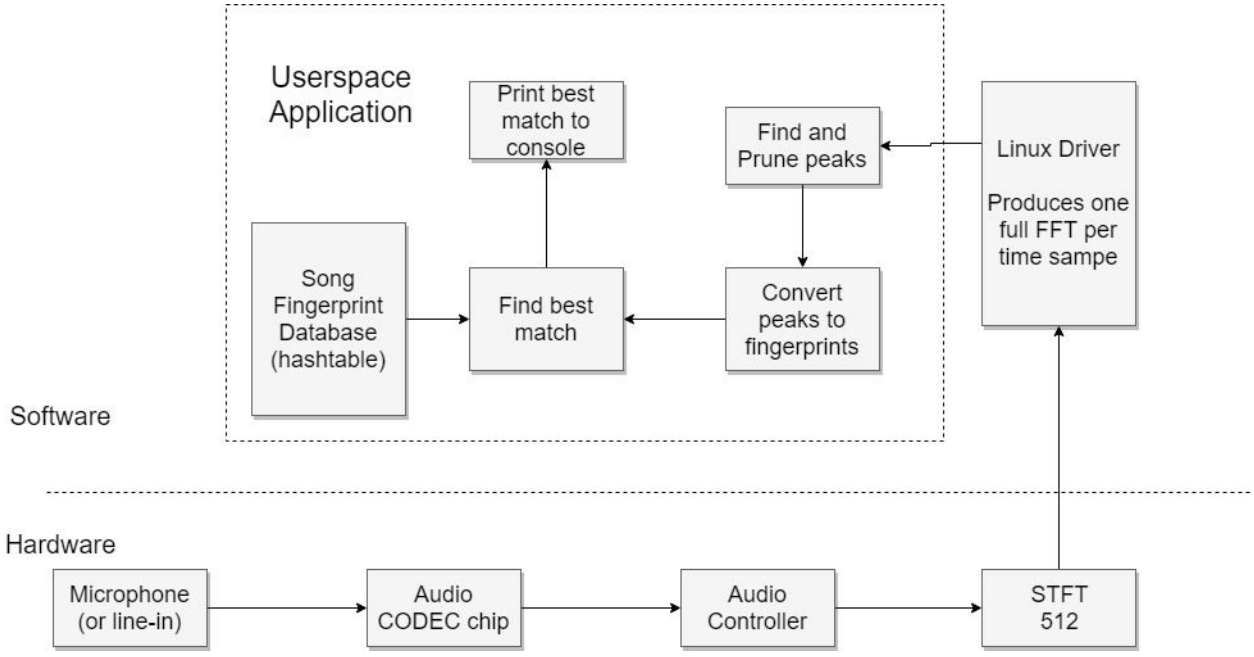
Eitan Kaplan (ek2928), Jose Rubianes (jer2201)
Tomin Perea-Chamblee (tep2116)
CSEEW4840 Embedded Systems
Columbia University
Final Report

Totally Not Shazam
Song Recognition

Summary:

By playing a song into the line-in port on the development board, our system is able to consistently correctly identify which song is being played. To accomplish this, we first take the FFT of the incoming audio on the FPGA, and then use those results to compare the observed sequence of notes to our database of song fingerprints.



Algorithm Overview:

Adding a song to the database:

1. Take STFT of audio
2. Find amplitude peaks of the STFT
3. Prune amplitude peaks to retain only the most significant peaks. This pruning step makes the algorithm robust against noise. At this point, amplitude data can be discarded, and what remains is a “constellation map”.
4. Create pairs of peaks that are in close time-proximity to each other. Form a fingerprint from each pair by concatenating the frequency of each peak, and the time delta between them.
5. Put each fingerprint in a hashtable where the key is the fingerprint and the value is the song name and the absolute time of the first peak in the pair (i.e., the anchor time).

Recognizing Unknown Song:

1. Repeat steps 1-4 of the databasing algorithm (STFT, peaks, pruning, and fingerprints) on the incoming unknown song sample
2. Look up each resulting fingerprint in the database to see if there is a matching fingerprint
3. Keep an addition hashtable where the keys are 3-tuples of [the sample fingerprint anchor time, the database fingerprint anchor time, and the song name in the database], and the keys are counts of how many times each 3-tuple key occurs amongst the matched fingerprints. Discard all matches whose corresponding count in this second hashtable is less than four. This second hashtable checks that not only pairs of notes match between the sample and the database songs, but that whole groups of notes match.
4. Sum the number of remaining matches for each song.
5. Choose the song with the highest number of such matches (in case of a tie, choose the databased song with the smaller number of entries in the database).

Note on the STFT:

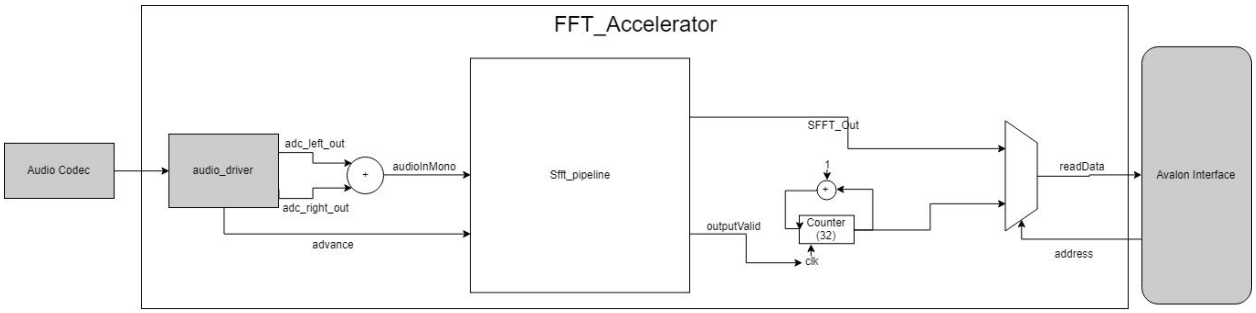
The Short Time Fourier Transform (STFT) is an operation that takes a time series (in this case audio amplitudes) and transforms it into a spectrogram -- a representation of how the frequency components of the audio change over time. The STFT is taken by doing successive Fourier Transforms on the incoming data. Each FFT is taken on audio data that overlaps with the previous audio data by a certain fixed number of samples -- typically half the number of audio samples used in each FFT (the NFFT).

The FFT is a complex operation, and produces complex results. However, for our purposes we found that simply taking the real portion of the results worked well. Additionally, due to symmetry properties of the FFT, when real input is given (and all of our audio input is real), the result is symmetrical -- the second half of the FFT is identical to the first half. Thus, we

safely discard the second half of the results. This is sometimes referred to (e.g., in software packages we used) as a “one-sided” FFT.

Hardware:

Hardware Overview:



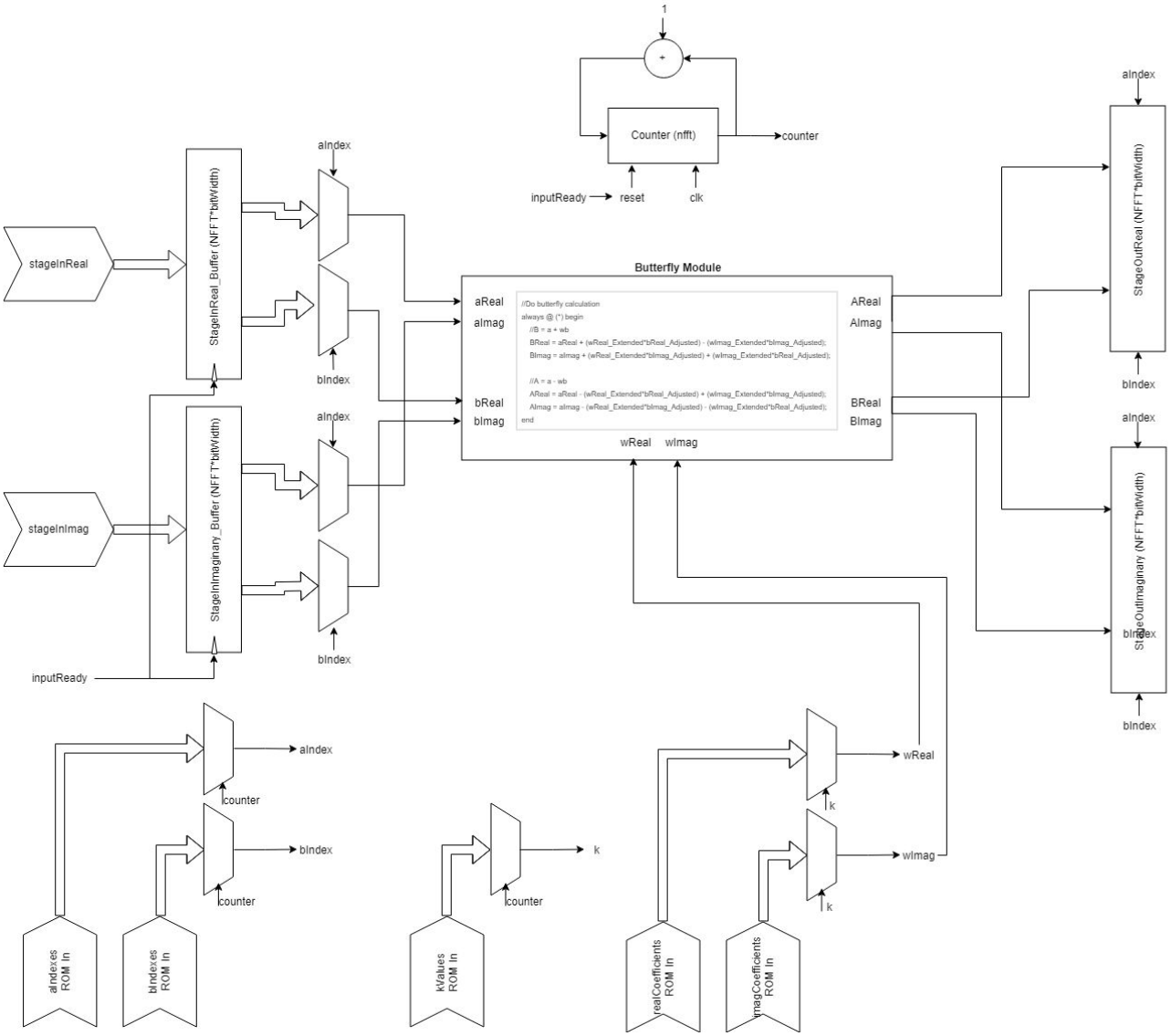
The hardware portion of our project is responsible for controlling the audio CODEC through the I2C bus, pulling data from the CODEC into BRAM buffers, computing FFTs from the incoming data, and interfacing with the Avalon bus to provide the FFT results to the processor when requested.

Audio CODEC Interface:

The audio CODEC is configured using the I2C bus. Rather than implement the I2C controller and the audio CODEC interface from scratch, we used and slightly modified existing SystemVerilog code provided by Altera and customized by Professor Scott Hauck at Washington University[1].

[1] Hauck, S. (n.d.). ECE271, Winter 2019. Retrieved from https://class.ece.uw.edu/271/hauck2/

Core FFT Module (pipelineStage):

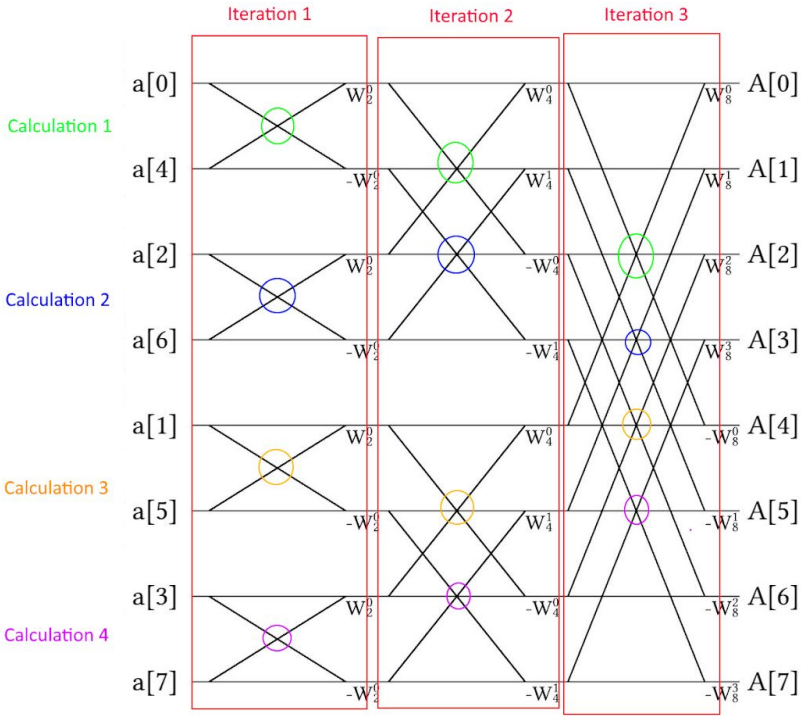


The core component of the hardware is the FFT module which computes FFTs on incoming raw audio data. We calculate the STFT by implementing the Cooley-Tukey algorithm, so we can avoid large matrix multiplications. While we had originally planned to implement a multi-stage pipeline to accomplish this, we found through testing that we only needed a throughput of 86 FFT results per second for the classification algorithm to work. Thus, we opted for a single-stage, sequential pipeline due to the ease of design and low throughput requirements.

