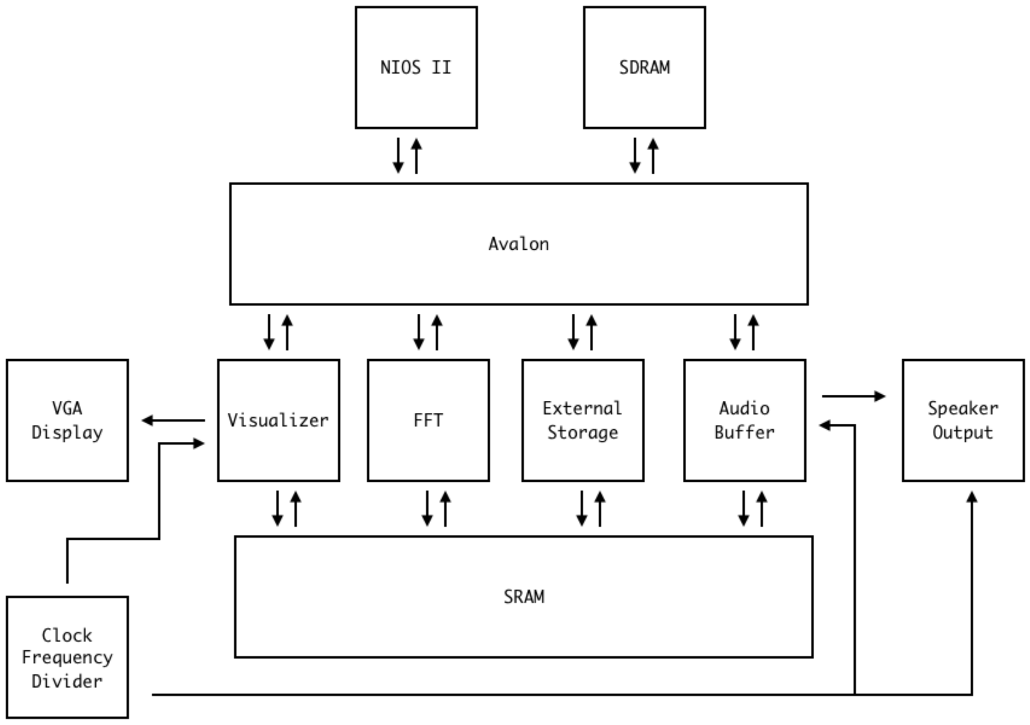
Kanto Audio Player - Design

David Benedetto - djb2167, Kavita Jain-Cocks - kj2264, Amrita Mazumdar - am3210
Zhehao Mao - zm2169, Darien Nurse - don2102, Jonathan Yu - jy2432

Project Overview:
Our project will be an audio player with an interesting visualization of the audio. The user will be able to play .wav audio files from an SD card. We will use a hardware-accelerated algorithm to play the audio on the FPGA's audio output and simultaneously display a visualization of the audio being played using a custom audio encoding. The user interface will be very minimal and allow the user to start and stop playback only.

High-Level Block Diagram:



Memory Requirements

- FFT will be computed using a radix-16 Cooley-Tukey FFT algorithm, using on-chip block RAM for storing coefficients. A radix-16 FFT requires 4*((n/16)^2 + 4n) bytes, as each coefficient is 4 bytes (2 bytes each for real and imaginary part), and if we use n = 256, our FFT will fit in 5.38 KB of block RAM.
- We will need to store the result of the FFT in SRAM so that the visualizer can access it. This will take 256 * 2 = 512 bytes.
- We are taking 44100 samples/sec, and we hope to keep 1 second of samples in the

audio buffer at any given cycle, or 88.2 KB of audio data in the audio buffer. This will be stored in the SRAM.

