ECE532 - Digital Hardware

Individual Project Report

Audio-to-MIDI Converter

James Shu-Hen Chen 991163904

Table of Contents

1. Introduction

Background

Musical Instrument Digital Interface (MIDI) is a standard in transmitting musical audio information in digital format. The standard is supported by most musical synthesizers, where the musical notes are synthesized and/or manipulated.

An Audio-to-MIDI (ATM) converter adds MIDI compatibility to non-MIDI instruments. It converts audio signals produced by conventional instruments into the MIDI standard, thus allowing digital manipulation of the musical notes. The audio information can be outputted to a MIDI synthesizer. Additionally, the ATM Converter can record music into compact MIDI data files.

Objective

The goal of this project is to implement an ATM converter on the Xilinx Vertex-II Multimedia Board. The board would capture audio signals and outputs, detect any event in the music, and output MIDI sequences corresponding to these events.

Please refer to the Group Project Report for detailed description about this project.

2. What I Did In This Project

Works I have done

Research: Music and MIDI Specification

I researched about the MIDI specification regarding how to represent music with MIDI sequences. I also found information on the Internet concerning the frequency of music notes.

Writing the FFT Wrapper

I wrote the first version of the FFT wrapper, including the logic and states required for loading/unloading data to/from the FSL and FFT core. The first version of the FFT wrapper was passed to my partner to be written in a style similar to the FSL Core Wrapper generated by XPS' Custom Function Wizard.

For more detail regarding the FFT wrapper, please refer to the project group report.

FFT Simulation

To fully understand how the generated FFT core functions, we decided to simulate only the generated core. I determined the appropriate test vectors to be fed into the FFT core and the expected results, while my partner worked on writing the simulation scripts. After both of us worked together to debug the simulation scripts, different test vectors were considered to test the following items which we were uncertain about:

1. Is the test script correct?
   A constant non-zero offset is fed to the FFT core as inputs and we observe whether or not the output has a non-zero value at the lowest frequency component and zero elsewhere.

2. Is the FFT core accepting signed or unsigned integers?
   Assuming the FFT core accepts signed integers, inputting a sine wave zero offset. We then observe whether or not the first output of the FFT (the lowest frequency component) is zero or a large number. If it is zero, the FFT core is accepting signed integers.
3. Does the FFT core output frequency domain representation for the first Nyquist or for both the first and second Nyquist?
4. Is the FFT core actually functioning correctly?
   Various test vectors, including relatively random ones, were fed into the FFT core. I used MATLAB to generate the FFT results from the same input vectors and compared the MATLAB results with the results outputted by the FFT core.

FFT Wrapper Simulation & Verification

I devised most of test cases to simulate and verify the functionality of the FFT wrapper, including the following:

1. Data is not available at the FSL when the FFT wrapper is ready to accept inputs, and was slowly fed into the FSL (one data in a few clock cycles). This test case best mimics the situation because the audio is captured at a relatively slow rate compared to the time it takes to fetch and process the audio data.
2. Data is available at the FSL while the FFT wrapper is ready to accept inputs, and continues to exist until the last data is loaded. This situation may occur with some earlier versions of the software.

We assumed the FSL FIFO with depth greater than 1024 will never be full at the output side of the FFT wrapper. This is because the software will always have to read 1024 points of FFT data before the FFT core receives enough data points from the software to perform the next set of FFT operation.

Debugging FFT Wrapper

This process took a very significant amount of time for both me and my partner. Because FFT was an indispensable part of the system, we spent as much time as possible to make it functional.

Both of us carefully inspected each signal in the simulation results for any timing or logical errors with the wrapper. We also tried debugging the FFT core on the hardware by making the wrapper write the state information to the FSL whenever it makes a state transition. As well, we made the wrapper write the same data to FSL whenever it writes any data to the FFT core.

