Music Visualization with Audio Beat-Matching

Maggie Reagan and Liz Schell

1. Overview

The goal for this project is to implement music visualization on a computer monitor using audio beat detection for real-time music playing. Most commercially available applications for this purpose are implemented in software, and are also limited in the visuals they provide. Hardware implementations that exist for this purpose are entirely custom, and are even more visually limited; the most common music visualization is done with LED strips. This project would offer a hardware implementation for music beat detection and a more interesting and visually pleasing visualization than currently exists on the market.

Our project has two major parts: audio beat detection and visual generator. The beat detection output will be as an input to a visual display generator that will have several stages. The first stage of the visual generator will rotate nested shapes on the screen. If we find that we have enough time, we will make the visual generator more complex by adding a fractal generator module to produce a background fractal image to put behind our simple image with nested shapes.

2. Design Decisions

2.1 Project Overview

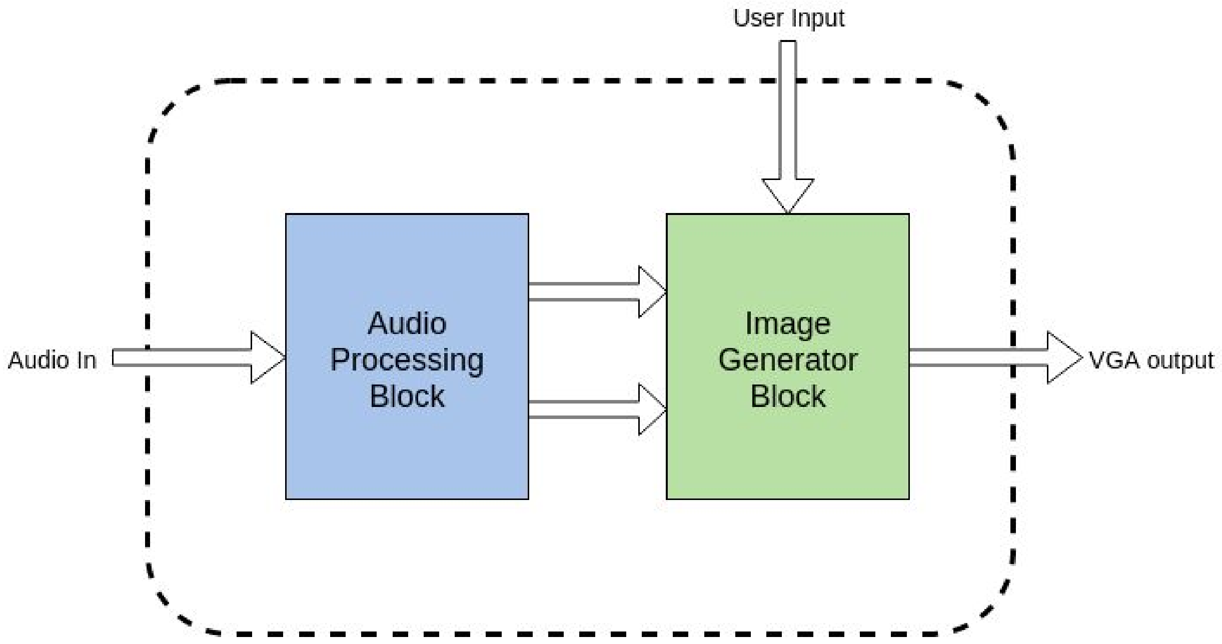

Figure 1: High-level block diagram of the system

The project as a whole can be modeled as two major blocks as shown in Figure 1: the audio processing block and the image generator block. The audio processing block samples musical input from either a microphone or direct audio input and outputs the tempo and phase of the song playing. These two outputs are given to the visual generation block, which changes the color of the display based on the phase and the speed of the rotation of the nested shapes based on the tempo.

2.2 Design Decisions and Motivation

We chose this project because we enjoy music and thought that it would be interesting to implement a colorful and unique visualization to go with our favorite music. In order for this project to be useful for a party or social event, we wanted to be able to process a song in real time or as close to real time as possible. We decided to go with an algorithm that takes a 2.2 second sample of a song to start generating the tempo and phase, to allow for beat detection as low as 60 bpm with some wiggle room. As new music samples come in, we will keep calculating the phase and tempo with the new data to keep up with changing tempos.

