**ECE532 - Digital Hardware**
**Individual Project Report**

# Audio-to-MIDI Converter

Sang-Joon Lee 990908354

Table of Content

# 1 Introduction

## 1.1 Objective

The initial objective of this project is to implement an Audio-to-MIDI (ATM) converter on the Xilinx Vertex-II Multimedia Board. The board would sample audio (music) signals as an input, and outputs MIDI sequences corresponding to the music in real time.

## 1.2 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 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.

# 2 Work Breakdown Structure

## 2.1 How was the project partitioned?

The Audio-to-MIDI converter project can be partitioned into two major components: the implementation of FFT core with FSL bus and software processor that analyzes audio signals in real-time for MIDI conversion.

The implementation of FFT involves generating the FFT core and importing it to Xilinx Platform Studio (XPS) and developing a wrapper for FFT to correctly communicate with FSL bus. The software processor involves analyzing result from FFT data to extract information on current music note being played.

My responsibility for this project was to implement FFT core in XPS and writing wrapper for the FFT core to communicate with FSL bus.

Due to lack of knowledge of the tool and unfamiliarity with design steps, one member's work was always reviewed by the other member for correctness. This method allowed both group members to be familiar with other member's system, and at the same time, it ensured that the project is well integrated with each other's work.

The following table lists a general work breakdown structure of this project:

**Research and Development Phase**

| | *Task Description* | *Responsibility* |
|---|---|---|
| 1 | Research Audio to MIDI conversion method<br>- MIDI specification and audio to MIDI conversion method<br>- Algorithm to detect and extract behaviour of current music note being played | James, Sang-Joon |
| 2 | Capture sound into ZBT memory and playback using AC97 Controller | Sang-Joon, James |
| 3 | Research on FFT implementation<br>- Number of points required for audio signal processing. | Sang-Joon James |
| 4 | Research on how FSL works | Sang-Joon , James |
| 5 | Generate FFT core and import to XPS<br>- Learn to generate Coregen tool under ISE environment<br>- Learn to import core into XPS.<br>- Write .mpd .bbd and .pao files | Sang-Joon |
| 6 | FFT Wrapper version 1 | James |
| 7 | FFT Wrapper version 2 | Sang-Joon |

|  |  |  |
|---|---|---|
|  | - Rewrote FFT wrapper with automated FSL wrapper generation tool provide in XPS.<br>- Correct FSM |  |
| 8 | Generate Netlist and Bitstream | Sang-Joon |

**Simulation and Verification Phase**

| *Task* | *Description* |
|---|---|
| Hardware IP: FFT Wrapper | - Setup testbench using ModelSim 6.0<br>- Create .do file and setup simulation parameters for each test cases.<br><br>Test case 1: single test input (PASS)<br>Test case 2: multiple test input (PASS)<br>Test case 3: low-frequency input (PASS)<br>Test case 4: 27 MHz test case (PASS)<br><br>- Simulate and debug VHDL code (FSM) using the test bench |
| Software IP: Sound Processor | - Perform software review and verify functionality of code by hard-coded input data and comparing with expected output. |

**Integration Phase**

| *Task* | *Description* |
|---|---|
| Hardware IP: FFT Wrapper | Use AC97 Controller to capture sound signals and push data into FSL FIFO to perform FFT on sound signals |
| Software IP: Sound Processor | - Integrate with Hardware FFT<br>- Write to FSL FIFO.<br>- Read from FSL FIFO. |

## 2.2 Problems Encountered

During the development stage of this project, we ran into numerous major as well as minor problems. The problems were mostly due to lack of knowledge on how the tool works. Also, there were some cases where problem was cause by wrong assumptions made during development and simulation phase.

The following sections describe major problems I have encountered during the development phase of our project in detail.

### 2.2.1 Incorrect FFT Core Problem

Problem: A FFT core with 512 point was generated with radix-4 architecture option. However, simulations showed that FFT core does not produce correct result. This has been verified with Matlab result as well as similar result from other group using the FFT core.

Solution: To avoid this problem, another FFT core was generated using radix-2 architecture option. Simulations show that this FFT functions correctly. This algorithm is slower than radix-4 implementation, however, it was decided that it is sufficient for this project.

### 2.2.2 Generating Netlist

Problem: To import generated FFT core into XPS, three data files, .mpd, .bbd and .pao file were created. However was not able to generated Netlist

What I tried:

- Tried to mimic a version of .mpd , .bbd and .pao file from other pcore directory.. however, this was not
- Tried to create a new set of files by referencing FSL example posted on the website. however was not able to generate FSL

Solution:

- Match the folder name and wrapper name, rather than matching generated core name to the folder name.
- Include .ngc netlist folder for FFT.

### 2.2.3 GENERATING BITSTREAM

Problem: Was not able to generate bitstream. This is because the Xilinx tool did not copy .ngc and .edn file to implementation folder properly for FFT cores. Therefore, an error was generated during ngdbuild phase.

Solution:
To avoid this problem .ngc and .edn file was manually copied to <root>/implementation folder before bitstream was generated. A batch file was also created to automate this process.

### 2.2.4 FFT SIMULATION TEST BENCH

Description: The FFT wrapper and core function correctly during simulation, however when FFT core is downloaded to hardware, FFT core does not produce correct result.

Problem: A potential cause of this problem is due to incorrect test bench. This is caused because the simulation test bench does not take the behaviour of FSL into account. We made an assumption that the FSL bus behave as shown in the Xilinx data sheet. Therefore, our simulation only included FFT wrapper communicating with FFT core and input directly feed into FFT wrapper. This test bench does not test FSL bus and its timing into analysis which results into incorrect result during the integration stage.

