Design Document - ZyloZinger

Sienna Brent - scb2197

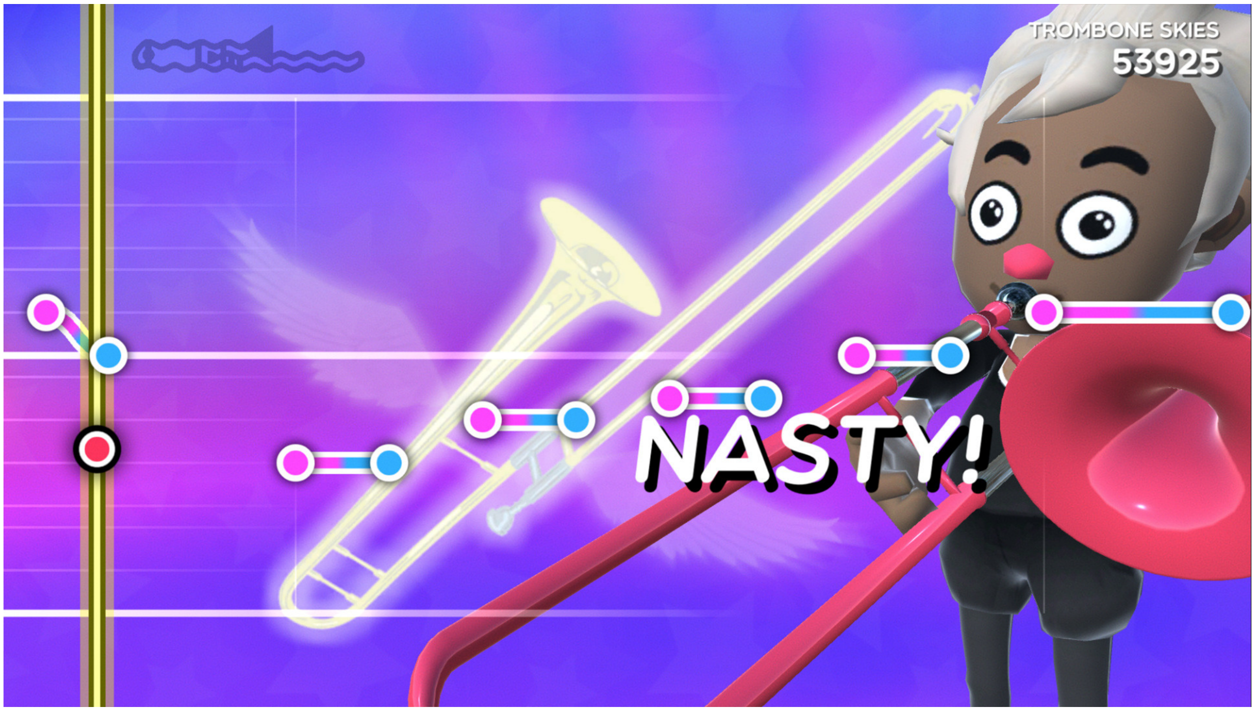
Alex Yu - asy2126

Riona Westphal - raw2183

Rajat Tyagi - rt2872

1. Introduction

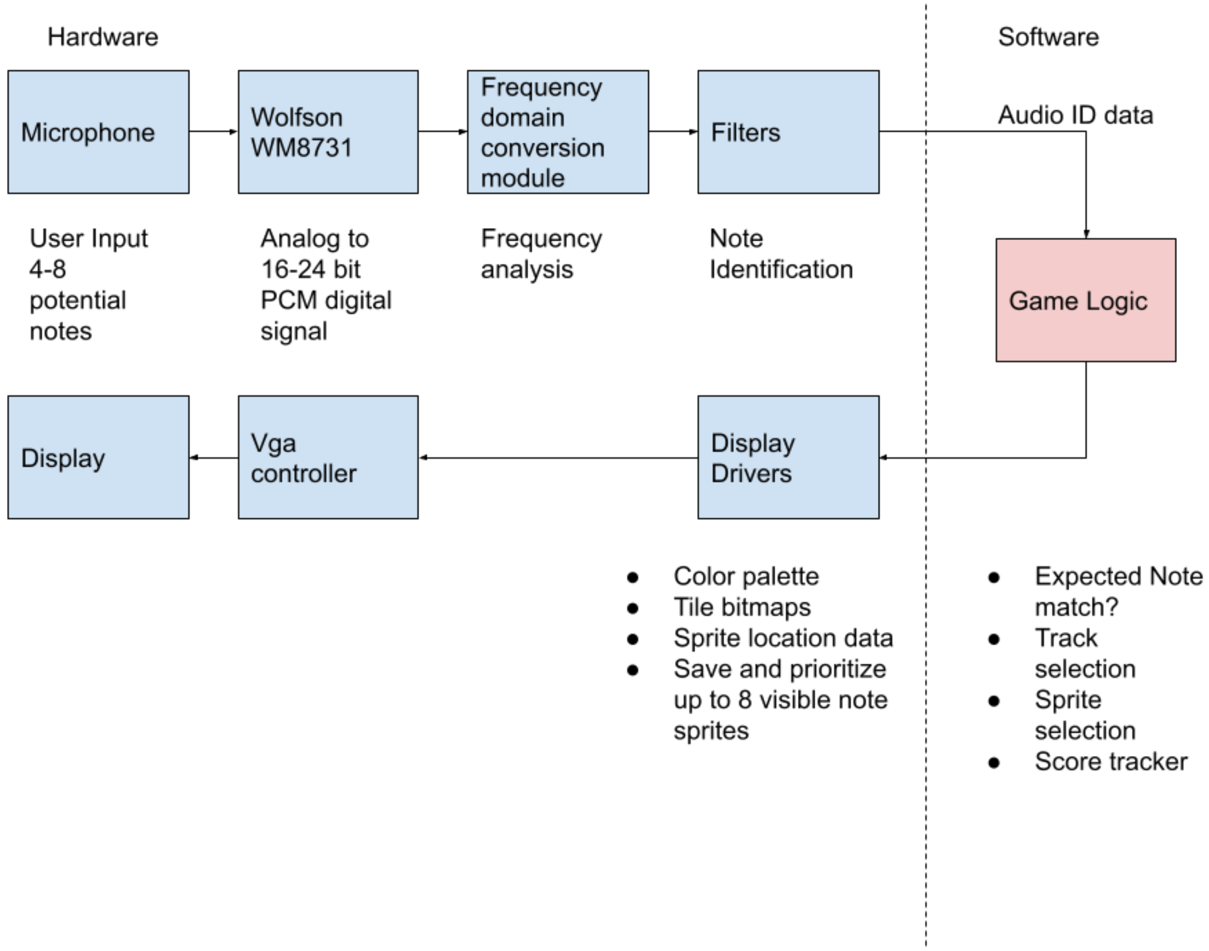
This device will use a microphone connected to the FPGA through the mic-in port to recognize four distinct tones being played on an external instrument. The incoming audio will be used in a video game simulation on the VGA display to earn “points” for the user if the correct notes are played at the correct time. Similar in spirit to guitar hero, trombone champ, or even DDR.



