Lab 4: Creative Project

CS M152A Lab 3
TA: Gu, Hongxiang
June 12, 2019

Nathan Maron,
Denise Wang,

Introduction

The goal of this lab is to design, implement, and demo a circuit for a project of our own choosing using the FPGA board. For this project, we have decided to create a simplified version of Guitar Hero.

Guitar Hero attempts to imitate many features of playing a real guitar. The players use a replica of a guitar as a controller to match notes that appear on the screen. The guitar controllers consist of five different colored fret buttons that correspond to the five different colors that represent notes that fall from the top of the screen. As each of these notes reach a specific line at the bottom of the screen, the player must match the notes that appears on the screen to the controller. For each note the player hits accurately, their scores will increase. After stringing a series of 10 successive notes, the score multiplier will increase and the player will get twice as many points for each point.

In our recreation of Guitar Hero, we have decided to use only four different colored fret buttons. The four buttons will be used on the FPGA board as the up, down, left, right buttons. The horizontal line at the bottom of the screen indicates the hit zone, where each player must press each button once it reaches the line. For each note the player hits correctly, they will receive _ point. After stringing a series of _ successive notes, the score multiplier will increase and the player will get twice as many points for each point.

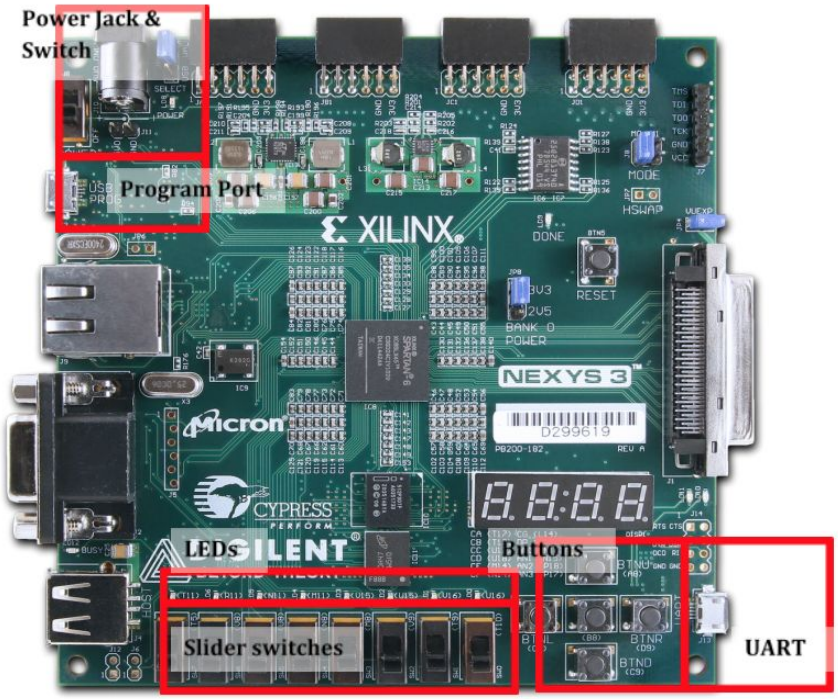

Figure 4.1- The only component used in this lab is are the buttons. The up, right, down, left buttons represent the notes that need to be pressed in the game.

Design Requirements

Our implementation of Guitar Hero is broken down into smaller components that would make the game come together as a whole. These components would make up for the grading rubric for the project.

Functionality	Percentage	Description
Hit Functionality	20%	If the player hits the corresponding note displayed on the screen on the board, their score increases and if the player misses, the score remains the same.
Streak Functionality	20%	If the player hits 10 successive notes, the player's score will double until they miss a note.
Score Functionality	20%	The player's score is based on the number of notes they hit correctly. For each note they hit, their scores will increase.
Display	20%	The display consists of four columns that will contain the notes and button needed to be pressed. It will also display the player's score.

Table 4.1- Grading rubric in the lab proposal.

Game Logic

The screen will display the notes that will be played, and the player will press the buttons on the board corresponding to those notes. The four columns will display which buttons need to be pressed. The notes will fall from the top of the display to the bottom, and the player must press the correct corresponding button before the notes reaches the bottom. If the wrong button is pressed or the note reaches the bottom of the display, it will be considered a missed note. Each player will receive a certain number of points for each note that they hit, and for every 10th successive note they hit, they get twice as many points until they miss a note.

