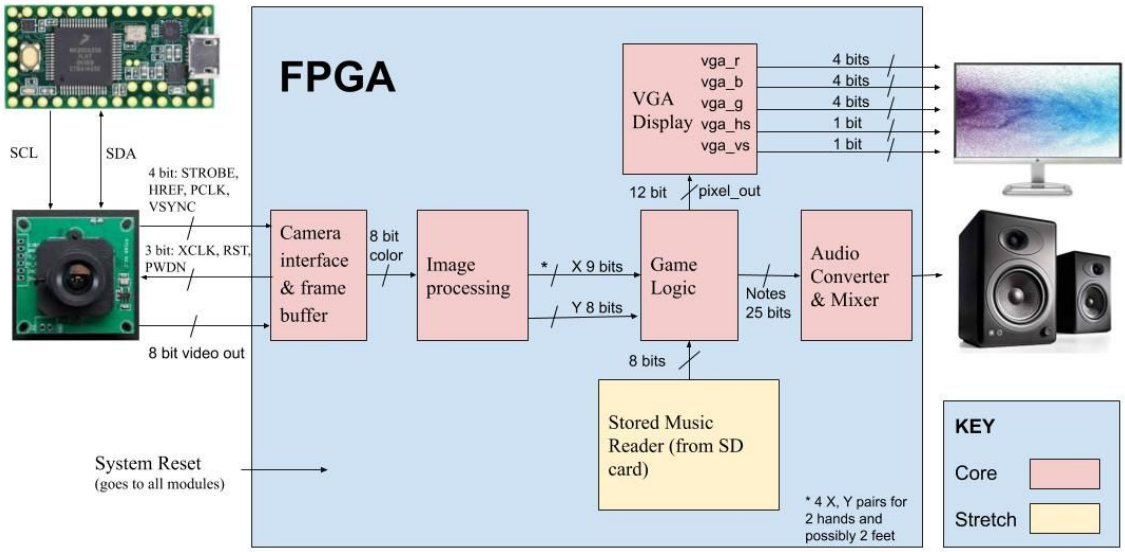
Sarah Spector and Alejandro Diaz
6.111 Final Project Proposal
10/29/19

FPGuitAr Hero

I. Overview

FPGuitAr Hero will combine camera-based motion tracking with a graphical interface and audio output to create a game that gets people moving through music. In *FPGuitAr Hero*, players wear colored tags on their hands which are tracked by a camera and mapped onto the graphical interface. On this interface, users must "hit" notes as they stream down the screen. Each note successfully hit will produce the corresponding audio output and earn a point.



II. External hardware

A. Camera and microcontroller

We will use an OmniVision OV07670 CMOS VGA (640x480) camera chip. To conserve FPGA resources, we will initialize registers on the camera using a pre-programmed microcontroller (Arduino Teensy) communicating over I2C. The camera will then send 8-bit color data to the FPGA.

B. Display and Speakers

Since the project involves both graphical and audio output, we will need a standard VGA display to act as a GUI for our game as well as a set of speakers connected to the FPGA to play the music generated by *FPGuitAr Hero*. The notes will be specified using a PWM module.

III. FPGA modules (all have a reset signal as an input and run on a 65Mhz clock)

A. Camera interface and frame buffer (Sarah)

We will need a module to interface with the camera, controlling camera input signals like XCLK (clock), RST (reset) and PWDN (power), and receiving video-related signals STROBE, HREF, PCLK, and VSYNC as well as the 8-bit color data. This should not require significant memory or arithmetic operations, and we can test by porting the camera output through the FPGA directly into the VGA output to visualize the camera image as a livestream.

After implementing the camera interface, we will then store 1 frame at a time of 8-bit color in BRAM before porting it into the image processing algorithm. Given that the camera resolution is 640x480 pixels and each pixel requires 8 bits to store, a 1 frame buffer should take up 307.2 KB of BRAM. We may go to a lower resolution via camera settings or downsampling if we find that we need to conserve BRAM. We maintain 8 bit throughput throughout the system pipeline to prevent data loss. We can test our frame buffer by porting it out into VGA output to visualize a single captured frame.

B. Image processing (Sarah)

Once a complete frame has been stored in the buffer, we port pixels into the image processing algorithm one at a time (8 bit wide wire). We divide the 640x480 camera output into a grid and add a small buffer of registers (or maybe BRAM) to the image processing module to hold the number of pixels present in one square of the grid. Pixels are sent into the algorithm from the earlier frame buffer one grid square at a time.

The image processing module uses color thresholding on this grid square of pixels to assign a 1 or 0 to that square based on whether enough of the target color is present. We aim to make these grid squares large enough such that we can use an evaluation of 1 in a square as an indication that our target color patch is present in that square. The image processing module then sends X and Y coordinates (9 and 8 bits respectively) of the detected colored square into the game logic module.

Color thresholding should not require significant computing power/operations since it can be implemented as a simple comparison between, for instance, the R value of the pixel and some constant threshold R value. We can test our color thresholding algorithm by visualizing the grid squares that evaluate positive on the VGA display as we move a colored blob around (live) in the camera's field of view.

C. Game logic (Alejandro)

Game Logic will take the x, y positions of the hands of the person. The x positions will be used to generate hand analogues on the screen (could be any shape or image). In order to create shapes and images the system will rely on the modules implemented in Lab 3. It is likely that circles or other shapes will be generated, which would require equations to describe the shapes or other such techniques.

