Keyboard Sound Synthesizer

List of Authors (Matthew Adams, Mohammed Shatit, Thomas DeSchutter, Anthony Hamm)
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
e-mails: mshatit@oakland.edu, matthewadams@oakland.edu, arhamm@oakland.edu, tmdeschutter@oakland.edu

***Abstract*—The purpose of this project is to use external peripherals to create a sound synthesizer using the Nexys A7-50T. This project is capable of playing 16 different notes corresponding to 16 keys on a keyboard in a similar orientation as a small-scale piano. The notes are played through a piezo buzzer and displayed on 7-segment displays. Notes play for a quarter of a second, unless a sustain button is asserted. Notes will continue to play until the button is let go. The findings of this project were that the Nexys A7-50T uses a scan code when interfacing with PS/2 keyboards, rather than ASCII as originally thought. In addition, it was discovered that the 100 MHz on board clock was too fast to achieve the desired note frequencies. This was addressed by using a clock divider which allowed for the lower frequencies needed for the notes. Overall, this project suggests that fpgas are capable of a wide range of applications.**

I. Introduction

This report covers the methodology and implementation of a sound synthesizer on a Nexys A7-50T using a keyboard, piezo buzzer and on board 7-segment displays. The motivation behind this project is to gain more knowledge in interfacing with external peripherals and pulse width modulation (PWM). This was accomplished by creating a keyboard sound synthesizer that acts as a small-scale piano. Many of the topics used in this design were learned in class and expanded upon. This includes interfacing with a keyboard and pulse width modulation. One specific example of this is that research needed to be done to determine that the Nexys A7-50T uses a scan code when a key on a PS/2 keyboard is pressed rather than ASCII code [2].

II. Methodology

For this project, the input device is a USB keyboard. The keyboard uses the PS/2 interface protocol, making use of the my_ps2keyboard circuit from unit 3. The 8-bit output of the my_ps2keyboard circuit is a scan code of whatever button was pressed. This 8-bit signal is sent to the LUT and 7-segment display module whenever a key is pressed. The LUT converts it to a 4-bit sel for a mux which selects between the 16 musical notes. The inputs of the mux are 16 PWM circuits with different frequencies. The output of the mux is connected to a piezo buzzer that plays the note. Notes only play for 0.25 seconds unless a push button on the board is held down. This is controlled by a finite state machine. The 7-segment display module decodes the scan code and uses a serializer to display the note on the 7-segment displays.

