Wow! Music ♫ — ECE241 Final Report

Team Members: Addisalem Semagn (1004042476), Eduardo Stecca Ortenblad (1004601549)
Teaching Assistant: Alex Rodionov
Date: December 3, 2018

1. Introduction

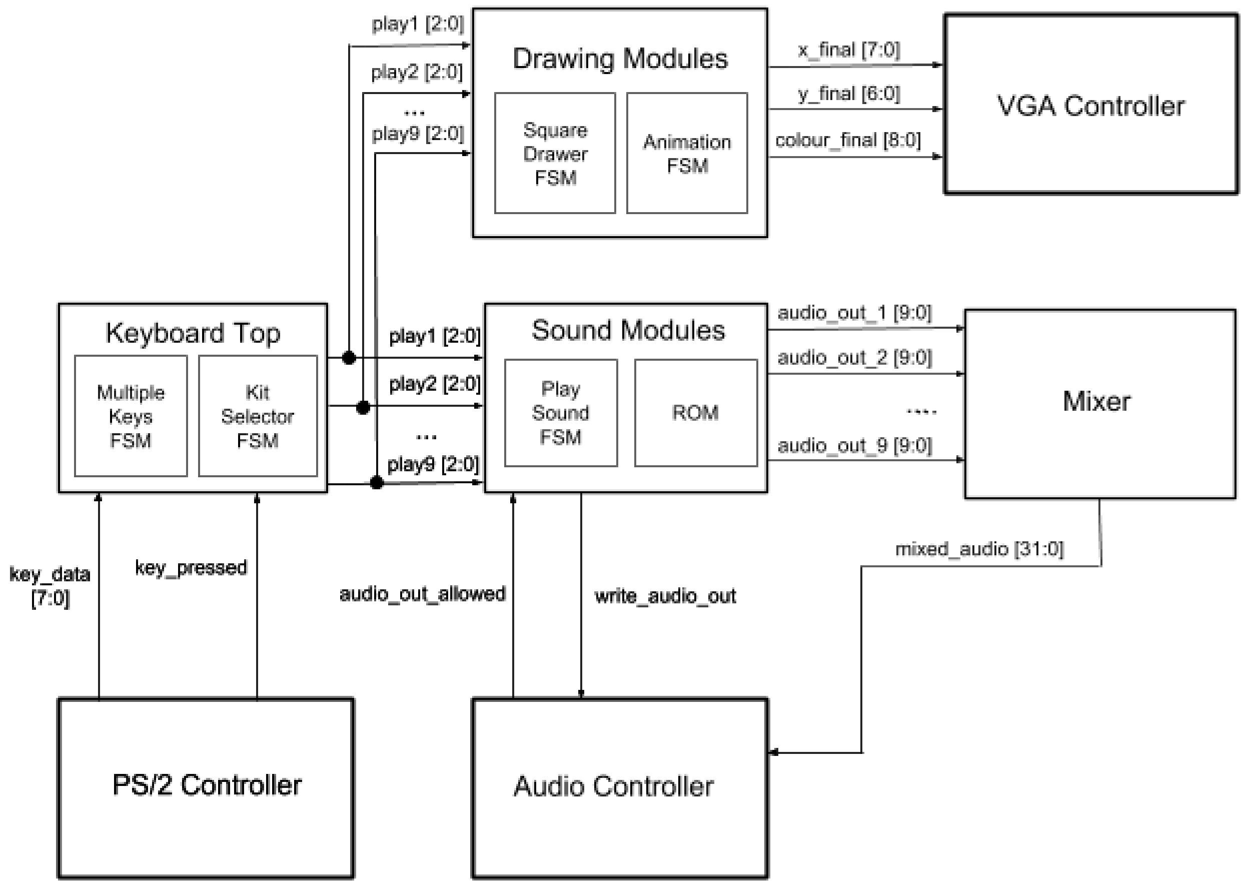
Our final project for Digital Systems (ECE241) was inspired by a mutual interest in music. Our original vision was to implement an electronic instrument that received user input from a hex keypad, played audio from sounds stored in ROMs, and a VGA display that shows the instrument that each key corresponds to. However, through improvements on each milestone, the final project used a PS/2 controller, audio controller with sound mixing, and VGA controller with improved graphics to create a fun music player with the option of two instruments: piano or drums.