Solution: Included the FSL bus into testbench. However, we were not able to test this correction in hardware due to timing constraints.

### 2.2.5 FSL Automated Wrapper generator

Problem : When a function provide by Xilinx XPS to automatically generate a template for FSL wrapper. However, when the FSL bus is deleted from the project, the hardware specification files does not delete the external connection parameters from the .mhs file. This generates error when generating netlist and bitstream.

Solution: Manually delete the lines from hardware specification file (.mhs).

### 2.2.6 MODELSIM 5.8 Compilation Issue

Problem: Older versions of ModelSim (ModelSim 5.8) does not allow compilation due to missing library files. It does not allow compilation nor simulation of any HDL code.

Solution: This problem was resolved by using ModelSim version 6.0 for compilation and simulation.

## 2.3 Other tools used

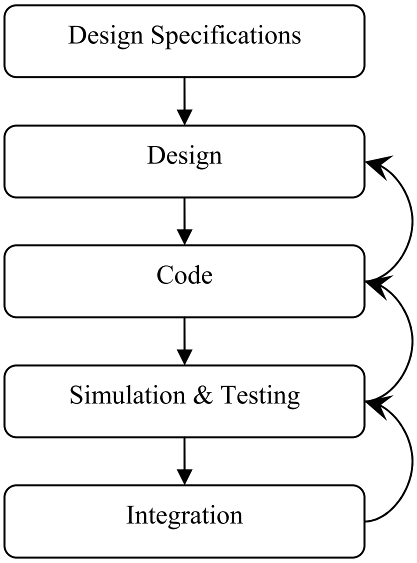
To verify the functionality of the FFT core, Matlab software simulation was used to compare the output results.

Also, to creating a test bench, an alternative method to ModelSim was considered. A test bench had been setup under ISE environment, however, it was difficult and more time consuming to modify test vectors in ISE than ModelSim. Therefore, ModelSim was chosen as test bench environment.

## 2.4 Design Methodology

### Design Flow

The following figure illustrates design flow that we took to developing this



First, the design specification and requirements of the system was determined. The system was divided into software and hardware modules. The functionality of each module and the interface and how module communicates with each other were determined.

After each module was design, each of the modules were tested by other team member. Also, to ensure correctness and due to limit of knowledge of the tool, a peer review was performed on each other's work at each stage of the design flow.

## Sources Code Control Method

Whenever we get something working, we sent 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. These working version were backup in a zipped file with a set of naming convention to note date and time of the latest version of the project.
Also, any medication made to source code were comment with initial of person who has modified the code. This comment has been removed from final version of the project.

## Simulation and Testing

The simulation and test was performed separately for two main component of this project: implementation of FFT in hardware and sound processor in software.

The implementation of hardware IP, FFT, was simulated as a module using a testbench under ModelSim environment. A set of test vectors were predetermined to ensure full coverage of test. Also, to verify the simulation results, the outputs of the FFT result were compared with outputs from Matlab simulation.

The software IP, sound processor, was tested using a dummy FFT data inputs. The outputs were compared with known expected outputs to verify the functionality of the sound processor.

## 2.5 What did you learn?

Through this project, I learnt:

- how to generate a core and import it into XPS
- design steps to generating bitstream setting up folder structures and writing files to import user pheriperal
- how to debug using GBD
- how to setup test bench using ISE and ModelSim
- how to write test script using
- how to wrap a core to interface with FFT

I learnt that a working simulation does not ensure that hardware implementation works as expected. During the integration stage, we have encountered various problems. Also, I assume that FSL core functions correctly as shown in the data sheet. This assumption lead to

## 2.6 Anything else you spent your time on (related to the project)?

In the initial stage of the project, we spent most of time reading FFT core data sheet to fully understand how FFT works. Also, when we realized that FFT should be implemented with FSL bus due to bandwidth limitation problems, we had to research and fully understand behaviour of FSL bus instance before moving on the design stage.

Also, during the design specification stage of the project, we spent significant time on researching MIDI specification to develop method to implement audio to MIDI converter in hardware. Initially, our goal of the project involved connecting a MIDI compatible musical instrument to Multimedia board. However, this implementation involved building a custom hardware that converts serial port to MIDI port. Due to time constraints as well as technical difficulties, we concluded that this idea was not feasible. Therefore, another portion of our researching time was spent on finding alternative solution to this problem. A number of solutions were proposed by each of our member. We chose that implementing on CONPORT using serial port was the easiest option.

## 3 Community Contribution

We have identified a problem with the radix-4 implementation of FFT generated with an outdated version of Coregen. This problem was found after another group has reported on the bulletin board. This problem was verified comparing the FFT result with a known set of expected outputs.

## 4 Course Feedback

I felt that it will be helpful if you had more guidance on how the tools work and what are the debugging steps to take, perhaps tutorial on different approach to debug. I felt that I spent most of the time trying to figure out how to do something even for a simple implementation. For every hour of implementation, I spent more than five hours on researching how the tool works and what are the steps I need to take to make the tool work.

Also, I found that the GDB and XMD tools are useful to debug software modules. However, it was difficult to figure out problems with XPS tools. For example, when I was attempting to import the FFT core, it was difficult to figure out what was wrong by looking at the error message on the XPS console.

I am very satisfied with the current grading structure. However, I think having a weekly lab section for marks is a good idea as well. I think this may have forced us to get small things working with the Multimedia board and understand the tool by attacking isolated problems.