ECE 385

Fall 2023

Final Project
My Talking Tom

Ziyi Lin
Zhuoyang Lai
TA: Jerry Wang

1.Introduction

Our final project design is based on a classic video game called My Talking Tom; it is a game which the cat in the screen will hear what you say and repeat in his own voice, also by touching the screen differently, you can react with the cat, our version of My Talking Tom realizes several key features in the original version, Tom can listen and play black the sound you record with a different tone, namely, a cat tone. Also, there are simply animation when the cat is listening, talking and some other inputs has been added.

2.Description of design

Our design can be split into two parts, animation engine and sound engine.

a) Animation engine:

Animation engine is realized by Mutiple pre-loaded memory file which contain the raw version of various picture representing the cat's states, several palates to map the color, a controller to control the logic between inputs, state and the picture displayed, IP VGA TO HDMI and module vga_comtroller as well as a wrapper top to finally transmitted the desired HDMI signal and display the cat on screen.

It is worth noting that most of the pictures are stretched and reshaped before putting in the desired position on the screen. This is intended to save memory space in FPGA. Also, palate bits size is also restricted for memory saving purposes, in our case, every pixel in the raw memory file will occupy 4 bits.

We will go into more detail at module description @ module cat_animation.

b) Sound engine:

Sound engine is used for audio display with a different tone compared to the sound inputted, the process can be split into several processes:

1. transfer the pdm type data to pcm type data

The PDM signal is shifted and accumulated, and the resulting PCM data is generated on the rising edge of the PCM clock (pcmClk).

2. down sampling

Why we need down sampling:

The human voice frequency band ranges from approximately 300 to 2000 HZ[1], While the vibration frequency of cat is about 3 times higher than human. Therefore we need to use down sampling to increase the pitch.

$$Y(\omega) = \frac{1}{D}\sum_{k=0}^{D-1} X\left(\frac{\omega - 2\pi k}{D}\right)$$

Fig 1. how down sampling change the frequency behavior

As is illstrated in fig1, the original frequency is expanded D times, for example, if one signal is bandlimited to pi/6, if we downsampling it by 2, the frequency is bandlimited to pi/3 now.

In time domain, we keep every D'th sample from x[n] to form y[n]. For example, if D = 5, we keep every 5th sample at x[0], x[5], x[10], and so on.

To implement the down sampling in time domain, we use a new clock cycle based on pcm_clk. The pcm data get updated at each rising edge of pcm clk, for example, if we make new_clk = pcm_clk*2, we discard x[1], x[3],x[5] …...

Which means we down sampling by 2. By changing the parameter D, we can control how many times we want to down sample.

3. filter

The filter is based on FIR Compiler vivado IP. By changing the coe file in matlab, We can set the stop frequency and reduce the noise. Before the data is processed, change the data flow speed by using fifo because filter IP only accept data that update each posedge of system_clk.

4. data storage

The data is stored in BRAM.

5. implement time stretching model

Why we need time stretching model:

after down sampling, part of data points is discarded. If we play the down sampled pcm file, not only the pitch is increased, but also the speed is increased. This effect is like play audio at double/triple speed.

Therefore, to recover the down sampled data, we must implement a module that change the speed while not changing the pitch.

Time stretching can be performed on an FPGA in real-time using various digital audio processing techniques. One can take the Fast Fourier Transform (FFT) of a signal. This is called frequency domain Time stretching. Approaching Time stretching in the frequency domain is mathematically intensive and therefore complicated to do in hardware.

Time stretching can also be done in the time domain using various techniques. Mathematically, the concepts are less complex than frequency domain Time stretching and therefore relatively easier to implement. We approached our solution in the time domain.

A simple example of time stretching a sine wave is shown below.