The musical visualization we will be implementing will have two parts, one of which is a stretch goal. The main visualization will be a series of rotating cubes and circles which rotate at a speed correlated to the tempo detected from the audio block. The colors of the cube and circle outline will also change color in time with the music; i.e. at every other beat. The color change will occur at every beat or every other beat depending on the tempo of the song; for some faster songs, what is most visually pleasing might be a slower color change rather than a strobe color effect. We decided on this as our main visualization because we feel that the tempo matching the rotation speed is an intuitive connection for someone viewing the visualization. The multiple color-outlined shapes we decided on allow us to create a rainbow, color shifting effect, while the black between the outlines keep the image from looking garish or overwhelming. The following image, Figure 2, is what we plan for the final image to look like.

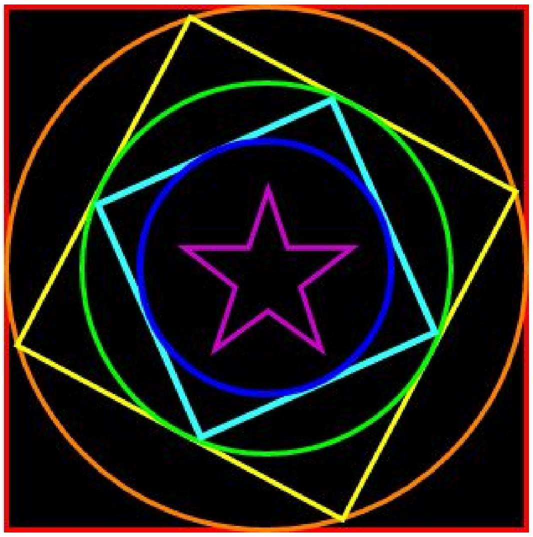

Figure 2: Music Visualization

The stretch goal for this music visualization will be a similar image, except instead of overlaying a black background, the rotating circles and squares will overlay a Mandelbrot fractal. We chose this as a background image because it is both computationally intense and also very interesting visually. This background would be non-moving for the most part, although would be able to be changed based on user input with some delay to zoom and pan left, right, up, and down. This is defined as a stretch goal because combining it with the moving music visualization will be fairly simple alpha blending, and because on its own it is a fairly complex mathematical calculation that will require significant infrastructure and planning, especially with regards to memory, which will be explained more in detail later on in this section. It is also a slow calculation; adding a moving aspect that updates in real-time with music would require more resources and time than is feasible in combination with the other aspects of this project. Figure 3 is an example of what the Mandelbrot fractal would look like.

