**Music Player Final Project Report**

**ECE 3300/L, Section 3, Prof. Mohamed Aly**

**By: Ethan Song and Meredith Toledo**

**Abstract**

This Verilog Project utilizes the Nexys A7 FPGA development board to create an interactive music player system. Three dedicated switches allow the user to select from three pre-made songs, with a corresponding numeric identifier displayed on the seven-segment display. There is also a VGA output that shows the corresponding number via a monitor. An LED light has been implemented to light up and turn off for every note, to show the duration of a note being played. A reset button is also incorporated to stop the playback and reset the song. The project utilizes the PWM Audio Amplifier in the constraints file to output audio. Songs are stored in case statements to play specific notes sequentially for a specified duration. The frequency of the notes has been calculated according to the clock speed to hit the right sound frequency when being played back. This project showcases the Nexys A7 FPGA’s potential with audio playback and provides a base foundation of being able to play sounds at specific frequencies for a period of time.

**Objectives**

Our objectives were to create some sort of way to generate sound and make a song out of the generated sounds. Furthermore, we added additional elements to the project, such as making it multiple songs and having an output that shows what song is being played.

**Methods**

The project comprises of Verilog modules that have been instantiated in a single top file. The primary code that we used is from a previous ECE 3300 lab document. We based our project off of this code and have modified it to have more music and to have different outputs, such as the LED and VGA outputs. Here are the following modules and its code:

**SongPlayer.v (top file)**

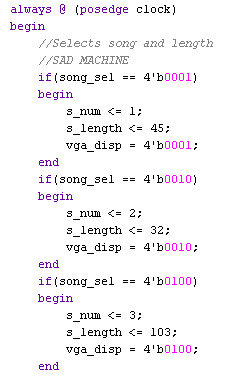
The following screenshots of code will represent the module “SongPlayer” that serves as our project’s top file. In this module is the ultimate declaration of inputs and outputs we will be using, including the clock, our reset (used by a button), playSound enable (used by a switch), and our songSel switches (to determine which track should be played. The outputs will be assigned accordingly to where they should be connected on the board via our constraints file for the Nexys A7 100T FPGA board, which will later be shown. All these variables being declared allow our final project to run and be interactive, as well as showcase all the project’s utilities. These correspond with other module instantiation such as the clock divider (clk\_div), which will allow the correct timing parameters to be applied per VGA and seven segment display, and the VGA connection (vga\_out) in order to ensure that what we want to display on the screen, does so properly.

In the loops displayed later within the SongPlayer module, we first come across the always block that contains 3 if statements. These if statements provide conditions where, if the enable for the song player is on, it will activate a certain song based on which switch is flipped. Alongside that, it will determine what number song is displayed on the monitor via VGA, and what number will display on the seven segment display, and also determine the length of the song which is specified in the music sheet. Next brings up a condition where if the enable is off, or the reset is on, no sound will display, and the default image on the monitor and default number on the seven segment will display. Otherwise, the music will play. In the condition that the music will play, we have a counter for frequency, and a counter for note duration which will increment to “parse” through a song and play it for however long we decided to make it on the MusicSheet. When the frequency counter hits the max frequency of the note and the time counter hits the max duration of the note, the counters will reset and toggle the audio being outputted, as well as the LED that serves the purpose of showcasing when a note is played and how long it is played. The very last if statement will make sure the number (representing the number in note order) will reset at the end of each song.

**A screenshot of a computer program

Description automatically generated A screenshot of a computer program

Description automatically generated**

** A screenshot of a computer program

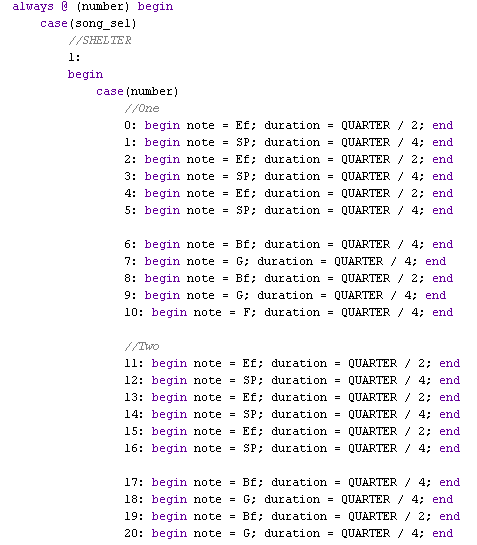
Description automatically generated**

**MusicSheet**

These proceeding screenshots will represent the MusicSheet module, which essentially carry the songs themselves to be played via the SongPlayer module. The input number will control which note is playing in the order that it is written and the song\_sel will refer to which song from the “music sheet” will play. The first 3 parameters declared set values for note timing (in reference to quarter notes, half notes, and whole notes) and the last few serve as parameters for frequency of desired notes played per song plus, one for “empty space” (SP). Within the always block will begin songs in a nested case statement, where case 1 will contain a number of cases representing the number of notes, the note itself, and the duration of the note, that will play in order, and case 2 and 3 will follow the same formatting. The parameters for each note have been calculated based on note charts that give fixed numbers of piano note frequencies, and dividing them respective to the frequency of the Nexys A7 100T FPGA board itself. These have been tested to make sure that the notes are as accurate as possible when the audio is played.