Once gameplay has begun the notes will stream down (changing x, y coordinates) from the top of the screen with a rectangular shape (could be more complex). The cadence at which the notes stream down will be in accordance with the BPM (beats per minute) of the song. The length of the notes will correspond to the relative amount of time the note will be played within the music. As an example a 50 pixel long note will be played for twice as long as a 25 pixel long note. The location of the notes on the x axis of the screen will also correspond to the relative pitch of the notes. Notes on the left side of the monitor will be at a low pitch and notes on the right side will be at a higher pitch with a gradient in between.

The hand analogues will be superimposed on a straight horizontal line on the screen in order to standardize timing in the game. When the notes and the hands intersect the note will be played. If a note has been hit then there will be an indication that the note has been hit with a change of color as well as the audio output.

Scoring will be some combination of notes hit and other gameplay features that may provide a multiplier or other score implications. The score will initially be displayed in hex on the hex_display for debugging and will be placed on the screen itself, time allowing.

This module will be tested by simply playing through the game and seeing if appropriate behavior is observed. For testing we will most likely hardcode a short song taking up perhaps about 100 registers. Initially x, y positions will be faked by using button presses to control hand positions.

There should be minimal arithmetic operations on the order of 10 +/- operations and 100 registers for keeping track of note positions and movement.

D. VGA Display (Alejandro)

The VGA Display module will rely heavily on the already implemented module that was used in the Pong Lab. This provides 16 different individual RGB values that can be displayed on the monitor. Each pixel is displayed one at a time on a 65MHz clock. A v_sync and a h_sync signal specify to the monitor where each pixel should be displayed. This module should not need to be tested since it has already been implemented. There should be minimal arithmetic operations on the order of 10 +/- operations and 100 registers for keeping track vcount, pixel, hcount etc.

E. Audio Converter & Mixer (Alejandro)

Will rely on the modules from the audio lab as a base. The sine generator will be used, but will have a modified lookup table that will contain a wave that is similar to a guitar wave. This also leaves room for changing the timbre of the guitar and adding different instruments such as the trumpet.

Notes will be received as a 25 bit array that allows for two octaves of possible notes being played simultaneously. Each bit will correspond to a different note that should be playing. The notes will correspond to specific phase shifts that encode the specific frequencies for each note.

$$\frac{\left(440\frac{\text{cycles}}{s}\right)\cdot\left(2\times10^{32}-1\frac{\text{phase}}{\text{cycle}}\right)}{48\times10^{3}\frac{\text{samples}}{s}}\approx 39370534\frac{\text{phase}}{\text{sample}}$$

Furthermore the module will have the capability of mixing multiple notes in order to create chords. To mix multiple notes we will experiment a variety of techniques including X-OR to mix together PWM signals from different notes.

This module will be tested by playing notes that are hardcoded first separately and then together as a chord. There should be minimal arithmetic operations on the order of 10 +/- operations and 100 registers for keeping track vcount, pixel, hcount etc.

IV. Extensions and stretch goals

A. Load a song

If time allows, we will replace hard-coded music notes in FPGA registers with the ability to load in songs and or notes from an SD card or other comparable storage mechanism. We would likely either manually write notes and music in an FPGA readable format or create a Python Script to write an MP3 music file into an understandable format for the FPGA.

B. More sophisticated image processing

We could implement more sophisticated algorithms that either do a better job of tracking and/or do not need to rely on brightly colored patches to identify x, y positions of hands and feet.

C. Game logic add-ons (in no particular order)

1. Menu Selection - The coordinates of one of the hands can be used to select different menu items withing the GUI in order to toggle different songs or difficulty. The right hand would be tracked by taking the x,y position that is farthest up and to the right within the corresponding right top quadrant. Option selection would then consist of maintaining the hand over a certain selection for a certain amount of time (~ 3 seconds).
2. Y coordinate - Since the hand is superimposed on a straight line eliminating the y coordinate of the hands, the y coordinate could be incorporated in some way, such as for volume control or a different style of gameplay.
3. Guitar Hero Gameplay - The system can attempt to mimic some features from Guitar Hero gameplay. For example, note streaks can be tracked in order to add to a score or affect some kind of multiplier. Also certain notes can be made to worth more or to add certain kinds of bonuses. For example there could be a meter that fills up based on correct notes played and then once it is full an extra multiplier is added with some cool gameplay effects such as inverted colors or rainbow RGB values for all notes.
4. Ghost Mode - A mode where you see the gameplay of a previous person as you are playing the same level. This could allow you to compete against another person more easily without actually playing them at the same time. This would most likely require the use of BRAM to save a players gameplay from a previous round.

5. Non-Discrete Notes - Noted do not have to be discrete. Glissandos could be incorporated(a glide from one note to another). This would most likely mean changing I/O from gameplay module to the audio module to perhaps be 4 independent notes with 32 bit precision where the game logic would specify slightly differing notes in order to achieve glissandos.
6. Feet Tracking - Feet could also be added in to add to gameplay. This could mean a split screen with notes for hands and notes for feet.
7. Two-Player Mode - A split screen and players have to compete.
8. Multiple Levels/Songs - Have multiple songs and/or difficulty levels. We could also keep track of high scores for each game.
9. Variable Gameplay Speed - different BPMs for different songs
10. Incorporated other gestures - Could incorporate jumping or clapping