The base hardware block used to find the FFT is the ButterflyModule, which performs a single radix-2 FFT calculation. This module takes in two complex inputs, in addition to a static complex coefficient, and produces two complex outputs. We use a single butterfly module in the computation stage, iterating through the input buffer and writing the results of the butterfly calculation to the same buffer location as the inputs. This allows us to cut the memory overhead of the calculation stage in half[2]. This is done down the entire buffer, and is repeated for log2(NFFT) iterations (9 times for out 512 point output). Our sequential approach to calculating FFTs is very similar to the method described by Cohen, and is explained more in-depth in his 1977 paper[3].

Iterations = log2(N)

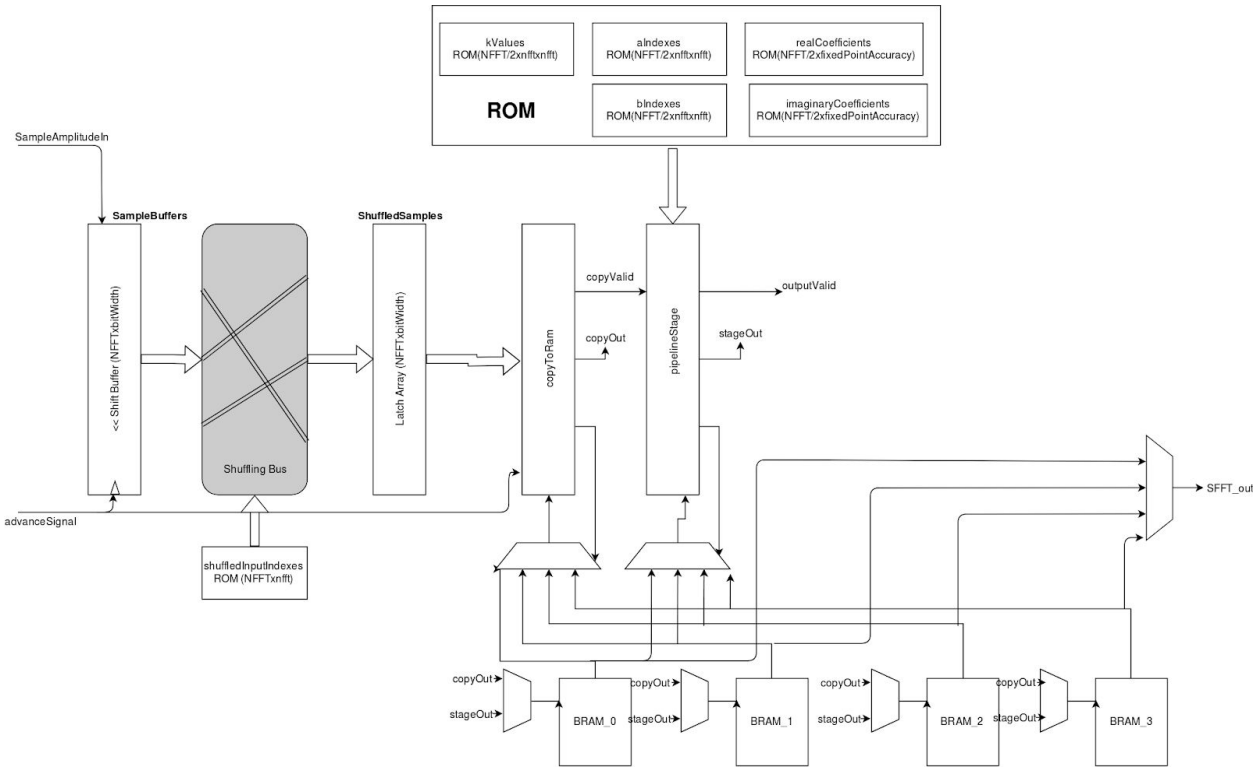
Butterflies Per Iteration = N/2



[2] NOTE: Output buffers are shown in block diagram for simplicity, but they are the same as the input buffers. Furthermore, the buffers are actually outside of the stage module as BRAM blocks within the pipeline.

[3] Cohen, Danny. (1977). Simplified control of FFT hardware. Acoustics, Speech and Signal Processing, IEEE Transactions on. 24. 577 - 579. 10.1109/TASSP.1976.1162854.

Supporting Infrastructure for FFT Module (Sfft_pipeline):

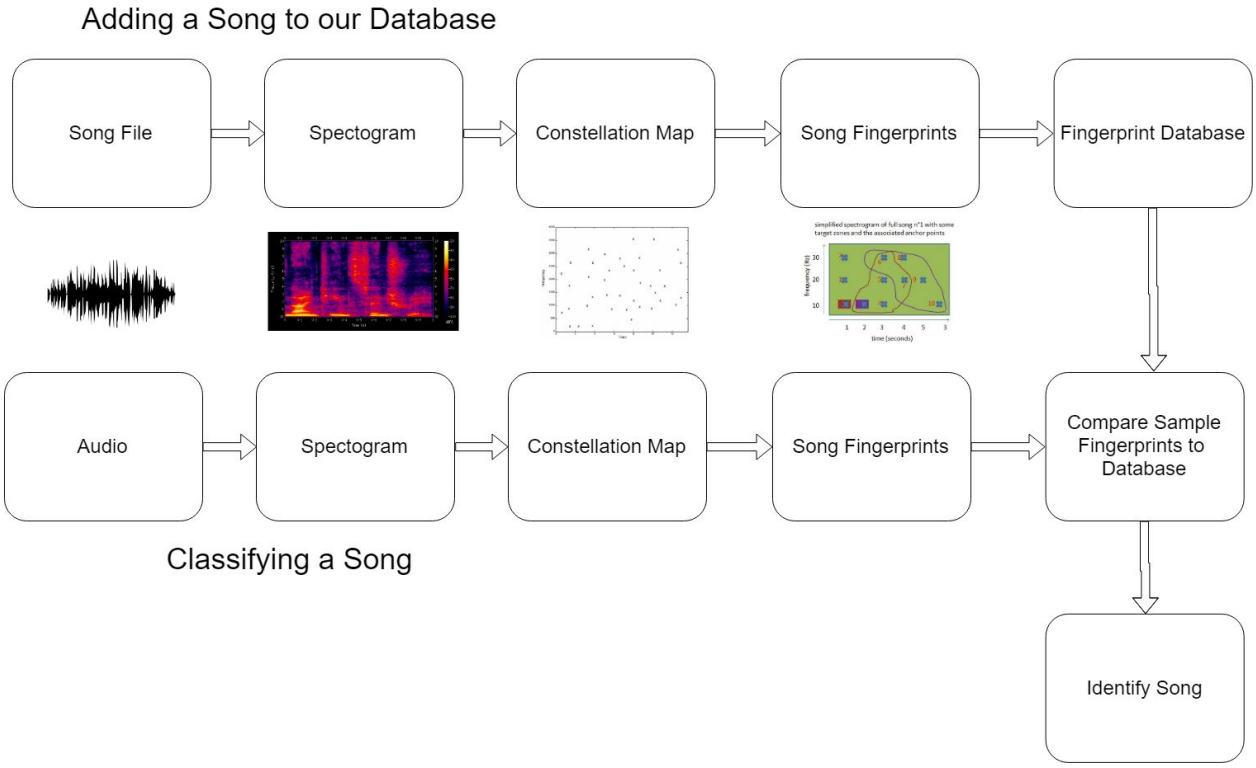


Every time a sample comes in, the first stage CopyToRam copies the values of the previous 512 samples into a BRAM block. It then signals the FFT Stage module to start computation, along with which BRAM block to use as it's memory buffer. These two stages iterate through the BRAM blocks, so that one can be copying while the other is computing.

We use 4 BRAM blocks, ensuring that once a complete calculation is written to one of them, it will not be overwritten for another 3 samples. This gives the driver time to read the outputs from the BRAM module even if it started to pull values mid-way through a calculation. The pipeline keeps track of where the most recent sample is stored, and gives the avalon-bus access to that BRAM block when a read is requested. If the BRAM block is overwritten during a driver read, the pipeline will notify the driver that the pulled-sample is corrupt using a valid bit.

Additionally, the hardware keeps a counter that tallies the number of STFT results thus far computed is placed in a buffer. The driver reads from the buffer to retrieve the data. This serves as a timestamp of the pulled sample. The hardware makes no guarantee that the next sample seen by the driver will be the next sample sequentially (i.e., if the driver is too slow retrieving data, some data may be dropped). Our algorithm is robust against occasional missed sample (as long as timing data is preserved -- hence the need for the sample counter).

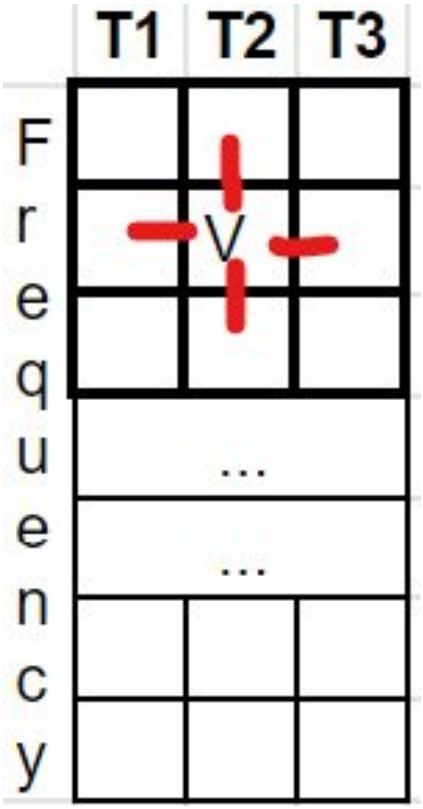
Software:



The algorithm for classifying songs involves finding local maxima in the spectogram, then using those peaks to generate fingerprints that act as keys (hashes) for a large database (hash table) of cataloged songs. This is accomplished by identifying the most important frequency ranges that humans can perceive (bins), and, then, finding the most significant frequencies that occur in those ranges in any song (local extrema within the selected bins of frequencies in the Fourier transform of the signal). Afterwards, the identified frequency peaks form a unique characterization of a song that is called a fingerprint which is stored in the hash table (for a more descriptive and detailed explanation of hash table generation, see Cristophe's Coding Geek article[4]). It is noise tolerant and time invariant. Also, it ensures consistency over recording hardware, volume, levels of background noise, etc). Generating these hashes for incoming audio allows for quick searching of possible song matches, rather than sequentially searching for a match across the entire catalog (songs which have already been fingerprinted according to the same algorithm, and so the catalog is generated or retrieved ahead of time).

[4] Cristophe. (2015, August 6). How does Shazam work. Retrieved from Coding Geek

Peak Finding/Compression Algorithm:



The STFT data comes in from the accelerator as a two-dimensional array. In software, we divide the 256 frequencies into 6 logarithmic bins. In each bin, we find all peak candidates, i.e., local maxima -- all entries in the array which are greater than all of their four neighbors. Then, for each bin, at each time slice, we keep only the peak with the highest amplitude of the peak candidates. At this point we have a maximum of 6 peaks per time slice (but usually fewer than six). At this point peaks are represented as 3-tuples consisting of the time, frequency, and amplitude. In order to further reduce the number of peaks we use, we prune in the following manner. We look at a time chunk of 500 time slices at a time, and take the average and the standard deviation of all peak amplitudes in that time chunk. Then we throw away all peaks with amplitudes that are less than the average plus a the standard deviation times a coefficient.

Choosing only the largest peaks makes the algorithm robust against noise -- hopefully, any noise will not be as loud as the song itself, and therefore the noise will be pruned.

Software Only Shazam Model

A significant portion of our efforts were devoted to creating a software only model of the Shazam algorithm. Instead of taking the STFT in real-time (which takes a long time in software -- hence the motivation for the hardware FFT accelerator) we precomputed the STFT and saved it to a file, and then just read it in when running our “recognize” program. This allowed us to verify the efficacy of the Shazam algorithm before we began the project, and, once the project was underway, it allowed us to develop and refine the software portion of the Shazam algorithm before the hardware FFT module was completed.

Our software model is able to recognize songs -- even with artificially added noise -- at 100% accuracy across the 30 songs we were working with.

Results and Analysis

Our system is able to recognize songs on a consistent basis, with 100% accuracy among the 30 songs in our database.

Originally, we prepared constellation maps for the songs in software -- i.e., we took an STFT of each song, found and pruned the peaks of the spectrogram, and wrote the resulting peaks to a file. We then copied those files to the board, where our software program could read them in to quickly build the database. However, found that our system had trouble recognizing songs when comparing against a database built from these software produced constellation maps.

Instead, we found that by generating the constellation files using the FFT generated by the hardware worked much better. This suggests that the software FFTs of the .wav files differ from the hardware FFTs of the incoming audio from the CODEC. We don't know with certainty why this is, but possibilities include: fixed point arithmetic or the occasional overflow produces sufficiently different FFTs; or data from audio CODEC is sufficiently different from the .wav files -- with it has inherent (and consistent) noise, or perhaps is scaled differently.