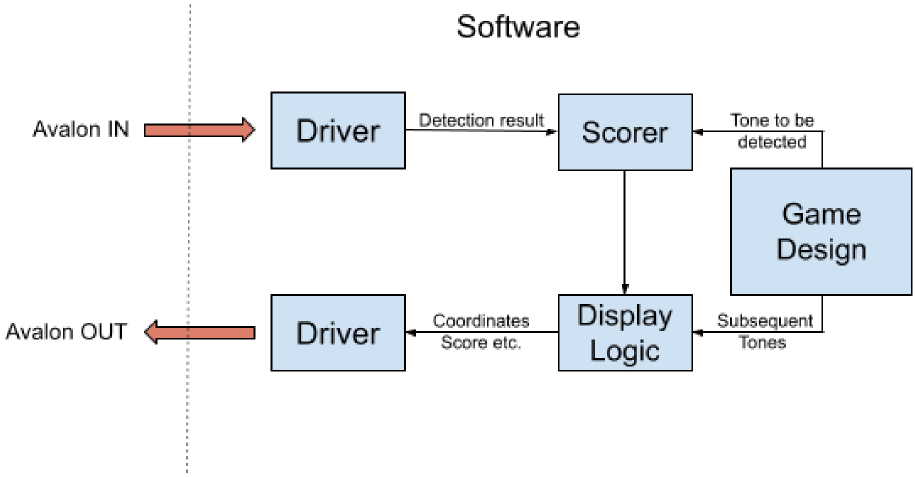
2. Block diagrams

On the hardware side: The block diagram shows that on the hardware side, a microphone feeds analog signal into the onboard audio codec. The audio codec converts the analog signal into a 16-24 bit Pulse Code Modulated digital signal. These bits are then sent into our FFT accelerator and filtered to then only send to software a hotcoded digital signal of 4-8 bits representing the specific note played for software to interpret.

Software does some magic in the game and then sends sprite and game state information to the PPU or picture processing unit which takes the sprite data and sends the correct raster line signal to the vga display.



Software



Game logic:

On start, the game will display a menu to select a song. This menu can be navigated by using keyboard input:

- Arrow up/down: choose song
- Enter: select song

After selecting a song, the display will change to that of the game itself:

- Sprites (ovals) will fall down the display
- When the ovals reach a green area near the bottom of the display, the user should play the appropriate note
- The appropriate note will be dictated by color and column.
    - For example, note “C” will always be blue and in the leftmost column
- When the user plays the appropriate note, their score (displayed in the upper right corner) will increase, the combo will increment by 1, and the sprite will disappear

    - If the user misses or plays an inappropriate note, their score will stay the same, the combo will revert to 0, and the sprite will fall to the bottom of the display

At the end of a song, a “score page” will be displayed. This will show:

- The total score
- The number of hit notes
- The number of missed notes
- Highest combo
- A percentage of the total of notes that were hit

From the score page, the user has the option of either returning to the menu or replaying the song. The desired option can be chosen using the arrow up/down keys and pressing enter on the desired choice.

3. Algorithms

We have two preliminary ideas for how to go about doing FFT calculations on the data:

1) A short time fourier transform (STFT) will be performed on the entire audio sample. Thus for each short set time frame, a FFT is performed and peaks are analyzed after a 1200 - 3600 Hz range (based on the current toy piano available) band pass filter. The resulting “fingerprint” produced by the STFT would be sent to the software at the end and the score would be calculated after the game is over. This continuous FFT analysis would require a complex FFT pipeline, as can be seen in the 2019 project TNShazam.

2) Alternative to the continuous STFT approach, we would have an internal timer that sets a “valid” time interval over which a DFT is performed. The user would be expected to play the note within the valid range, and produce one peak at the correct frequency in the DFT. This method would be less computationally intensive - instead of performing dozens of FFT’s per second, we would only be doing one per note, but would probably not be as robust against noise as the

former option. Additionally, this approach would require more memory for the increased size of the audio sample being taken before the FFT is taken.

In either case, the FFTs will be done using the Cooley-Tukey algorithm and with the ButterflyModule as the basic building block.

For display, a similar approach to the NES sprite display solution will probably be implemented:

Sprites will be split into a spire attribute table where horizontal and vertical position, bitmap name, are stored and a list of pattern maps which the name will refer to as seen in the sprite lecture. Per raster line, the display module will fetch the color palette, fetch the tiles it will cover, fetch the tile bitmaps according to sprites, search for location data of all sprites to check if they exist in the line, and save the first 8 visible sprites. We will likely use fewer than 8 different sprites at a time.

4. Resource budgets

Resource budget is difficult to estimate when size of sprites have not been decided upon and size of audio sample to be analyzed have been decided on.

