**ECE 385**

Fall 2021

Final Project

**FPGA Hero**

Steven Dimov & Owen Shin

Section ABE

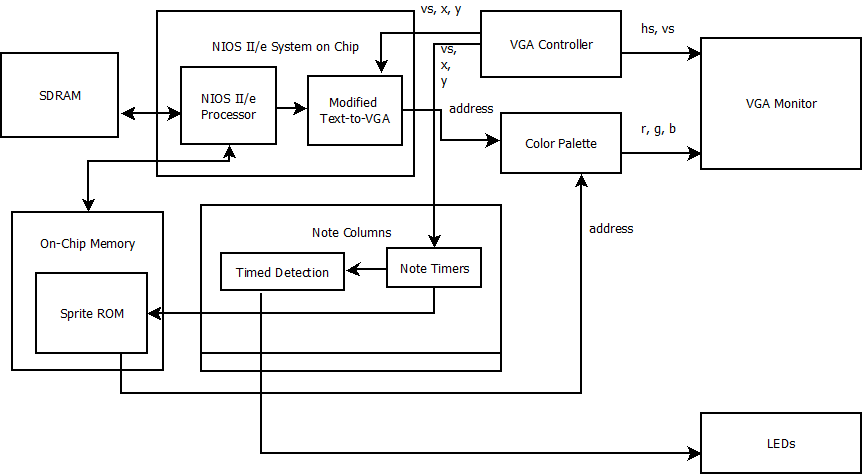
TA: Abigail Wezelis

**Introduction:**

The idea of this project was to recreate the classic guitar hero game onto the FPGA. The design outputs sprites and text to a VGA display. The sprites are displayed by reading a ROM, and their locations are taken from hardware timers. The text is displayed using the system on chip which contained a modified text-to-VGA IP. A separate piece of hardware was made to output audio to the FPGA shield’s 3.5 mm port. This could There were many features that weren’t implemented, but the current hardware serves as a skeleton for an interactive game.

**Description of Final Project:**

Block diagram



Block diagram of the Guitar Hero game. The system on chip controls text, and other hardware modules control notes in-game and take in user input or output to a VGA monitor.

System-on-chip

The system on chip of this lab is a copy of the lab 7 system on chip. It uses the NIOS II/e processor and a modified version of the IP created in that lab. The IP still writes text to the VGA monitor, but does so by outputting the address for a color palette to the top-level module. This allows all of the combinational logic for this and the palette to be used by other hardware modules. In addition to the customized IP, an SDRAM controller and PLL, on-chip memory, system ID, and a timer peripheral.

The system on chip runs software that initializes by filling the screen with text from top to bottom, then runs the random screensaver from lab 7 part 2. This is used to demonstrate the abilitiy to make blank characters in VROM transparent and to make transparent backgrounds or foregrounds for non-null characters.

Module Descriptions

Module: topLevel

Inputs: Clk, KEY[1], KEY[0]

Outputs: VGA\_HS, VGA\_RS, VGA\_R, VGA\_G, VGA\_B

Description: This module contains five instantiations of the timersColumn module, and routes all outputs from these. It also instantiates a singleTimer1 module to count to one-sixth of the time that the timerColumn modules count to, which is used to signal when these modules should start their timers. It also contains some combinational logic and a palette ROM to determine which graphical outputs from modules have priority and which ones are transparent, and how their outputs should be translated into colors. It also instantiates the system on chip to give graphical output for text and a VGA controller module to control and give coordinates for the VGA monitor.

Purpose: This module integrates all of the other modules into one so that the necessary connections can be made for the project’s functionality.

Module: strumWithinRange

Inputs: timerActive, frameClk, strum, Y

Outputs: score, addHit, addMiss

Description: This module checks the location of its corresponding timer when a strum signal is set high to check whether this happened within the desired range. The desired range for a strum is determined by parameters. The module pulses a signal to add to a count of misses or hits based on whether a strum signal was pulsed within the desired range. It also outputs a score value that is larger the closer the timer is to the desired parameter location.

Purpose: This module determines the valid range of when a note should be hit onscreen in order for the game to be able to able to produce a score for the player, which is the incentive of the game.

Module: note\_rom

Inputs: addr

Outputs: data

Description: The input addr is taken in to match a certain row of the rom, determined through combinational logic. The data at that location of the rom is then sent as an 49-bit output, holding information for the sprites that will be drawn onscreen.

Purpose: this module provides the note information of the game, determining when and where a note will be. Specifically, it is used for determining which columns will produce a sprite at the top of the screen, which is then carried through to the bottom.

