Programmable Audio Visualizer

Dany Qumsiyeh, Mike Spindel

November 6, 2005

Abstract

This project is a programmable audio visualizer that can display synchronized graphics on multiple displays. It is capable of executing a user-specified program each frame that can contain mathematical functions as well as configuration commands for a set of special purpose graphics rendering hardware. The implementation is composed of three parts: a frontend audio processing module, a specialized microprocessor, and an output display module. The frontend processor maintains a 1s audio sample FIFO and executes an FFT and beat-detection algorithm. This data is then available to the the graphics processor, which executes a user-defined program to render the visualization to a framebuffer. The output module maps parts of the framebuffer to appropriate external displays.

Contents

List of Figures

1 Description

1.1 Display

The system is designed to support multiple synchronized displays. This will be done by having the effects engine operate on a single, large, virtual buffer. Displays are then mapped onto sections of this buffer with positions that reflect their relative physical locations. Even with a single display, having a larger virtual buffer hides edge effects and allows patterns to emerge from off-screen.

1.2 Memory

The system will maintain two large virtual buffers in one or more SRAM chips. While the display modules read from one buffer, the engine writes to the other, and the mapping switches when a frame is completed. The mapping from global coordinates to the DRAMs is handled by a memory_manager module. This module provides read ports (in the pixel coordinate system) for each display module, and both read and write ports for the effects module. To meet SRAM port limitations, restrictions can be enforced such has having displays not share a single SRAM module.

1.3 Processor

Effects are created by a processor which loops through instructions in BRAM. Example code that could be compiled is shown in figure 1. The processor can execute the usual arithmetic commands, along with a simple "if" that can skip a certain number of instructions. The special commands "convolve" and "generate", however, run long operations on the buffers in memory, during which pc is halted. Predefined registers supply the parameters for these commands.

1.4 Effects

The system will be able to apply a variety of programmable effects to the buffer for every frame. In general, a convolution can be applied, and any number of generators used, which can overlay patterns and waveforms on the screen. A number of motion effects (such as rotate, translate, and zoom) can

Figure 1: Example pseudo-code.

```
//initialize
if (resetbutton) skip 1
rANGLE <= 0

//configure motion effects
reg(motion1type) <= (rotate)
reg(motion1param1) <= rANGLE
reg(motion2type) <= (translate)
reg(motion1param1) <= rX
reg(motion1param2) <= rY

//specify convolution filter
reg(convolve00) <= 1
reg(convolve01) <= 2
...

//do the convolution
convolve

//configure parallel generators
reg(gen1type) <= (waveform)
reg(gen1param1) <= rSIZE
reg(gen2type) <= (scope)

//run generators
generate

//configure generators again
reg(gen1type) <= (circle)
reg(gen2type) <= (disabled)

//run new generators
generate

//vary parameters
rANGLE <= rANGLE + 1 //spin
r1 <= (rX < rY)
if (r1) skip 1
r3 <= r1 * r2
//etc
```

modify the convolution and generators, with the limitation of the number of motion module instances. The 2D convolution filter is small, but fully specified by user registers, and can implement effects such as blur or edge-detect.