5. A detailed plan for the hardware/software interface: every register and bit

In our top-level hardware system verilog file, three main modules will be instantiated: The audio control, the VGA control, and the FFT accelerator. The audio control is what handles the Wolfson WM8731 audio CODEC. Video will be controlled with a VGA controller. The FFT accelerator is internal to the hardware and does not require an interface.

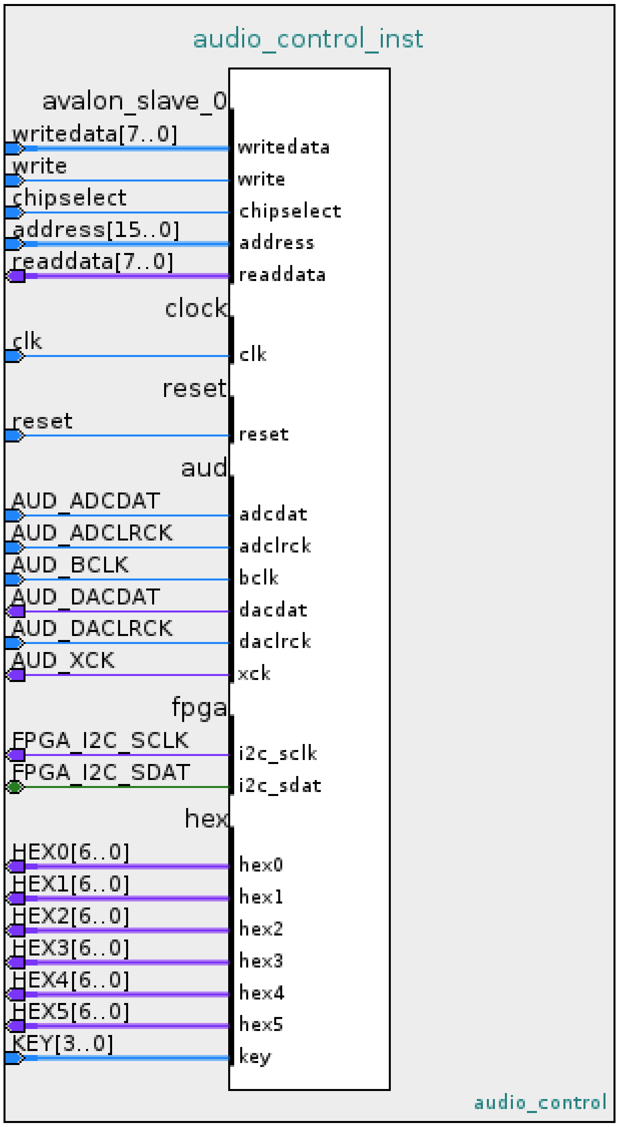


Image of the audio controller.

Analog audio signal is input from a microphone into the Wolfson WM8731 CODEC and converted to 16 bit PCM digital signal representing amplitude in the time domain. Instead of configuring the Wolfson WM8731 with I2C ourselves, we will use audio drivers sourced from Altera and modified by Professor Scott Hauck and Kyle Gagner from their audio tutorial for the De1 board. Left and Right stereo at 24 pins, it will be halved and summed to create mono audio (since they come from the same source).

The FFT module will intake the 16 most-significant bits of data from audio_control and output the result to the top-level, which will map the frequency to a 4 bit hotcoded note data packet and send it to the software on readdata. If the FFT module writes 0110, it means the two middle notes were detected. Software is what will determine which notes are new and should be scored.

There should probably be one more bit indicating if a note was struck twice in a row or bleeding through from the previous strike.

