Lab 5 - Taylor Cowley and Andrew Okazaki

Chapter 3: Game Audio

Section 2.1: WAV Files and the AC97

3.1.a: WAV File Conversion

WAV files are organized very simply: there is a header that declares it as a WAV file, and tells the bit encoding and the framerate of the sound. Then there is a data section which consists of data sub-blocks with small headers telling how long it is, and then just the sound data.

Initially MATLAB was used in an attempt to retrieve the raw sound data from the WAV files, but as explained in the “Bug Report” section, this attempt was unsuccessful.

So instead, a simple C program was used. It takes two parameters, an input filename and an output filename. The C program simply reads the input file until reaching binary for the letters ‘d’,‘a’,‘t’,‘a’ consecutively. Then it has reached the data section. It simply reads the raw data, converts it to the ascii representation, and dumps it in the output file seperated by a comma and a space.

The user then is required to go into the output file and make it valid C. The final file should look as follows:

> *int32_t alienKillSoundRate = 11025;// sample rate*
>
> *int32_t alienKillSoundFrames = 3370;// size of array*
>
> *int32_t alienKillSound[] = { 49, 13, 0, [**11025 samples of data**]*
>
> *};*

This file's data can then be used by declaring variables as extern in the other functions

> ***extern** int32_t alienKillSoundRate;*

3.1.b: AC97 Operation

The AC97 has capabilities for both recording and playing audio from/to outside sources. It is used here for simply playing audio. For playing audio, it has a FIFO data structure that holds the currently-playing audio samples. It has capabilities to raise an interrupt when the FIFO is almost empty, which can be responded to by inserting more audio. Samples are submitted to the FIFO by means of a memory register- the data can continuously be written to the register and the AC97 will automatically insert it into the FIFO. Care must be taken not to overflow the FIFO, which is why usage of the AC97 interrupt mechanism is preferred, as well as only inserting 100-256 samples at a time.

In our Space Invaders, the AC97 has interrupt capabilities activated, and the interrupt handler calls a function to add more sound data to the card. Space Invaders knows which sound should be playing at all times, and loads the correct sound into the card 100 samples at a time. If no sound should be currently playing, it resets the interrupt and clears the FIFO so the AC97 can “play” silence.

3.1.c: Sound Triggering

The events in Space Invaders that currently trigger sounds are

1. The tank firing a shot
2. The tank dying
3. The mother ship’s theme when it is on the screen
4. The mother ship or a normal alien dies
5. The aliens move

The above list is in order of the priority in which the sounds are played. Space Invaders enforces this priority by having a large if/else block that determines which sound to send to the AC97- the sounds that are earlier in this block have higher priority than later sounds. For example, if the mother ship is on the screen, the aliens move, and the tank fires a shot all at the same time, the tank shot sound is played- the AC97 is always raising interrupts, and in the interrupt handler it detects that a shot was fired, gives that sound to the AC97, and returns, not worrying about the mother ship nor alien sounds.

Bug Report

The most difficult part of this section was converting the WAV files. MATLAB was the initial choice-- because of its build-in audio reading functions, it seemed obvious. However, when the data produced from the MATLAB was played by the AC97, it was very static-y and loud. Much time was spent thinking that the interface to the AC97 was incorrect, but after a copy of sound data was borrowed from someone else’s working code, the MATLAB script was identified as the culprit. A quick C program was written to reextract the sound data, and all was well.

The second hardest part was figuring out how the AC97 worked. The provided documentation provided no explanation for how to use it, and did not even mention the provided C functions to interface with it. After discussion in class, with the TA, and with other class members, proper usage was learned, and it successfully output nice-sounding audio.

A note on the code: all of the source code is provided, but only spaceInvadersRUN.c, sound.h, and sound.c had noticable differences. All of the other files had simple calls added to activate the sound, but aside from correcting simple coding standard errors, no other changes were made. wavKiller.c is the C file used to extract the sound data from the wav files.