- The visualizer will use on-chip block RAM for computation. We are going to have a rudimentary FFT bar graph visualization, which will show the amplitude of each frequency as the song plays. We intend to show 256 frequency slots with amplitude levels ranging from 0-15, so we will use 4*256/8 = 128 bytes for the visualizer. We will also need some overhead for VGA configuration storage, which is trivial enough to also be included in on-chip block RAM.
- The clock frequency divider will need to store an internal count, up to 1134, (50 MHz / 44.1 kHz = 1134) so we will need 11 bits of on-chip block RAM to maintain this counter.
- We will use SDRAM for the NIOS II data and instruction memory so as to free up the SRAM for use by the hardware peripherals

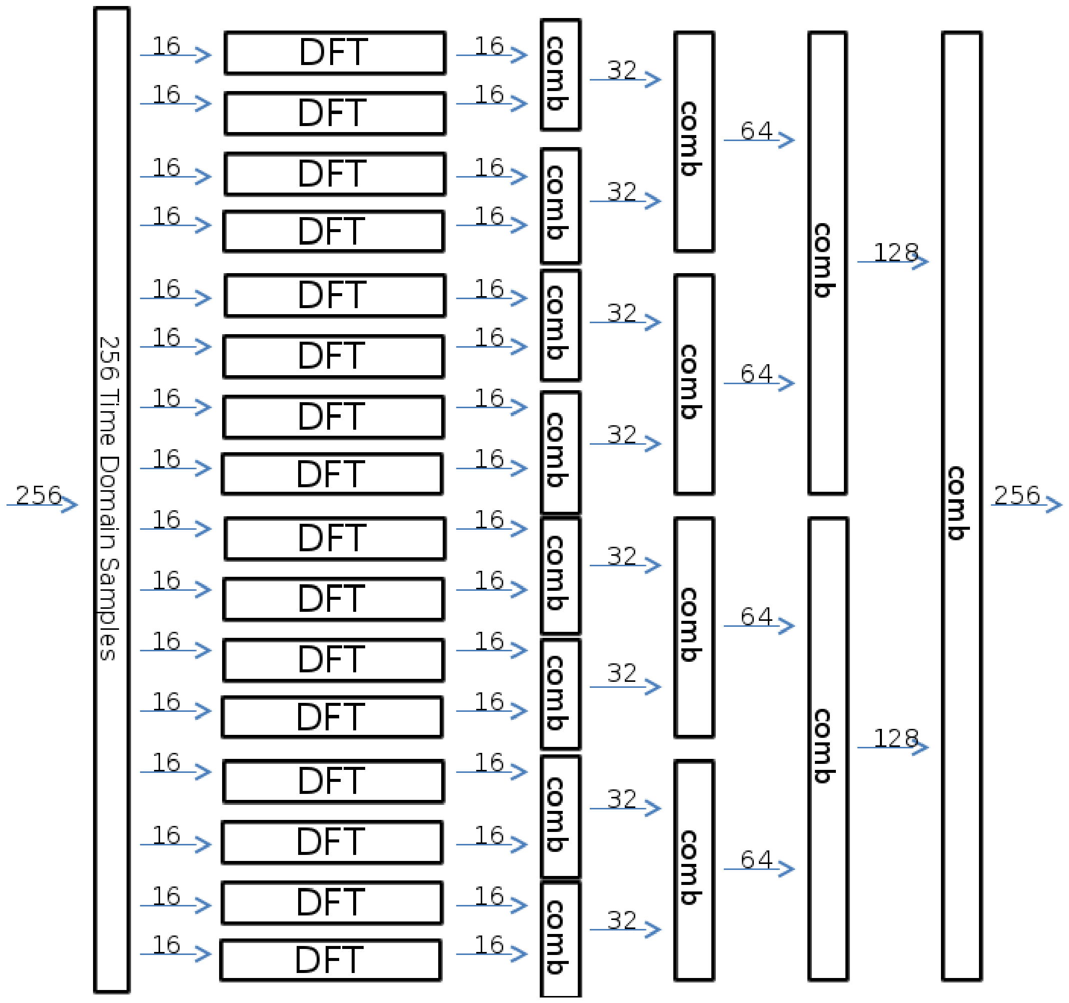
Critical Path:

The most computationally-intensive operation we do in one clock cycle would be a combinational multiplication in the FFT unit. This would thus be our critical path.