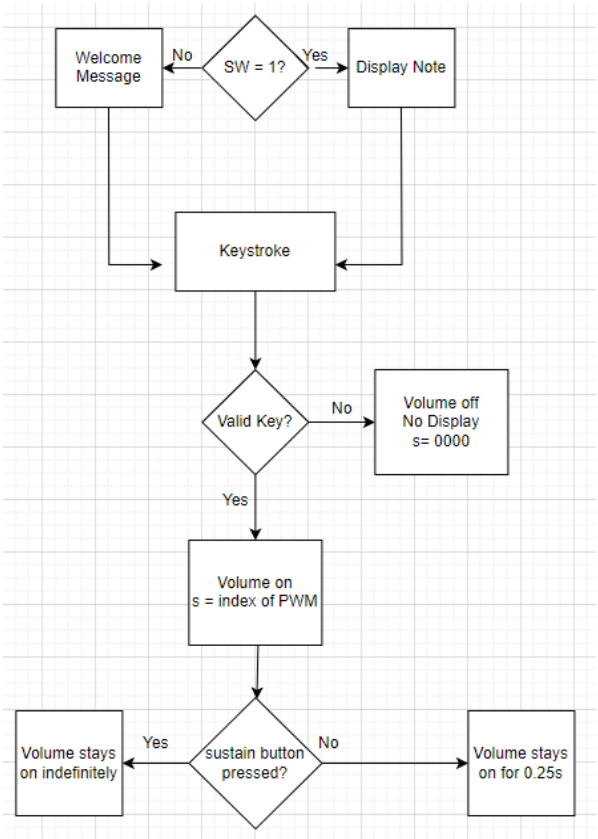

Figure 1: Sound Synthesizer Flow Chart

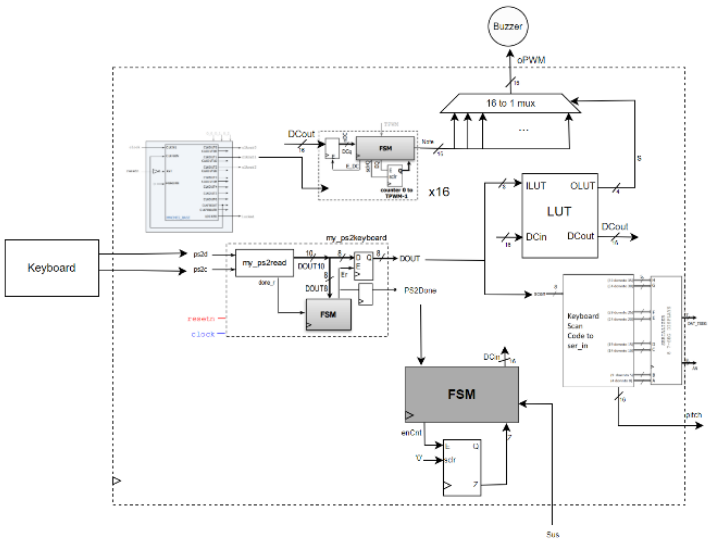

Figure 2: Sound Synthesizer Block Diagram

A. PS/2 Keyboard Interface

The code for this circuit was available on the course website [1]. The PS/2 interface transfers data synchronously. A chip, PIC24FJ128, inside the Nexys board translates the USB signal into an old-style PS/2 bus. This bus carries two signals, one for the clock and one for the data. The clock is used to synchronize the communication. The data is sent to the board in bits one after the other wrapped in between a start and a stop bit. The data will be received and processed by the board by the ps2read circuit which takes the clock and the input data bit by bit to then output a 10 bits signal. The PS/2 circuit is controlled by an FSM to account for the start bit of the data and to keep track of the count of the number of bits received. A filter is used to make sure that the PS2 clock is constant for at least 8 clock cycles. To interface with a keyboard, the ps2read circuit is used with another finite state machine to account for key holding and shifting functionalities. It is also used with a register to output the data when a 'done' signal is high.

B. Key Code to Tone Index LUT

The LUT will control the sel bits of the mux. Again, this circuit was available for use from the course website [1], but modifications were needed to accomplish our goals. One input (*ILUT*) to the LUT will be the 8-bit output of the PS2 circuit (*scancode*) which will contain the scancode of the current musical note. Based on the note a different frequency will need to be played on the piezo buzzer. So, the LUT will output the correct select bits (*OLUT*) for the mux to choose the PWM circuit that has the desired frequency. Since there are 16 different notes, the corresponding output of the LUT will be 4 bits.

The LUT also takes a second input (*DCin*), which is the duty cycle dictated by the FSM. The scancode is again compared to the 16 designated key values. If the scancode matches one of the 16 key values, then the LUT will output the unchanged input duty cycle (essentially allowing the volume set by the FSM to pass). If the scancode does not match one of the designated values, then the LUT outputs a duty cycle of zero (ensuring no note will be played by undesignated keys).

