Music Visualizer Proposal: Bradley Edwards, Aston Motes, and Stephen Oney

Abstract:
Our project seeks to produce a system for visualizing arbitrary audio sources in real time. These visualizations will, given a frequency analysis of the audio signal, display a visually appealing image on a television, which appears to synchronize with the music. Our system will comprise a number of subsystems including: an analog to digital converter, Fourier transformations, video output, and a processor, running a high level programming language and other lesser important necessities.

The analog to digital converter will provide the discrete samples to use in our system. The input to the ADC will be from a portable audio source, like an iPod or a CD player. We are looking at various options for the ADC, including the ADC on the 6.111 kit as included in the ac97 chip. The ADC will feed its discrete outputs as inputs to a Fast Fourier Transform module; however, the bridge to communicate between these two devices will need to be created. Another module will satisfy this condition. This bridge module will perform possible computation of the discrete signals and provide an output that will be fed directly into the FFT.

There are a number of options for the FFT. Using the FFT module from the Xilinx IP Core Generator is the first option, and probably the best one. We will research other creating the FFT from scratch. A simple test of the FFT will involve inputting a sine wave and making sure that we get a spike on the output. The output of the FFT will be fed into a bucketizer that normalizes the frequency response, and groups the outputs based on certain ranges of data. The buckets feed into an input buffer for the CPU.

44100Hz audio sampled at 44100 Hz

We need 1/44100 = 2.267x10-5 s as the frequency of the data fetch

If we had a 1MHz processor, we would have 1x10-6 seconds per cycle, meaning about 100 clock cycles to do processing, so processing power should not be an issue.

The input buffer uses DRAM memory, and only needs to be able to store enough for the amount of buckets, assuming the CPU can handle the speed requirements. The memory will store the current state, to let the CPU fetch it at any given time to do its processing. The speed of the memory has to be above (44100 Hz)*(the number of buckets) per read AND right operation.

The input buffer will only store the state of the buckets for one audio sample. Once a new sample is ready, then the input buffer will clear, and remember the new value, without being completely sure that the CPU was able to read the previous value.

The memory is easy to test. Once we have a set amount of buckets, we can simply

test it by writing arbitrary data to it and reading that data from it, keeping a check to make sure that the original data we wrote is the data that comes out of this memory module.

The CPU will be based on verilog code for some core. This way, we can compile C code for the visualizer into assembly language for the CPU. Right now, the RISC instruction set is the most likely core, as there is verilog code for a limited RISC code set on the web site opencores.org. The main constraint on the CPU is that the processing for the visualization takes under 100*(CPU clock speed to the visualization). This was shown in
the previous calculation

The CPU will read the value of each bucket from the input buffer and output the appropriate data for the visualization. It can take in inputs to decide which input to use. For example, we might use the switches to control how any given visualization looks.

Testing the CPU will be one of the most difficult aspects. The CPU is the hardest component to test. Once we get the architecture set up, we have to try out different C compilers, such as gcc, to make sure that we can write C code for our compiler.

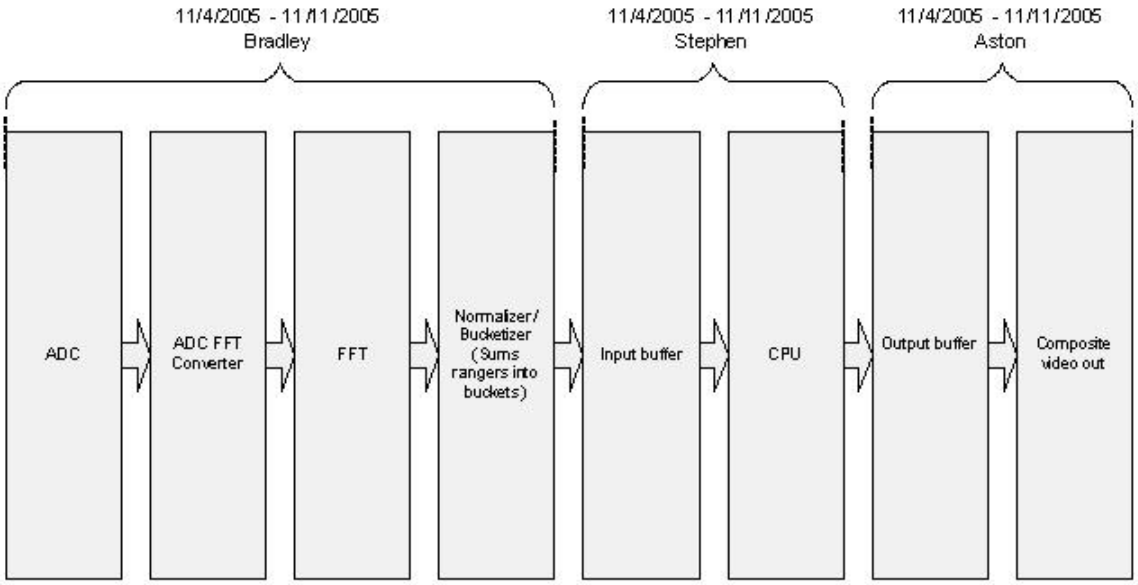
The composite video output module is responsible for producing the video output for our system. We will be using the ADV7194 Video Encoder chip which has been included as part of the Labkit to do the hard work of proper digital to analog conversion and formatting the output for the NTSC televisions we plan our system to work with. In addition to this chip, this module will also contain a submodule responsible for producing proper horizontal sync, vertical sync, and blanking signals as well as converting the colors from RGB (as the CPU for the system uses) to YCrCb format for pixel data.

The pixel data for the video output will be stored in a shared dual port memory between the CPU and the video output module, with each pixel on screen as a memory location and the width of the memory corresponding with our pixel color depth. Because the CPU is producing pixel values which must be read while still valid by the video output module, some synchronization between the video out and the CPU is needed, whether it be a signal that a line has finished drawing (and so the CPU can fill in that line) or perhaps a signal noting the full screen blanking, so that the CPU can refresh the entire memory, assuming this period is long enough. In the meantime, the CPU can hold the pixel data in a self contained temporary memory block before transferring it to the shared location.

In terms of timing, the video output DAC typically runs at 27Mhz, so having a memory which allows reads at this speed is likely a good idea in order to keep the pipeline filled between the memory and the video output. By pushing data as quickly as possible, more time is opened up for the CPU to place data into the memory in the period of time

between the start of one frame drawing and the start of the next given the non-drawing periods and knowledge about where the horizontal and vertical traces are located.

Testing of the video output module and accompanying memory will be done in stages to ensure correctness. The first step will be to attempt to create a simple blank (all black) screen which an NTSC television accepts as a proper source, then try various monochromatic fills to ensure that the color conversion works correctly. Also important will be a screen test which places a one pixel border around the outside of the screen to check the small but crucial boundary cases. Once these basic video tests are done, filling the memory with bitmapped graphics (created by hand on a PC) should allow the showing of arbitrary pictures on the screen. From here, there is the issue of integration with the CPU that must be handled, which will most entail testing of proper input/output timing such that data is not overwritten before being read but the frame rate of the total system is acceptable to the human eye. This will likely involve simple animation in addition to basic output.


11/4/2005 - 11/11/2005
Bradley
11/4/2005 - 11/11/2005
Stephen
11/4/2005 - 11/11/2005
Aston
ADC
ADC FFT Converter
FFT
Normalizer/ Bucketizer (Sums rangers into buckets)
Input buffer
CPU
Output buffer
Composite video out