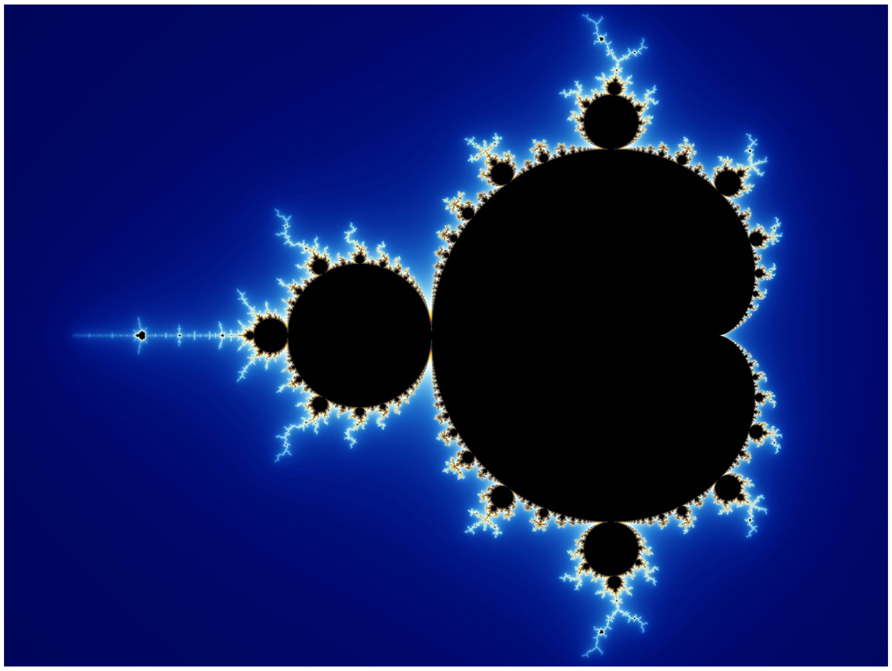

Figure 3: Mandelbrot Fractal

One of the main challenges we were faced with in coming up with this design was available memory for storing images; while we could use a lower quality image, such as a 640x480 pixel image or lower, and use less memory, this would not be as visually pleasing as a high quality image. In addition, we want to be able to update the image in real-time or almost real-time; therefore, as a compromise between high image quality and ability to update the image quickly, we decided to use SVGA quality, or 800x600 16-bit pixels. Storing a current and next image on the FPGA will involve using both ZBT SRAMs, and possibly an additional BRAM buffer for updating in real time. In addition, we need to separately store a non-rotating fractal background image. Because this is a stretch goal, there is still some flexibility with how this will be implemented. We could use flash memory and a BRAM buffer that stores part of the image at a time, which is possibly too slow due to flash read and write times, or forego any memory other than a BRAM holding part of the image, which is possibly more slow than the first option due to computation times. Currently, our plan is to use flash memory and BRAM buffer, which is the more structurally difficult of the two options, allowing us to fall back on the second option of foregoing flash memory if necessary.

3. Implementation

3.1 Audio Processing Block

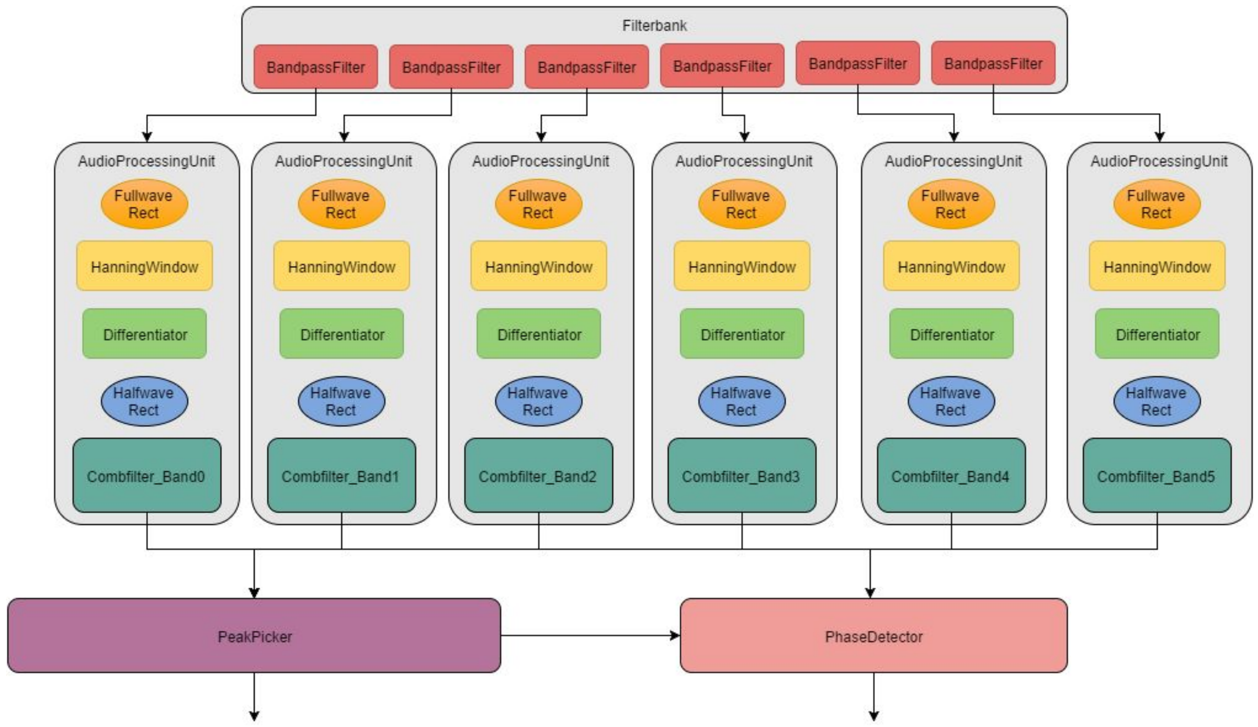

Figure 4: Audio Processing Block Diagram

The purpose of the audio block is to process the input audio and output the frequency and phase of the audio to the visual block. The frequency tells the visual block what the tempo of the music is, and the phase tells the visual block how long it will be until the next beat so that the visual block can generate images in phase with the audio. To do this, the input audio is first split up into six different frequency bands by the frequency filterbank so that they can be processed separately. Each of these different bands is sent to its own audio processing unit. The audio processing unit contains an envelope extractor, a differentiator, a half-wave rectifier, and then a comb filter. These first few modules will accentuate the tempos in the signal, and then the comb filters act as resonators for different possible frequencies of the audio. After the comb filters, we calculate the energy of their outputs to find the energy of different frequencies within the band that is being processed by that audio processing unit. These energies are sent on to the peak-picker, which compares the different energies to determine which frequency is the fundamental tempo of the audio. This will be the tempo whose comb filter output has the highest energy.

