Glenn Norris Jr

December 7, 2016

CMPE 3432

Dr. Lei

The Music Board

The idea of creating a keyboard that can output music notes at various frequencies like an actual piano is an idea worth pursing and in Digital Design Lab I was able to pursue this idea. Using the knowledge acquired over the span of the quarter implementing this project was time consuming and fairly difficult. The idea was to use user input from a keyboard and map it to keys around the middle “A” frequency which is around four hundred and forty hertz. The project needed very few parts and acquiring them was not a hard task and implementing them was straight forward. Creating a state diagram was fairly straight forward, but through unseen errors I had to quickly change methods and approach the project differently. This project shows understanding of using the FPGA board and the possibilities of using Verilog to create a program that acts as a circuit in hardware. Programming in Verilog has its benefits and having the FPGA to work off of made approaching this task a lot simpler than it would have in other situations.

The components needed for this project was essential to its completion, but were very simple to find and put together. The first component I needed was the FPGA board which I could use to create this makeshift piano. The keyboard was another essential part of the project being that I needed to get user input or key presses in order to map music notes to certain keys. A bread board was also essential for this project since I would ultimately need to output a signal and configure a circuit if necessary. Also driving a speaker with this signal is important if we want to hear the sound and gauge how clearly and close to what the music notes actually are in case an inspiring musician ever wants to use it. The need for other various parts like connecting wires were necessary for creating the circuit and running the signals correctly in order to get a clean sound. These components come together to help create a circuit that will ultimately produce sound for the project. The other part of the project was discerning what keys on the key board to use and what frequency of music notes available to use. Knowing that the clock on the FPGA was around one hundred megahertz I was able to mathematically deduce what values I would need to reproduce music sounds. With the components and math in hand I was then able to combine all these things in order to get a working music board.

The process of this project is fairly straight forward where the first thing that had to done was create a state diagram. The state diagram is a good visual on how to approach the problem being that one of the requirements of this project was to use a state machine in some manner. The first state machine consisted of around sixteen states that represented the various states that you could be able to go into. The way the project was initially set up a key press would correlate directly to a certain state, for example the keys a,s,d,f,q,w,e,r,j,k,l,;,u,i and o would represent the music notes and the other keys would do nothing and just be there. These keys also correlate to the states ranging from one to fifteen with zero being a silent state where nothing is happening and the sixteenth state being an output stage. So if one of these designated keys are pressed output sound, if none of the keys are pressed remain the in the silent state which is state 0. The way the project was set up the key press would move from state zero to the correlating state, once in that state it would collect information and then ultimately move to state 15 where it would output the sound. It is worth mentioning how the keyboard works in this particular situation and I will touch on it soon. The state machine that was initially initiated in this project was very chaotic and hard to follow and often throughout the process I would notice values changing as they moved through the states in a manner I did not want. Due to this very odd behavior I decided it was not beneficial to go through two to three states before output and this approach had to be abandoned for a much simpler approach that could be followed.

Coming up with a simple state diagram allowed me to create a program around such an idea. This state diagram consisted of three states which represented a silent state, a keypress state and then an output state. Where the silent state is essentially listening or waiting for a key to be pressed once the key has been pressed you transition into the key identification state. Once in the identification state the key is verified as one of the keys that have been mapped, once this is verified we can then move to output. I also incorporated a few fail safes where if a key has been pressed and it is not one of the keys we care about it the state machine would go back to the silent state. In the key verification state it is also discerned if the key has been pressed or is being held where essentially if the key is being held down output would continue until released, whereas when the key is simply pressed a sound would be output for a certain time. Also while in this transition state values about what key is being pressed are being stored and being sent to the module that creates the signal that the speaker will output. Now that understanding of how the state diagram works it is now essential to understand how the keyboard works and how it interacts with this state machine.