A screenshot of a computer code

Description automatically generated



**vga\_out**

This next module allows our external monitor output to show the corresponding track number based on what song we are playing on the board. In the screenshots below, firstly, there are multiple inputs, outputs, and registers declared. We have a clock input (which has a later on connection in the top file), the 4-bit input for our switches, an output for the horizontal and vertical sync signals, and 4-bit outputs for red, blue, and green colors. The registers serve as temporary values for our x and y-axis counters, and the 4-bit values for our colors. In the first 2 always blocks, we are ensuring that our x and y counters (which are later assigned to the signals themselves) fall within the parameters of “addressable video time”, or in other words, what will actually allow the values to display on the screen, with this picture as a reference.



The proceeding lines then provide cases that display images depending on which switch is activated (for a song) to print “SONG 1..2..3” and “SONG” as a default screen. The if conditions go row by row filling pixels with different colors to draw what we wanted it to display upon activation. In the end, the signals for red, blue, and green are assigned so that they only activate if the counters are within range of the video time so they can display properly, otherwise, they will not.

This is its final output:

A purple screen with white text

Description automatically generated And the number changes based on which track plays.

A screenshot of a computer program

Description automatically generated A screenshot of a computer code

Description automatically generated

A screenshot of a computer code

Description automatically generated

**clk\_div**

The module clk\_div serves as a generic clock divider respective to the Nexys A7 100T FPGA board's internal clock speed and maximum frequency, that toggles the clock within a certain parameter to allow projects on the board to work within a certain timing range. The clock toggles when the counter hits the parameter number. In our project, we instantiate the clock divider with different parameters within our top file to allow the seven segment display and external output monitor to showcase track numbers accordingly within the time frame that they normally work at.

A screenshot of a computer program

Description automatically generated

A screenshot of a computer code

Description automatically generated

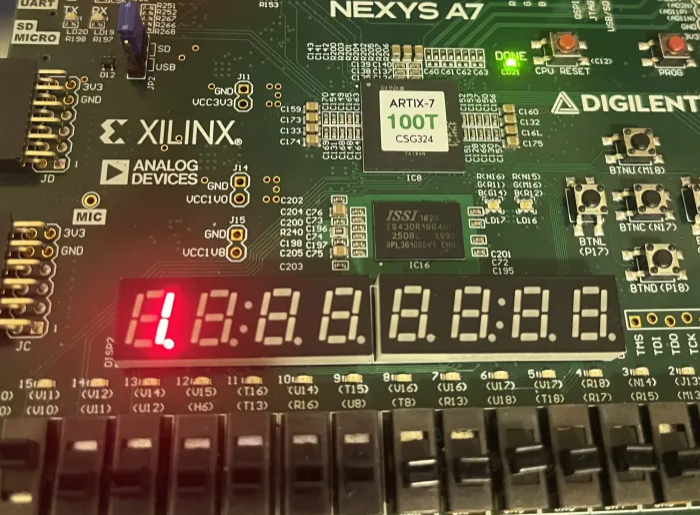
**ssd\_driver**

The ssd\_driver is a module that allows us to activate only one specific segment from the 8 given by the board, and have its internal led segments display number 0-3 corresponding with the track choice. In the module there is a case statement to choose which segments will light up based on the number being inputted. This module is also instantiated in the top file to work with everything else on the board.

A screenshot of a computer program

Description automatically generated

This is how it displays depending on switch input.



**Constraints File**

The following screenshot will show our connections in the constraint file with inputs and outputs from the top file so that the project would actually carry out on the board.

A screenshot of a computer

Description automatically generated

A close-up of a document

Description automatically generated

A close-up of a list of text

Description automatically generated

A close-up of a computer code

Description automatically generated

A screenshot of a computer code

Description automatically generated

A close up of a number

Description automatically generated

**S**

**Results**

In this section, we’ll show our final testbench, as our demo of the lab working on YouTube.

<https://youtube.com/shorts/OQdSCrJ9yLY?si=noQ9J4UFGpF_Ec_p>

**Conclusion**

In this project, we were able to create a music player based off a lab from previous semesters. We used the code as basic framework and translated sheet music to be played in the FPGA. We also achieved in modifying the code so that we have more functionality, such as more music, an indicator to the note playing, and a seven segment display and a VGA to show what song we are on.