FPGA Sound Synthesizer

Creating a sound synthesizer and effects bank in VHDL

Armela Gjokaj, Forrest Mason, Mehvi Mehvi, Tyler Waddell
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
e-mails: armelagjokaj@oakland.edu, forrestmason@oakland.edu, mehvi@oakland.edu, waddell@oakland.edu

***Abstract*— The purpose of this project was the implementation through VHDL of a sound synthesizer. The design philosophy utilizes a state machine, a multiplexor, counter, and pulse width modulations module to drive a buzzer through one of PMOD ports on the Artix-7 FPGA boards. The components of this final project include several essential VHDL coding techniques.**

INTRODUCTION

The report goes into further detail on the planned design architecture of the FPGA Sound Synthesizer project and how it was implemented through VHDL in the Xilinx Vivado Design Suite.

The project implements a sound synthesizer through VHDL. For our project we used an external device, which is the buzzer, that we connected through the PMOD ports. The goal of the project was to play a sound that can be heard from humans, based on 8 different frequencies. The user can use SW[0] to hold the system in its current state. Otherwise, the user can use SW[1] to play the notes (C, D, F, E, G, A, B, C). Notes ascend from C4 to C5; notes are considered natural because they have no sharps or flats.

I. METHODOLOGY

Before the system is programmed to our specification, some math is necessary to achieve desirable frequencies. Individual musical notes have a unique resonant frequency, and if we want to implement a system that plays back these notes, we need to calculate the frequency at which our component modules should operate at. Because the FPGA board used in this demonstration operated at a clock rate of 100MHz, that can be used as a reference clock in our calculation. The method used to play back the audio involved a PWM module that utilizes a count that's in reference to the period of the clock rate, so that is our desired value to calculate. By dividing the clock rate by the note frequency, we can get a number of periods that the PWM module should count to for output. This calculation is as follows

$$TPWM\ (unitless) = \frac{10^8\ (Hz)}{Note\ Frequency\ (Hz)}$$

In the sake of brevity, a table of these values has been included.

Table 1: TPWM and DC Values

Clock Rate	Note	Note Frequency	TPWM Count	DC
100MHz	C	261	383142	191571
	D	293	341297	170648
	E	329	303951	151976
	F	349	286533	143266
	G	392	255102	127551
	A	440	227273	113636
	B	493	202840	101420
	C	523	191205	95602

The DC value is then calculated as a method of volume control, implemented as the TPWM count divided by a factor of two. Because it helped in clarity later on, all values for the TPWM count in program were rounded to an even number so as to avoid decimals in resultant calculations.

A counter is used to buffer the output of the states to a point that it is discernible to the human ear. Having the states change at the rate set by the clock system would result in a blur and not produce easily recognizable notes. By implementing a counter and connecting its output as an enable function in the FSM, we can modulate the cycling from one state to another. In this case, the counter counts to 25,000,000. This number was chosen because it results in a quarter note being played at 60 beats per minute, which is a simple tempo that is easy to keep up with and gives the user/listeners ample time to recognize note changes.

VHDL language will be used in order to successfully program the Artix-7 Nexys FPGA board. The process of building this circuit consists of two primary systems that can each be represented in diagrams. The first is a state diagram that describes the desired output of the program based on user input, and the second a block diagram which uses the state

diagram as a component. As such, the state diagram will be explained in detail first.

A. State Diagram

The objective of this project was to have a user-variable frequency played through a buzzer speaker. Four frequencies within the human-audible range (20-20,000Hz) were chosen and are selected be four unique states. The state diagram is presented at top right. In this current configuration, the user selects between four potential input states and is constrained to always increment the states. That is, if the state is S1 the only next viable state is S2. If the state is S2, the only next viable state is S3. In this way, the user controls a ramping style input into the buzzer speaker until it resets back to the lower frequency value and starts again.

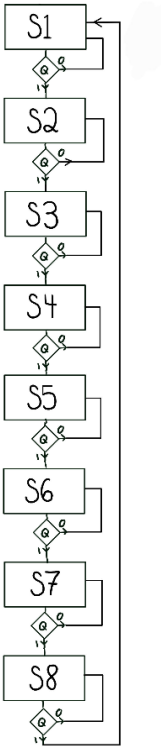

Figure 1: System State Diagram

In addition to this "main" finite state diagram, the PWM module contains its own individual diagram as well to control its outputs. This is listed below.

