Piano Simulation
ECE241 Final Project report
Prepared by Adam Muir & Shuyi Wang
December 2, 2019

Introduction

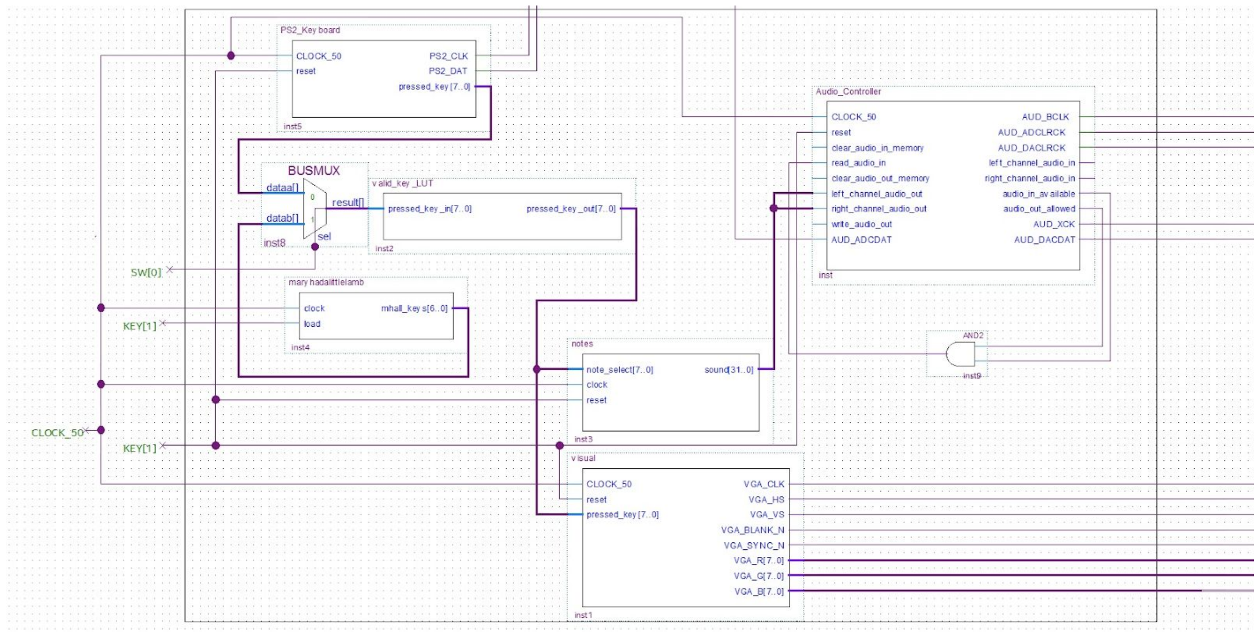
The final project for Digital Systems(ECE241) was a comprehensive and accumulative test of our knowledge, skills, and creativity applied onto hardware design with Verilog. We set our goals for the three week timeline as the following.

- Week 1: Understanding audio output for DE1-SoC board and get corresponding audio output from input switches on the board
- Week 2: Use PS2 keyboard as the primary input and output corresponding audio to the board
- Week 3: Have visual representation of the specific key being pressed on a piano keyboard via VGA display
- Extra Feature! : Preprogrammed song with VGA display for each note when played.

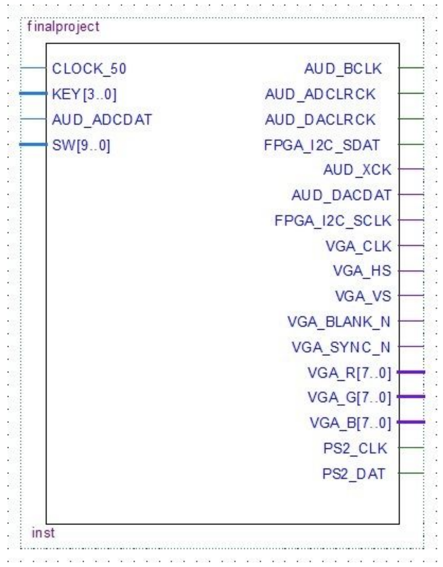
The underlying motivation for the project was to create an interactive and satisfying project for the user, something they could enjoy and learn from. The game was inspired by a real piano which ironically both of the team members cannot play. It was equally important to us to demonstrate the power of hardware to mimic and model an instrument. That is to say, when someone thinks of modelling, there is more of an association to software than hardware. Doing this project on hardware circuits seemed like an interesting idea that would also showcase the concepts taught throughout the course.

