Musical Generator Peripheral to the SCOMP System

Victor Alfaro
William Hao
Tyler Jeng
Lining Song
Andrew Moore

Submitted
December 6th, 2022

Introduction

This document is an overview of our audio peripheral to the SCOMP System. It contains the device's functionality and modifications of the peripheral to achieve the best design for a user to create for their own applications. The assembly can also be modified to determine whether a note is a flat - as well as control the octaves, waveforms, and volume of the outputted note. My project specialization was working on the assembly code and testing the peripheral. I understood where the issues were in the code and made adjustments. The team was able to accomplish all baselines and additional functionalities presented in our proposal.

Device Functionality

Our device decodes signals provided by user-generated assembly code into notes that are playable on audio devices. We modified our peripheral to simplify creating new code for it. Users input their settings including starting octave, note length, or waveform in a section of the assembly code that executes upon resetting or starting the system. Then, the user specifies a subroutine in which they will code their song. We created subroutines that perform actions such as sending signals to the peripheral and calling delays to simplify the coding process for users.

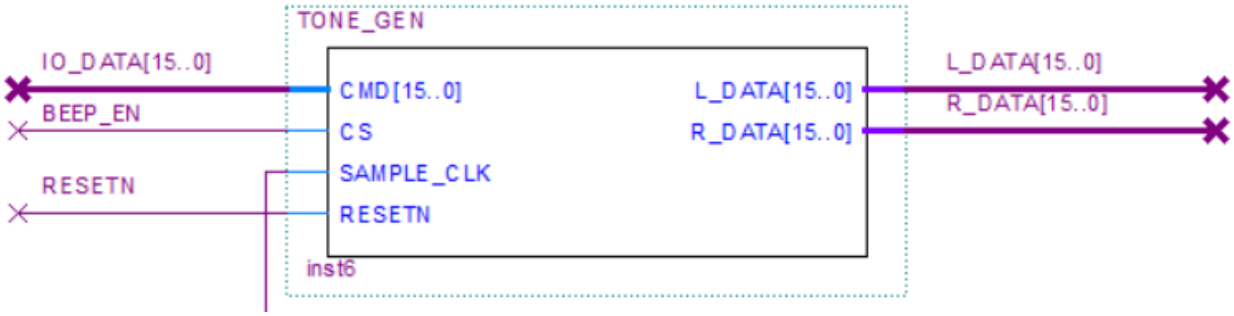

Figure 1. Schematic block diagram highlighting input and output signals for the peripheral TONE_GEN. IO_DATA is an input signal from the assembly, and the L_DATA and R_DATA output signals control the sound produced by the system.

Design Decisions and Implementations

Our design utilizes a tuning word register size of 12 bits, a phase register size of 15 bits, and a PROM containing 256 values. We obtained these values from the formula $F_0 = (M*F_{clk})/(2^N)$, where F_0 is the frequency, M is the adjusted tuning word, F_{clk} is the system's clocking frequency rate (48,000 Hz in our case) and N is the size of the phase register, in bits. Each of the distinct waveforms is implemented as a separate ROM with its own .mif file, and we decided that the base 256 value ROM was sufficient to produce an accurate sound within a 1% error of the note's frequency.

Our peripheral decodes I/O signals programmed in assembly. Starting with the most significant bit, we designated the first seven bits to play distinct notes in the second octave of the C major scale. Each of these seven options, in tandem with the flat option, corresponds to hard-coded assignments of the tuning word within the VHDL file as described in Figure 2. This design is effective because tuning words of higher notes are left-shifted tuning words of lower notes.

Notes	Frequency (Hz)		Adjusted Tuning Word	Binary Represenation
C2	65.41		44.65322667	101100
D2flat	69.3		47.3088	101111
D2	73.42		50.12138667	110010
E2flat	77.78		53.09781333	110101
E2	82.41		56.25856	111000
F2	87.31		59.60362667	111011
G2flat	92.5		63.14666667	111111
G2	98		66.90133333	1000010
A2flat	103.8261744		70.87866839	1000110
A2	110		75.09333333	1001011
B2flat	116.5409404		79.55861531	1001111
B2	123.4708		84.28939947	1010100

Figure 2. Table outlining notes in octave two with their corresponding frequencies, adjusted tuning words, and binary representations.

The other nine bits represent modifiers for each note. As shown in the table below, three bits determine whether the system plays no sound or plays octaves 2-8, two bits decrease the left and right speaker channels' volumes, and two bits choose the waveforms sine, square, sawtooth, or piano.

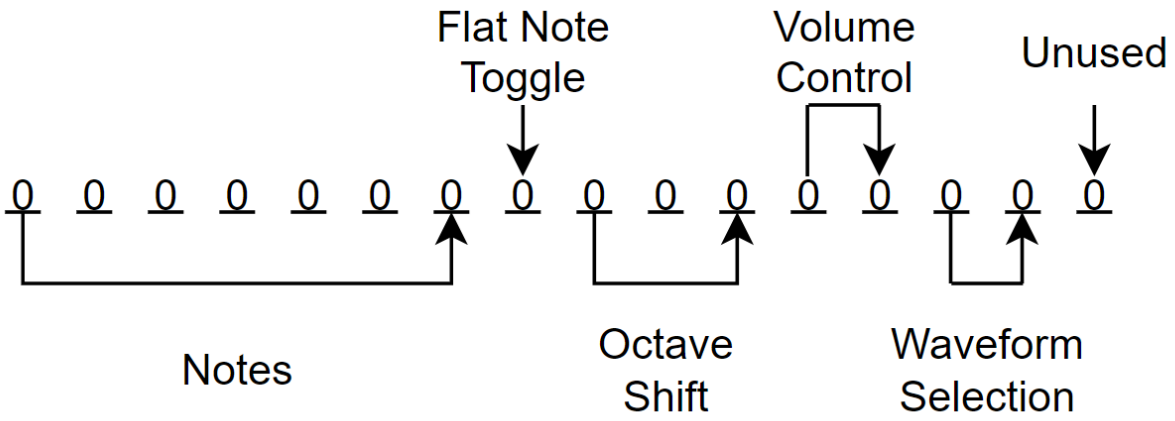

Figure 3. Diagram dissecting the input signal that is sent to the peripheral

Though the least significant bit of the signal is not used in VHDL, we utilized the bit in assembly to assert a note for a shorter duration than the default 500ms.

Conclusions

Overall, the team was able to successfully achieve all requirements from the project including the needed additional functionalities through modifying the peripheral. My recommendation to future engineers is to make use of the unused bit at the end of the assembly code. They can modify the behavior in the VHDL code to create their own additional functionalities as the assembly was coded in a one-hot state machine style that allows for ease-of-access in terms of user accessibility (when they modify the assembly or peripheral). There are comments in all blocks of code in the VHDL and Assembly that allows for better understanding of what each line does.