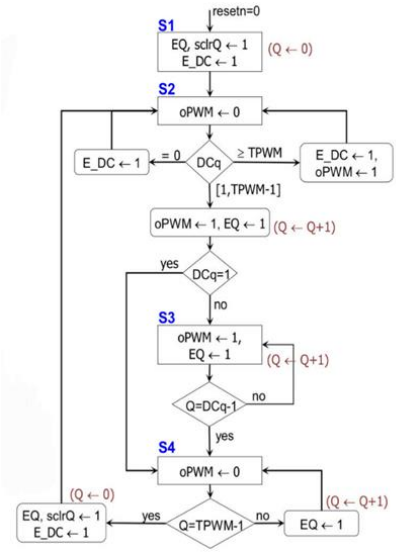

Figure 2: PWM State Diagram

B. Block Diagram

Expanding upon the previously mentioned state diagram is the block diagram that describes the overall functionality of the system.

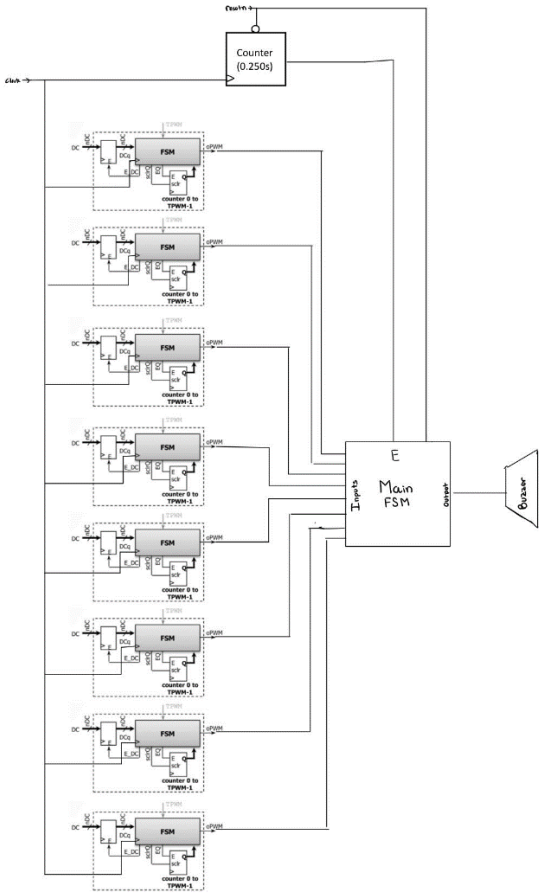

Figure 3: System Block Diagram

Eight individual PWM modules, each with their own assigned values for tPWM and DC based on the calculations shown in Table 1, feed into the Main FSM component and are internally selected. The output is then buffered by the counter.

II. Experimental Setup

As stated previously, we used the Nexys Artix A7 programmable logic board to synthesize the sound. We worked in the Vivado VHDL programming environment to build and code. We created source files for all of our main competes such as the counter, multiplexor, state machine and pulse width modulations.

III. Results

Overall, the board worked and functioned as it was desired to. By flipping the SW[0] switch from low to high the user can halt the ramp and prevent it from continuing the cycle, instead holding its current position, and playing that frequency for as long as the switch is high.

Conclusions

Because the implementation of this design is rudimentary, further improvement on the system should likely be focused on making the system more robust and user operatable. To keep with the synthesizer theme, it would be interesting to add components such that the FPGA board can be operated with added affects such as delay, vibrato, flanger, or background operations like an arpeggiator or rhythm line. Further states can be implemented to increase the versatility of sounds, going from only quarter notes to eight, sixteenth, whole or otherwise by changing what the counter counts to or increasing the number of counters in the system.

References

IV. References

[1] D. Llamocca, "VHDL Coding for FPGAs," Reconfigurable Computing Research Laboratory, [Online]. Available: http://www.secs.oakland.edu/~llamocca/VHDLforFPGAs.html. [Accessed 31 March 2022].

[2]D.Llamocca, "ECE4710: Computer Hardware Design," [Online]. Available: http://www.secs.oakland.edu/~llamocca/Winter2021_ece4710.html [Accessed 31 March 2022].

[3] D. Llamocca, Unit 3- External Peripherals: Interfacing