Module: palette\_ROM

Inputs: addr

Outputs: data

Description: This module intakes a given address, addr, which corresponds to a location in the rom that holds RGB information for that specific color. This address’s information is then fed to the 14-bit output, data through combinational logic.

Purpose: This module is the ROM that holds the color palette information for providing color to the screen. It is fed in a selection for which color to transform into RGB information for the VGA to display.

Module: vga\_controller

Inputs: Clock, Reset

Outputs: hs, vs, pixel\_clk, blank, Draw\_X, Draw\_Y

Description: The clock input operates at the FPGA’s 50MHz signal. The reset signal sets both the vertical count and the horizontal count to zero indicating the setup of drawing a new image. The hs signal synchronizes the start of each new row and the vs signal synchronizes with the beginning of each new frame, done by modulating between high and low based off of the vertical and horizontal counts. The pixel\_clk signal is a 25MHz clock signal, as it is the 50MHz clock input divided by two. It controls the frequency that vs and hs operate on. The blank signal is an active low signal that indicates the blanking period of the vga output in which there should be no data as the ‘electron gun’ needs time to reset. The Draw\_X and Draw\_Y signals match with the 2-dimensional counting signals, vc and hc, which count the horizontal rows and vertical columns that the electron gun’s location is at a given moment.

Purpose: This module controls the vga output by creating an operating clock frequency, synchronizing the output signals appropriately for the VGA, and providing the location of where pixels are being drawn at a given moment. This allows the other module to interface with the operation of the VGA display to utilize it for a graphical output.

Module: vga\_text\_avl\_interface\_3

Inputs: CLK, RESET, AVL\_READ, AVL\_WRITE, AVL\_CS, AVL\_BYTE\_EN, AVL\_ADDR, AVL\_WRITEDATA, Draw\_X, Draw\_Y

Outputs: AVL\_READDATA, reg\_idx

Description: This module is an IP that is a modified version of the IP from part 2 of lab 7. Like lab 7, it uses the avalon bus to receive characters and foreground/background selections to store in VRAM, but does not contain its color palette in the IP. Instead, the IP outputs the address of a color in the palette module instantiated in the top-level entity. There is also some extra combinational logic to check whether the character currently being drawn (according to the inputted x and y coordinates Draw\_X and Draw\_Y) is null. If this is the case, all palette addresses for pixels in this character’s area are null, which means a black pixel. This allows for transparency where characters are null.

Purpose: This IP was modified to allow the system on chip to write characters without taking up unnecessary resources (a palette and some combinational logic) and have transparency for null or non-null characters.

Module: timersColumn

Inputs: DrawX, DrawY, M, gameClk, noteStart, strum, frameClk

Outputs: hcount, mcount, regAddr

Description: This module instantiates multiple timers to represent multiple notes in a column. Upon each rising edge of the gameClk clock, it checks whether the noteStart signal is instructing it to start a new note in the game and increments which of the timers it may start a note on. If the noteStart signal is high, it sends a start signal to the relevant timer module. The timer modules count up by the input number M to the prescribed limit using the clock signal frameClk. There are as many strumWithinRange modules instantiated to check whether a player input was given while a timer’s value was within the desired range. These strumWithinRange modules increment a set of registers that count hits or misses accordingly, and the values of these registers are outputted as hcount and mcount. The timersColumn module also checks the DrawX and DrawY with the locations of the timers, and determines whether the pixel being drawn on the VGA monitor should represent a note sprite. If a note sprite should be represented, combinational logic is used to address a row in the noteSpriteROM module, and based on the relevant bit in that row the module outputs a color palette address.

Purpose: This module was created to be instantiated once per column of notes in the game, each with the horizontal location determined by a parameter. Each contains the timers for notes, and the hardware to check whether a note is being drawn or being played by the user input. Each column outputs the relevant pixels for a note being drawn or the counted responses to user inputs.

Module: singleTimer1

Inputs: M, Clk, start

Outputs: timerCount, timerActive, timerEnd

Description: This timer module takes in a parameter and counts up to that value upon the start signal being high, incrementing the counter by the input M and using the clock signal Clk. This allows for the speed to be varied by M. The timerCount output holds the current value of the timer, and the timerActive is an active-high signal indicating whether the timer is counting or has halted. The timerEnd signal pulses when the timer halts, which can be connected to the start signal of the same timer for a continuously counting timer.

Purpose: This module is being used to track the location of notes and start notes at zero when they should be entered into the game.