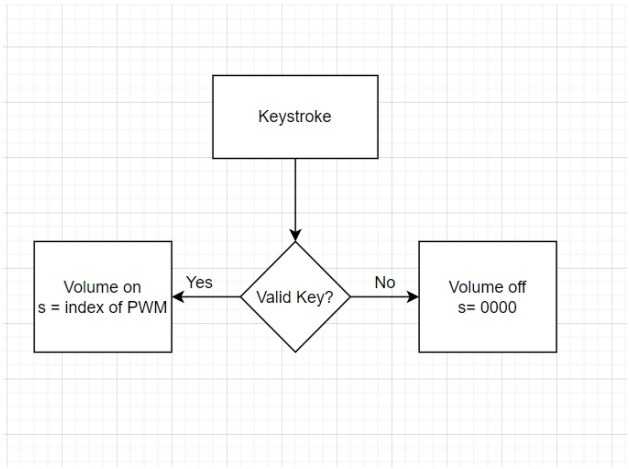

Figure 3: LUT Flow Chart

C. 7-Segment Display Module

The 7-segment display module is responsible for controlling the startup message as well as displaying each note when it is played. The code for this circuit is also available on the course website but needed to be modified to accomplish our goals [1]. When SW0 is set to 0, the startup up message is displayed. This is an 8-word message with each message displaying for one second. This is done using a 1 second counter and a FSM. The state machine has 9 states, the last state outputs nothing so there is a break before the message loops. In each state, the z output of the counter is checked to determine if the count has been reached. Once it has, a 40-bit output is sent to the 7 segment serializer. Every 5 bits of this output represent a character and is one of the inputs to the serializer.

The serializer [1] is composed of a mux, counter, fsm, 3-to-8 decoder, and a 7-segment decoder. The 7-segment decoder is a modified version of the hex to 7-segment decoder to allow for more characters to be displayed. The z output of the counter goes high every 0.001s. Whenever this occurs, the state machine outputs a 3-bit select and moves to the next state. This select is the input to a 3-to-8 decoder as well as an 8-to-1 mux. The 3-to-8 decoder controls which 7 segment display is on by controlling the AN bits. The mux selects which input or character is sent to the 7-segment decoder. Since the count is reached so fast, it appears that all 7 segment

displays are on at the same time and the desired message is displayed.

When SW0 is set to 1, the notes are displayed based on the key pressed. This is done by a decoder which takes the 8-bit scan code from the ps/2 keyboard circuit and converts it to a 40-bit output depending on the key pressed. The process for how this output is displayed is the same as with the startup message. A mux which uses SW0 as the select bit controls which of the two 40-bit outputs to send to the serializer. For all notes, they are displayed on the two right displays and if it is sharp, it displays that as well. When any other key is pressed, nothing is displayed.

D. PWM Circuit Array, Multiplexor, and Clock Divider

Each musical note corresponds to a unique frequency that needs to be sent to the buzzer pin. The method chosen to generate these different frequency signals was to utilize the PWM circuit outlined on page 12 of Unit 3 notes [3] and available on the course website [1]. Since the goal is to achieve 16 distinct musical notes, the factor that should vary is the period, which is determined by the TPWM signal. Unfortunately, the TPWM signal is not an input of the PWM circuit, but rather it is a parameter and as such, it cannot be varied once the circuit is generated during synthesis. With this in mind, a collection of 16 separate PWM circuits will be used, each with a distinct TPWM parameter set to correspond to the musical note that it is to generate. The duty cycle input of each of these PWM circuits will be set to the same value (determined by the FSM), so that the volume from each PWM circuit is uniform. This collection of PWM circuits will all output to one 16 to 1 multiplexor. The output of this multiplexor will be selected using a select input generated by a LUT, mentioned previously. A piezo buzzer will be used as the audio output. This is accomplished by wiring one pin of the buzzer to ground and the other pin to pin JA1 on the Nexys A7 board. The oPWM signal is assigned to JA1 in the XDC file.

In order to calculate the TPWM value for each individual circuit, a calculation needed to be performed. First, one problem needed to be addressed. The clock on the board is 100 MHz, and the TPWM parameter is an integer value (max value of 65,535). As shown below, the clock is much too fast to achieve the desired musical notes in this way:

$$\frac{100 \times 10^6}{65{,}535} = 1525.9\ Hz$$

This shows that providing a maximum TPWM value with a 100 MHz clock yields a high frequency of 1525.9 Hz, while our desired range of notes is about 520 Hz - 1244.6 Hz [5].