The Design

Schematic Diagram



Our top level module (finalProject.v) instantiates and connects the submodules that control the core game design. The sub modules include the FSM for VGA display (visual.v), the module for keyboard control (ps2_keyboard.v), the module to output chosen note (notes.v), the multiplexer that chooses between user controlled keyboard or the preprogrammed song (finalProject.v), and the shift register that outputs preprogrammed song (maryhadalittlelamb.v).



Top level module

The top level module instantiates and connects the submodules to each other as well as the appropriate output pins.

Notes Module

Inside the notes submodule, an 8 bit hex code is passed as an input, where it is assigned a frequency in a lookup table. To output the correct pitch, each frequency had to correspond to a counter that we calculated. The counters would be used to create square waves of different wavelengths. In total, there are 36 notes that could be played.

Keyboard controller module

A PS2 keyboard is connected to the De1-SoC board to be used as a more **intuitive?** input. When a key is pressed, the PS2 controller module reads the input and creates a matching hex code. We used a lookup table to determine if the entered key was a valid hex code used on our keyboard, wherein the signal would be passed to the appropriate modules.

Visual Module

This module contains a finite state machine that contains four states: idle, draw, pause, and erase. The idle state will wait for an input before moving to draw. Draw and erase each call ROM modules for the black key, left key, middle key, and right keys to draw the inputted key onto the display. The pause state will create a delay after drawing so that the user can see the image before it is erased.

Preprogrammed song module

The preprogrammed song module utilized a 294 bit wide shift register that shifted 7 bits at a time. The register was parallel loaded with a predetermined sequence of hex values that correspond to the notes in a song. It also uses a rate divider at 3Hz to set the length that each note is held.

The Successes

Overall, we succeeded in meeting all milestones with its corresponding deadlines and ended up with the desired result. There was nothing in particular that we failed to achieve from our goals. All the logic and physical implementation worked as designed during presentation.

However, there were many obstacles that we had to overcome along the way. From not knowing how audio was implemented on the DE1-Soc board to the final outcome, we had to self-teach ourselves the entire workings of audio in Verilog. We modified the audio demo code from the eecg.utoronto.ca website that was written for the DE2 board. Many of the ports for the pins and switches were different on the two boards that had to be modified for our uses. The audio was a crucial part of our project, specifically objective one. After multiple attempts to modify the given code, we rewrote the audio controller file that fixed the issue.

One of the biggest issues we've encountered was the VGA display. While the code was drawing the particular key, it had a hard time erasing the key. It would produce a variety of diagonal black stripes going down the key or just not erase at all. After experimenting with different ranges and counters, we figured out the best way to fix the issue was to debug via ModelSim by looking at the signals on waveforms. By doing so, we diagnosed that the problem was with the counter not incrementing at the right time. The solution was to have a case statement and a reset within the clock registers.

Future improvements

If we were given the opportunity to re-do this project, we would like to make sure to figure out to write our keyboard inputs into a Finite State Machine in the beginning. Originally our plan was to use case switch blocks to navigate switching the pitch between different keyboard inputs. Later on, as we progressed through the objectives, it had to be changed due to the VGA displays. In order to have a different visual indication of each changed pitch, a finite state machine had to be implemented. It became very time consuming to migrate all the code from case switch blocks to finite state machines. If we got the display FSM working as soon as possible, it would become much easier to implement the rest of the project on top of it. It could provide instant feedback for the program and would make debugging much easier.

On the topic of debugging, instead of using ModelSim and brute force to debug keyboard inputs, it would be much easier to write a testbench. Instead, we tested by compiling and testing it on the board every time we coded of any change that could potentially fixed the problem. ModelSim wasn't very effective for testing inputs, since the do files forced the input to begin with. Writing a testbench might have saved us a lot of time wasted compiling and board-testing the code.

We would also wanted to implement additional features such as a keyboard tutorial to beginner users if time allowed. It would serve as an nice introduction for novice keyboard users and familiarize users with how our project worked.

Conclusion

Through this project, we have demonstrated and applied our knowledge gained from the course by creating a functional application. It applies all the key concepts that were introduced in the labs, including multiplexers, lookup tables, shift registers, rate dividers, and finite state machines. In addition, this project was also an opportunity to learn and explore new concepts that we could implement to enhance the end product. These concepts include input via PS2 keyboard, VGA, and audio adapters which were different output devices. If time permits, there would’ve been more features and concepts we would’ve wished to implement and include. Given the limited timeframe of the project, we were satisfied and proud with the outcome of our final project.

Appendix

For better readability, all code can be found on the github repository.
https://github.com/shuyi-wang/ece241