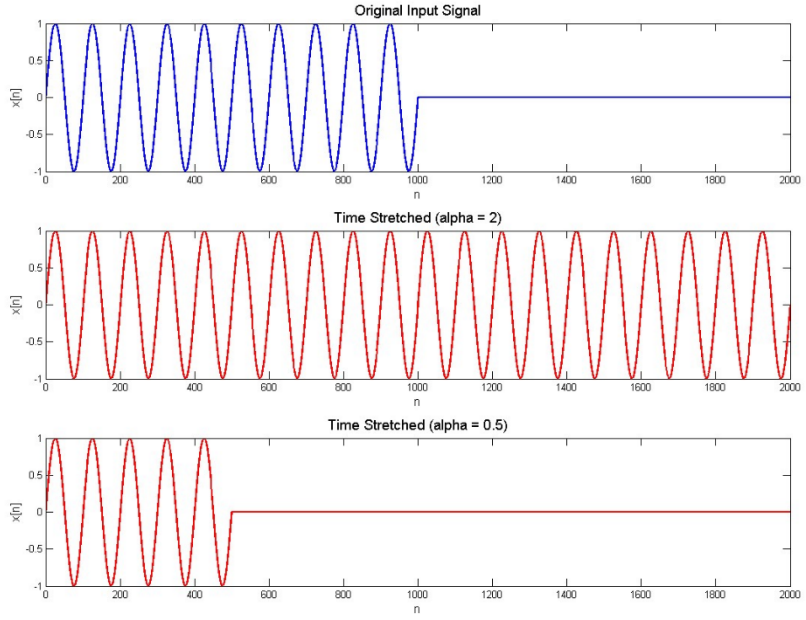


Figure 2. Time stretching

An effective algorithm for time stretching called Synchronous Overlap and Add (SOLA). It involves segmenting the input signal x[n] into smaller overlapping blocks $x_i[n]$, each time shifted by S a samples. These blocks are then overlapped again with a new time shift of a*S a and added together to produce the time stretched signal. To produce a high-quality time stretched signal, the parts of the blocks that actually overlap must be faded in and out as they are being added together. Below is a series of figures which shows the steps of this algorithm.

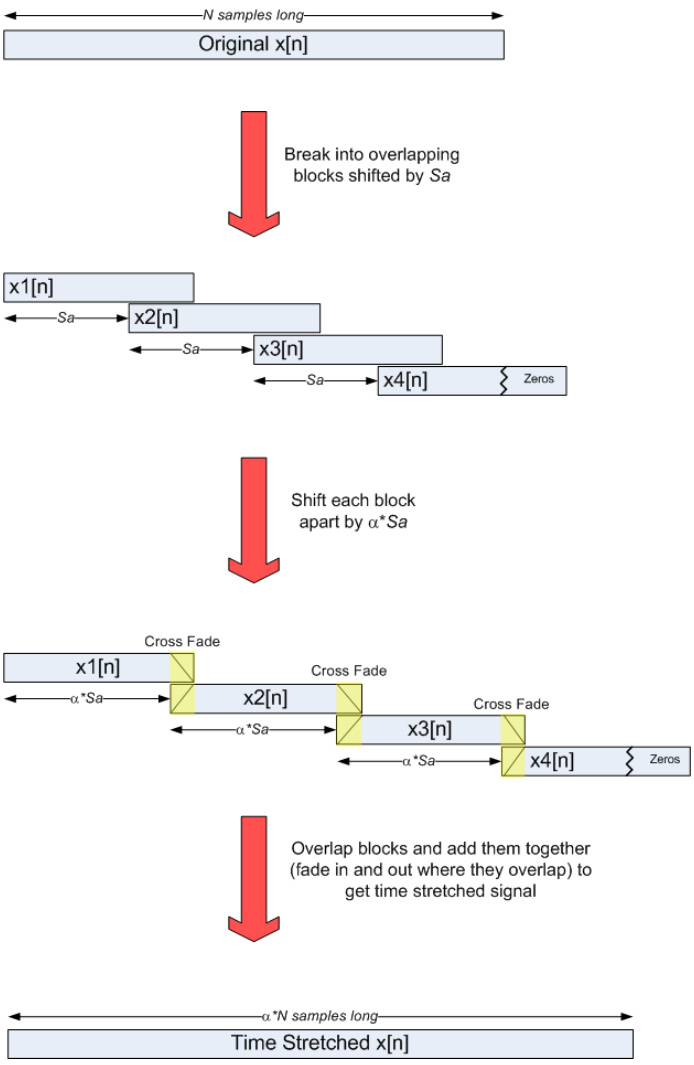


Figure 3. The basic time stretching algorithm

As it turns out, we were not able to implement this exact algorithm in hardware due to its complexity and the timing constrain of our project. Instead, we used a simplified time stretching method. Rather than overlapping the data blocks and cross-fading between them, we decided to repeat or truncate an audio block a set number of times to produce a signal with longer/shorter duration while preserving the frequency [2].