How to Use the System to Recognize Songs

Generate ROM files with MATLAB, then use Quartus to synthesize the hardware[5], and copy the rbf and dtb to the boot partition of the board[6]. Boot the board.

Once the board has the FPGA configured and boots, go the the "software" folder. Run "make" and "./makeRecognize.sh" to compile the software system. Then, insert the driver module by running "insmod fft_accelerator.ko". The system is now set up.

To recognize songs using a database that was created using FFTs from the board, enter the "constellationFiles_board" folder. Run "../recognize_board" to start the program. Play the music through a mic or line-in, and hit enter when prompted.

To recognize songs using a database that was created using FFTs in software with our python script, enter the "constellationFiles_software" folder. Run "../recognize" to start the program. Play the music through a mic or line-in, and hit enter when prompted. This method of running the system does not perform as well as the recognize_board program, as described above.

[5] recompileHardware.sh script
[6] updateBootPartition.sh script

How to Use the Software Model to Recognize Songs in .wav Files

Enter SoftwareShazamModel directory and run make. Then enter constellationFiles directory and run “../recognize”.

Future Work

-Test with microphone.

-Ideally, the amount of time required to listen to an incoming song would be determined dynamically; i.e., the system would only listen for as long as necessary to find enough matches to recognize the song with confidence. This would require creating a multithreaded program so that one thread can begin matching against the database while the other thread continues to poll the driver for more FFT data. While this would be a nice feature, and would moderately improve the user experience, for our project we determined that the program would just listen to all songs for the same fixed amount of time. We chose an amount of time (about 30 seconds) that would be sufficient to produce enough matches to recognize the song in most cases.

-There is some amount of parameter tuning that could be done that may improve the accuracy of the system further.

List of Files

Hardware:

- FFT_accelerator.sv: Top level hardware module, communicates with audio codec via i2c as well as with the segment displays (used frequently in debugging). Passes mono audio and advance signals to SFFT Pipeline.
- SfftPipeline_SingleStage.sv: Top level SFFT pipeline module, samples the input signal at the rising edge of the advance signal. Outputs only real components of calculated FFT result. Raises valid bit for one cycle when calculation ends. Connects to internal BRAM module where results are stored.
- bram.sv: BRAM modules used as pipeline memory buffers
- soc_system.tcl: Top level SystemVerilog file includes FPGA pin assignments.
- soc_system_top.sv: Defines top-level system module

Software:

- FFT_accelerator.h: Header file for FFT accelerator device driver.
- FFT_accelerator.c: FFT accelerator device driver. Pulls FFT results from data bus.
- usr.c: User space test program for testing driver results.
- fft.py: Python file used to generate FFTs in software. Written to files to be read by C++ program.
- recognizeSongs.py: Preliminary Python prototype for classification algorithm
- SoftwareShazamModel/recognize.cpp: Pure C++ implementation of the classification algorithm
- song_list.txt: List of song names in database.
- *constellationFiles/**: contains sparse representations of songs, used for building database
- db.cpp: generates constellation files from FFTs pulled from device
- software/recognize_board.cpp: recognizes songs from audio played to device, using database made from hardware generated FFTs
- software/recognize.cpp: recognizes songs from audio played to device, using database made from software generated FFTs
- generate_constellations.cpp: generates constellation files from fft files.

- GenerateRomFiles.m: Generates the ROM files used in the FFT pipeline
- generateKs.m: Used by GenerateRomFiles to generate K factors
- suffleIndices.m: Used by GenerateRomFiles to generate shuffle indexes for FFT inputs
- q2dec.m: Used by GenerateRomFiles to convert numbers to hex

- FFT_Testing.m: Testing script for FFT algorithm and comparison with hardware results
- myFFT.m: Used to test hardware FFT algorithm
- myButterfly.m: Software version of butterfly module.
- runMatlabScript.sh: Shell script to start execution of GenerateRomFiles.m

Code:

Driver:

fft_accelerator.h:

```
#ifndef _FFT_ACCELERATOR_H
#define _FFT_ACCELERATOR_H

#include <linux/ioctl.h>
#include <linux/types.h>

#define BINS 6
#define FREQ_WIDTH_BYTES 1
#define FREQ_WIDTH_BITS 1
#define AMPL_WIDTH_BYTES 4
#define AMPL_WIDTH_BITS 32
#define AMPL_FRACTIONAL_BITS 7
#define COUNTER_WIDTH_BYES 4
#define SAMPLING_FREQ 48000
#define DOWN_SAMPLING_FACTOR 256
#define N_FREQUENCIES 256

#define AMPLITUDES_SIZE (N_FREQUENCIES * AMPL_WIDTH_BYTES)


typedef int32_t ampl_t;

typedef struct {
        ampl_t fft[N_FREQUENCIES];
        uint32_t time;
        uint8_t valid;
} fft_accelerator_fft_t;


typedef struct {
  fft_accelerator_fft_t *fft_struct;
} fft_accelerator_arg_t;

#define FFT_ACCELERATOR_MAGIC 'p'

/* ioctls and their arguments */
```

```
#define FFT_ACCELERATOR_READ_FFT  _IOR(FFT_ACCELERATOR_MAGIC, 2,
fft_accelerator_arg_t *)

#endif
```

fft_accelerator.c

```
/* * Device driver for the VGA video generator
 *
 * A Platform device implemented using the misc subsystem
 *
 * Stephen A. Edwards
 * Columbia University
 *
 * References:
 * Linux source: Documentation/driver-model/platform.txt
 *               drivers/misc/arm-charlcd.c
 * http://www.linuxforu.com/tag/linux-device-drivers/
 * http://free-electrons.com/docs/
 *
 * "make" to build
 * insmod fft_accelerator.ko
 *
 * Check code style with
 * checkpatch.pl --file --no-tree fft_accelerator.c
 */

#include <linux/delay.h>
#include <linux/module.h>
#include <linux/init.h>
#include <linux/errno.h>
#include <linux/version.h>
#include <linux/kernel.h>
#include <linux/platform_device.h>
#include <linux/miscdevice.h>
#include <linux/slab.h>
#include <linux/io.h>
#include <linux/of.h>
#include <linux/of_address.h>
#include <linux/fs.h>
#include <linux/uaccess.h>
#include "fft_accelerator.h"
```

```
#define DRIVER_NAME "fft_accelerator"

/* Device registers */
#define AMPLITUDES(x)    (x)
#define TIME_COUNT(x)    (AMPLITUDES(x) + AMPLITUDES_SIZE)
#define VALID(x)         (TIME_COUNT(x) + COUNTER_WIDTH_BYES)
#define READING(x)       (VALID(x) + 1)

/*
 * Information about our device
 */
struct fft_accelerator_dev {
        struct resource res; /* Resource: our registers */
        void __iomem *virtbase; /* Where registers can be accessed in memory */
} dev;


// TODO rewrite this.  Right now it is a sketch. Double check subleties with bit widths,
// and make sure addresses match with hardware.
static int fft_accelerator_read_sample(fft_accelerator_fft_t *sample_struct) {
        int i;
        static uint32_t prev_time = 0;

        // THIS PART IS NOT FULLY PARAMETRIZED --
        // ioread*() calls may need to change if bit widths change.
        int tries = 0;
        while (1) {
                while ((sample_struct->time = ioread32(TIME_COUNT(dev.virtbase))) ==
prev_time) {
                        tries++;
                        if (tries > 15){
                                printk("\tSample time delta: %d\n", sample_struct->time -
prev_time);
                                return -1;
                        }
                        usleep_range(1000, 2000);
                }
                iowrite8(0x1u, READING(dev.virtbase));
                for (i = 0; i < N_FREQUENCIES; i++) {
                        sample_struct->fft[i] = ioread32(AMPLITUDES(dev.virtbase) +
i*AMPL_WIDTH_BYTES);
                }
                sample_struct->valid = ioread8(VALID(dev.virtbase));
```

```
                iowrite8(0x0u, READING(dev.virtbase));
                if (sample_struct->valid){
                        break;
                } else {
                        tries++;
                }
        }
        printk("\tSample time delta: %d\n", sample_struct->time - prev_time);
        prev_time = sample_struct->time;
        return 0;
}


/*
 * Handle ioctl() calls from userspace:
 * Read or write the segments on single digits.
 * Note extensive error checking of arguments
 *
 *
 */
static long fft_accelerator_ioctl(struct file *f, unsigned int cmd, unsigned long arg)
{
        fft_accelerator_fft_t *sample;
        fft_accelerator_fft_t *dest;
        fft_accelerator_arg_t arg_k;

        if (copy_from_user(&arg_k, (fft_accelerator_arg_t *) arg, sizeof(fft_accelerator_arg_t)))
                return -EACCES;

        switch (cmd) {

        case FFT_ACCELERATOR_READ_FFT:
                sample = kmalloc(sizeof(fft_accelerator_fft_t), GFP_KERNEL);
                if (sample == NULL) {
                        printk("nomem");
                        return -ENOMEM;
                }
                printk("About to read sample\n");
                if (fft_accelerator_read_sample(sample) == -1) {
                        kfree(sample);
                        return -EIO;
                }
                printk("Read Peaks\n");
```

```
                dest = arg_k.fft_struct;
                if (copy_to_user(dest, sample,
                                sizeof(fft_accelerator_fft_t))) {
                        kfree(sample);
                        return -EACCES;
                }
                kfree(sample);
                break;

        default:
                return -EINVAL;
        }

        return 0;
}

/* The operations our device knows how to do */
static const struct file_operations fft_accelerator_fops = {
        .owner          = THIS_MODULE,
        .unlocked_ioctl = fft_accelerator_ioctl,
};

/* Information about our device for the "misc" framework -- like a char dev */
static struct miscdevice fft_accelerator_misc_device = {
        .minor          = MISC_DYNAMIC_MINOR,
        .name           = DRIVER_NAME,
        .fops           = &fft_accelerator_fops,
};

/*
 * Initialization code: get resources (registers) and display
 * a welcome message
 */
static int __init fft_accelerator_probe(struct platform_device *pdev)
{
        int ret;

        /* Register ourselves as a misc device: creates /dev/fft_accelerator */
        ret = misc_register(&fft_accelerator_misc_device);

        /* Get the address of our registers from the device tree */
        ret = of_address_to_resource(pdev->dev.of_node, 0, &dev.res);
        if (ret) {
```

```
		ret = -ENOENT;
		goto out_deregister;
	}

	/* Make sure we can use these registers */
	if (request_mem_region(dev.res.start, resource_size(&dev.res),
			       DRIVER_NAME) == NULL) {
		ret = -EBUSY;
		goto out_deregister;
	}

	/* Arrange access to our registers */
	dev.virtbase = of_iomap(pdev->dev.of_node, 0);
	if (dev.virtbase == NULL) {
		ret = -ENOMEM;
		goto out_release_mem_region;
	}

	return 0;

out_release_mem_region:
	release_mem_region(dev.res.start, resource_size(&dev.res));
out_deregister:
	misc_deregister(&fft_accelerator_misc_device);
	return ret;
}

/* Clean-up code: release resources */
static int fft_accelerator_remove(struct platform_device *pdev)
{
	iounmap(dev.virtbase);
	release_mem_region(dev.res.start, resource_size(&dev.res));
	misc_deregister(&fft_accelerator_misc_device);
	return 0;
}

/* Which "compatible" string(s) to search for in the Device Tree */
#ifdef CONFIG_OF
static const struct of_device_id fft_accelerator_of_match[] = {
	{ .compatible = "csee4840,fft_accelerator-1.0" },
	{},
};
MODULE_DEVICE_TABLE(of, fft_accelerator_of_match);
```

```
#endif

/* Information for registering ourselves as a "platform" driver */
static struct platform_driver fft_accelerator_driver = {
	.driver  = {
		.name  = DRIVER_NAME,
		.owner = THIS_MODULE,
		.of_match_table = of_match_ptr(fft_accelerator_of_match),
	},
	.remove	= __exit_p(fft_accelerator_remove),
};

/* Called when the module is loaded: set things up */
static int __init fft_accelerator_init(void)
{
	pr_info(DRIVER_NAME ": init\n");
	return platform_driver_probe(&fft_accelerator_driver, fft_accelerator_probe);
}

/* Calball when the module is unloaded: release resources */
static void __exit fft_accelerator_exit(void)
{
	platform_driver_unregister(&fft_accelerator_driver);
	pr_info(DRIVER_NAME ": exit\n");
}

module_init(fft_accelerator_init);
module_exit(fft_accelerator_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Eitan Kaplan, Columbia University");
MODULE_DESCRIPTION("FFT Accelerator driver");
```