Our milestones were as follows:

1. Audio controller functionality (one sound to play from one ROM, controlled by a key on the FPGA)
2. Hex keypad functionality (upgraded to the PS/2 controller) and sound mixing
3. Multiple sound kits, and any VGA interface (changed from our original idea of a simple graphic that showed which key was pressed and a picture of the corresponding instrument to having an animation that corresponded with the sound played)

Key takeaways from this project for the future were learning the difference between designing hardware and software, as well as the importance of well planned schematics and thorough simulations. The opportunity to bring together many aspects of the course content gave us a deeper understanding of each hardware component we implemented and how they communicate with each other.

2. The Design



Top Level Module
User input is received by the PS/2 Controller and interpreted by the keyboard FSM as a play signal that is sent to the display and sound modules to determine which sound should be played based on the current kit selected and which animation should begin. The display modules send a final (X, Y) coordinate to be drawn on the VGA Controller, as well as a colour. The memory modules for each sound send a 10-bit audio out signal to the mixer module. The mixer module then sends the 32-bit mixed audio to be played by the Audio Controller.

IP Cores
The VGA Controller was taken from the seventh lab assignment, and the Audio and PS/2 Controllers were from the course provided resource[1].

Keyboard Module
Two controlpaths interpreted the signals received from the PS/2 Controller. An FSM for each key allowed multiple keys to be pressed at the same time. Once the make code is received for a key, a play signal for the sound is enabled until the break code for the corresponding key is received. Another module determined which kit was selected, differentiating between each kit when the keys 1 or 2 were

[1] http://www.eecg.toronto.edu/~pc/courses/241/DE1_SoC_cores/

pressed. It outputted a one-hot code of which kit was currently selected, to select the corresponding set of sounds. (Refer to keyboard.v)

Sound Modules
Our sound modules work through the communication of a datapath, a controlpath and a ROM (one set of these per sound, 6 sounds per kit, 2 kits; a total of 12 sets). The controlpath waited in an idle state until a play signal from the PS/2 controller was received. This signal triggered a counter in the datapath that sent its output to the corresponding ROM to increment through each address, until either the user released the key or the maximum address was reached (a number that was hard coded into each of the 12 instantiations of the sound module). A slight variation of the controlpath was used for the sounds that looped, where the address reset to 0 and restarted once maximum address was reached. For each address, the 10-bit output from the ROM was sent to the mixer module. (Refer to Appendix A for schematic).

Mixer Module
This module takes the outputs of 8 sounds (a power of 2, though only 6 sounds are used), adds them, and shifts right by 3 bits to divide by 8. While the addition of negative and positive numbers is straightforward, our simulation showed that simply adding all the inputs resulted in an incorrect sum for some combinations of inputs. As a result, we add the positive numbers and the absolute value of the negative numbers separately. We then convert the negative sum back into it's signed form, and add it to the positive sum for a final sum that can be used to derive the quotient. The quotient is then padded by 20 trailing bits, to produce the final 32 bit wire that is sent to the audio controller.

Memory Modules
We created memory initialization files (.mif) files to store the 10-bit wide data for each sound and 12 corresponding read-only memory (ROM) modules.

Display Modules
The foundation of the design for the display modules was taken from Lab 7, specifically part 2 (drawing a 4 by 4 square) and part 3 (getting the square to move across the screen). A play signal from the keyboard triggers the beginning of the animation. Another signal determines if the animation should be drawn up or down, based on the sound. This controls if the Y coordinate should be incremented or decremented, and the colour of the 8 by 8 square drawn (black when drawing down, a non-black colour when drawing up). A similar module draws a 16 by 16 button that lights up when a key is pressed. A modified clock controls two frame counters, one every 4 frames (to “draw” the animation for the sound) and one every single frame (to “erase” the animation when user releases the key for a smoother look).

The VGA can only receive one (X, Y) coordinate and colour at a time. When multiple keys are pressed, each instantiation of the drawing modules sends its own coordinate and colour output. When using a single clock (CLOCK_50), each module updates it's coordinates at the same time. For a slight delay, eight clocks are used (CLOCK_50, CLOCK2_50, CLOCK3_50, CLOCK4_50 and their complements). A counter that counts from 0 to 13 acts as a pseudo 14 to 1 multiplexer, selecting a different set of coordinates and colour output to send to the VGA on each count.