The way that this problem was solved was to use the MMCM clock divider circuit from page 19 of Unit 4 notes [4] which is also available on the course website [1]. This unit simply takes an input of the clock to be divided and outputs 3 clock signals (one identical to the input, one that is half the speed of the input, and one that is a quarter of the speed of the input). In order to have the TPWM values at roughly the same scale, both the 25 MHz output and the 50 MHz output of the clock divider were used, the former for the lower frequency notes and the latter for the higher frequency notes. Below is a sample calculation of how the TPWM parameter was calculated for the C#5 note [554 Hz]:

$$\frac{25 \times 10^6}{TPWM} = 554\ Hz$$

$$45{,}126 = TPWM$$

C_5	523.25	65.93
$C^{\#}_5/D^{b}_5$	554.37	62.23
D_5	587.33	58.74
$D^{\#}_5/E^{b}_5$	622.25	55.44
E_5	659.25	52.33
F_5	698.46	49.39
$F^{\#}_5/G^{b}_5$	739.99	46.62
G_5	783.99	44.01
$G^{\#}_5/A^{b}_5$	830.61	41.54
A_5	880.00	39.20
$A^{\#}_5/B^{b}_5$	932.33	37.00
B_5	987.77	34.93
C_6	1046.50	32.97
$C^{\#}_6/D^{b}_6$	1108.73	31.12
D_6	1174.66	29.37
$D^{\#}_6/E^{b}_6$	1244.51	27.72

Figure 4: Note Frequencies (Hertz on left column) [5]

E. Control Circuit (FSM)

The FSM consists of two states, one that mutes the volume while waiting for input and the other to keep the volume on for a certain length of time. When in state 1, the FSM mutes the volume by setting *DCin* to zero and waits for the done signal from the PS/2 circuit (*PS2Done*) before proceeding to state 2. When in state 2, the FSM turns the volume on by setting *DCin* to a constant value (20,000). This value was chosen because it results in roughly half volume for many of the notes, however since

it is a fraction of TPWM the volume will have some variation depending on the note being played. The FSM then checks if the sustain button is pressed by checking if *sus* is high or low. If the sustain button is being pressed the FSM remains in state 2, sustaining the note as long as the user decides. If the sustain button is not being pressed, then the FSM remains in state 2 for 0.25 seconds (until the counter expires and *z* goes high) before returning to state 1, resulting in a 0.25s note. This counter component is a modulo-n counter available on the course website [1].

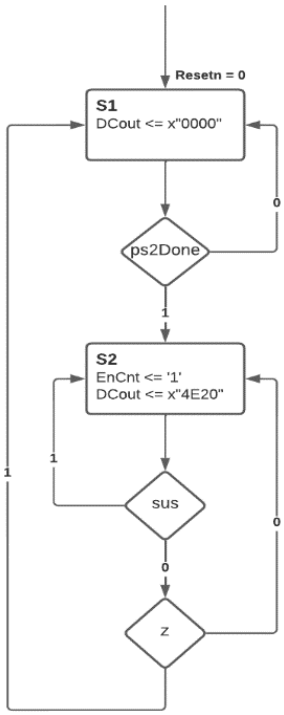

Figure 5: Main Finite State Machine

III. Experimental Setup

In order to verify the functionality of this project, a simple test bench was created. The goal of the testbench is to test two different scenarios: one in which a key is pressed and the sustain button is not pressed and one in which a key is pressed and the sustain button is pressed and held down. The expected result is that in the first case, the FSM will enter state 2 when PS2Done goes high. At this time the display should show the note being played and the FSM should remain in state 2 until the counter expires (z =1), at which point it will return to state 1. In order to observe this result in simulation, the counter needed to be scaled down significantly (0.25 seconds is much too long) and a value of 30 clock cycles was chosen for simulation purposes. The expected result in the second case is that the note played would be held if the sustain button is pressed. The counter would be ignored in this case and the note will stop playing when the sustain button is released. This can be seen in the simulation. figure 2 where sus (sustain button) is high.

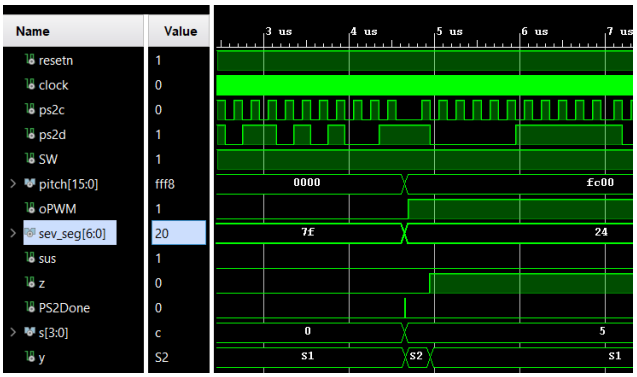

Figure 6: Case 1 Simulation Results

The image above is the simulation results of the first case from the experimental setup section. As expected, once the PS2Done signal goes high, the seven segment display changes to display the note being played and the FSM enters state 2 for 30 clock cycles before returning to state one (at which point the note is muted). Upon returning to state one, the note played remains on the seven-segment display until a new note is played.