recognize.cpp

```
/*
 */

#include <iostream>
#include <fstream>
#include <string>
#include <cstring>
#include <sstream>
#include <list>
```

```
#include <set>
#include <vector>
#include <unordered_map>
#include <algorithm>
#include <cfloat>
#include <cmath>
#include "fft_accelerator.h"
#include <sys/ioctl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <fcntl.h>

#define NFFT 512
#define NBINS 6
#define BIN0 0
#define BIN1 10
#define BIN2 20
#define BIN3 40
#define BIN4 80
#define BIN5 160
#define BIN6 240

#define PRUNING_COEF 1.4f
#define PRUNING_TIME_WINDOW 500
#define NORM_POW 1.0f
#define STD_DEV_COEF 1.25
#define T_ZONE 4

struct peak_raw {
        float ampl;
        uint16_t freq;
        uint16_t time;
};

struct peak {
        uint16_t freq;
        uint16_t time;
};

struct fingerprint {
        uint16_t anchor;
        uint16_t point;
        uint16_t delta;
```

```
};

struct song_data {
	std::string song_name;
	uint16_t time_pt;
	uint16_t song_ID;
};

struct hash_pair {
	uint64_t fingerprint;
	struct song_data value;
};

struct count_ID {
	std::string song;
	int count;
	int num_hashes;
};

struct database_info{
	std::string song_name;
	uint16_t song_ID;
	int hash_count;
};

std::list<hash_pair> hash_create(std::string song_name, uint16_t song_ID);

std::list<hash_pair> generate_fingerprints(std::list<peak> pruned,
	std::string song_name, uint16_t song_ID);

std::unordered_map<uint16_t, count_ID> identify_sample(
	const std::list<hash_pair> & sample_prints,
	const std::unordered_multimap<uint64_t, song_data> & database,
	std::list<database_info> song_list);

std::list<peak> generate_constellation_map(std::vector<std::vector<float>> fft, int nfft);

std::list<peak> read_constellation(std::string filename);

std::vector<std::vector<float>> get_fft_from_audio(float sec);

std::list<hash_pair> hash_create_from_audio(float sec);
```

```
float score(const struct count_ID &c) {
	return ((float) c.count)/std::pow(c.num_hashes, NORM_POW);
}

bool sortByScore(const struct count_ID &lhs, const struct count_ID &rhs) {
	return lhs.count == rhs.count ? score(lhs) > score(rhs) : lhs.count > rhs.count;
}

int fft_accelerator_fd;

int main()
{
	/*
	 * Assumes fft spectrogram files are availible at ./song_name, and that
	 * song_list.txt exists and contains a list of the song names.
	 */

	std::unordered_multimap<uint64_t, song_data> db;
	std::list<database_info> song_names;
	std::unordered_map<uint16_t, count_ID> results;
	struct hash_pair pair;
	std::pair<uint64_t, song_data> temp_db;
	struct database_info temp_db_info;
	std::string temp_match;
	std::string temp_s;

	std::string output;
	int hash_count;

	std::fstream file;
	std::string line;
	std::vector<std::string> song_file_list;

	uint16_t num_db = 0;

	// open device:
	static const char filename[] = "/dev/fft_accelerator";
	if( (fft_accelerator_fd = open(filename, O_RDWR)) == -1) {
		std::cerr << "could not open " << filename << std::endl;
		return -1;
	}

	file.open("song_list.txt");
```

```
while(getline(file, line)){
  if(!line.empty()){
        // skip most songs since board does not have enough ram to handle 30
        if (num_db % 6 != 2) { num_db++; continue;}

        num_db++;
        temp_s = "./"+ line;
        hash_count = 0;

        std::list<hash_pair> temp;
        temp = hash_create(temp_s, num_db);

        for(std::list<hash_pair>::iterator it = temp.begin();
                 it != temp.end(); ++it){

                temp_db.first = it->fingerprint;
                temp_db.second = it->value;
                db.insert(temp_db);

                hash_count++;
        }

        temp_db_info.song_name = temp_s;
        temp_db_info.hash_count = hash_count;
        temp_db_info.song_ID = num_db;
        song_names.push_back(temp_db_info);

        std::cout <<  "(" << num_db << ") ";
        std::cout << temp_s;
        std::cout << " databased.\n Number of hash table entries: ";
        std::cout << temp.size() << std::endl;
        std::cout << std::endl;
        std::cout << std::endl;
  }
}
file.close();

/*DEBUG*/
std::cout << "Full database completed \n\n" << std::endl;


while(true)
{
```

```
                std::cout << "Ready to identify. Press ENTER to identify the song playing.\n";
                std::cin.ignore();

                temp_s = line;
                std::list<hash_pair> identify;
                // identify = hash_create_noise(temp_s, num_db);
                identify = hash_create_from_audio(25);
                std::cout << "Done listening.\n";


                results = identify_sample(identify, db, song_names);

                std::vector<count_ID> sorted_results;
                for(auto iter = results.begin();
                        iter != results.end(); ++iter){
                        sorted_results.push_back(iter->second);
                }
                std::sort(sorted_results.begin(), sorted_results.end(), sortByScore);
                for (auto c = sorted_results.cbegin(); c != sorted_results.cend(); c++) {
                   std::cout << "-" << c->song << " /" << score(*c) << "/" << c->count << std::endl;
                }

        }

        return 0;
}


std::unordered_map<uint16_t, count_ID> identify_sample(
        const std::list<hash_pair> & sample_prints,
        const std::unordered_multimap<uint64_t, song_data> & database,
        std::list<database_info> song_list)
{
        std::cout << "call to identify" << std::endl;

        std::unordered_map<uint16_t, count_ID> results;
        //new database, keys are songIDs concatenated with time anchor
        //values are number of appearances, if 5 we've matched
        std::unordered_map<uint64_t, uint8_t> db2;
        uint64_t new_key;
        uint16_t identity;

        for(std::list<database_info>::iterator iter = song_list.begin();
```

```
        iter != song_list.end(); ++iter){
        //scaling may no longer be necesary, but currently used
        results[iter->song_ID].num_hashes = iter->hash_count;
        results[iter->song_ID].song = iter->song_name;
        //set count to zero, will now be number of target zones matched
        results[iter->song_ID].count = 0;

}

//for fingerpint in sampleFingerprints
for(auto iter = sample_prints.begin();
        iter != sample_prints.end(); ++iter){

   // get all the entries at this hash location
   const auto & ret = database.equal_range(iter->fingerprint);

   //lets insert the song_ID, time anchor pairs in our new database
   for(auto  it = ret.first; it != ret.second; ++it){

            new_key = it->second.song_ID;
            new_key = new_key << 16;
            new_key |= it->second.time_pt;
            new_key = new_key << 16;
            new_key |= iter->value.time_pt;

            db2[new_key]++;
   }

}
// second database is fully populated


//adds to their count in the results structure, which is returned
for(std::unordered_map<uint64_t,uint8_t>::iterator
                    it = db2.begin(); it != db2.end(); ++it){

        //full target zone matched
        if(it->second >= T_ZONE)
        {
                //std::cout << it->second << std::endl;
                identity = it->first >> 32;
                results[identity].count += (int) (it->second);
        }
```

```
        }

        return results;

}

std::list<hash_pair> hash_create(std::string song_name, uint16_t song_ID)
{
        std::cout << "call to hash_create" << std::endl;
        std::cout << "Song ID = " << song_ID << std::endl;

        std::list<peak> pruned_peaks;
        pruned_peaks = read_constellation(song_name);

        std::list<hash_pair> hash_entries;
        hash_entries = generate_fingerprints(pruned_peaks, song_name, song_ID);

        return hash_entries;
}

std::list<hash_pair> hash_create_from_audio(float sec)
{
        uint16_t song_ID = 0;
        std::string song_name = "AUDIO";
        std::cout << "call to hash_create_from_audio" << std::endl;
        std::vector<std::vector<float>> fft;
        fft = get_fft_from_audio(sec);

        std::list<peak> pruned_peaks;
        pruned_peaks = generate_constellation_map(fft, NFFT);

        std::list<hash_pair> hash_entries;
        hash_entries = generate_fingerprints(pruned_peaks, song_name, song_ID);

        return hash_entries;
}

std::list<hash_pair> generate_fingerprints(std::list<peak> pruned,
        std::string song_name, uint16_t song_ID)
{
        std::list<hash_pair> fingerprints;