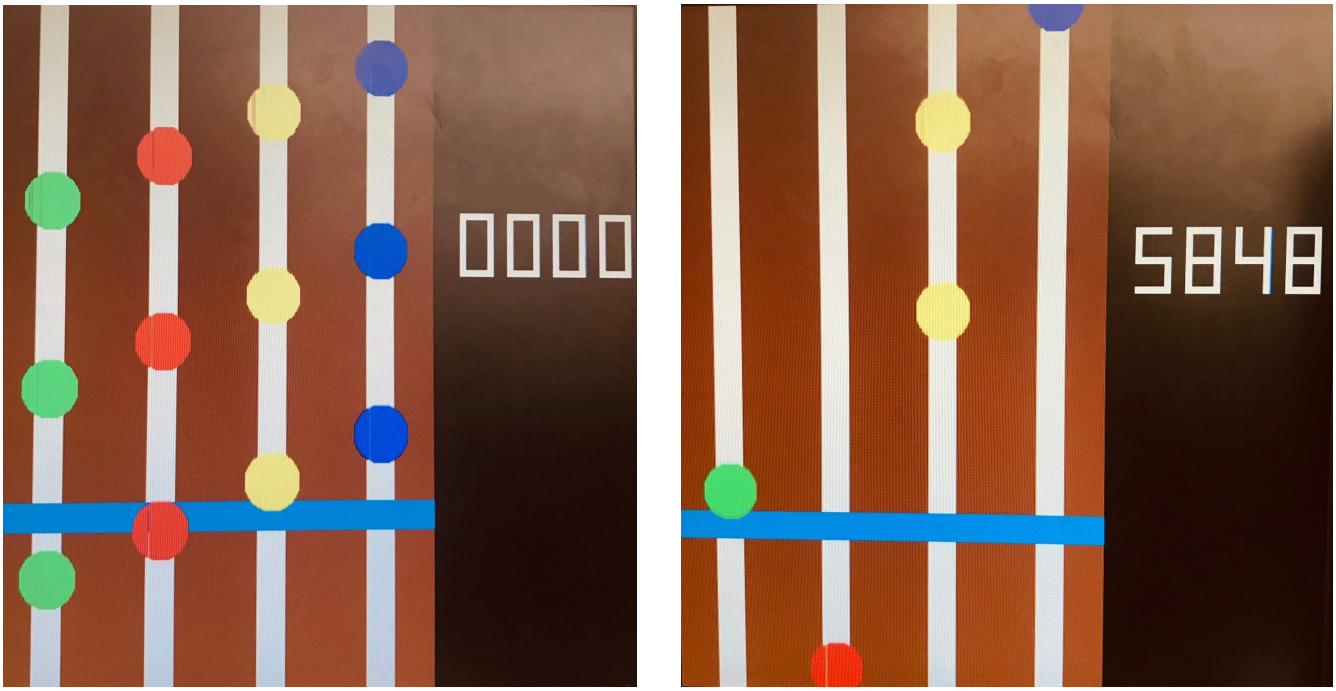

Figure 4.2- Our implementation of the Guitar Hero display. The four different colored circles represent the notes that needed to be played, and the horizontal blue line indicates when these notes need to be hit. The four digits on the right display the score.

Design Description

Nerp_demo_top.v
The Nerp_demo_top.v contains the source code from the nerp demo. We added the four directional buttons to the input of the top module and sent the buttons as input to the vga640x480 module, and also added debouncing for the buttons in the top module using a slower clock.

vga640x480.v
Uses nerp demo vga640x480.v was the foundation. We changed the display to look like a guitar by changing the if statements that control the pixel colors in certain ranges. We connected the pixel clock, the clr, and the button pushes to the input of the gamer controller module and output display bits and enable bits which control the score shown on screen and when the dots should move down the screen. We added 12 if statements to the display combinational always block to make dots move down the screen. We also included 28 if statements to control a 4 digit 7-segment display on screen. The if statements for the digits is controlled by the output display bits from the game controller.

dotcontroller.v
The dotcontroller.v takes as input the debounced button presses, clr, the enable bits, and the input pixel clock dclk and outputs pixel positions for the 12 dots and a dotclk which is a clock divided to get the speed of the dots reasonable. The dotclk was made by adding one to an 18-bit register at the posedge of dclk and taking the most significant bit of the 18-bit register as our dotclk. Then for every posedge of the dotclk we update the run bits and the registers that contain the dot positions. Run bits is sent to the vga module to control whether the if statements for the dots are shown or not. Run bit logic uses the enable bits from the game controller and sets the run bit for a specific dot to 1 if the enable bit is set and the run bit for that dot continues to be 1 until the dot reaches the bottom of the screen. The dot positions are iterated if its enable bit is set until the dot reaches the bottom of the screen.