3. Report on Success

Our project was a success! Not only were we able to achieve all three of our milestones, we made improvements to our design with the addition of the VGA animation and the replacement of the hex keypad for the PS/2 Controller. The final outcome was a usable, interactive electronic music player with a fun animation. All functionality worked as desired. The keyboard was able to receive all 6 inputs simultaneously and the mixer produced a sound that accurately resembled a mix of the sounds played. We had two functioning kits, one for percussion and one for piano. The only limiting factor to how many instruments we could implement was the on chip memory.

4. What would we do differently?

While the final outcome was as desired, we ran into many avoidable problems in the process as a result of hardware issues due to misinformation, disorganized code structure that resulted in debugging difficulty, and insufficient memory.

First, for our first two milestones, several days were spent debugging hardware issues (controller not working due to missing files, poor audio quality due to that were easily solved through guidance of our TA and could not have been foreseen in our simulations. We should have instead dedicated this time to implementing additional testable functionality.

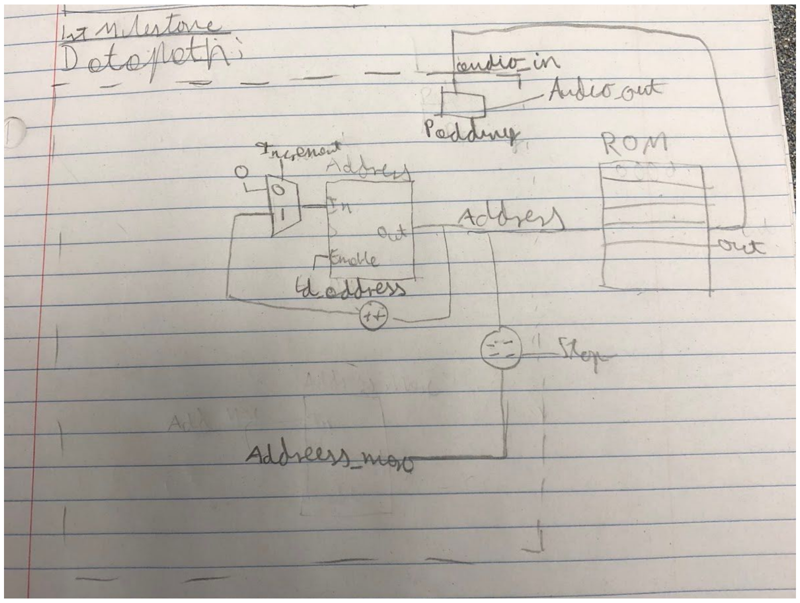
Second, there is a clear lack of readability in our code, which consequently resulted in increased debugging difficulty. Given the chance to redo this project, certain changes should be made to the code architecture. Since we implemented the modules that controlled the Audio, VGA, and PS/2 separately, combining the corresponding, already inefficient top level modules resulted in a disorganized final top level module. There is a repetition of groups of the same set of modules, which should have been made into modules of their own that could have been easily reused. This resulted in many repetitive wires, and made it more difficult to modify the functionality of any module these group of modules (adding and removing inputs and outputs) as the same change had to be reflected in each instance of the bundle. These changes to the structure would have been implemented had we had more time.

Third, while initially we heavily simulated our design to confirm theoretical functionality before testing it on the FPGA, inadequate knowledge of ModelSim resulted in simulations for the animation on the VGA that proved to be unhelpful. The alternative debugging method used was trial and error, which was an inefficient use of time and we would have changed.

Lastly, our design initially had three kits with nine sounds each. A majority of the corresponding .mif files were made before actually implementing them on the day preceding the deadline, when we realized we did not have nearly as much storage available as we needed. This could have been avoided if calculations for the amount of storage we needed had been made before creating the ROMs.

5. Appendix

Appendix A: Schematic of Sound Module


1st Milestone
Datapath:
audio_in
Audio_out
Padding
ROM
Increment
Address
In
Out
Enable
Ld_address
Address
out
+1
==
Stop
Address_max