Music Time

FPGA Piano

List of Authors (Kulin Damani, David Lewis, Chris Viray)
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
e-mails: kldamani@oakland.edu, djlewis2@oakland.edu, cviray@oakland.edu

I. Introduction

This report will cover how we will make a musical keyboard reproduce sounds with the utilization of a FPGA, Nexys4 Board. The motivation for this project is for the Final project for Llammoca's Computer Hardware class. The implications of this project is to learn more about the audio jack and the PS2 input on the Nexys4 Board.We will also be learning about how the data transfer works between the keyboard and the FPGA.

From the class, this project will cover how we can capture and generate sounds using the FPGA. Topics learned on our own will cover how we can generate a sound during a keypress from the keyboard. Applications of this project will be making music and having a more portable musical keyboard to carry around wherever you may travel.

II. Methodology

For this project we will be interfacing with a PS/2 keyboard to generate a signal that will control an output frequency for the mono audio jack on the board. The frequency will be held in a decoder file. We plan on using the full PWM range (from 0 to 255) to select and create 4 separate octaves. 2 switches on the board will be used to create a select signal for a demux. The demux will output the signal from the keyboard to one of four decoders. Each decoder will correspond to one of the four octaves of notes. Then the corresponding frequency of that note will be converted and sent to the mono audio jack included on the FPGA. The audio jack then converts the digital signal into a single channel analog audio signal which will be received and played by a speaker.

A. PS2Keyboard

The figure 1 shows the full diagram of the interfacing with keyboard that has two 1-bit input which are the ps2c and ps2d and an output of 8-bits scan codes. When the key is pressed, it will go through the ps2read to generate it from the keyboard and it will produce 10-bits, but the bits will cut-off to 8-bits output to go to the register with enable and the FSM which the control circuit. However, when the key is released, an 'FO' key-up is sent ahead of the scan codes so there is always extended key that is sent before and after you pressed the key.

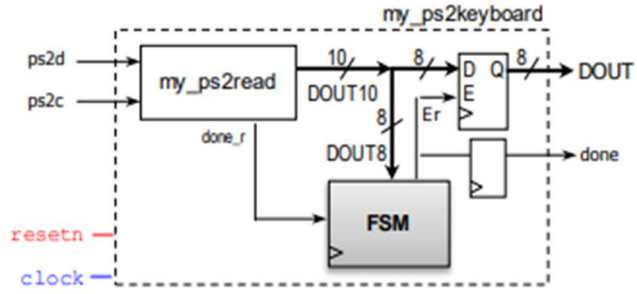

Figure1: PS2keyboard Diagram

The FSM has two state that will control the output of the ps2keyboard component in figure 2. The two input will come from the ps2read to produce an output Er which enable the register when the key is pressed. the done_r will only be '1' when the ps2read produced output of key-up code and scan codes and it will only go to state two when the dout8 is equal to 'FO'. When the key is pressed, the scan codes will trigger and generate an output of 8-bits scan codes.

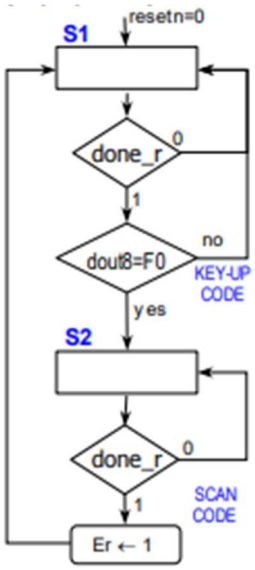

Figure2: Keyboard FSM

Each key has a hexadecimal representation or 8-bits in binary. We will only be using 7 key scan codes which are A, B, C, D, E, F, and G. Those letters will be based on the key of the piano tone so each key will have different tone to it and the other scan codes will be the pedal tone to minimize the sound.

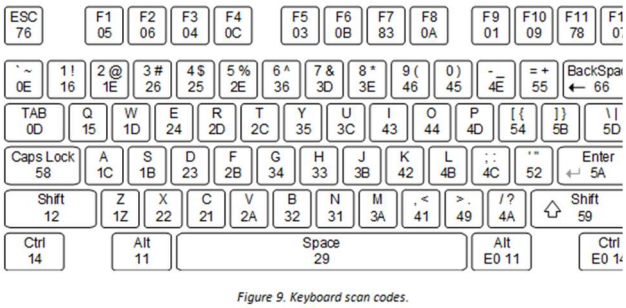

Figure3: Keyboard Scan Codes

B. Keyboard

From the keyboard scan codes we can determine which data we need to use to create a signal that comes out when pressing one of the keys, A-G. Since each letter has the same key up code, we only needed to know the HEX representation on the keyboard. This keyboard will then ultimately control how the tone changes in the hands of the user.

C. Decoder

The decoder was used to map the keyboard press to the correct frequency. The frequency was then connected to the audio files created by Dr. Llamocca to vary the frequency being outputted. Since we had multiple octaves, we used a switch to switch between the octaves, so that we are still utilizing only the seven keys. Below is Figure4 which shows one part of our decoder.

```
if sw = "01" then
case key is
   when "00110010" => frq <= 65;--b
   when "00011100" => frq <= 73;--a
   when "00110100" => frq <= 82;--g
   when "00101011" => frq <= 86;--f
   when "00100100" => frq <= 97;--e
   when "00100011" => frq <= 109;--d
   when "00100001" => frq <= 123;--c
```

Figure4: Decoder code

D. PWM Audio

The audio files were provided by Dr. Llamocca as stated previously. The circuits can be seen in Figures 5 and 6. We ran into some issues using it such as having the Duty Cycle being backwards. This means that as the frequency got higher the Duty Cycle was lower, which should not be the case. It should be as the frequency got higher the Duty Cycle is also higher. We had a simple fix and mirrored the inputs so that C, where the keyboard starts, has the higher frequency and B, where it ends, has the lowest. This is shown above in Figure 4.