gamecontroller.v
The game controller takes as input the pixel clock dclk, dotclk, clr, the debounced directions buttons, and the positions for the dots as they travel down the screen. The output is the enableBits, and the 4-digit seven-segment display bits. We made a sendclk to make a time between each dot as it comes down from the top of the screen by using a 24-bit counter and looking at the most significant bit in the 24 bits. The sendclk controlled a finite state machine which determine which enable bits to set. This finite state machine has 12 stages that drop the dots one at a time onto the screen and it resets once all of the dots leave the screen. We made invalid press and valid press signals based on if the dot position was in between some value and the button is pushed or if it was pushed when the dot was not in range. We made four score digits that were initialized to zero and increased by the value streak if a there was a valid press. If there were multiple valid button presses in a row the we incremented a streak variable which resets to 1 if there is an invalid press. The score digits were put into the digittodisplay module to get the output segment bits from the module.

digittodisplay.v
Takes four input digits and output the 7-digit binary for what display segments should be lit. Made using always blocks with a case statement for each of the four digits.

Simulation Documentation

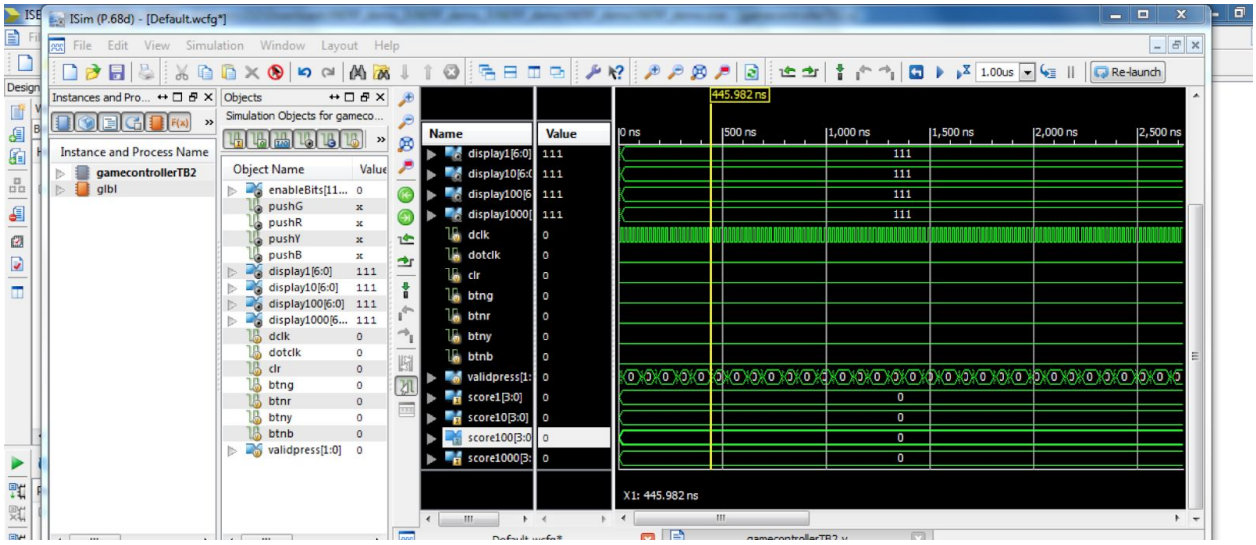

Figure 4.3- This screenshot displays the waveform of the gamecontroller module. The inputs include the pixel clock dclk, dotclk, clr, the debounced directions buttons, and the positions for the dots. The right side represents the resulting simulation of the output of the score.