The keyboard is an important component of this project and understanding how the FPGA reads from the keyboard was an essential part of this project. The keyboard runs on its on clock where when a key is pressed a scan code,“F0” and leading bit is read and when a key is released a similar action happens where “F0” is read and also another scan code and a trailing bit. Understanding that this process can produce thirty three bits and how to handle this situation is essential to reading from the keyboard. Also if a key is pressed it takes roughly ten microseconds to repopulate all bits as it cycles through scan code and “F0”. Knowing this you can construct a counter that stores values on the negative edge of the signal and store all thirty three bits into a register, in my case constructing a register and shifting each bit to the right to get the scan code which would be at bit eight to bit one. Once this register has been populated you now have been successful in getting keyboard values and can move on to other parts of the program. It is also worth mentioning that in order to move from state to state you need to construct a counter of some sort that runs on the keyboards clock and not the FPGA’s clock. In my case I initially was using the FPGA’s clock and had to manually clock in order to move states which was not very efficient. The new counter uses the keyboards clock and counts thirty three times which is how long it should take for the key press to populate the register with the thirty three bits. Once thirty clock cycles has happened the counter is free to reset and move states if necessary when transitioning from state to state or key press to key press in order to output sound without manually clocking. The next step in this process is to create a module that will take the output value and discern what sound to play.

The module used in order to output the sound is known as the music module that takes in a few outputs and outputs to the speaker. This module needs values from the state machine, like if the last state was reached, the value corresponding to the state and the clock to name a few. The importance of the value corresponding to the key that was pressed is essential in order to output the sound correctly. In order to properly output the sound the module consists of a counter that takes the FPGA’s clock and using it to create a sound signal. In order to do this I had to know the frequency in hertz of each musical note which was gained from a site that describes the various frequencies these notes are at. For example I chose values around the middle “A” key which correlates to around four hundred and forty hertz, knowing this value and knowing that the FPGA clock runs at one hundred hertz you can find values that relate to the counter that needs to be constructed in order to output that sound. The counter had to be nineteen bits in order to accommodate the values ranging from 382,219 to 101,238, being that the values range so far the most significant bit we would use can range from nineteen, eighteen and seventeen. It is crucial to keep in mind there are fourteen values which correlate to a music note, which in turn relates directly which key is pressed and needing that value passed in from the main module is important. The value helps cycle through the many if else statements that will allow the program to use the correct value that relates to the key that has been pressed. Once the correct counter value has been picked we can then let the counter run its course and reset once it has reached the desired value and ultimately output the signal. Due to the signal being driven by the most significant bit we must also select the correct bit that relates to which key is pressed. As mentioned before the most significant bit can be either nineteen, eighteen or seventeen and picking the wrong one can give a totally different sound that would ruin the sounds that are being outputted. In order to select the right bit the OR logic is used where if the nineteen bit is one we output that bit. Yet if the eighteen bit is one and the nineteen bit is zero we must select that bit, but if the seventeenth bit is one and the others are zero we must select that bit. If all the bits are zero no signal will be generated which acts as a failsafe in case a key that is mapped somehow makes it that far. These components then come together in order to output sound to the speaker correctly, yet there is always room for improvement on this project.

The project outputs sounds to the speaker yet through debugging I have noticed that the sound is emitted as a step wave. A step wave is how the signal is outputted and does not sound as good as one that is outputted more as a sine wave. Improving this can greatly produce better sounding waves which are even closer to the note they are attempting to sound like. Another improvement can be made on the key press, I have noticed that certain keys have debouncing issues, if this can be taken care I can ensure the value that has been pressed and there will not be some weird value popping up in the register that stores the scan code. Similarly, if I can get the key presses to enable and sound right when a key is released I would be able to play sounds in a much tighter time period. The problem with the keyboard as it sits, there is a delay between key presses of a few seconds which is quite noticeable. If this problem can be resolved somewhere in the code it would ensure rhythms and beats can be produced and the delay would not be so noticeable. All in all this project tested the material learned this quarter in the Digital Design Lab class this quarter and other past classes. Using two components like the speaker and keyboard in order to create music was a fun, stressful and time consuming task but getting it work was well worth it. Being able to implement a simple state diagram allowed for smoother code and debugging. Using the methods learned in class debugging was a fairly straight forward task where using the FPGA board outputs like the seven segment, LED lights or the signal analyzer to gauge what is happening on the board and correcting it so that we can produce the best signal possible. The signal that was ultimately produced was rough and can be synthesized to sound better which is well within my knowledge to do so.