Once the fundamental tempo has been found, the phase detector will take in the output of the comb filters and the tempo of the audio. It will look at the signal from the comb filter that corresponds to the calculated tempo and determine the phase of this signal. Using this, it will predict the time until the next beat, which will be sent onto the visual generator in the form of a phase shift.

3.2 Image Generator Block

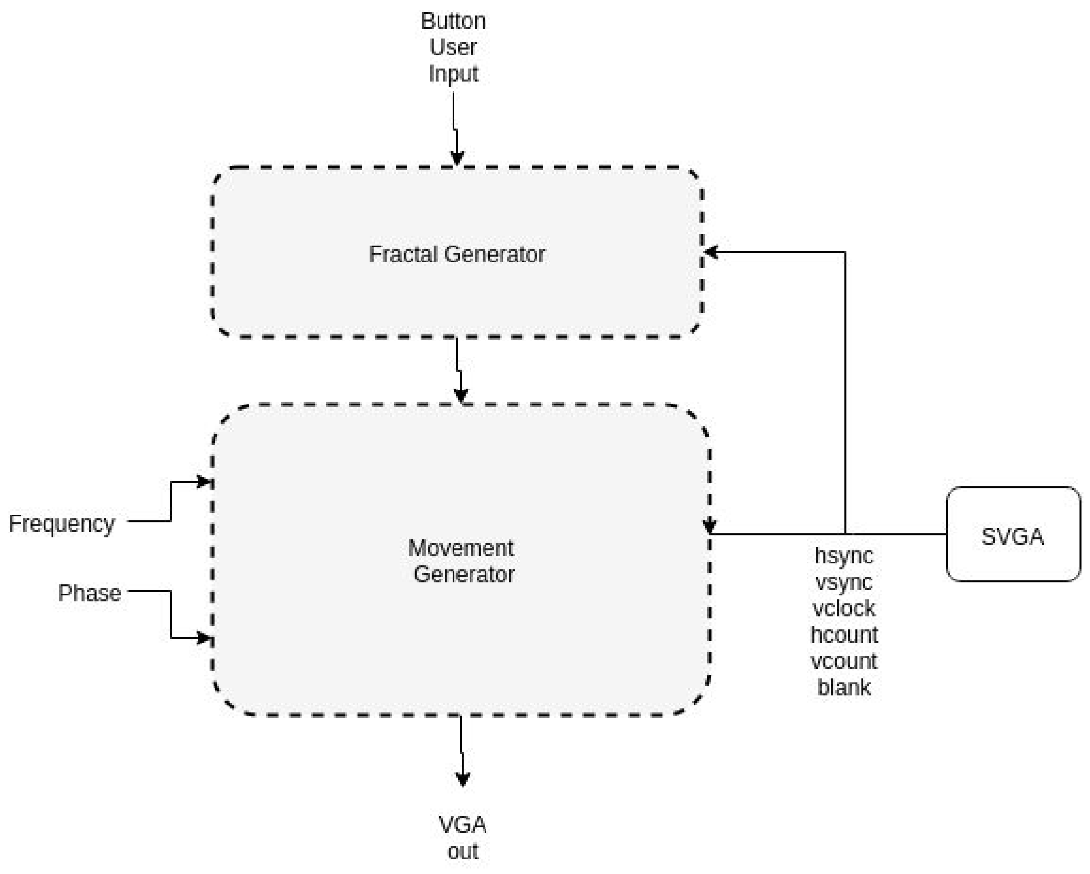

Figure 5: Image generator block

The visual processing block will take in the frequency, or tempo, of the song and the phase, or time until next beat, from the audio processing block. The tempo or frequency will be used to calculate the speed at which the foreground square (see Figure 2) rotates. The phase will determine exact time of the color change-- that is, when phase is zero, the color shift will occur. The VGA block supplies the necessary inputs to the movement generator block such as hsync, vsync, vclock, hcount, and vcount. We are choosing to use SVGA resolution (800x600 pixels) rather than XVGA due to the amount of memory required to store the fractal image. The movement generator will take in the pixel values generated by the fractal generator and implement alpha blending to combine the two images. The movement generator block will contain several submodules, as shown in Figure 6.