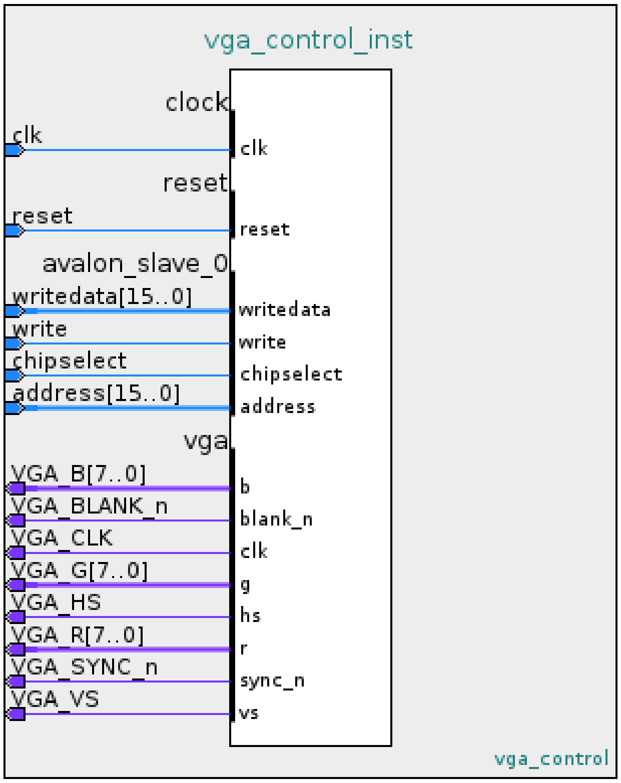
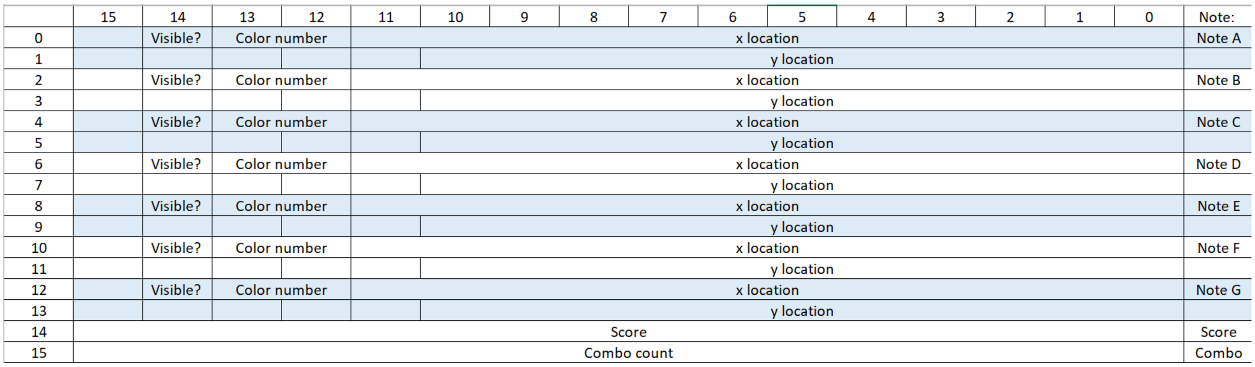


Image of VGA controller

VGA controller takes in and address and writedata dependent on our software to hardware data packet. The 16 addresses will hold all sprite attribute tables and print to display accordingly.

The following will be the information sent from software -> hardware during the game. This information will be sent via the Avalon bus:



	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Note:
0		Visible?	Color number		x location												Note A
1						y location											
2		Visible?	Color number		x location												Note B
3						y location											
4		Visible?	Color number		x location												Note C
5						y location											
6		Visible?	Color number		x location												Note D
7						y location											
8		Visible?	Color number		x location												Note E
9						y location											
10		Visible?	Color number		x location												Note F
11						y location											
12		Visible?	Color number		x location												Note G
13						y location											
14	Score																Score
15	Combo count																Combo

For the menu stage, each selectable song (encased within a rectangle) will be represented by a sprite. A red rectangle (the same parameter as the selectable song's rectangle) will surround the currently selected song on a higher layer.

For the score page, each shown parameter (ex: total score, highest combo) will be represented by a sprite.

Milestones

03/31: **1: Get the microphone communicating with the FPGA**

04/07: Get display communicating with FPGA and software

04/14: **2: FFT working - FPGA should be able to recognize predefined tone**

04/21: Software driver interfacing with hardware

04/28: **3: Graphics complete**

05/04: Finished