Super Sound Regenerator (SSR)

Group member: Junbin Ma

Introduction:

Sound effects on Human voice is very popular and a lot of companies makes great money on doing so. However, the “cost” of applying sound effects is quite lot for both the hardware and software, and usually after applying the effects on real time or audio files, the sounds will be "unnatural" because those software/hardware usually treat sounds as discrete data, which for sure, not the case for real people’s sound (and most sounds in audio files).

The challenge & innovation here is:

Challenge & innovation branch 1:

1: we cannot trade the quality of sounds for the efficiency, because otherwise the SSR products will be meaningless (or just act like the other products);

Approach -> Using FFT conversion to make the audio file from discrete data to continuously function/data, and it will almost lose no accuracy (which based on how many bits the hardware can support).

2: the FFT is not very slow at the software side, however, when it comes up to large data FFT conversion, the software will be slower for sure because the two-dimensional loop takes O(n^2) operating times; the FFT in hardware side may act a little slower for small FFT conversions because it will take some time for data transfer from software to hardware or memory map I/O, but since the total amount of data for the FFT to deal with is constant, the hardware side will be faster for larger segmentation of the total data.

Approach -> Try to find a relative large segmentation number N for FFT do the conversion. (Since 1.5n^2 + 1000000n will be faster than 2n^2 + n for a large n.)

3: the data cannot be insanely large to make the performance seemingly apparently tend to hardware side, since the sample number in the audio file is not infinite!

Approach -> In my approach, the data number is 2^16, which is a very "good" number but still not large enough to make the performance act "favor of" SSR, but this number 65536 for almost the best quality wave file 48kHz is already a segmentation greater than 1.36 secs!

Challenge & innovation branch 2:

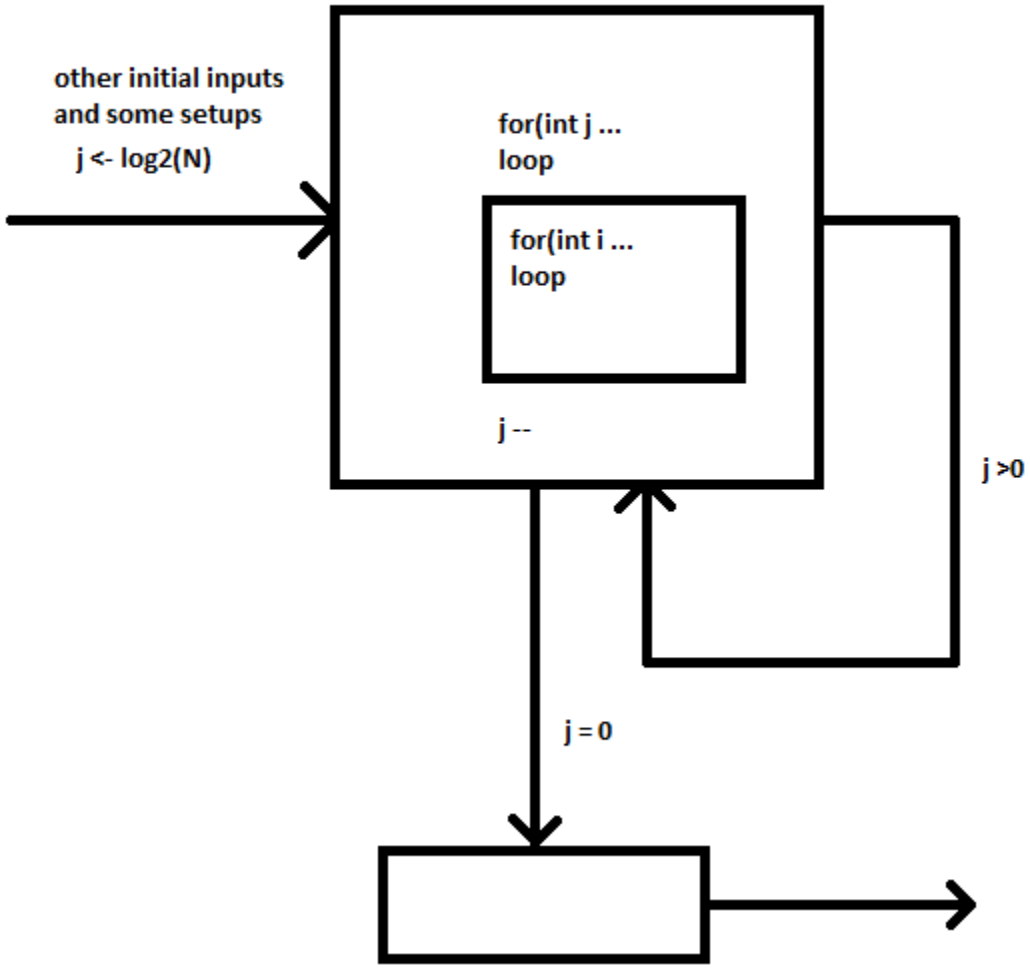
1: hard to test for the real sound, hard to tell whether the effects are "good", "bad", "better" or "worse"!