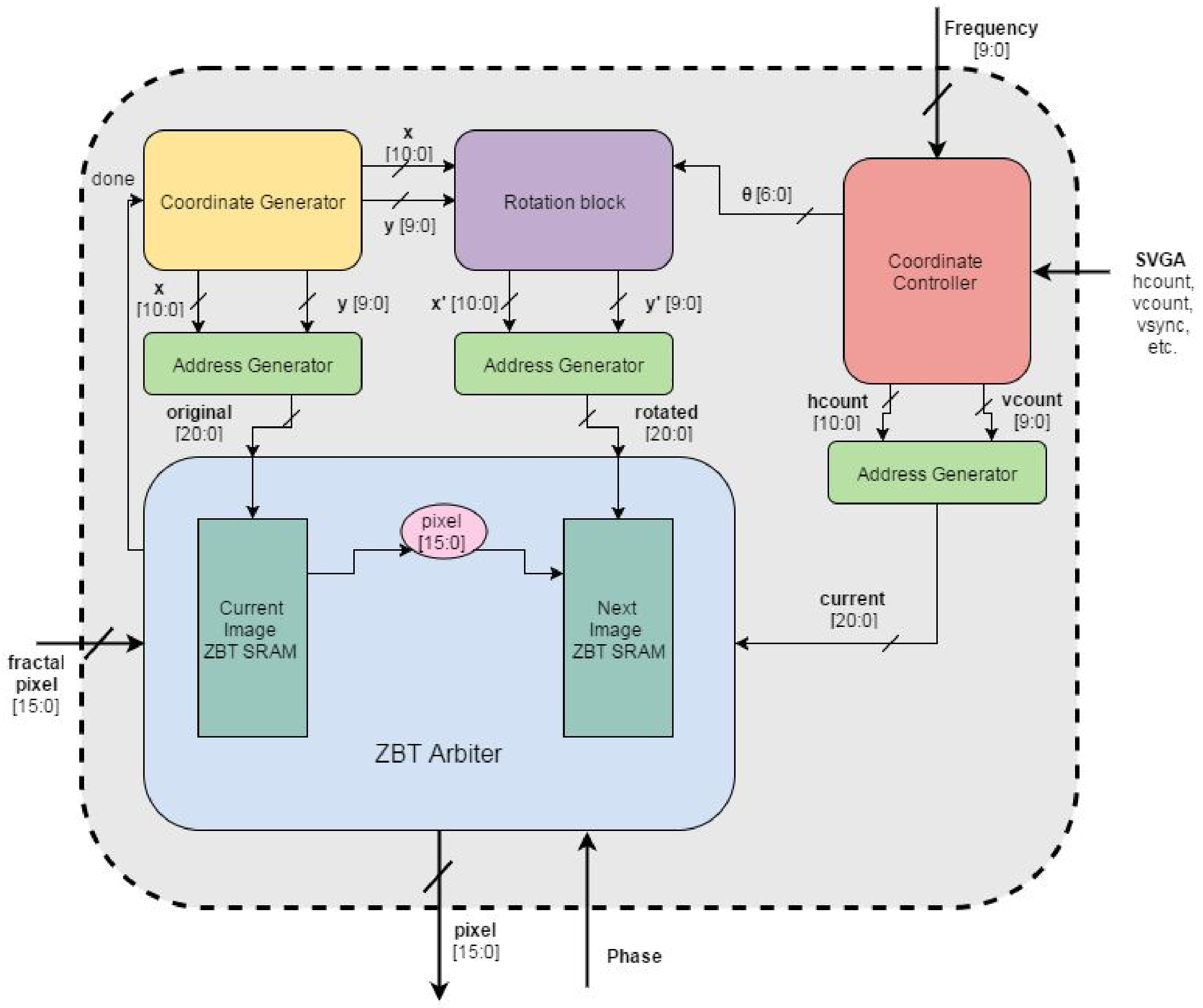

Figure 6: Movement Generator

The movement generator will comprise of a coordinate generator, several address generators, a rotation block, a coordinate controller, and a ZBT arbiter. The coordinate generator generates the next x-y coordinate to be rotated, and sends that to the rotation block. The rotation block performs the rotation of the x-y coordinate by multiplying the given x-y value by sinθ or cosθ values stored in LUTs. The angle θ by which the image is rotated by is given by the coordinate controller module, which maps the input frequency or tempo of the song given by the audio processing block to a θ value. The original and rotated image coordinates are passed through an address generator and given to the ZBT Arbiter, where the pixel value at the original coordinate address is stored at the rotated coordinate address in two ZBT SRAMs. This means that we will only be reading from one ZBT (the current image) and only writing to one ZBT (the next image) at a time. The hcount and vcount from the SVGA block are given to the coordinate controller, which again sends this through an address generator to be given to the ZBT arbiter, which is able to read the current pixel from the current image ZBT. If we are able to implement fractal generation, the fractal pixel input to the ZBT Arbiter is alpha-blended with the current moving pixel, and outputted to the monitor. The phase input to the ZBT arbiter will cause a color shift in the pixel value; that is, on the rising edge of the phase input we will implement a color shift by adding or subtracting to the pixel value.

The fractal generator block will contain several submodules, as shown in Figure 7.