        struct fingerprint f;
        struct song_data sdata;
```

```
        struct hash_pair entry;
        uint16_t target_zone_t;
        uint64_t template_print;
        struct peak other_point;
        struct peak anchor_point;

        int target_offset = 2;

        target_zone_t = T_ZONE;


        for(std::list<peak>::iterator it = pruned.begin();
         std::next(it, target_zone_t + target_offset) != pruned.end(); it++){

                anchor_point= *it;

                for(uint16_t i = 1; i <= target_zone_t; i++){

                        other_point = *(std::next(it, i + target_offset));

                        f.anchor = anchor_point.freq;
                        f.point = other_point.freq;
                        f.delta  = other_point.time - anchor_point.time;

                        sdata.song_name = song_name;
                        sdata.time_pt = anchor_point.time;
                        sdata.song_ID = song_ID;

                        template_print = f.anchor;
                        template_print = template_print << 16;
                        template_print |= f.point;
                        template_print = template_print << 16;
                        template_print |= f.delta;

                        entry.fingerprint = template_print;
                        entry.value = sdata;

                        fingerprints.push_back(entry);
                }
        }

        return fingerprints;
}
```

```
uint32_t sec_to_samples(float sec) {
	return (int) sec*(SAMPLING_FREQ/DOWN_SAMPLING_FACTOR);
}

float samples_to_sec(uint32_t samples) {
	return  ((float) samples)/SAMPLING_FREQ;
}

float ampl2float(ampl_t fixed) {
	// divide by 2^(fixed point accuracy) 2^7.
	return ((float) fixed) / std::pow(2.0, AMPL_FRACTIONAL_BITS);
}

#define ERR_IO 0xFFFFFFFFFFFFFFFFu
#define ERR_NVALID 0xFFFFFFFFFFFFFFFEu

uint64_t get_sample(std::vector<float> & fft) {
	fft_accelerator_arg_t vla;
	fft_accelerator_fft_t fft_struct;
	vla.fft_struct = &fft_struct;

	fft.clear();
	fft.reserve(N_FREQUENCIES);

	if (ioctl(fft_accelerator_fd, FFT_ACCELERATOR_READ_FFT, &vla)) {
		perror("ioctl(FFT_ACCELERATOR_READ_FFT) failed");
		return ERR_IO;
	}
	if(!fft_struct.valid) {
		return ERR_NVALID;
	}
	for (int i = 0; i < N_FREQUENCIES; i++) {
		//std::cout << ampl2float(fft_struct.fft[i]) << " ";
		fft.push_back(ampl2float(fft_struct.fft[i]));
	}
		//std::cout << std::endl;
	return fft_struct.time;
}


std::vector<std::vector<float>> get_fft_from_audio(float sec) {
```

```
	uint32_t samples = sec_to_samples(sec);
	std::cout << samples << std::endl;
	std::vector<std::vector<float>> spec;
	spec.reserve(N_FREQUENCIES);
	for (int i = 0; i < N_FREQUENCIES; i++) {
		std::vector<float> vec;
		vec.reserve(samples);
		spec.push_back(vec);
	}
	std::vector<float> fft_temp;
	uint64_t time;

	for (uint32_t i = 0; i < samples; i++) {
		time = get_sample(fft_temp);
		//this assumes we miss nothing

		for(uint32_t j = 0; j < N_FREQUENCIES; j++){
			//std::cout << fft_temp[j] << " ";
			spec[j].push_back(std::abs(fft_temp[j]));
		}
			//std::cout << std::endl;
		if (time == ERR_IO || time == ERR_NVALID) {
			std::cout << "Could not get audio fft\n";
		   // spec[0].size < samples
			return spec;
		}
		 //copy contents of fft_temp into spec[i], averaging if there are missed times.

	}
	return spec;
}

// Eitan's re-write:

inline int freq_to_bin(uint16_t freq) {
	if (freq <  BIN1)
		return 1;
	if (freq <  BIN2)
		return 2;
	if (freq <  BIN3)
		return 3;
	if (freq <  BIN4)
		return 4;
```

```
        if (freq <  BIN5)
                return 5;
        if (freq <  BIN6)
                return 6;
        return 0;
}

std::list<peak_raw> get_raw_peaks(std::vector<std::vector<float>> fft, int nfft)
{
   std::list<peak_raw> peaks;
   uint16_t size_in_time;

   size_in_time = fft[0].size();
   for(uint16_t j = 1; j < size_in_time-2; j++){
        // WARNING not parametrized by NBINS
        float max_ampl_by_bin[NBINS + 1] = {FLT_MIN, FLT_MIN, FLT_MIN, FLT_MIN,
FLT_MIN, FLT_MIN, FLT_MIN, FLT_MIN};
        struct peak_raw max_peak_by_bin[NBINS + 1] = {};
      for(uint16_t i = 0; i < fft.size() - 1; i++){
        if(     fft[i][j] > fft[i][j-1]                    && //west
                fft[i][j] > fft[i][j+1]                       && //east
                (i < 1          || fft[i][j] > fft[i-1][j]) && //north
                (i >= fft.size() || fft[i][j] > fft[i+1][j])) { //south
                if (fft[i][j] > max_ampl_by_bin[freq_to_bin(i)]) {
                   max_ampl_by_bin[freq_to_bin(i)] = fft[i][j];
                   max_peak_by_bin[freq_to_bin(i)].freq = i;
                   max_peak_by_bin[freq_to_bin(i)].ampl = fft[i][j];
                   max_peak_by_bin[freq_to_bin(i)].time = j;
                }
        }
      }
        for (int k = 1; k <= NBINS; k++) {
           if (max_peak_by_bin[k].time != 0) {
           peaks.push_back(max_peak_by_bin[k]);
           }
        }
   }
   return peaks;
}

std::list<peak> prune_in_time(std::list<peak_raw> unpruned_peaks) {
        int time = 0;
        float num[NBINS + 1] = { };
```

```
	float den[NBINS + 1] = { };
	float dev[NBINS + 1] = { };
	int bin;
	unsigned int bin_counts[NBINS + 1] = { };
	unsigned int bin_prune_counts[NBINS + 1] = { };
	std::list<peak> pruned_peaks;
	auto add_iter = unpruned_peaks.cbegin();
	auto dev_iter = unpruned_peaks.cbegin();
	for(auto avg_iter = unpruned_peaks.cbegin(); add_iter != unpruned_peaks.cend(); ){

		if (avg_iter->time <= time + PRUNING_TIME_WINDOW && avg_iter !=
unpruned_peaks.cend()) {
			bin = freq_to_bin(avg_iter->freq);
			den[bin]++;
			num[bin] += avg_iter->ampl;
			avg_iter++;
		} else {

			while(dev_iter != avg_iter){
				if (dev_iter->time <= time + PRUNING_TIME_WINDOW
					&& dev_iter != unpruned_peaks.cend()) {

					bin = freq_to_bin(dev_iter->freq);
					if(den[bin]){
						dev[bin] += pow(dev_iter->ampl - num[bin]/den[bin],
2);

					}
					else{
						dev[bin] = den[bin];
					}
				}
				dev_iter++;
			}
			for (int i = 1; i <= NBINS; i++)
			{
				if(den[i]){
					dev[i] = sqrt(dev[i]/den[i]);
				}
				//std::cout << dev[i] << " ";
			}
			//std::cout << std::endl;
			while (add_iter != avg_iter) {
				bin = freq_to_bin(add_iter->freq);
```

```
                                if (den[bin] && add_iter->ampl > STD_DEV_COEF*dev[bin] +
num[bin]/den[bin]  ) {
                                        pruned_peaks.push_back({add_iter->freq,
add_iter->time});
                                        bin_counts[freq_to_bin(add_iter->freq)]++;
                                } else {
                                        bin_prune_counts[freq_to_bin(add_iter->freq)]++;
                                }
                                add_iter++;
                        }
                        memset(num, 0, sizeof(num));
                        memset(den, 0, sizeof(den));
                        time += PRUNING_TIME_WINDOW;
                }
        }
        for (int i = 1; i <= NBINS; i++) {
                std::cout << "bin " << i << ": " << bin_counts[i] << "|  pruned: " <<
bin_prune_counts[i] << std::endl;
        }
        return pruned_peaks;
}



std::list<peak> generate_constellation_map(std::vector<std::vector<float>> fft, int nfft)
{
        std::list<peak_raw> unpruned_map;
        unpruned_map = get_raw_peaks(fft, nfft);
        return prune_in_time(unpruned_map);
}

std::list<peak> read_constellation(std::string filename){

        std::ifstream fin;
        std::list<peak> constellation;
        uint32_t peak_32;
        struct peak temp;
        std::streampos size;
        char * memblock;
        int i;

        fin.open(filename+"_48.realpeak", std::ios::binary | std::ios::in
                        | std::ios::ate);
```

```
        i = 0;
        if (fin.is_open())
        {
            size = fin.tellg();
            memblock = new char [size];

            fin.seekg (0, std::ios::beg);
            fin.read (memblock, size);
            fin.close();

            while(i < size)
            {

                    peak_32 = *(uint32_t *)(memblock+i);
                    temp.time = peak_32;
                    temp.freq = peak_32 >> 16;
                    constellation.push_back(temp);
                    /* MUST incremet by this amount here*/
                    i += sizeof(peak_32);
            }


            delete[] memblock;
        }

        return constellation;

}
```