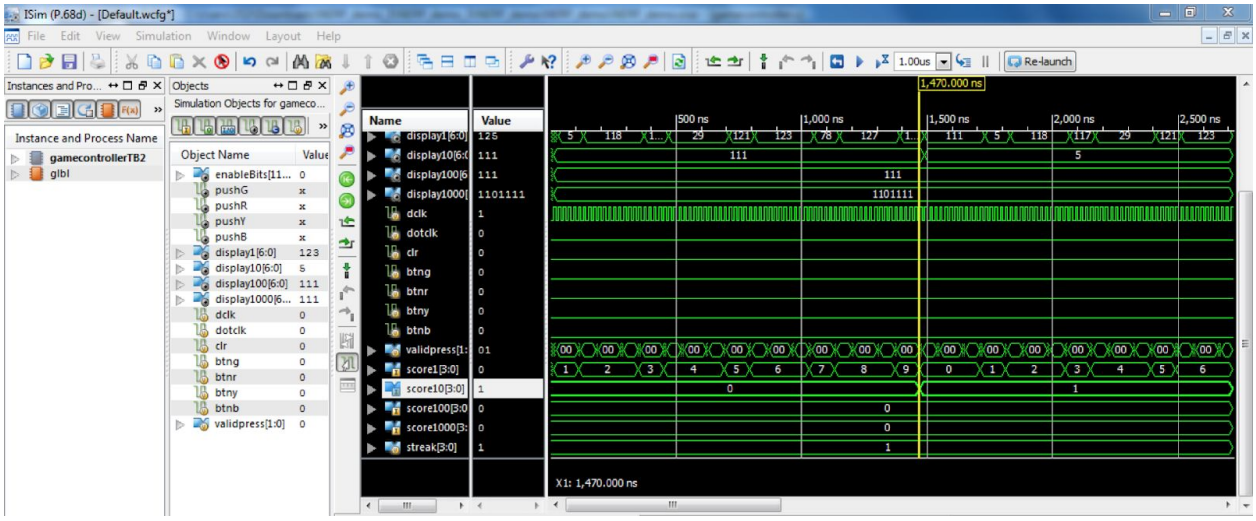

Figure 4.4- This screenshot displays the waveform of the gamecontroller module. The inputs include the pixel clock dclk, dotclk, clr, the debounced directions buttons, and the positions for the dots. The right side represents the resulting simulation of the output of the streak.

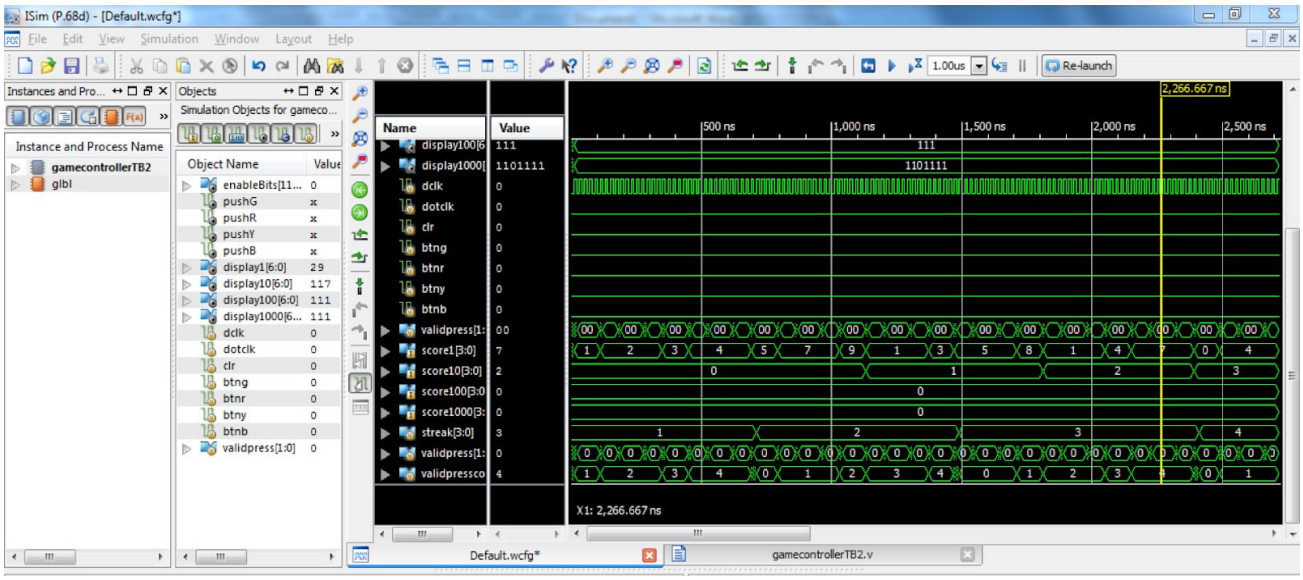

Figure 4.5- This screenshot displays the waveform of the gamecontroller module. The inputs include the pixel clock dclk, dotclk, clr, the debounced directions buttons, and the positions for the dots. The right side represents the resulting simulation of the output of both the score and streak.

Conclusion

Overall, each of the modules combined to create our simplified version of Guitar Hero. Using what we have learned from previous labs, we have designed and implemented the game using the FPGA board that imitates the real game.

Some of the problems we face was that we failed to implement to score correctly. In the test benches we saw proper behavior of streak and score, but in reality when we pushed the button in range we saw the score shoot up by a couple thousand and after that some button pressed affected on the tens and ones digit but most pushes increased the score by a couple thousand. At first we thought it was the the logic, but after moving the code around to created more detailed test benches we realized that all the functionality for the logic worked with the test benches. Then we thought that it must be a button debouncer issue because the logic for increasing the score was working properly in test bench, but by changing the button two switches and seeing the score constant increase when the switch was asserted, we realized it was not a button debouncer issue. After researching the problem further, we realized that we were using an edge triggered always block instead of a level triggered combinational block. Combinational block are used when the output is not necessary changed every clock cycle, but by other levels or changes in other variables in the sensitivity list. After this change, nothing improved as we were still getting the massively increasing score, and we had nowhere else to look for the issue, so our score unfortunately remained buggy.