As for the overall latency of our system, the longest path would be processing samples from the SD Card to display on the VGA display, as this would require the data to pass through the storage controller, the FFT unit, and the visualization unit.

Peripherals:

- FFT Unit
    - Block Diagram



This is a radix-16 DFT, so we split the inputs into 16 different parts and perform a DFT on each group according to the equation

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i2\pi k \frac{n}{N}} \qquad k = 0, \dots, N-1.$$

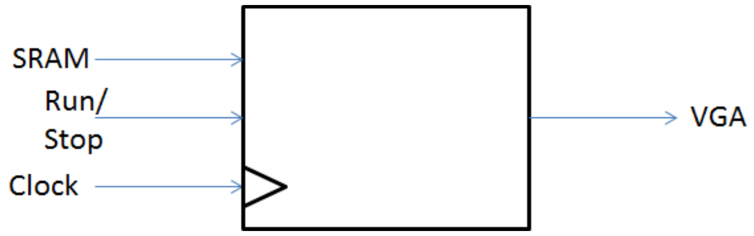
We then recombine the individual DFT outputs into the final outputs in 4 stages, using the following equation for each stage

$$X_k = \begin{cases} E_k + e^{-\frac{2\pi i}{N}k} O_k & \text{if } k < N/2 \\ E_{k-N/2} - e^{-\frac{2\pi i}{N}(k-N/2)} O_{k-N/2} & \text{if } k \geq N/2. \end{cases}$$

The numbers on each of the arrows indicate the number of samples being transferred, not the number of bits. In the first stage (direct from input), the samples are 16 bit (audio samples). In the next few stages, samples are 32 bits (16 bits each for real and imaginary parts).

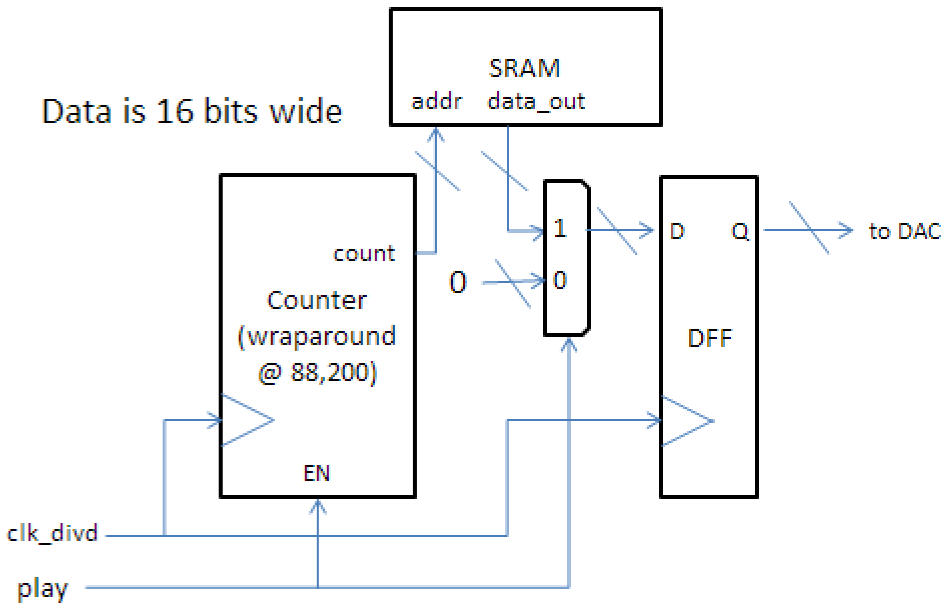
- I/O Locations
    - NIOS II Signals
        - read address (where in SRAM)
        - begin computation (control)
        - computation complete (status)
    - External Signals
        - SRAM signals (read time domain, write frequency domain)