Module: I2S\_interface

Inputs: LRCLK, SCLK, freq\_mod, I2S\_DOUT

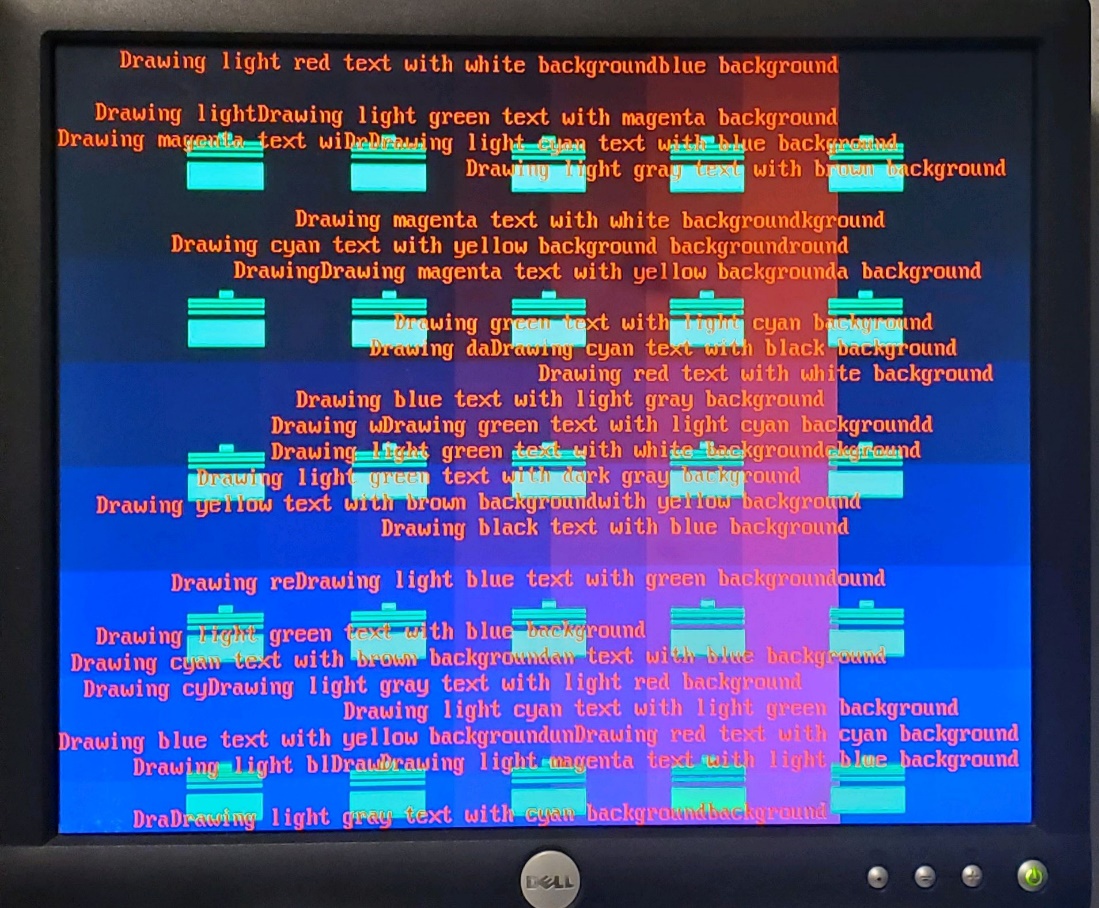
Outputs: I2S\_DIN

*(I2S\_DIN & I2S\_DOUT are technically inouts)*

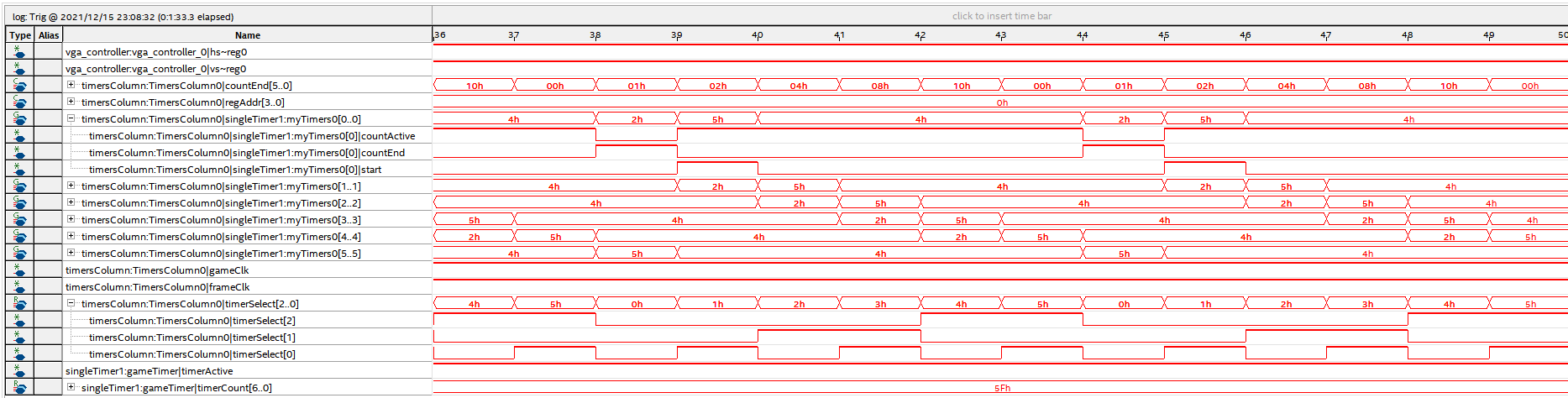
Description: The module can be set to either take in the I2S data from the IC’s line in or from the signal generator. The main ports, I2S\_DIN, I2S\_DOUT, serve to collect and output data to the headphone out and provide data from the line in respectively, while they both operate on a single bit wide bitstream. The SCLK signal controls the shift register for this operation, as the data is moved by one bit on each cycle. The LRCLK signal handles the interpretation of this flow of data as it establishes when data corresponds to the left channel on active low and right channel on active high. These two clock signals are provided by the SGTL5000 when it is fed a 12.5MHz signal, which is done in the top level module by dividing the main 50MHz clock. The freq\_mod one-hot signal modulates the square wave signal frequency by offsetting the counter by an amount determined by the input of the 5 frets.

Purpose: This module handles the routing and data that is to be played on the SGTL5000. This module is the interface that allows sound to be read or written to the SGTL5000 IC using the I2C/I2S protocol. It also contains the signal generator for the five different sounds intended for the 5 ‘frets’ on the game.

Display on VGA monitor:



Start and End Signals for Timers in a Single Column



Design Statistics Table

|  |  |
| --- | --- |
| LUT | 2,880 |
| DSP | 0 |
| Memory (BRAM) | 55,296 |
| Flip-Flop | 2,353 |
| Frequency | 144.84 MHz |
| Static Power | 96.18 mW |
| Dynamic Power | 0.69 mW |
| I/O Power | 9.03 mW |
| Total Power | 105.90 mW |

**Conclusion**

Our proposal’s design was fairly complex, requiring a lot of time and effort. Unfortunately, we were unable to reach our project’s full potential, but we were able to construct a portion of the intended functionality, but done under separate files not integrated into the same project. At the final state of our project we had a minimal graphical output that could draw sprites that moved down the screen drawn from a rom. The graphics also included background visuals and text