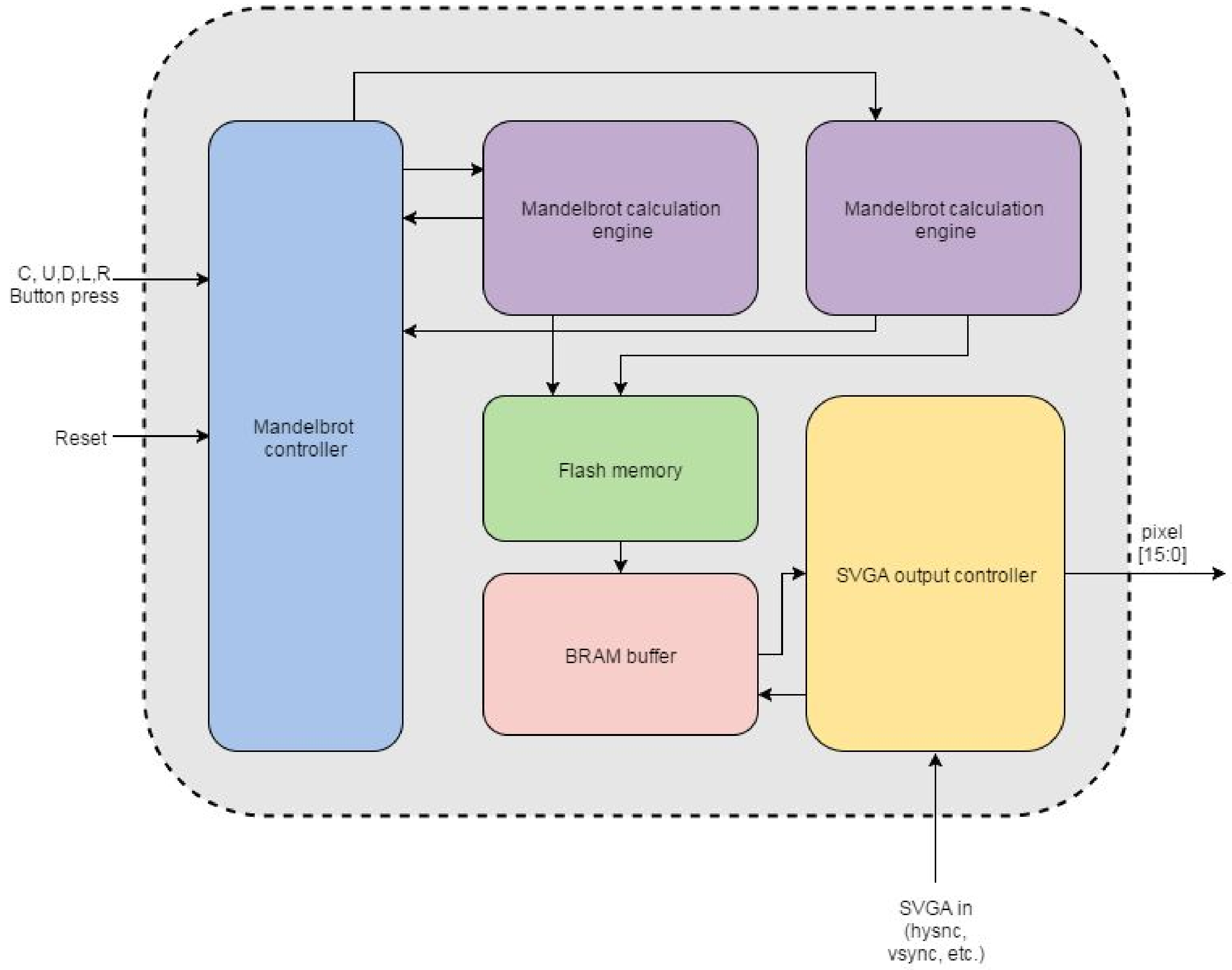

Figure 7: Background Fractal Image generator

Each mandelbrot calculation engine will take in the next x-y coordinate provided by the Mandelbrot controller, and convert this x-y coordinate into Mandelbrot coordinates. Once converted into Mandelbrot coordinates, the calculation will perform the iterative Mandelbrot calculation. The number of iterations performed until a condition is met determines the color of this pixel. The color of this pixel is then stored into flash memory, and the Mandelbrot controller provides the next pixel x-y coordinate to be determined. The SVGA output controller reads pixel values from the BRAM buffer at the appropriate 60Hz frame refresh rate. Because the calculations for the Mandelbrot background will take a significant amount of time, any user input causing a change in the background image will have a noticeable delay. The Mandelbrot controller handles any user input such as zooming and panning, providing updated parameters for the Mandelbrot coordinate conversion and pixel color calculation.

4. Timeline, Testing, and Division of Responsibility

	11/2	11/9	11/16	11/23	11/30	12/7
1) Finalize block diagram and shared signals	🟪					
2a) Write and test filterbank and audio processing unit	🟦	🟦				
2b) Write and test coordinate controller address generator, coordinate generator, and rotation block	🟩	🟩				
3a) Write and test peak-picker and phase detector		🟦	🟦			
3b) Write and test ZBT arbiter		🟩	🟩			
4)Integrate audio and visual subsystems			🟪	🟪		
5a) Work on stretch goals: Mandelbrot controller and calculation engine				🟪	🟪	
5b) Work on stretch goals: Memory architecture				🟪	🟪	
6) Debugging					🟪	🟪
7) Final report, video, and checkoff						🟪

Figure 8: Proposed Schedule

The above figure shows our proposed schedule of milestones for completing this project, with Liz's tasks in blue, Maggie's tasks in green, and shared tasks in purple.. In this section, we'll explain the finer details of these milestones.

1) First we will finalize a high level block diagram of the entire system and lower level block diagrams of each the visual and audio blocks. We will decide on signals that are shared between the two blocks so that each block can be written and tested separately before integration.

