM controller from the Xilinx Multimedia Board Examples web page. This was functionally simulated with the behavioral model of the ZBT RAM chip and then tested in the XPS system by performing reads and writes to the memory. Even in this early stage, there were timing issues with the clocks to the RAM, but it was my other team members who investigated and resolved these issues at this stage of the project.

Nevertheless, the controller was extended to support additional memory interfaces as per our project design. During this process, the internals of the module were modularized because each of the interfaces was fundamentally the same logic replicated. The functional simulation was extended to test memory operations on each of the interfaces and also to test what happens when memory operations are requested at multiple interfaces simultaneously. The latter test was primarily a test of the arbitration logic in the controller.

As discussed in section 2.2 about the VGA controller, the RAM controller was later improved to support overlapping and bursting operations. This involved substantial modifications to the read/write state machine in the controller. The support for overlapping and bursting operations also required for new simulation test vectors, specifically memory requests which overlapped and used bursts.

After the modified controller was functionally verified, it was integrated into the system and, as discussed previously, the VGA controller was modified to take advantage of the new functionality. Unfortunately, on-chip testing revealed anomalies in the pixel output again. This time, however, the problem was attributed to the RAM controller because pixel colours were not correct when using bursts of four words but were correct when just reading one pixel per operation.

Confident that the controller was functionally correct, I presumed the problem to be timing related. On the suggestion from Gary, I checked to ensure that all RAM outputs

had a register in its output pad. I used the Xilinx FPGA Editor to look at what the ISE tools generated for the design. As it turned out, the original design of the ZBT RAM controller had logic after the last register for some of the output signals and thus some of the output pads were driven by registers elsewhere on the FPGA. This undesired placement was easily corrected by pushing the logic after the register to the register input.

Unfortunately, having all the registers in the output pads did not fix the pixel colours. Also, a fix similar to the VGA timing problem could not be replicated here because the ZBT RAM clocks were already generated by a separate DCM. However, examining the ZBT RAM clock outputs in the FPGA editor showed that the clocks seemed to be routed on normal routing nets. Also, the three clock output pins were not in the same area so the edges of the three clocks were probably not aligned. More importantly, there was no guarantee that the clock edges were aligned with the data generated by the internal logic that was sent to the ZBT RAM chips.

In what may be called a last resort, I decided to try different phase offsets in the DCM generating the output clocks. The thinking was that there might be some phase offset that aligns the clock edges with the data. This process was primarily trial-and-error and eventually a phase offset was found that made everything work! All these RAM timing problems illustrate the difficulty of interfacing with external chips and all the details that need to be accounted for to get a functioning system.

2.4 Render module

My work on the render module was limited to the parameterization of the sub-modules within it. Initially, most the sub-modules used hard-coded values for their effects. For example, the clear buffer always cleared to black and the fade buffer effect faded colours by a fixed amount. I simply added a bit of logic to read parameters from the FSL bus and then incorporated it into the effect module.

Functional simulations were performed using existing simulations, just slightly modified to include additional FSL parameters in the test vectors. On-chip testing was performed by using various parameters and visually identifying the difference.

2.5 Software

On the software side of things, I spent time developing the smooth colour gradient that can be seen in the visualization. Most of my software development time was spent on tweaking colour brightness values, fade rates, and so on to get an appealing visualization.

I also performed some refactoring near the end of the project to clean up the source code.

2.6 Conclusions

Most of the problems I encountered and consequently where most of my time was spent were related to the timing issues with the VGA display and the ZBT RAM controller. Resolving those issues further reinforced the view that timing issues are just not fun to deal with! However, a lot of time was saved because I could make an educated guess that it was a timing problem and not a functional problem.

It was definitely an excellent experience to use the ModelSim tool to verify functional correctness through test benches integrating multiple components, including behavioral models of things that I never included in simulation before. Perhaps, next time, the use of timing simulations will solve the types of problems encountered even more quickly.

3 Community Contribution

I have not contributed to the bulletin board.

4 Xilinx Tools Feedback

Overall, I found the Xilinx tools easy enough to use, although I have had prior experience with other FPGA tools. One thing that should be changed is that previous XPS compile output should be cleared when starting a new compile. It is hard to look through the output after several compiles because it requires slower scrolling to ensure that I do not accidentally go too far and look at the output of some previous compile.

In a similar vein, having a more visually pleasing output display would be a nice feature to have, similar to the output reports found in ISE. Important information such as device utilization and the timing status of the design should be extracted from the text output and displayed in an easily accessible manner.

Another unpleasant problem was the need to manually clean everything in order to get a pcore HDL change to be compiled. It would have been more convenient for XPS to clean everything automatically (or better yet, just clean what needs to be recompiled) if it detected a change to a user-supplied pcore.

5 Course Feedback

Overall I found this course to be an enjoyable experience. In this domain of digital design, the act of doing is the best path to encountering and learning the details involved.

However, many of the details, such as those involving off-chip timing, are not readily identified or understood when first encountered. The course does devote a significant

amount of time to timing analysis and various scenarios, but I think it is done rather late in the course to be of much help to the project. Perhaps an earlier introduction to some of the timing issues that may present themselves on these development boards will be helpful, although in reality I guess there are just too many things to cover in one course on this topic.

Nevertheless, there are a lot of positives to be taken from this course. The use of ModelSim as the simulator instead of the built-in simulator is a definite plus, compared to what was used in previous courses in the undergraduate curriculum. The tutorials are very helpful in introducing the tools and their various functions. And last but not least, I think the idea of a course primarily based on a full-term project gives a more complete view on project development than the traditional three- to four-week end-of-term projects in other courses.