- Visualizer
    - Block Diagram

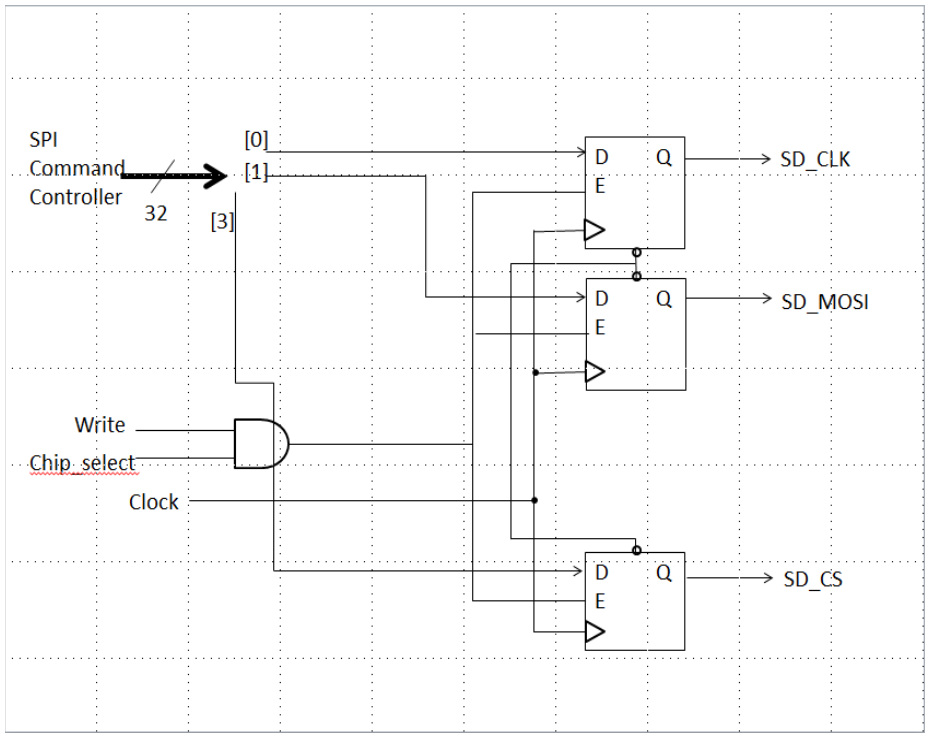


This portion is similar in implementation to the bouncing ball of lab 3. We will refresh this with the frequency-divided clock signal so it is synchronous with the music. The horizontal direction is divided into 256 sections for the frequency blocks, each of these blocks will get a bar whose height will be decided by the amplitude. The frequency and amplitude will be found in the SRAM and portrayed on the screen.

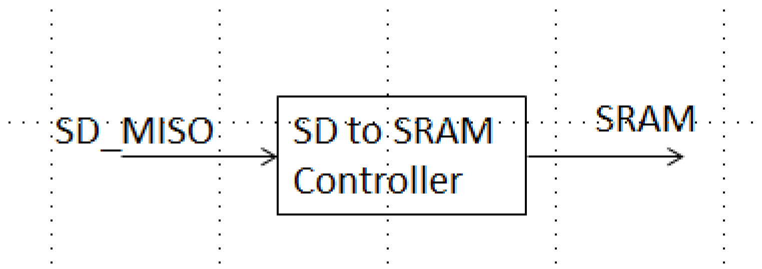
    - I/O Locations
        - NIOS II Signals
            - Run/Stop
        - External Signals
            - SRAM signals (read frequency domain)
            - Clock
- Audio Buffer
    - Block Diagram - data is 16 bit



- I/O Locations
    - NIOS II Signals
        - Pause/Play
    - External Signals
        - SRAM signals (read audio samples)
        - Clock
- SD Controller
    - Block Diagram



SPI Command Controller sends the appropriate initialization commands to the SD card from Nios II.



SD_MISO comes from the SD Card. Normally this goes into the SPI interface however in this case we are writing it straight to SRAM instead.

- I/O Locations
    - Nios II Signals
        - read address (where in storage device)
        - counter (number of bytes to read)
        - write address (where in sram)
        - begin read
    - External Signals
        - SD Card SPI signals
        - SRAM signals (writing bytes read)

Milestones:

- Milestone 1
    - RTL level design of all peripherals
- Milestone 2
    - Have all individual peripherals finished (written in VHDL)
- Milestone 3
    - Build interfaces between all peripherals and finish synchronization software