2) Then we will begin the development of the two large blocks by writing and testing the modules that they're composed of. For Liz, this starts with the filterbank and audio processing unit. For Maggie, this means the coordinate controller and generator, the address generator, and the rotation block.
3) We will then move on to writing and testing the peak-picker and phase detector for Liz, and the ZBT arbiter module for Maggie.
4) Next, the full audio and visual blocks will be integrated individually and then integrated together to create the full system. Once basic design goals are working, we can begin working on our stretch goals.
5) The stretch goals will involve working on the implementation of the Mandelbrot set algorithm itself, and the memory architecture for storing pixels and streaming the video output.
6) The last full week will be dedicated to buffer room and debugging of the full system.
7) Finally, we will get our project checkoff, record the demo video, and finish up our project report.

The division of responsibility is fairly simple; Liz will be working on the audio processing block, while Maggie will be working on the image generation block. We will both work on implementing the stretch goal of the Mandelbrot Set generator.

5. Resources

Our project doesn't require much hardware other than the supplied labkit and computer monitor. We can supply the audio input with either an aux cable into the labkit or by a microphone connected to the labkit. These parts can be easily bought or acquired on their own.

6. Conclusion

This project is technically challenging as well as interesting. The overall goal of this project is to implement a music visualizer that will display interesting images while either listening to music casually or during a party. Both of us have seen beat-matching implemented in LED displays for usage during parties; we wanted to come up with a more technically challenging and visually interesting alternative to this. The parts of our project are also versatile, because the audio beat detector could be paired with other visualization generators for different effects, and the visualization generator could be set up to change with something other than music. We hope to build something that we can use frequently for parties or other social events.