Software Sound Processor

I wrote the software to allow MicroBlaze processor to coordinate the various pieces of hardware, as well as process the FFT data to determine the correct MIDI sequences to be sent. For more detail regarding the software sound processor, please refer to the project group report.

Testing the Software Sound Processor

I also tested and debugged the software sound processor by writing a function that generates hard-coded FFT results. From the hard-coded FFT results, the software sound processor was able to determine the correct MIDI sequences to be sent.

Difficulties Encountered

The first major hurdle we have encountered is that the generated radix-4 FFT core was not generating expected output. For a long time we thought we are incorrectly feeding data to the FFT core, but after we discussed with another group that were also using the FFT core, we found out they have encountered similar problem. We avoided this problem by replacing the FFT core with a radix-2 version.

Originally we planned to run the software from BRAM. However, as the software grew large, it did not fit into the BRAM with the default size. This caused the XMD to display some error messages that I did not quite understand at the time when I was downloading the executable file to BRAM. When I figured out what caused the problem, I attempted to increase the size of the BRAM but for some unknown reason it still gave me the same error. The problem was avoided by running the software from ZBT memory instead of the BRAM.

Other Tools Used

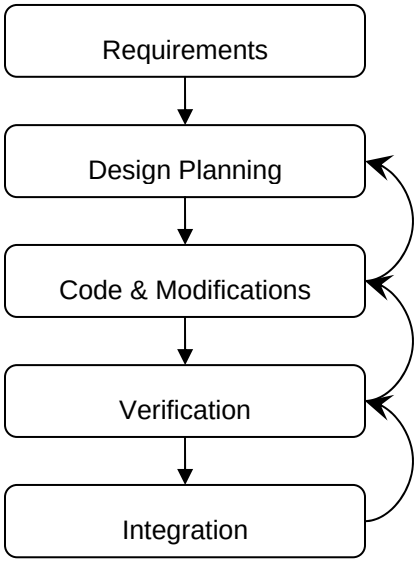
I have used MATLAB to verify the FFT results generated by the FFT core.

Project Management

Since the FFT is an critical part of our project, we tried to spend as much time as possible on getting the FFT to work. However, we still failed to come out with a fully functional version of the FFT. Please refer to the *Lessons Learned* section on how we can improve our project development process to ensure success.

Design Flow

We followed a design flow similar to the Waterfall model. The model is illustrated in the following figure:



First, we list out the requirements of the system. Then we plan our design based on the requirements. This includes defining the organization of the system, functionalities of each module, and how each module communicate with each others.

Followed by the design planning is coding in VHDL (hardware) or C (software). We then verify the developed code, isolating them from the rest of the system when possible, by simulation, testing, and/or inspection. Finally, we integrate the developed code with the rest of the system.

At many points, we have to go back to the previous step. For instance, after integration, we may need to verify the code once more. After successful verification, we would then go back to develop code for something else. Likewise, we will need to go back to fix our code if verification showed any inconsistencies with the requirements.

We may occasionally need to go back to design planning if we see that the plans are not feasible.

Source Code Control

Because the team only consists of two people, source code control was not a difficult task. We simply inform each other (usually verbally or through online instant messenger) about which portion of the code one is modifying.

Whenever we get something working, we will send the files and/or the entire project to each other and save it into the hard drive. Whenever we want to make modifications to these working copies, we will duplicate the files into another directory to avoid these files to be overwritten.

Simulation & Testing

I have performed simulation and testing on the FFT wrapper and the software sound processor. For more detailed information, please refer to the "Works I have done" section in this report.

In order to minimize the effort required to develop and test our software and hardware, we attempt to simulate or test each developed code as soon as a significant portion is completed. For instance, we simulated the FFT wrapper as soon as we developed the portion to initialize the FFT core to accept incoming data.

Lessons Learned

I have learned that there can be many unexpected events in an embedded system's development process, as these unexpected events caused significant delays in our project development.

