**Audio Sample Player**

Our design features an audio sample player which is able to play back short snippets of sound through an off-chip digital-to-analog converter (DAC). The audio samples are stored in an off-chip flash memory chip accessed via a dedicated, read-only SPI interface. The sounds are triggered by the CPU using memory-mapped IO. The address of the memory-mapped instruction determines which of the sound channels should be triggered. Apart from this simple interface, the sample player operates independently of the main CPU. Please refer to [FIGURE X] to get an idea of how the sample player incorporates with the rest of the system.

Our current implementation supports the simultaneous playback of four independent sound channels. However, there is sufficient bandwidth between the sample player and the flash memory to support more channels if we later deem that to be appropriate.

Although the Cirrus Logic CS4344 DAC supports stereo sound at sample rates of up to 192kHz and a maximum depth of 24 bits, we have opted to use 8-bit samples at a rate of 32kHz. This is a decidedly lo-fi audio system, but we feel that it works well for our target drum machine application and for the kinds of retro video games our system is designed to support. Again, due to the amount of excess SPI bandwidth available, we could opt to increase the fidelity of the system at the cost of reduced total sample time.

**Sound Storage**

Our design uses an 8Mbit flash memory chip to store the sample data (since for our application the memory is read-only, we will refer to it as a ROM). At our target sample rate of 32kHz, this equates to about 32 seconds of total recorded sound. In order to reduce design complexity, the sound clips are equally sized and placed at fixed offsets in the sample ROM. Our design currently limits the length of each sound to 32768 samples, or about 1 second. Therefore, our current setup supports up to 32 individual sounds, placed at 32KB offsets from the start of the ROM.

Since our sounds are stored in a socketed off-chip memory, it is possible to change the sample set by simply swapping the sample ROM. As an interesting historical note, this is actually how some of the earliest digital drum machines were designed and used, until the advent of swappable ROM cards and eventually fully-programmable drum samplers in the mid-1980s.

**Sound Playback**

Each sound clip is stored as a series of signed 8-bit samples in our off-chip ROM. To reproduce the sound, each successive sample needs to be loaded from the ROM and sent the audio DAC at our predefined sample rate of 32kHz. We have implemented a finite state machine to handle this process and to support simultaneous playback of multiple individual sounds.

The “playback position” of a sound dictates the address of the next sample that needs to be loaded. When a sound is triggered, the playback position is reset to the beginning of the clip – this corresponds to address 0 relative to the 32KB ROM offset for each particular sound as discussed in the Sound Storage section. Each sample period, the playback position is incremented by one and playback proceeds through the clip at the predefined sample rate until the end of the 32KB segment is reached. Sounds can be retriggered in the middle of playback by resetting the playback position to 0.

To support playback of multiple independent sounds, it’s necessary to fetch multiple samples each sample period. The FSM handles this fetching and accumulates the individual sample values in a dedicated register. The I2S interface latches this accumulated sample data each sample period and continuously streams the serialized data to the DAC.