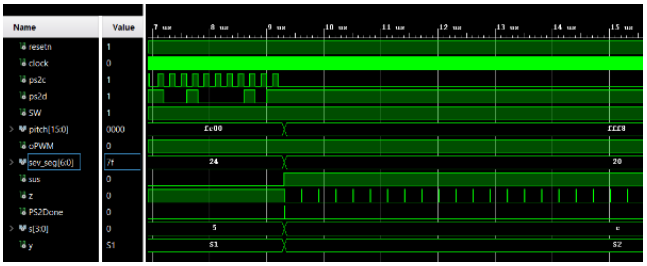

Figure 7: Case 2 Simulation Results

The image above is the simulation result of the second case from the experimental setup section. As expected, once the PS2Done signal goes high the seven segment display changes to display the note being played. However, since the sustain button is being held down, this time the FSM remains in state 2 long beyond 30 clock cycles and as a result the note is played longer.

IV. Results

After uploading the code onto the Nexys A7-50T, the program starts in its default state. This consists of no sound being played through the speaker and the 7-segment display outputting the Welcome Message on a loop. Once the switch is flipped, the 7-segment display will output the note that was last played. If there was no

note played, then the display will remain blank. The program will react to keystrokes from an attached keyboard. Once a key is pressed, the system checks if it is a valid key. If so, the volume will be turned into a set value and the selected key's tone will be played through the piezo speaker for a quarter of a second. If not, the volume will be set to zero and the display will be blank. The sustain button can be utilized to hold the note for as long as the user pleases. The note will play when the key is pressed and will stop once the key is released. The user may play any of the sixteen notes individually but not as chords.

V. Challenges & Improvements

One of the challenges that struck out during the implementation of this project was the volume control of the notes. Because our method of generating different notes is generating different frequencies from the master clock and using a constant duty cycle (DCin) for these notes, it was hard to make all notes in one volume. One way we found to overcome this, is to use a look-up table to map duty cycle values for each frequency of each note to have them in sync in terms of the volume and to also be able to raise and lower the volume of the notes as a whole, but with the limitation of the memory, this was not achievable. It can be done with the use of a different FPGA that allows more memory than the Artix-7. An improvement that could have been made to enhance the quality of the project is to use a speaker instead of a buzzer and tie that with our hardware to produce better quality sounding notes.

VI. Conclusion

In Conclusion, implementing the Keyboard Sound Synthesizer was an effective way to put our learned topics into practice and gain hands-on experience with designing and constructing a digital system with external peripherals like the keyboard, 7-segment display, and buzzer and a control unit to drive the system. The product could be expanded upon in the future to add more features such as more notes, volume control, and better-quality sounds of different instruments.

VII. References

[1] VHDL Coding for FPGAs. [Online]. Available: http://www.secs.oakland.edu/~llamocca/VHDLforFPGAs.html. [Accessed: 22-Apr-2022].

[2] A. Brown, "Nexys A7 Reference Manual," *Nexys A7 Reference Manual - Digilent Reference*. [Online]. Available: https://digilent.com/reference/programmable-logic/nexys-a7/reference-manual. [Accessed: 22-Apr-2022].

[3]Unit 3- External Peripherals: Interfacing [Online]. Available:https://moodle.oakland.edu/pluginfile.php/7446753/mod_resource/content/15/Notes%20-%20Unit%203.pdf [Accessed: 22-Apr-2022].

[4] Unit 4-Special Purpose Circuit & Techniques [Online]. Available: https://moodle.oakland.edu/pluginfile.php/7446764/mod_resource/content/14/Notes%20-%20Unit%204.pdf

[5]Physics of Music-Notes [Online]. Available: https://pages.mtu.edu/~suits/notefreqs.html [Accessed 23-Apr-2022].