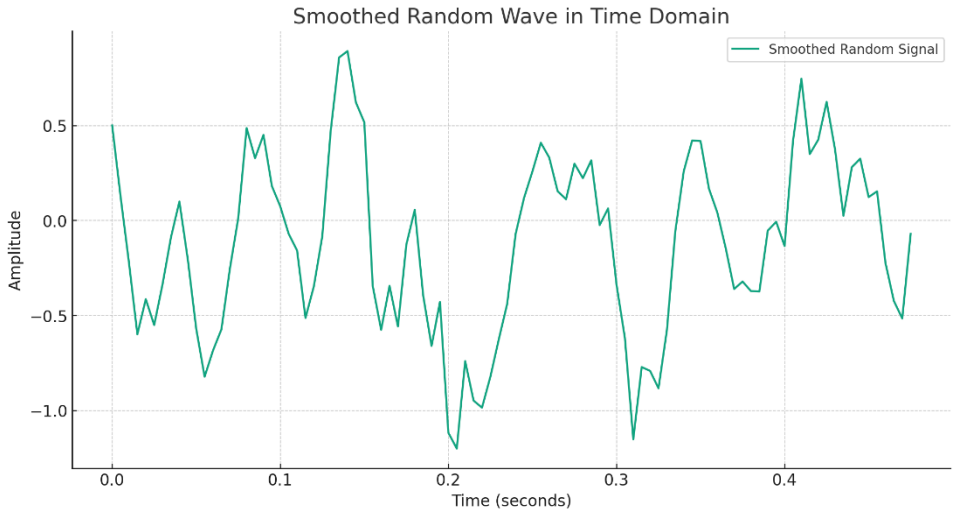


Figure 4. random wave

If we play the first segment twice and then the second segment twice
We can get a pcm file that time stretched twice.

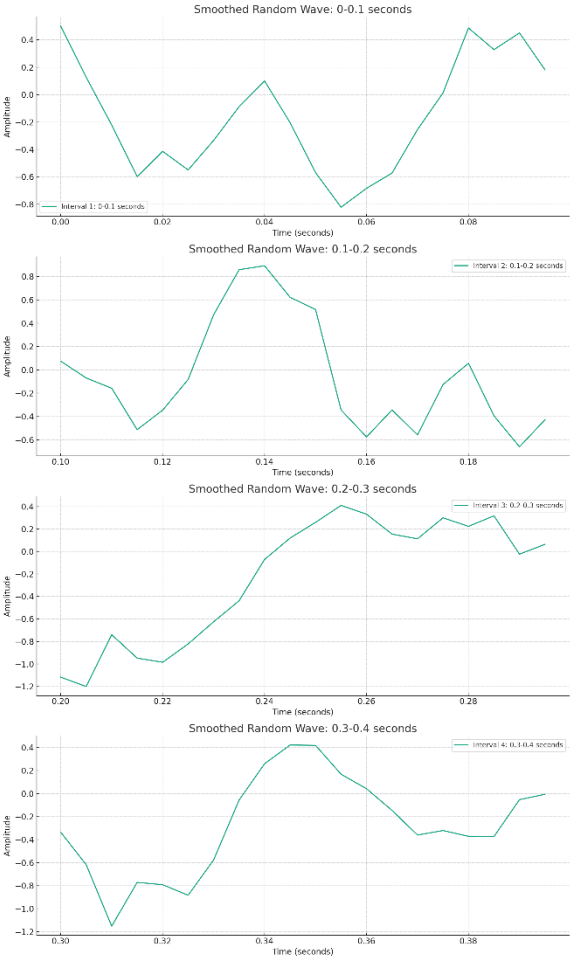


Figure 5. grid of the previous wave

The hardware implementation is described below.

Use clk_slice to slice the data set. By changing the period of clk_slice, we can control how many data point is store in each segment. A proper choice of period clk_slice is important,

if data number contain in each segment is too small, distortion will occur. If data number contains in each segment is too large, then information inside the sound can't be reconstructed. There is a tradeoff. After trying different parameters, I find a grid that holds 1024 sample points is the most suitable choice.