recognize_board.c

```
/*
 */

#include <iostream>
#include <fstream>
#include <string>
#include <cstring>
#include <sstream>
#include <list>
#include <set>
#include <vector>
#include <unordered_map>
```

```
#include <algorithm>
#include <cfloat>
#include <cmath>
#include "fft_accelerator.h"
#include <sys/ioctl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <fcntl.h>

#define NFFT 512
#define NBINS 6
#define BIN0 0
#define BIN1 10
#define BIN2 20
#define BIN3 40
#define BIN4 80
#define BIN5 160
#define BIN6 160

#define PRUNING_COEF 1.4f
#define PRUNING_TIME_WINDOW 500
#define NORM_POW 1.0f
#define STD_DEV_COEF 1.25
#define T_ZONE 4

struct peak_raw {
        float ampl;
        uint16_t freq;
        uint16_t time;
};

struct peak {
        uint16_t freq;
        uint16_t time;
};

struct fingerprint {
        uint16_t anchor;
        uint16_t point;
        uint16_t delta;
};

struct song_data {
```

```
	std::string song_name;
	uint16_t time_pt;
	uint16_t song_ID;
};

struct hash_pair {
	uint64_t fingerprint;
	struct song_data value;
};

struct count_ID {
	std::string song;
	int count;
	int num_hashes;
};

struct database_info{
	std::string song_name;
	uint16_t song_ID;
	int hash_count;
};

std::list<hash_pair> hash_create(std::string song_name, uint16_t song_ID);

std::list<hash_pair> generate_fingerprints(std::list<peak> pruned,
	std::string song_name, uint16_t song_ID);

std::unordered_map<uint16_t, count_ID> identify_sample(
	const std::list<hash_pair> & sample_prints,
	const std::unordered_multimap<uint64_t, song_data> & database,
	std::list<database_info> song_list);

std::list<peak> generate_constellation_map(std::vector<std::vector<float>> fft, int nfft);

std::list<peak> read_constellation(std::string filename);

std::vector<std::vector<float>> get_fft_from_audio(float sec);

std::list<hash_pair> hash_create_from_audio(float sec);

float score(const struct count_ID &c) {
	return ((float) c.count)/std::pow(c.num_hashes, NORM_POW);
}
```

```
bool sortByScore(const struct count_ID &lhs, const struct count_ID &rhs) {
	return lhs.count == rhs.count ? score(lhs) > score(rhs) : lhs.count > rhs.count;
}

int fft_accelerator_fd;

int main()
{
	/*
	 * Assumes fft spectrogram files are availible at ./song_name, and that
	 * song_list.txt exists and contains a list of the song names.
	 */

	std::unordered_multimap<uint64_t, song_data> db;
	std::list<database_info> song_names;
	std::unordered_map<uint16_t, count_ID> results;
	struct hash_pair pair;
	std::pair<uint64_t, song_data> temp_db;
	struct database_info temp_db_info;
	std::string temp_match;
	std::string temp_s;

	std::string output;
	int hash_count;

	std::fstream file;
	std::string line;
	std::vector<std::string> song_file_list;

	uint16_t num_db = 0;

	// open device:
	static const char filename[] = "/dev/fft_accelerator";
	if( (fft_accelerator_fd = open(filename, O_RDWR)) == -1) {
		std::cerr << "could not open " << filename << std::endl;
		return -1;
	}

	file.open("song_list.txt");
	while(getline(file, line)){
	  if(!line.empty()){
```

```
            num_db++;
            temp_s = "./"+ line;
            hash_count = 0;

            std::list<hash_pair> temp;
            temp = hash_create(temp_s, num_db);

            for(std::list<hash_pair>::iterator it = temp.begin();
                        it != temp.end(); ++it){

                        temp_db.first = it->fingerprint;
                        temp_db.second = it->value;
                        db.insert(temp_db);

                        hash_count++;
            }

            temp_db_info.song_name = temp_s;
            temp_db_info.hash_count = hash_count;
            temp_db_info.song_ID = num_db;
            song_names.push_back(temp_db_info);

            std::cout <<  "(" << num_db << ") ";
            std::cout << temp_s;
            std::cout << " databased.\n Number of hash table entries: ";
            std::cout << temp.size() << std::endl;
            std::cout << std::endl;
            std::cout << std::endl;
      }
}
file.close();

/*DEBUG*/
std::cout << "Full database completed \n\n" << std::endl;


while(true)
{
            std::cout << "Ready to identify. Press ENTER to identify the song playing.\n";
            std::cin.ignore();

            temp_s = line;
            std::list<hash_pair> identify;
```

```
                // identify = hash_create_noise(temp_s, num_db);
                identify = hash_create_from_audio(30);
                std::cout << "Done listening.\n";


                results = identify_sample(identify, db, song_names);

                std::vector<count_ID> sorted_results;
                for(auto iter = results.begin();
                        iter != results.end(); ++iter){
                        sorted_results.push_back(iter->second);
                }
                std::sort(sorted_results.begin(), sorted_results.end(), sortByScore);
                for (auto c = sorted_results.cbegin(); c != sorted_results.cend(); c++) {
                   std::cout << "-" << c->song << " /" << score(*c) << "/" << c->count << std::endl;
                }

        }

        return 0;
}


std::unordered_map<uint16_t, count_ID> identify_sample(
        const std::list<hash_pair> & sample_prints,
        const std::unordered_multimap<uint64_t, song_data> & database,
        std::list<database_info> song_list)
{
        std::cout << "call to identify" << std::endl;

        std::unordered_map<uint16_t, count_ID> results;
        //new database, keys are songIDs concatenated with time anchor
        //values are number of appearances, if 5 we've matched
        std::unordered_map<uint64_t, uint8_t> db2;
        uint64_t new_key;
        uint16_t identity;

        for(std::list<database_info>::iterator iter = song_list.begin();
                iter != song_list.end(); ++iter){
                //scaling may no longer be necessary, but currently used
                results[iter->song_ID].num_hashes = iter->hash_count;
                results[iter->song_ID].song = iter->song_name;
                //set count to zero, will now be number of target zones matched
```

```
            results[iter->song_ID].count = 0;

        }

        //for fingerpint in sampleFingerprints
        for(auto iter = sample_prints.begin();
                iter != sample_prints.end(); ++iter){

           // get all the entries at this hash location
           const auto & ret = database.equal_range(iter->fingerprint);

           //lets insert the song_ID, time anchor pairs in our new database
           for(auto  it = ret.first; it != ret.second; ++it){

                   new_key = it->second.song_ID;
                   new_key = new_key << 16;
                   new_key |= it->second.time_pt;
                   new_key = new_key << 16;
                   new_key |= iter->value.time_pt;

                   db2[new_key]++;
           }

        }
        // second database is fully populated


        //adds to their count in the results structure, which is returned
        for(std::unordered_map<uint64_t,uint8_t>::iterator
                        it = db2.begin(); it != db2.end(); ++it){

                //full target zone matched
                if(it->second >= T_ZONE)
                {
                        //std::cout << it->second << std::endl;
                        identity = it->first >> 32;
                        results[identity].count += (int) (it->second);
                }
        }

        return results;

}
```

```
std::list<hash_pair> hash_create(std::string song_name, uint16_t song_ID)
{
	std::cout << "call to hash_create" << std::endl;
	std::cout << "Song ID = " << song_ID << std::endl;

	std::list<peak> pruned_peaks;
	pruned_peaks = read_constellation(song_name);

	std::list<hash_pair> hash_entries;
	hash_entries = generate_fingerprints(pruned_peaks, song_name, song_ID);

	return hash_entries;
}

std::list<hash_pair> hash_create_from_audio(float sec)
{
	uint16_t song_ID = 0;
	std::string song_name = "AUDIO";
	std::cout << "call to hash_create_from_audio" << std::endl;
	std::vector<std::vector<float>> fft;
	fft = get_fft_from_audio(sec);

	std::list<peak> pruned_peaks;
	pruned_peaks = generate_constellation_map(fft, NFFT);

	std::list<hash_pair> hash_entries;
	hash_entries = generate_fingerprints(pruned_peaks, song_name, song_ID);

	return hash_entries;
}

std::list<hash_pair> generate_fingerprints(std::list<peak> pruned,
	std::string song_name, uint16_t song_ID)
{
	std::list<hash_pair> fingerprints;
	struct fingerprint f;
	struct song_data sdata;
	struct hash_pair entry;
	uint16_t target_zone_t;
	uint64_t template_print;
	struct peak other_point;
	struct peak anchor_point;
```

```
        int target_offset = 2;

        target_zone_t = T_ZONE;


        for(std::list<peak>::iterator it = pruned.begin();
         std::next(it, target_zone_t + target_offset) != pruned.end(); it++){

                anchor_point= *it;

                for(uint16_t i = 1; i <= target_zone_t; i++){

                        other_point = *(std::next(it, i + target_offset));

                        f.anchor = anchor_point.freq;
                        f.point = other_point.freq;
                        f.delta  = other_point.time - anchor_point.time;

                        sdata.song_name = song_name;
                        sdata.time_pt = anchor_point.time;
                        sdata.song_ID = song_ID;

                        template_print = f.anchor;
                        template_print = template_print << 16;
                        template_print |= f.point;
                        template_print = template_print << 16;
                        template_print |= f.delta;

                        entry.fingerprint = template_print;
                        entry.value = sdata;

                        fingerprints.push_back(entry);
                }
        }

        return fingerprints;
}


uint32_t sec_to_samples(float sec) {
        return (int) sec*(SAMPLING_FREQ/DOWN_SAMPLING_FACTOR);
}
```

```
float samples_to_sec(uint32_t samples) {
	return  ((float) samples)/SAMPLING_FREQ;
}

float ampl2float(ampl_t fixed) {
	// divide by 2^(fixed point accuracy) 2^7.
	return ((float) fixed) / std::pow(2.0, AMPL_FRACTIONAL_BITS);
}

#define ERR_IO 0xFFFFFFFFFFFFFFFFu
#define ERR_NVALID 0xFFFFFFFFFFFFFFFEu

uint64_t get_sample(std::vector<float> & fft) {
	fft_accelerator_arg_t vla;
	fft_accelerator_fft_t fft_struct;
	vla.fft_struct = &fft_struct;

	fft.clear();
	fft.reserve(N_FREQUENCIES);

	if (ioctl(fft_accelerator_fd, FFT_ACCELERATOR_READ_FFT, &vla)) {
		perror("ioctl(FFT_ACCELERATOR_READ_FFT) failed");
		return ERR_IO;
	}
	if(!fft_struct.valid) {
		return ERR_NVALID;
	}
	for (int i = 0; i < N_FREQUENCIES; i++) {
		//std::cout << ampl2float(fft_struct.fft[i]) << " ";
		fft.push_back(ampl2float(fft_struct.fft[i]));
	}
		//std::cout << std::endl;
	return fft_struct.time;
}


std::vector<std::vector<float>> get_fft_from_audio(float sec) {
	uint32_t samples = sec_to_samples(sec);
	std::cout << samples << std::endl;
	std::vector<std::vector<float>> spec;
	spec.reserve(N_FREQUENCIES);
	for (int i = 0; i < N_FREQUENCIES; i++) {
```

```
                std::vector<float> vec;
                vec.reserve(samples);
                spec.push_back(vec);
        }
        std::vector<float> fft_temp;
        uint64_t time;

        for (uint32_t i = 0; i < samples; i++) {
                time = get_sample(fft_temp);
                //this assumes we miss nothing

                for(uint32_t j = 0; j < N_FREQUENCIES; j++){
                        //std::cout << fft_temp[j] << " ";
                        spec[j].push_back(std::abs(fft_temp[j]));
                }
                        //std::cout << std::endl;
                if (time == ERR_IO || time == ERR_NVALID) {
                        std::cout << "Could not get audio fft\n";
                      // spec[0].size < samples
                        return spec;
                }
                 //copy contents of fft_temp into spec[i], averaging if there are missed times.

        }
        return spec;
}

// Eitan's re-write:

inline int freq_to_bin(uint16_t freq) {
        if (freq <  BIN1)
                return 1;
        if (freq <  BIN2)
                return 2;
        if (freq <  BIN3)
                return 3;
        if (freq <  BIN4)
                return 4;
        if (freq <  BIN5)
                return 5;
        if (freq <  BIN6)
                return 6;
        return 0;
```

```
}

std::list<peak_raw> get_raw_peaks(std::vector<std::vector<float>> fft, int nfft)
{
   std::list<peak_raw> peaks;
   uint16_t size_in_time;

   size_in_time = fft[0].size();
   for(uint16_t j = 1; j < size_in_time-2; j++){
         // WARNING not parametrized by NBINS
         float max_ampl_by_bin[NBINS + 1] = {FLT_MIN, FLT_MIN, FLT_MIN, FLT_MIN,
FLT_MIN, FLT_MIN, FLT_MIN};
         struct peak_raw max_peak_by_bin[NBINS + 1] = {};
      for(uint16_t i = 0; i < fft.size() - 1; i++){
         if(     fft[i][j] > fft[i][j-1]                    && //west
                  fft[i][j] > fft[i][j+1]                      && //east
                  (i < 1          || fft[i][j] > fft[i-1][j]) && //north
                  (i >= fft.size() || fft[i][j] > fft[i+1][j])) { //south
                  if (fft[i][j] > max_ampl_by_bin[freq_to_bin(i)]) {
                     max_ampl_by_bin[freq_to_bin(i)] = fft[i][j];
                     max_peak_by_bin[freq_to_bin(i)].freq = i;
                     max_peak_by_bin[freq_to_bin(i)].ampl = fft[i][j];
                     max_peak_by_bin[freq_to_bin(i)].time = j;
                  }
         }
      }
         for (int k = 1; k <= NBINS; k++) {
            if (max_peak_by_bin[k].time != 0) {
            peaks.push_back(max_peak_by_bin[k]);
            }
         }
   }
   return peaks;
}

std::list<peak> prune_in_time(std::list<peak_raw> unpruned_peaks) {
         int time = 0;
         float num[NBINS + 1] = { };
         float den[NBINS + 1] = { };
         float dev[NBINS + 1] = { };
         int bin;
         unsigned int bin_counts[NBINS + 1] = { };
         unsigned int bin_prune_counts[NBINS + 1] = { };
```

```
	std::list<peak> pruned_peaks;
	auto add_iter = unpruned_peaks.cbegin();
	auto dev_iter = unpruned_peaks.cbegin();
	for(auto avg_iter = unpruned_peaks.cbegin(); add_iter != unpruned_peaks.cend(); ){

		if (avg_iter->time <= time + PRUNING_TIME_WINDOW && avg_iter != 
unpruned_peaks.cend()) {
			bin = freq_to_bin(avg_iter->freq);
			den[bin]++;
			num[bin] += avg_iter->ampl;
			avg_iter++;
		} else {

			while(dev_iter != avg_iter){
				if (dev_iter->time <= time + PRUNING_TIME_WINDOW
					&& dev_iter != unpruned_peaks.cend()) {

					bin = freq_to_bin(dev_iter->freq);
					if(den[bin]){
						dev[bin] += pow(dev_iter->ampl - num[bin]/den[bin], 
2);
					}
					else{
						dev[bin] = den[bin];
					}
				}
				dev_iter++;
			}
			for (int i = 1; i <= NBINS; i++)
			{
				if(den[i]){
					dev[i] = sqrt(dev[i]/den[i]);
				}
				//std::cout << dev[i] << " ";
			}
			//std::cout << std::endl;
			while (add_iter != avg_iter) {
				bin = freq_to_bin(add_iter->freq);
				if (den[bin] && add_iter->ampl > STD_DEV_COEF*dev[bin] + 
num[bin]/den[bin]  ) {
					pruned_peaks.push_back({add_iter->freq, 
add_iter->time});
					bin_counts[freq_to_bin(add_iter->freq)]++;
```

```
                        } else {
                                bin_prune_counts[freq_to_bin(add_iter->freq)]++;
                        }
                        add_iter++;
                }
                memset(num, 0, sizeof(num));
                memset(den, 0, sizeof(den));
                time += PRUNING_TIME_WINDOW;
        }
    }
    for (int i = 1; i <= NBINS; i++) {
        std::cout << "bin " << i << ": " << bin_counts[i] << "|  pruned: " <<
bin_prune_counts[i] << std::endl;
    }
    return pruned_peaks;
}



std::list<peak> generate_constellation_map(std::vector<std::vector<float>> fft, int nfft)
{
    std::list<peak_raw> unpruned_map;
    unpruned_map = get_raw_peaks(fft, nfft);
    return prune_in_time(unpruned_map);
}

std::list<peak> read_constellation(std::string filename){

    std::ifstream fin;
    std::list<peak> constellation;
    uint32_t peak_32;
    struct peak temp;
    std::streampos size;
    char * memblock;
    int i;

    fin.open(filename+".boardpeak", std::ios::binary | std::ios::in
                | std::ios::ate);

    i = 0;
    if (fin.is_open())
    {
        size = fin.tellg();
```

```
            memblock = new char [size];

            fin.seekg (0, std::ios::beg);
            fin.read (memblock, size);
            fin.close();

            while(i < size)
            {

                peak_32 = *(uint32_t *)(memblock+i);
                temp.time = peak_32;
                temp.freq = peak_32 >> 16;
                constellation.push_back(temp);
                /* MUST incremet by this amount here*/
                i += sizeof(peak_32);
            }


            delete[] memblock;
        }

        return constellation;

}
```