2: lots of bugs software side and hardware side! (see bugs&solution.docx)

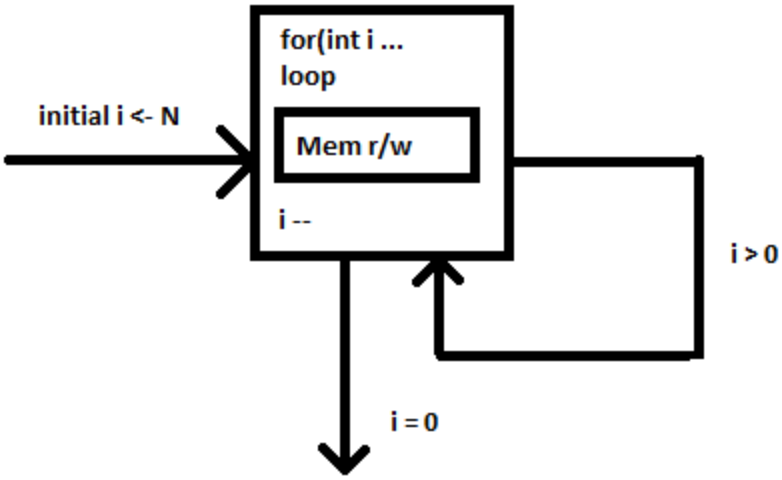
3: the understanding of high/low filter is limited! We usually take the frequencies to be the "filter" and the real data to be "filtered", however, for the higher purpose of my project, we need to do something directly on the frequency data!

Ideas for FFT software->hardware transfer:

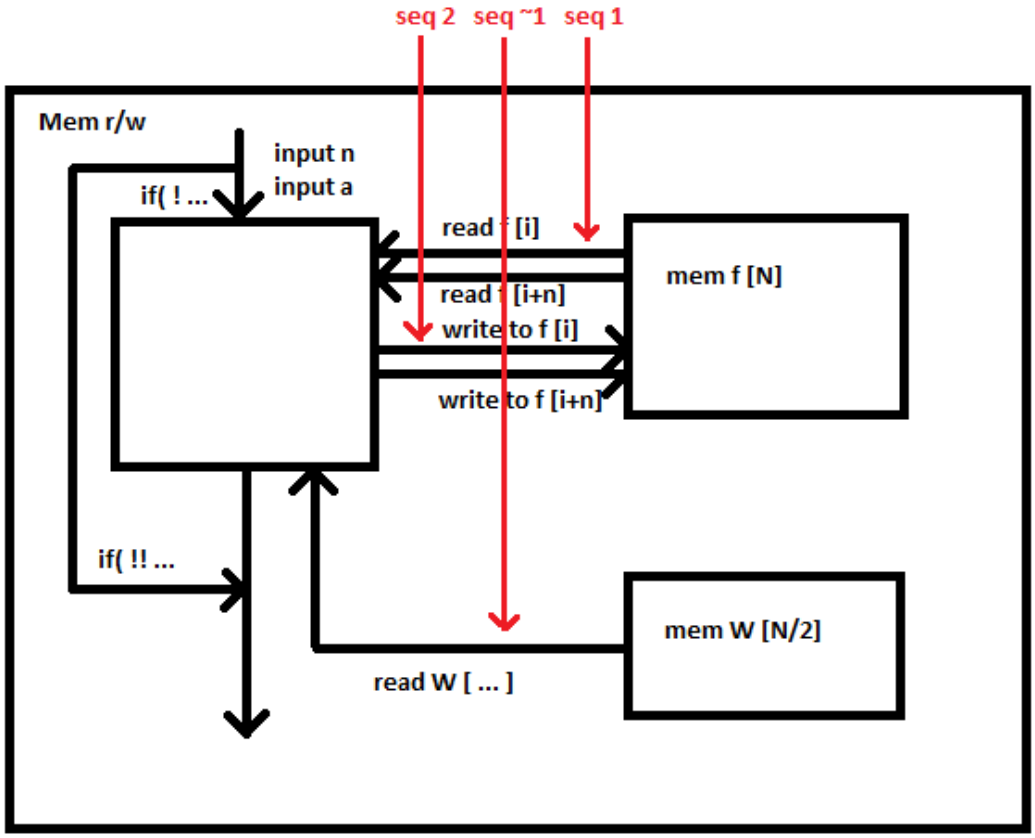
For int j block:



for int i block:



Mem r/w: <- this part



Files included and works did:

final_project.xpr ------ hardware side for FFT transform

// only implied faked RAM, solutions included in bugs&solution.docx

SoundLoop.sv ------ final_project.xpr related source file

SoundLoopTest.sv ------ SoundLoop test file

// the tests only include the inner loop of the FFT transform

main.cpp ------ software demo for the sound file IO, high/low filter, FFT transform

bugs&solution.docx ------ bugs that detected and possible solutions to be applied

// software bugs can't figure out yet

wave.c ------ sound file parse file

wave.h ------ sound file parse file header

111.wav ------ sample input file

Results and checklist:

Software side:

Supports 2 channels (2 bytes ea.), 44.1kHz .wav file sample input	√
Audio file read correctly but with a little format confusion	√
File data into c buffer	√
Write audio file (data "noising")	×
High/low filter seems work	√
FFT/IFFT works in software side	√

Hardware side:

FFT transform into Verilog expression	√
States works/logic works	√
Too many I/Os	×
Write the possible solutions in bugs&solution.docx	×

Cross interface:

Memory map I/O written in bugs&solution.docx	×
BRAM/RAM usage not implied	×

Tests:

Performance test not implied	×
Software side stack on the middle part	×
Include SoundLoopTest.sv (inner loop test)	√
Need much more tests on hardware side!	×

Conclusion:

Although the hardware programming is totally different from software programming, it worth a try and gives me a lot of inspiration on my future programming methods. Also, I believe this project can be implemented on fields other than audio effects, because the advantage of the hardware is not fully presented by the limitation of the real data. Although lots of my project failure included, after doing all these works, I know that the FFT on hardware side can do much better on a larger amount of data transforming process! What's more, it gives me the idea of reducing the cost by improve my own design, the hardware possibility of the higher performance is beyond my past

imagination! Once more, I am aware of that, the knowledge of the hardware in future programming life is so important and the ideas behind it is the abstract algorithm implementation and the deep understanding of the components behind the screen.

Great course and experiments!

Listed I/Os:

All I/Os are between software and hardware or inside software or inside hardware

Related work1:

"Methods for Efficient, High Quality Volume Resampling in the Frequency Domain"

https://dl.acm.org/citation.cfm?id=1034424

High-low pass filters. Which we are going to use before processing to our effects. It is going to modify the sound at different ranges when it is too load or too low. We will use it to get rid of the sound that is either too high or too low, improving the sound quality. Cutting off unwanted sound is important because we don't want some noise in our sound if we are modifying our audio input. Otherwise, the noise will be included into the modification.

Related work2:

"Frequency-domain compatibility in digital filter BIST"
https://dl.acm.org/citation.cfm?id=266278

It is an approach to fast FFT, allowing us to access frequency domain. It will help us to make changes to a certain range of frequency, so that we can perform different effect on the audio input. For example, if we want to do an autotune to make the sound into a certain pitch, we will require FFT.

Reference:

Aili Li & Klaus Mueller & Thomas Ernst, "Frequency-domain compatibility in digital filter BIST", https://dl.acm.org/citation.cfm?id=1034424

FFT in c,

https://en.wikipedia.org/wiki/Fast_Fourier_transform

https://stackoverflow.com/questions/8801158/fft-in-a-single-c-file

Laurence Goodby & Alex Orailoglu, "Methods for Efficient, High Quality Volume Resampling in the Frequency Domain", https://dl.acm.org/citation.cfm?id=266278

Rashi Agrawal, "High Pass Filtering in Frequency Domain", https://www.youtube.com/watch?v=z_dkB7vtDL0