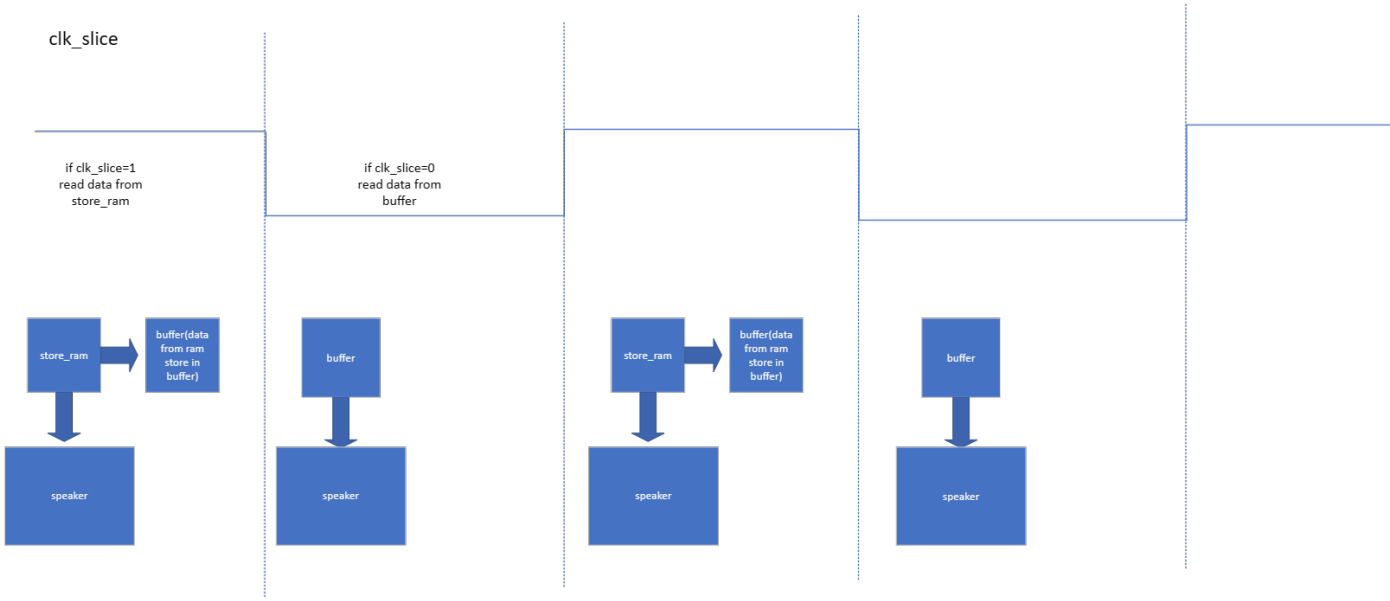


Fig6. Data flow diagram based on clk

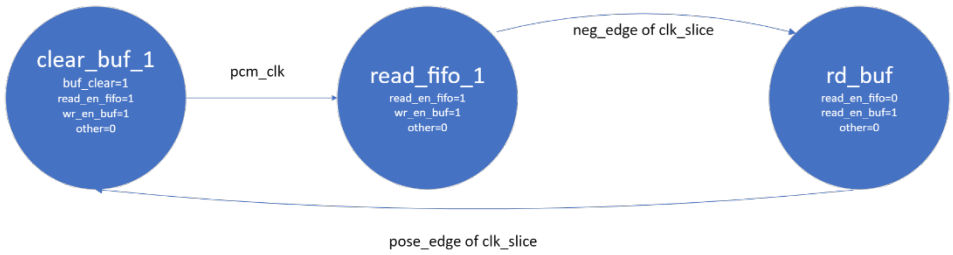


Fig6. State diagram

5. implement window function in time domain

This step diminishes the noise that occurs at the junction of two audio files. Crossfade is an audio editing technique used to smoothly fade in and out between two audio segments, reducing the discontinuities and abrupt noises at the junction point.