db.cpp

```
/*
 */

#include <iostream>
#include <fstream>
#include <string>
#include <cstring>
#include <sstream>
#include <list>
#include <set>
#include <vector>
#include <unordered_map>
#include <algorithm>
#include <cfloat>
#include <cmath>
#include "fft_accelerator.h"
#include <sys/ioctl.h>
```

```
#include <sys/stat.h>
#include <sys/types.h>
#include <fcntl.h>

#define NFFT 512
#define NBINS 6
#define BIN0 0
#define BIN1 10
#define BIN2 20
#define BIN3 40
#define BIN4 80
#define BIN5 160
#define BIN6 160

#define PRUNING_COEF 1.4f
#define PRUNING_TIME_WINDOW 500
#define NORM_POW 1.0f
#define STD_DEV_COEF 1.25
#define T_ZONE 4

struct peak_raw {
	float ampl;
	uint16_t freq;
	uint16_t time;
};

struct peak {
	uint16_t freq;
	uint16_t time;
};

struct fingerprint {
	uint16_t anchor;
	uint16_t point;
	uint16_t delta;
};

struct song_data {
	std::string song_name;
	uint16_t time_pt;
	uint16_t song_ID;
};
```

```
struct hash_pair {
	uint64_t fingerprint;
	struct song_data value;
};

struct count_ID {
	std::string song;
	int count;
	int num_hashes;
};

struct database_info{
	std::string song_name;
	uint16_t song_ID;
	int hash_count;
};

void write_constellation(std::list<peak> pruned, std::string filename);

std::list<hash_pair> hash_create(std::string song_name, uint16_t song_ID);

std::list<hash_pair> generate_fingerprints(std::list<peak> pruned,
	std::string song_name, uint16_t song_ID);

std::unordered_map<uint16_t, count_ID> identify_sample(
	const std::list<hash_pair> & sample_prints,
	const std::unordered_multimap<uint64_t, song_data> & database,
	std::list<database_info> song_list);

std::list<peak> generate_constellation_map(std::vector<std::vector<float>> fft, int nfft);

std::list<peak> read_constellation(std::string filename);

std::vector<std::vector<float>> get_fft_from_audio(float sec);

std::list<hash_pair> hash_create_from_audio(float sec);

std::list<peak> create_map_from_audio(float sec);

float score(const struct count_ID &c) {
	return ((float) c.count)/std::pow(c.num_hashes, NORM_POW);
}
```

```
bool sortByScore(const struct count_ID &lhs, const struct count_ID &rhs) {
	return lhs.count == rhs.count ? score(lhs) > score(rhs) : lhs.count > rhs.count;
}

int fft_accelerator_fd;

int main()
{
	/*
	 * Assumes fft spectrogram files are availible at ./song_name, and that
	 * song_list.txt exists and contains a list of the song names.
	 */

	std::unordered_multimap<uint64_t, song_data> db;
	std::list<database_info> song_names;
	std::unordered_map<uint16_t, count_ID> results;
	struct hash_pair pair;
	std::pair<uint64_t, song_data> temp_db;
	struct database_info temp_db_info;
	std::string temp_match;
	std::string temp_s;

	std::string output;

	std::fstream file;
	std::string line;
	std::vector<std::string> song_file_list;


	// open device:
	static const char filename[] = "/dev/fft_accelerator";
	if( (fft_accelerator_fd = open(filename, O_RDWR)) == -1) {
		std::cerr << "could not open " << filename << std::endl;
		return -1;
	}


	std::cout << "Full database completed \n\n" << std::endl;


	while(true)
	{
		std::string song_name;
```

```
                std::cout << "Ready to create db song entry. Enter the name of the song
playing.\n";
                std::cin >> song_name;

                temp_s = line;
                std::list<peak> pruned;
                pruned = create_map_from_audio(125);
                std::cout << "Done listening.\n";
                write_constellation(pruned, song_name + ".board");
                std::cout << "Wrote constellation map for " << song_name << ".\n";
        }

        return 0;
}


std::unordered_map<uint16_t, count_ID> identify_sample(
        const std::list<hash_pair> & sample_prints,
        const std::unordered_multimap<uint64_t, song_data> & database,
        std::list<database_info> song_list)
{
        std::cout << "call to identify" << std::endl;

        std::unordered_map<uint16_t, count_ID> results;
        //new database, keys are songIDs concatenated with time anchor
        //values are number of appearances, if 5 we've matched
        std::unordered_map<uint64_t, uint8_t> db2;
        uint64_t new_key;
        uint16_t identity;

        for(std::list<database_info>::iterator iter = song_list.begin();
                iter != song_list.end(); ++iter){
                //scaling may no longer be necessary, but currently used
                results[iter->song_ID].num_hashes = iter->hash_count;
                results[iter->song_ID].song = iter->song_name;
                //set count to zero, will now be number of target zones matched
                results[iter->song_ID].count = 0;

        }

        //for fingerpint in sampleFingerprints
        for(auto iter = sample_prints.begin();
                iter != sample_prints.end(); ++iter){
```

```
            // get all the entries at this hash location
            const auto & ret = database.equal_range(iter->fingerprint);

            //lets insert the song_ID, time anchor pairs in our new database
            for(auto  it = ret.first; it != ret.second; ++it){

                        new_key = it->second.song_ID;
                        new_key = new_key << 16;
                        new_key |= it->second.time_pt;
                        new_key = new_key << 16;
                        new_key |= iter->value.time_pt;

                        db2[new_key]++;
            }

        }
        // second database is fully populated


        //adds to their count in the results structure, which is returned
        for(std::unordered_map<uint64_t,uint8_t>::iterator
                                it = db2.begin(); it != db2.end(); ++it){

                //full target zone matched
                if(it->second >= T_ZONE)
                {
                        //std::cout << it->second << std::endl;
                        identity = it->first >> 32;
                        results[identity].count += (int) (it->second);
                }
        }

        return results;

}

std::list<hash_pair> hash_create(std::string song_name, uint16_t song_ID)
{
        std::cout << "call to hash_create" << std::endl;
        std::cout << "Song ID = " << song_ID << std::endl;

        std::list<peak> pruned_peaks;
```

```
	pruned_peaks = read_constellation(song_name);

	std::list<hash_pair> hash_entries;
	hash_entries = generate_fingerprints(pruned_peaks, song_name, song_ID);

	return hash_entries;
}


std::list<peak> create_map_from_audio(float sec)
{
	std::list<peak> pruned_peaks;
	std::cout << "call to create_map_from_audio" << std::endl;
	std::vector<std::vector<float>> fft;
	fft = get_fft_from_audio(sec);
	pruned_peaks = generate_constellation_map(fft, NFFT);
	return pruned_peaks;
}

std::list<hash_pair> hash_create_from_audio(float sec)
{
	uint16_t song_ID = 0;
	std::string song_name = "AUDIO";
	std::cout << "call to hash_create_from_audio" << std::endl;
	std::vector<std::vector<float>> fft;
	fft = get_fft_from_audio(sec);

	std::list<peak> pruned_peaks;
	pruned_peaks = generate_constellation_map(fft, NFFT);

	std::list<hash_pair> hash_entries;
	hash_entries = generate_fingerprints(pruned_peaks, song_name, song_ID);

	return hash_entries;
}

std::list<hash_pair> generate_fingerprints(std::list<peak> pruned,
	std::string song_name, uint16_t song_ID)
{
	std::list<hash_pair> fingerprints;
	struct fingerprint f;
	struct song_data sdata;
	struct hash_pair entry;
```

```
        uint16_t target_zone_t;
        uint64_t template_print;
        struct peak other_point;
        struct peak anchor_point;

        int target_offset = 2;

        target_zone_t = T_ZONE;


        for(std::list<peak>::iterator it = pruned.begin();
         std::next(it, target_zone_t + target_offset) != pruned.end(); it++){

                anchor_point= *it;

                for(uint16_t i = 1; i <= target_zone_t; i++){

                        other_point = *(std::next(it, i + target_offset));

                        f.anchor = anchor_point.freq;
                        f.point = other_point.freq;
                        f.delta  = other_point.time - anchor_point.time;

                        sdata.song_name = song_name;
                        sdata.time_pt = anchor_point.time;
                        sdata.song_ID = song_ID;

                        template_print = f.anchor;
                        template_print = template_print << 16;
                        template_print |= f.point;
                        template_print = template_print << 16;
                        template_print |= f.delta;

                        entry.fingerprint = template_print;
                        entry.value = sdata;

                        fingerprints.push_back(entry);
                }
        }

        return fingerprints;
}
```

```
uint32_t sec_to_samples(float sec) {
	return (int) sec*(SAMPLING_FREQ/DOWN_SAMPLING_FACTOR);
}

float samples_to_sec(uint32_t samples) {
	return  ((float) samples)/SAMPLING_FREQ;
}

float ampl2float(ampl_t fixed) {
	// divide by 2^(fixed point accuracy) 2^7.
	return ((float) fixed) / std::pow(2.0, AMPL_FRACTIONAL_BITS);
}

#define ERR_IO 0xFFFFFFFFFFFFFFFFu
#define ERR_NVALID 0xFFFFFFFFFFFFFFFEu

uint64_t get_sample(std::vector<float> & fft) {
	fft_accelerator_arg_t vla;
	fft_accelerator_fft_t fft_struct;
	vla.fft_struct = &fft_struct;

	fft.clear();
	fft.reserve(N_FREQUENCIES);

	if (ioctl(fft_accelerator_fd, FFT_ACCELERATOR_READ_FFT, &vla)) {
		perror("ioctl(FFT_ACCELERATOR_READ_FFT) failed");
		return ERR_IO;
	}
	if(!fft_struct.valid) {
		return ERR_NVALID;
	}
	for (int i = 0; i < N_FREQUENCIES; i++) {
		//std::cout << ampl2float(fft_struct.fft[i]) << " ";
		fft.push_back(ampl2float(fft_struct.fft[i]));
	}
		//std::cout << std::endl;
	return fft_struct.time;
}


std::vector<std::vector<float>> get_fft_from_audio(float sec) {
	uint32_t samples = sec_to_samples(sec);
```

```
	std::cout << samples << std::endl;
	std::vector<std::vector<float>> spec;
	spec.reserve(N_FREQUENCIES);
	for (int i = 0; i < N_FREQUENCIES; i++) {
		std::vector<float> vec;
		vec.reserve(samples);
		spec.push_back(vec);
	}
	std::vector<float> fft_temp;
	uint64_t time;

	for (uint32_t i = 0; i < samples; i++) {
		time = get_sample(fft_temp);
		//this assumes we miss nothing

		for(uint32_t j = 0; j < N_FREQUENCIES; j++){
			//std::cout << fft_temp[j] << " ";
			spec[j].push_back(std::abs(fft_temp[j]));
		}
			//std::cout << std::endl;
		if (time == ERR_IO || time == ERR_NVALID) {
			std::cout << "Could not get audio fft\n";
		// spec[0].size < samples
			return spec;
		}
		 //copy contents of fft_temp into spec[i], averaging if there are missed times.

	}
	return spec;
}

// Eitan's re-write:

inline int freq_to_bin(uint16_t freq) {
	if (freq <  BIN1)
		return 1;
	if (freq <  BIN2)
		return 2;
	if (freq <  BIN3)
		return 3;
	if (freq <  BIN4)
		return 4;
	if (freq <  BIN5)
```

```
                return 5;
        if (freq <  BIN6)
                return 6;
        return 0;
}

std::list<peak_raw> get_raw_peaks(std::vector<std::vector<float>> fft, int nfft)
{
   std::list<peak_raw> peaks;
   uint16_t size_in_time;

   size_in_time = fft[0].size();
   for(uint16_t j = 1; j < size_in_time-2; j++){
        // WARNING not parametrized by NBINS
        float max_ampl_by_bin[NBINS + 1] = {FLT_MIN, FLT_MIN, FLT_MIN, FLT_MIN,
FLT_MIN, FLT_MIN, FLT_MIN};
        struct peak_raw max_peak_by_bin[NBINS + 1] = {};
      for(uint16_t i = 0; i < fft.size() - 1; i++){
        if(     fft[i][j] > fft[i][j-1]                 && //west
                fft[i][j] > fft[i][j+1]                     && //east
                (i < 1          || fft[i][j] > fft[i-1][j]) && //north
                (i >= fft.size() || fft[i][j] > fft[i+1][j])) { //south
                if (fft[i][j] > max_ampl_by_bin[freq_to_bin(i)]) {
                   max_ampl_by_bin[freq_to_bin(i)] = fft[i][j];
                   max_peak_by_bin[freq_to_bin(i)].freq = i;
                   max_peak_by_bin[freq_to_bin(i)].ampl = fft[i][j];
                   max_peak_by_bin[freq_to_bin(i)].time = j;
                }
        }
      }
        for (int k = 1; k <= NBINS; k++) {
           if (max_peak_by_bin[k].time != 0) {
           peaks.push_back(max_peak_by_bin[k]);
           }
        }
   }
   return peaks;
}

std::list<peak> prune_in_time(std::list<peak_raw> unpruned_peaks) {
        int time = 0;
        float num[NBINS + 1] = { };
        float den[NBINS + 1] = { };
```

```
	float dev[NBINS + 1] = { };
	int bin;
	unsigned int bin_counts[NBINS + 1] = { };
	unsigned int bin_prune_counts[NBINS + 1] = { };
	std::list<peak> pruned_peaks;
	auto add_iter = unpruned_peaks.cbegin();
	auto dev_iter = unpruned_peaks.cbegin();
	for(auto avg_iter = unpruned_peaks.cbegin(); add_iter != unpruned_peaks.cend(); ){

		if (avg_iter->time <= time + PRUNING_TIME_WINDOW && avg_iter !=
unpruned_peaks.cend()) {
			bin = freq_to_bin(avg_iter->freq);
			den[bin]++;
			num[bin] += avg_iter->ampl;
			avg_iter++;
		} else {

			while(dev_iter != avg_iter){
				if (dev_iter->time <= time + PRUNING_TIME_WINDOW
					&& dev_iter != unpruned_peaks.cend()) {

					bin = freq_to_bin(dev_iter->freq);
					if(den[bin]){
						dev[bin] += pow(dev_iter->ampl - num[bin]/den[bin],
2);
					}
					else{
						dev[bin] = den[bin];
					}
				}
				dev_iter++;
			}
			for (int i = 1; i <= NBINS; i++)
			{
				if(den[i]){
					dev[i] = sqrt(dev[i]/den[i]);
				}
				//std::cout << dev[i] << " ";
			}
			//std::cout << std::endl;
			while (add_iter != avg_iter) {
				bin = freq_to_bin(add_iter->freq);
```

```
                                if (den[bin] && add_iter->ampl > STD_DEV_COEF*dev[bin] +
num[bin]/den[bin]  ) {
                                        pruned_peaks.push_back({add_iter->freq,
add_iter->time});
                                        bin_counts[freq_to_bin(add_iter->freq)]++;
                                } else {
                                        bin_prune_counts[freq_to_bin(add_iter->freq)]++;
                                }
                                add_iter++;
                        }
                        memset(num, 0, sizeof(num));
                        memset(den, 0, sizeof(den));
                        time += PRUNING_TIME_WINDOW;
                }
        }
        for (int i = 1; i <= NBINS; i++) {
                std::cout << "bin " << i << ": " << bin_counts[i] << "|  pruned: " <<
bin_prune_counts[i] << std::endl;
        }
        return pruned_peaks;
}



std::list<peak> generate_constellation_map(std::vector<std::vector<float>> fft, int nfft)
{
        std::list<peak_raw> unpruned_map;
        unpruned_map = get_raw_peaks(fft, nfft);
        return prune_in_time(unpruned_map);
}

std::list<peak> read_constellation(std::string filename){

        std::ifstream fin;
        std::list<peak> constellation;
        uint32_t peak_32;
        struct peak temp;
        std::streampos size;
        char * memblock;
        int i;

        fin.open(filename+"_48.magpeak", std::ios::binary | std::ios::in
                        | std::ios::ate);
```

```
        i = 0;
        if (fin.is_open())
        {
            size = fin.tellg();
            memblock = new char [size];

            fin.seekg (0, std::ios::beg);
            fin.read (memblock, size);
            fin.close();

            while(i < size)
            {

                    peak_32 = *(uint32_t *)(memblock+i);
                    temp.time = peak_32;
                    temp.freq = peak_32 >> 16;
                    constellation.push_back(temp);
                    /* MUST incremet by this amount here*/
                    i += sizeof(peak_32);
            }


            delete[] memblock;
        }

        return constellation;

}

void write_constellation(std::list<peak> pruned, std::string filename){

        std::ofstream fout;
        uint32_t peak_32;
        struct peak temp;

        fout.open(filename+"peak", std::ios::binary | std::ios::out);
        for(std::list<peak>::iterator it = pruned.begin();
                        it != pruned.end(); it++){

                temp = *it;

                peak_32 = temp.freq;
```

```
                peak_32 = peak_32 << 16;
                peak_32 |= temp.time;

                fout.write((char *)&peak_32,sizeof(peak_32));
        }

        fout.close();
}
```