onscreen. The functionality of detecting when a valid strum was created but not integrated with the project. It would have used the keyboard for this operation, which was also not integrated into the project as intended. Sound that varied with pitch according to different switch inputs was able to be produced, done through a signal generator instead of data provided by the sd card, which was another component that was not able to be included in the project as originally intended. There were several difficulties and not many resources upon the effort of setting up the sd card to work. The program dd that was necessary to set up the data onto the sd card was not functional on our windows machines. But most notably, the sd card was simply difficult to debug due to a lack of visual feedback.

The final project could be extended to the original scope, which would require implementing keyboard inputs, fixing the timing detection and creating an interface between the system on chip and the game hardware. Keyboard implementation would allow for user input, which would consist of note-playing in the game and menu navigation. Originally, the intent was to allow the user to set which keys correspond to which notes and store those keycodes in registers, since some keyboards have different keycodes. Fixing the timing detection would allow for the game to properly count the number of hits or misses from the user, which could be communicated to the system on chip, which would display it in text. This would require communication between hardware modules instantiated in the top-level entity and the system on chip, which would probably be achieved through the avalon bus. Beyond that, there were ideas for extra features that would have added to the user experience. Modifying ROM and the way it was addressed would allow for animated sprites. Use of hardware to read from an SD card would have allowed game data (sequences of notes to be played) for different games to be stored and used without the need to recompile. Integrating the hardware modules and software for the SGTL5000 audio IC would have allowed for different sounds to be played when corresponding notes are played, or if it was integrated with the SD card a song could be played during a game.