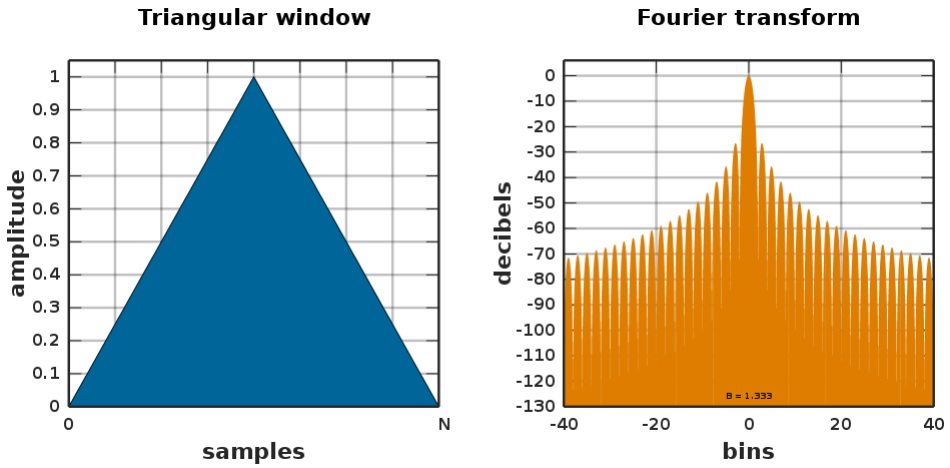


Fig7. Triangular window

The window that we implememnt is a triangular window.

6. transfer the pcm data to pdm data

The inverse process of step1.

3.Block diagram

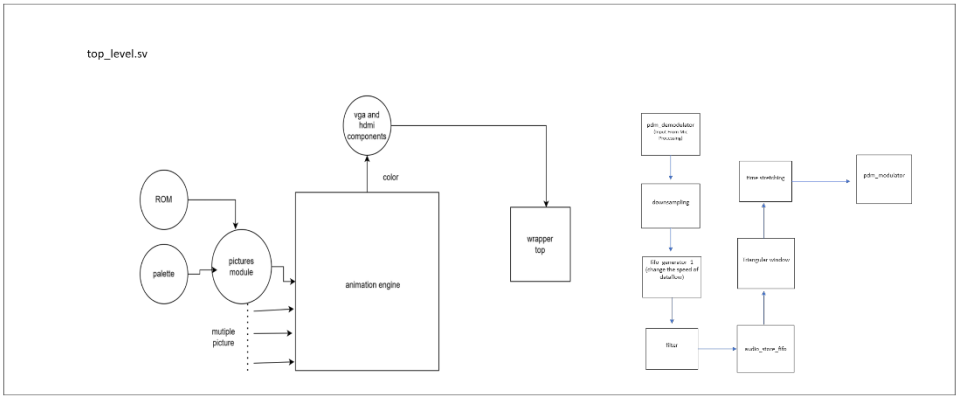
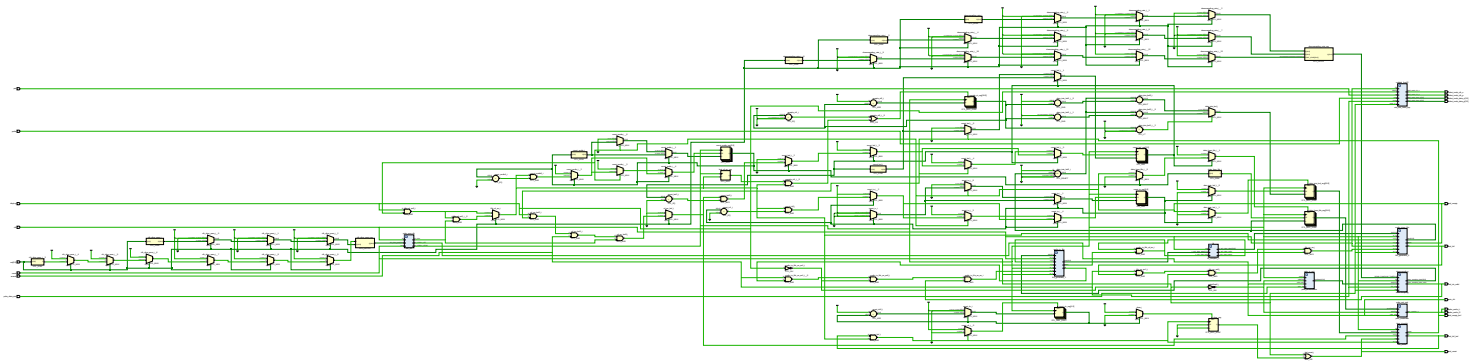


Fig8. Block diagram



4.Module description

a) animation engine

Module: mb_usb_hdmi_top

Input: Clk, reset_rtl_0, record, display, pet, poke

Output: hdmi_tmds_clk_n, hdmi_tmds_clk_p, [2:0]hdmi_tmds_data_n, [2:0]hdmi_tmds_data_p

Description: It is a simplified wrapper top module from previous lab, namely, lab6, It is used for wrapping animation engines with significant HDMI and vga components.