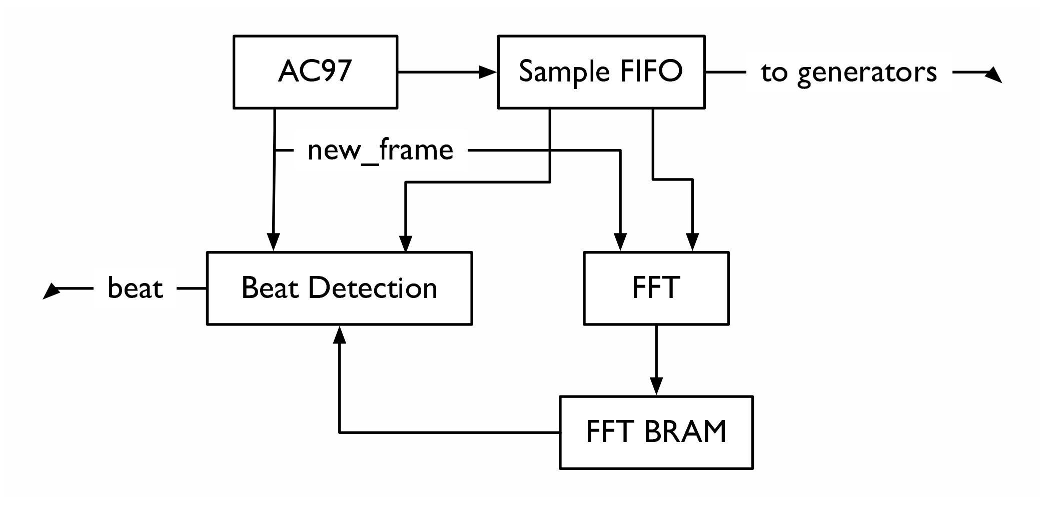

Figure 2: Preprocessing Block Diagram

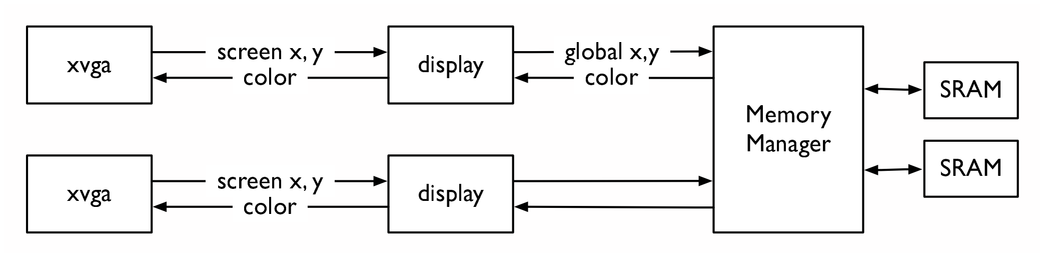

Figure 3: Display Block Diagram

2 Module Descriptions

2.1 Pre-processing Block

ac97 sound module provides samples at regular intervals.