A fatal mistake we made in this project is that we did not think of simulating the FFT wrapper with the FSL until a day before the project demo. We assumed if all the signals behaved correctly, it is going to work with FSL. However, it is quite likely that we made be some misinterpretation when we read the FSL's datasheet and that FSL actually works differently than how we think it works.

Another mistake we made was to design a project that heavily depends on the FFT, which we could not make it work in the given timeframe. Again, this is linked to the fact that we did not anticipate the unexpected events. However, we should have planned a project such that it does not depend so heavily on a function that we are not exactly sure how to implement.

Something else I spent my time on

I have recorded sounds from various instruments and performed FFT transform on these sound samples with MATLAB. Instruments that I have considered include guitar (what we originally planned to use to demonstrate our system), violin, piano, and flute. It turns out that guitar is not a very ideal instrument to use because each note it produces would have a frequency component that is almost as loud as the main frequency component. This may occasionally confuse our Audio-to-MIDI converter.

Also, I have tried to find inexpensive, off-the-shelf RS232 to MIDI converters, as well as researched about how to build a serial to MIDI converter.

3. Community Contribution

We verified some problem with the radix-4 implementation of FFT generated with a slightly outdated version of CoreGen.

I have discovered a trick to compile XPS projects on the local hard drives on the ECF PC machines. This can significantly speed up the process of generating netlist and bitstream, especially when the network usage is high. The trick goes as follows:

1. Click on Start Menu → Programs
2. Double-click on any one of the folders under Programs.
   An explorer window will pop up, showing the content of that folder. The folder should be located somewhere on the C drive.
3. Click on the “Folders” icon near the top of the explorer window.

Because the current folder is on the C drive, the C drive will show up in the tree shown on the left side of the window. Now, one can copy the project directory to the local hard drive and opens the XPS project by double-clicking the system.xmp file. When generating the netlist and/or bistream, the I/O operations are performed on the local hard drive rather than a network drive.

4. Feedback to Xilinx

There seem to be a bug with regards to the XMD window. When I type the “stop” command to stop the processor from running, the XMD window will appear to be frozen for a long time if the execution cannot be stopped. If I close the XMD window by clicking on the “x” button at the top right corner while the XMD window appears to be frozen, the entire computer occasionally will restart itself. I have encountered this situation in the design centre, the ECF PC lab, and at the microprocessor lab in Bahen Centre.

Also, it seems like the compilation process will go into an infinite loop (I waited for more than an hour and it did not terminate) after I modify a .vhd file and try to generate the bitstream. At once, this even occurred when I clean the project before clicking the “generate bitstream” menu item. The compilation process goes from generating netlist to generating bistream, back to generating netlist, and so on.

5. Course Feedback

It would be helpful if we have had more guidance on how to use the tools. It is, of course, difficult to do so because these tools are relatively new. However, most of the time we have spent on this project is on trying to figure out how to do something that could be very simple if someone has taught us how to.

As a personal opinion, I think it could be beneficial to have a project proposal draft that does not count for marks due in the first two weeks of the term. This draft would consider two or more projects, and students can get feedback on how feasible the projects are.

Also, I think most of the materials covered during the first half of the term can be covered within a slightly shorter time so that we can spend more time on the other things, or perhaps squeeze in a few lectures on how to use the various tools we may utilize when developing our project.

Having to start the project earlier would definitely help spread out the workload, but it could also mean that we would have to propose our project at an earlier time when we do not have much idea about how the tools work and how to go about implementing the project. Having more guidance in selecting the project may be necessary if the project is to be started earlier.

I am happy with the grading structure, as well as the “open” lab concept. If we were to be graded every week, we may be forced to spend a vast amount of time in some weeks when we encounter unexpected problems. I believe the “open” lab concept can benefit most students in planning their time (instead of having unexpected problems messing up their schedule).

I also find the weekly progress reports helpful in terms of helping us to keep our project on schedule.