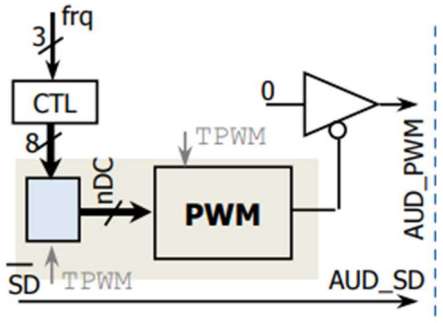

Figure5: PWM Audio Diagram

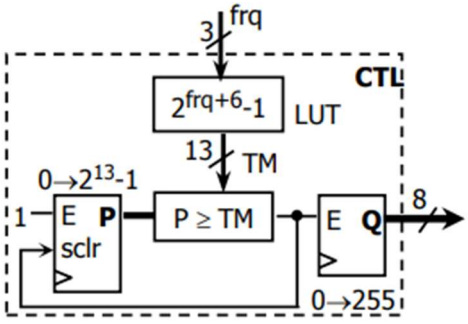

Figure6: PWM Audio CTL

Figure 7 is a full overview of our circuit and a clear look at how everything is connected together.

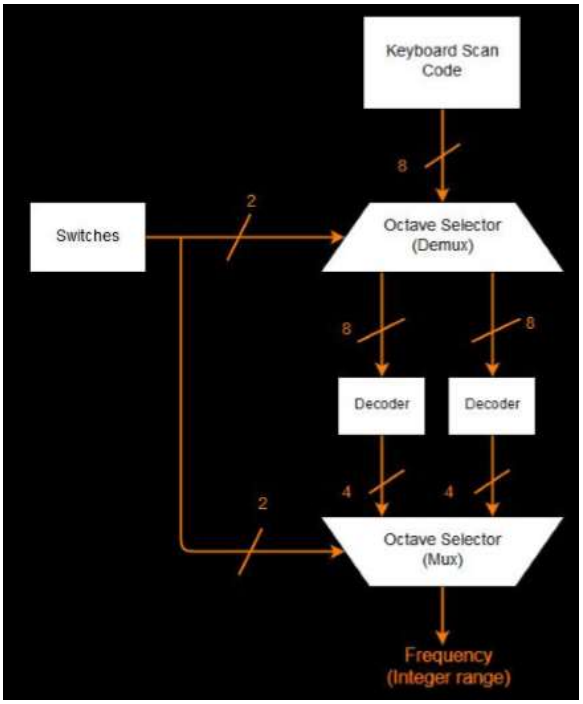

Figure 7: Circuit

III. EXPERIMENTAL SETUP

Hardware tools used in this project include a FPGA Nexys4 Board a PS2 keyboard, and a speaker. Software that we used in the project is the VHDL language which will be coded in Vivado. We verified these tools by how we can use an input to generate a specific output.

A keyboard with multiple inputs would be a great input tool for the project. The speaker will allow us to hear the frequencies of the sounds we are testing. Using the FPGA is required for the Computer Hardware class.

The results of this project were successful at reproducing sound to replicate that of a piano keyboard.

IV. RESULTS

The simulation In figure # come from the ps2keyboard component with the our testbench and it shows the input and output simulation. The ps2d and ps2c are only 1-bit input, so we tried to input the value in a certain seconds until we get the 11-bit input which include the start, scan codes, parity, and stop bit. We started the value of the key-up code as 'FO' since ps2read will produce it before of the scan codes so after 'FO' the scan code will be input such as'1C' = A. Therefore, the Dout is producing 0 in the beginning until it reach the input scan code and then it will produce the 1C which our scan code. This will extended longer until the next the scan code will generate.

We also fixed the issue where the duty cycle was backwards. It still is, although it is mapped correctly to its frequency based on its duty cycle. So now it will more accurately represent a piano keyboard.

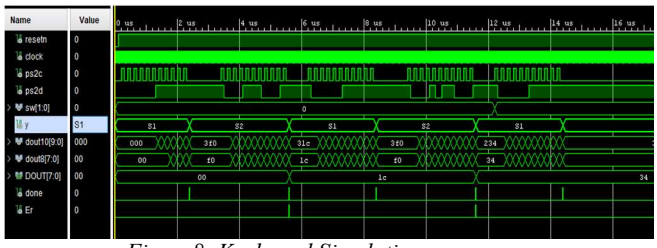

Figure8: Keyboard Simulation

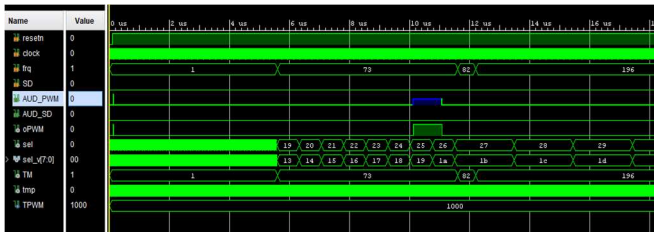

Figure9: PWM Audio Simulation

CONCLUSIONS

The overall results of this project was great. We accomplished our scope of the project over the time given to us. Using skills we learned throughout the semester and using resources provided by Dr. Llamocca, we were able to create and provide a creative and fun project. In this project, we got the chance to learn more about the PS2 interface, as well as the mono audio jack on the board. We also learned how we can connect both together to manipulate them in such a way where we can create something fun.

REFERENCES

1. Llamocca, Daniel. "VHDL Coding for FPGA's." Reconfigureable Computing Research Laboratory. N.p., n.d. Web. 23 Mar 2019
2. "Diligent" A National Instrument Company. 22 Apr 2019