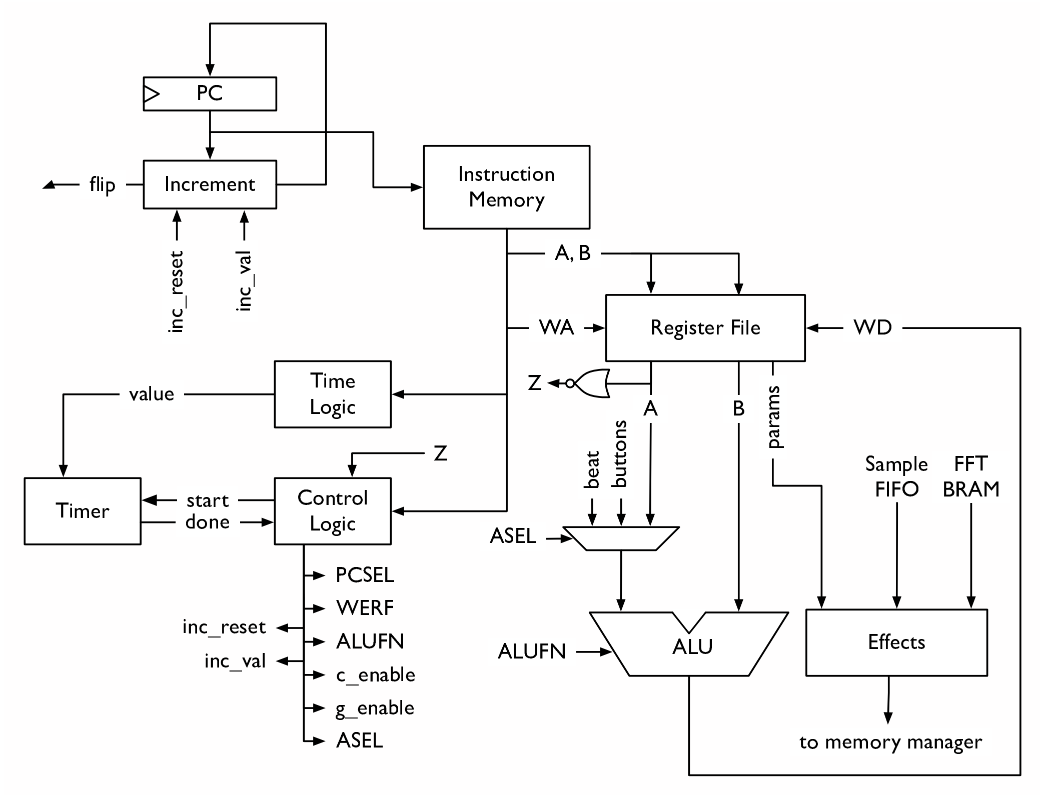

Figure 4: Processor Block Diagram

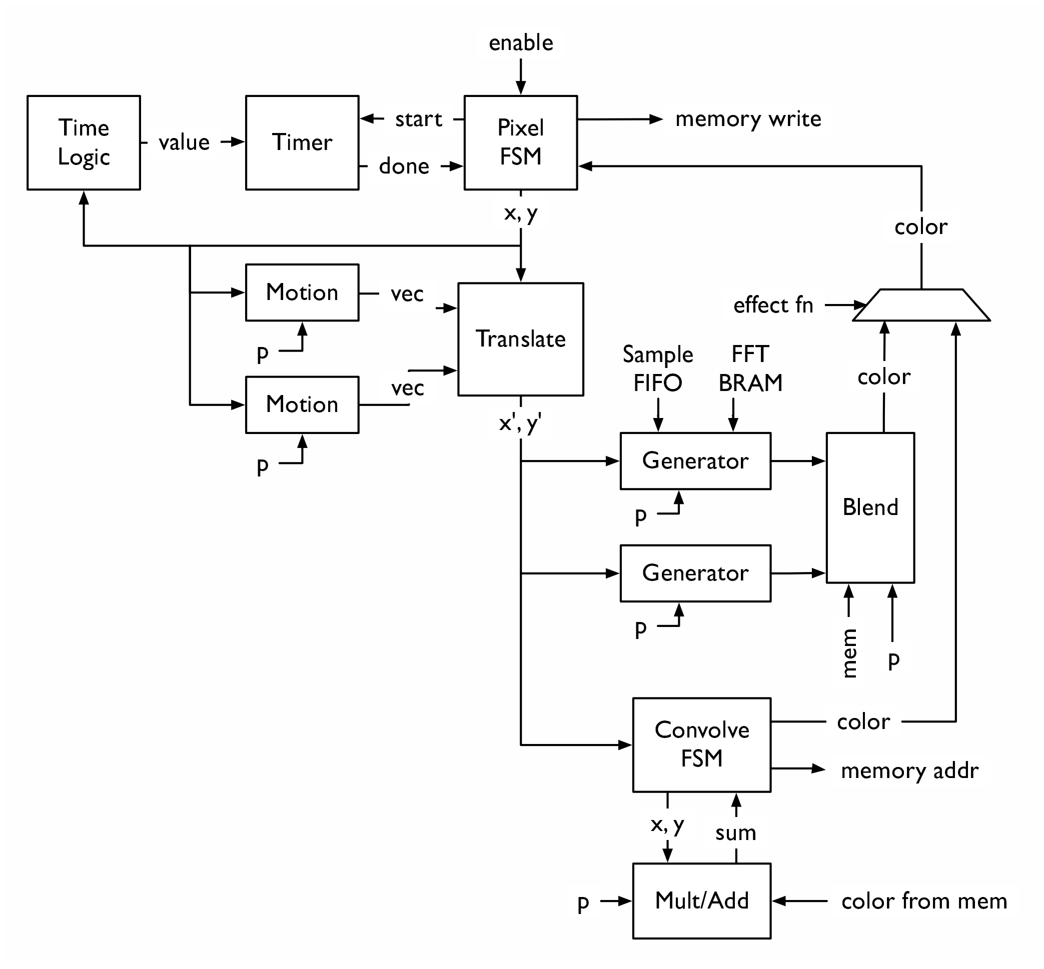

Figure 5: Effects Module Diagram

sample_fifo maintains a running buffer of the most recent samples in BRAM, and provides multiple read ports for fft, beat_detect, and generator modules.

fft does fft on the sample buffer on every new_frame signal, writing the results to fft_bram.

beat_detect reads data from fft_bram and sample_buffer and produces a beat value for the current sample.

2.2 Display Block

xvga[n] display interface modules which request pixels in sequence and take in a color value.

display[n] translates screen coordinates to absolute coordinates in the global buffer.

memory_manager translates read and write requests for global buffer coordinates to the correct SRAM requests.

2.3 Processor Block

pc register holds the value of the current instruction address.

increment increments pc by inc_val. if inc_reset is asserted (when an empty instruction is encountered), it resets pc to the first instruction, and toggles the value of flip (used by the memory_manager to determine which buffer is which).

instruction_mem BRAM holding the visualizer program.

time_logic interprets the current instruction and outputs how many clock cycles the instruction will take.

timer times out the value produced by time_logic, to hold pc still while long computations take place.

control_logic maps the current instruction to the listed control signals.

register_file register file with one write port and many read ports. Two read addresses come from the current instruction, and the others are fixed to the configuration registers for the convolve and generator modules.

asel mux chooses whether to substitute inputs such as beat or button values for the register A value.

alu the usual arithmetic functions, with the result written to a register.

effects takes parameters from the fixed read ports of register_file and communicates with memory_manager to update the buffer in memory. submodules described below.

2.4 Effects Module

timer times out the value produced by time_logic.

time_logic takes parameters from register_file and the current coordinates, and outputs how many clock cycles are required for the convolution or generators to finish.

translate adds a translation vector to the pixel coordinates, obtained by summing the vectors produced by the motion modules.

motion[n] takes parameters from register_file and coordinates from pixel_fsm, and produces the vector for the motion effect requested.

generator[n] takes parameters from register_file, and uses data from fft_bram or sample_buffer to produce a color for the current pixel coordinates.

blend combines the colors from the generator[n] modules with the current pixel from memory_manager based on the parameters.

conv_fsm steps through the coordinates nearby the current pixel, for doing the convolution. It expects a final convolution sum at the end, and passes that color value up to pixel_fsm. It requests from memory_manager the old pixel values, which are actually received by mult_add.

mult_add takes the pixel value from memory_manager, and offset coordinates. based on the parameters from register_file, multiplies the pixel by the appropriate value and outputs the running sum.

3 Division of Labor

Dany Qumsiyeh will write the display blocks, and processor blocks (without effects). Mike Spindel will write the preprocessor blocks, and effects blocks. We will share writing motion and generator modules.