Module: cat_x_example (x ranges from 1 to 16)

Input: vga_clk, [9:0] DrawX, DrawY, blank,

Output: [3:0] red, [3:0]green, [3:0] blue

Description: It wraps ROM and palette file and output color of the picture which ROM stored

Module: cat_x rom (x ranges from 1 to 16)

Input: [13:0] a or ([13:0]addra, clka)

Output: [3:0] spo or ([3:0]douta)

Description: Because we exhaust the distributed memory here, we also use BRAM as rom, thus, there are different kinds of inputs and outputs

Module: cat_x palette (x ranges from 1 to 16)

Input: [3:0] index

Output: [3:0] red, [3:0]green, [3:0] blue

Description: used as palette

Module: cat_animation_engine

Input: logic Clk, vga_clk, vsync, [9:0] DrawX, [9:0] DrawY, listen, speak, pet, poke

Output: [3:0] red, [3:0]green, [3:0] blue

Description: This module is the animation engine and outputs the color needed to hdmi component. It accepts various kinds of input including listen, speak, pet, poke, also the state will also change thanks to the 60 HZ clock vsync. Particularly, the cat will alter listening gesture once listen signal is high. Also, the screen will loop the pictures with cat changing mouth position once speak signal is high, these two singals are linked to the sound engine as well. The screen will loop over cat performing comfortable gesture pictures, when pet signal is hign, it will also increase cat's liveness Also, the cat's liveness will decrease indicated by a line of red heart displayed on the screen, the value of liveness is controlled by vsync, and if

Pet signal is low for a long time, the liveness of the cat drops too much, it will change

himself to frustrated gestures as default gesture. If the liveness drops to zero, a cat's fall and dizzy with stars animation will be displayed, the cat will remain fainted until pet signal is high for enough time.

b) Sound engine

Module: pdm_demodulator
Input: pcmClk, pdm_mclk, pdmRstn, pdm_mic
Output: pcmData[15:0]
Description: designed to process a PDM input signal, demodulate it using an accumulator, and produce a PCM audio output signal.

Module: diff_clk.sv
Input: clk, reset, clk_slice_para
Output: clk_100Mhz_d32, clk_100Mhz_d2048, clk_slice
Description: generate pcm_clk(48khz), pdm_clk, and clk_slice for time_strentch

Module: downSampling.sv
Input: rstn, clk, DOWN_SAMPLING_TIME, din_valid, din
Output: down_sampling_dout_valid, down_sampling_dout
Description: perform downsampling on an input signal. In words, we keep every D'th sample from x[n] to form y[n]. For example, if D = 5, we keep every 5th sample at x[0], x[5], x[10], and so on.

Module: audio_in_fifo
Input: wr_clk, rd_clk, wr_rst, rd_rst, din, rd_en, wr_en
Output: dout, full, empty
Description: change the dataflow speed.

Module: fir_compiler_1
Input: aclk, s_axis_data_tready,
m_axis_data_tdata, m_axis_data_tvalid
Output: s_axis_data_tdata, s_axis_data_tvalid,
Description: reduce the high frequency noise

Module: audio_store_fifo
Input: wr_clk, rd_clk, wr_rst, rd_rst, din, rd_en, wr_en
Output: dout, full, empty
Description: store the pcm file on FPGA

Module: fifo_buffer
Input: wr_clk, rd_clk, wr_rst, rd_rst, din, rd_en, wr_en

Output: dout, full, empty
Description: store a single grid of pcm data.

Mudule: pdm_modulator
Input: pdm_mclk, pdmRstn, pcmData
Output: pdm_speaker_o
Description: transfer pcm file to pdm file

5.Conclusion

Design Resources and Statistics

LUT	13871
DSP	2
Memory(BRAM)	72.5
Flip-Flop	1244
Frequency	94MHZ //WNS<0
Static Power	0.076w
Dynamic Power	0.315w
Total Power	0.39w

The code can be found on github.

6. Reference

1. *"Definition: Voice frequency".*
2. *Real-time FPGA pitch shifter.*
   https://people.ece.cornell.edu/land/courses/ece5760/FinalProjects/f2007/dah64_dp239/