Hardware

FFT_accelerator.sv

```
/*
Columbia University
CSEE 4840 Design Project
By: Jose Rubianes(jer2202) & Tomin Perea-Chamblee(tep2116) & Eitan Kaplan(ek2928)

This file contains the top level module for our FFT accelerator.
Takes in audio samples from codec, and coninuously computes the FFT.

Design parameters and functional behaviour can be adjusted in global_variables.sv
*/


`include "global_variables.sv"
`include "./AudioCodecDrivers/audio_driver.sv"
`include "SfftPipeline_SingleStage.sv"


module FFT_Accelerator(
                input logic clk,
                input logic reset,

                input logic [3:0]       KEY, // Pushbuttons; KEY[0] is rightmost

                // 7-segment LED displays; HEX0 is rightmost
                output logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5,

                //Audio pin assignments
                output logic FPGA_I2C_SCLK,
                inout FPGA_I2C_SDAT,
                output logic AUD_XCK,
```

```
            input logic AUD_DACLRCK,
            input logic AUD_ADCLRCK,
            input logic AUD_BCLK,
            input logic AUD_ADCDAT,
            output logic AUD_DACDAT,

            //Driver IO ports
            input logic [7:0] writedata,
            input logic write,
            input chipselect,
            input logic [15:0] address,
            output logic [7:0] readdata
            );


//Instantiate audio controller
reg [23:0] dac_left_in;
reg [23:0] dac_right_in;

wire [23:0] adc_left_out;
wire [23:0] adc_right_out;

wire advance;

reg [23:0] adc_out_buffer = 0;

reg [24:0] counter = 0;  //downsample adance signal

audio_driver aDriver(
        .CLOCK_50(clk),
        .reset(reset),
        .dac_left(dac_left_in),
        .dac_right(dac_right_in),
        .adc_left(adc_left_out),
        .adc_right(adc_right_out),
        .advance(advance),
        .FPGA_I2C_SCLK(FPGA_I2C_SCLK),
        .FPGA_I2C_SDAT(FPGA_I2C_SDAT),
        .AUD_XCK(AUD_XCK),
        .AUD_DACLRCK(AUD_DACLRCK),
        .AUD_ADCLRCK(AUD_ADCLRCK),
        .AUD_BCLK(AUD_BCLK),
        .AUD_ADCDAT(AUD_ADCDAT),
```

```
		.AUD_DACDAT(AUD_DACDAT)
		);

	//Convert stereo input to mono
	reg [23:0] audioInMono;
	always @ (*) begin
		audioInMono = (adc_right_out/2) + (adc_left_out/2);
	end

	//Determine when the driver is in the middle of pulling a sample
	logic [7:0] driverReading = 8'd0;
	always @(posedge clk) begin
		if (chipselect && write) begin
			driverReading <= writedata;
		end
	end

	wire sampleBeingTaken;
	assign sampleBeingTaken = driverReading[0];

	//Instantiate SFFT pipeline
	wire [`SFFT_OUTPUT_WIDTH -1:0] SFFT_Out ;
	wire SfftOutputValid;
	wire outputReadError;
	wire [`nFFT -1:0] output_address;
	assign output_address = address[`nFFT +1:2];
	wire [`SFFT_OUTPUT_WIDTH -1:0] Output_Why;

	SFFT_Pipeline sfft(
		.clk(clk),
		.reset(reset),

		.SampleAmplitudeIn(audioInMono),
		.advanceSignal(advance),

		//Output BRAM IO
		.OutputBeingRead(sampleBeingTaken),
		.outputReadError(outputReadError),
		.output_address(output_address),
		.SFFT_OutReal(SFFT_Out),  //This port in unused, but things break if you delete
it. Don't ask me why
		.OutputValid(SfftOutputValid),
		.Output_Why(Output_Why)
```

```
		);

	//Sample counter
	reg [`TIME_COUNTER_WIDTH -1:0] timeCounter = 0;
	always @(posedge SfftOutputValid) begin
		timeCounter <= timeCounter + 1;
	end


	//Instantiate hex decoders
	hex7seg h5( .a(adc_out_buffer[23:20]),.y(HEX5) ), // left digit
		h4( .a(adc_out_buffer[19:16]),.y(HEX4) ),
		h3( .a(adc_out_buffer[15:12]),.y(HEX3) ),
		h2( .a(adc_out_buffer[11:8]),.y(HEX2) ),
		h1( .a(adc_out_buffer[7:4]),.y(HEX1) ),
		h0( .a(adc_out_buffer[3:0]),.y(HEX0) );


	//Map timer(Sample) counter output
	parameter readOutSize = 2048;
	reg [7:0] timer_buffer [3:0];
	integer i;
	always @(posedge clk) begin
		if (sampleBeingTaken == 0) begin
			//NOTE: Each 32bit word is written in reverse byte order, due to
endian-ness of software. Avoids need for ntohl conversion

			//Counter -> address 0-3. Assuming 32 bit counter
			timer_buffer[3] <= timeCounter[31:24];
			timer_buffer[2] <= timeCounter[23:16];
			timer_buffer[1] <= timeCounter[15:8];
			timer_buffer[0] <= timeCounter[7:0];
		end
	end

	//Read handling
	always @(*) begin
		if (address < `NFFT*2) begin
			//Convert input address into subset of SFFT_Out
			//NOTE: Each 32bit word is written in reverse byte order, due to
endian-ness of software. Avoids need for ntohl conversion
			if (address % 4 == 0) begin
				readdata = Output_Why[7:0];
```

```
                end
                else if (address % 4 == 1) begin
                        readdata = Output_Why[15:8];
                end
                else if (address % 4 == 2) begin
                        readdata = Output_Why[23:16];
                end
                else if (address % 4 == 3) begin
                        readdata = Output_Why[31:24];
                end
        end
        else if (address[15:2] == `NFFT/2) begin
                //Send the timer counter
                readdata = timer_buffer[address[1:0]];
        end
        else begin
                //Send the valid byte
                readdata = {7'b0, ~outputReadError};
        end
    end


    //Sample inputs/Audio passthrough
    always @(posedge advance) begin
        counter <= counter + 1;
        dac_left_in <= adc_left_out;
        dac_right_in <= adc_right_out;
    end

    always @(posedge counter[12]) begin
        adc_out_buffer <= adc_left_out;
    end

endmodule


//Seven segment hex decoder
module hex7seg(input logic [3:0] a,
        output logic [6:0] y);

    always @ (a) begin
        case(a)
                0 : y = 7'b100_0000;
```

```
                        1 : y = 7'b111_1001;
                        2 : y = 7'b010_0100;
                        3 : y = 7'b011_0000;
                        4 : y = 7'b001_1001;
                        5 : y = 7'b001_0010;
                        6 : y = 7'b000_0010;
                        7 : y = 7'b111_1000;
                        8 : y = 7'b000_0000;
                        9 : y = 7'b001_1000;
                        10 : y = 7'b000_1000;  //a
                        11 : y = 7'b000_0011;  //b
                        12 : y = 7'b100_0110;  //c
                        13 : y = 7'b010_0001;  //d
                        14 : y = 7'b000_0110;  //e
                        15 : y = 7'b000_1110;  //f
                        default: y = 7'b011_1111;
                endcase
        end
endmodule
```

fft_accelerator.tcl

```
# TCL File Generated by Component Editor 18.1
# Thu May 02 19:41:15 EDT 2019
# DO NOT MODIFY


#
# fft_accelerator "FFT Accelorator" v1.0
#  2019.05.02.19:41:15
#
#

#
# request TCL package from ACDS 16.1
#
package require -exact qsys 16.1


#
# module fft_accelerator
#
```

```
set_module_property DESCRIPTION ""
set_module_property NAME fft_accelerator
set_module_property VERSION 1.0
set_module_property INTERNAL false
set_module_property OPAQUE_ADDRESS_MAP true
set_module_property AUTHOR ""
set_module_property DISPLAY_NAME "FFT Accelorator"
set_module_property INSTANTIATE_IN_SYSTEM_MODULE true
set_module_property EDITABLE true
set_module_property REPORT_TO_TALKBACK false
set_module_property ALLOW_GREYBOX_GENERATION false
set_module_property REPORT_HIERARCHY false


#
# file sets
#
add_fileset QUARTUS_SYNTH QUARTUS_SYNTH "" ""
set_fileset_property QUARTUS_SYNTH TOP_LEVEL FFT_Accelerator
set_fileset_property QUARTUS_SYNTH ENABLE_RELATIVE_INCLUDE_PATHS false
set_fileset_property QUARTUS_SYNTH ENABLE_FILE_OVERWRITE_MODE false
add_fileset_file FFT_Accelerator.sv SYSTEM_VERILOG PATH FFT_Accelerator.sv
TOP_LEVEL_FILE


#
# parameters
#


#
# module assignments
#
set_module_assignment embeddedsw.dts.group fft
set_module_assignment embeddedsw.dts.name fft_accelerator
set_module_assignment embeddedsw.dts.vendor csee4840


#
# display items
#
```

```
#
# connection point clock
#
add_interface clock clock end
set_interface_property clock clockRate 0
set_interface_property clock ENABLED true
set_interface_property clock EXPORT_OF ""
set_interface_property clock PORT_NAME_MAP ""
set_interface_property clock CMSIS_SVD_VARIABLES ""
set_interface_property clock SVD_ADDRESS_GROUP ""

add_interface_port clock clk clk Input 1


#
# connection point reset
#
add_interface reset reset end
set_interface_property reset associatedClock clock
set_interface_property reset synchronousEdges DEASSERT
set_interface_property reset ENABLED true
set_interface_property reset EXPORT_OF ""
set_interface_property reset PORT_NAME_MAP ""
set_interface_property reset CMSIS_SVD_VARIABLES ""
set_interface_property reset SVD_ADDRESS_GROUP ""

add_interface_port reset reset reset Input 1


#
# connection point avalon_slave_0
#
add_interface avalon_slave_0 avalon end
set_interface_property avalon_slave_0 addressUnits WORDS
set_interface_property avalon_slave_0 associatedClock clock
set_interface_property avalon_slave_0 associatedReset reset
set_interface_property avalon_slave_0 bitsPerSymbol 8
set_interface_property avalon_slave_0 burstOnBurstBoundariesOnly false
set_interface_property avalon_slave_0 burstcountUnits WORDS
set_interface_property avalon_slave_0 explicitAddressSpan 0
set_interface_property avalon_slave_0 holdTime 0
set_interface_property avalon_slave_0 linewrapBursts false
set_interface_property avalon_slave_0 maximumPendingReadTransactions 0
```

```
set_interface_property avalon_slave_0 maximumPendingWriteTransactions 0
set_interface_property avalon_slave_0 readLatency 0
set_interface_property avalon_slave_0 readWaitTime 1
set_interface_property avalon_slave_0 setupTime 0
set_interface_property avalon_slave_0 timingUnits Cycles
set_interface_property avalon_slave_0 writeWaitTime 0
set_interface_property avalon_slave_0 ENABLED true
set_interface_property avalon_slave_0 EXPORT_OF ""
set_interface_property avalon_slave_0 PORT_NAME_MAP ""
set_interface_property avalon_slave_0 CMSIS_SVD_VARIABLES ""
set_interface_property avalon_slave_0 SVD_ADDRESS_GROUP ""

add_interface_port avalon_slave_0 writedata writedata Input 8
add_interface_port avalon_slave_0 write write Input 1
add_interface_port avalon_slave_0 chipselect chipselect Input 1
add_interface_port avalon_slave_0 address address Input 16
add_interface_port avalon_slave_0 readdata readdata Output 8
set_interface_assignment avalon_slave_0 embeddedsw.configuration.isFlash 0
set_interface_assignment avalon_slave_0 embeddedsw.configuration.isMemoryDevice 0
set_interface_assignment avalon_slave_0 embeddedsw.configuration.isNonVolatileStorage 0
set_interface_assignment avalon_slave_0 embeddedsw.configuration.isPrintableDevice 0



#
# connection point FFT
#
add_interface FFT conduit end
set_interface_property FFT associatedClock clock
set_interface_property FFT associatedReset ""
set_interface_property FFT ENABLED true
set_interface_property FFT EXPORT_OF ""
set_interface_property FFT PORT_NAME_MAP ""
set_interface_property FFT CMSIS_SVD_VARIABLES ""
set_interface_property FFT SVD_ADDRESS_GROUP ""

add_interface_port FFT KEY keyinput Input 4
add_interface_port FFT HEX1 display1 Output 7
add_interface_port FFT HEX0 display0 Output 7
add_interface_port FFT HEX3 display3 Output 7
add_interface_port FFT HEX4 display4 Output 7
add_interface_port FFT HEX5 display5 Output 7
add_interface_port FFT FPGA_I2C_SCLK i2c_clk Output 1
add_interface_port FFT FPGA_I2C_SDAT i2c_data Bidir 1
```

```
add_interface_port FFT AUD_XCK aud_xck Output 1
add_interface_port FFT AUD_DACLRCK aud_daclrck Input 1
add_interface_port FFT AUD_ADCLRCK aud_adclrck Input 1
add_interface_port FFT AUD_BCLK aud_bclk Input 1
add_interface_port FFT AUD_ADCDAT aud_adcdat Input 1
add_interface_port FFT AUD_DACDAT aud_dacdat Output 1
add_interface_port FFT HEX2 display2 Output 7
```

bram.sv

```
/*
 * Construct BRAM modules for pipeline memory buffers
 */

`include "global_variables.sv"

/*
 * Dual read/write port ram
 */
module myNewerBram (
	input logic clk,
	input logic [`nFFT -1:0] aa, ab,
	input logic [(`SFFT_OUTPUT_WIDTH*2) -1:0] da, db,
	input logic wa, wb,
	output logic [(`SFFT_OUTPUT_WIDTH*2) -1:0] qa, qb);

	logic [(`SFFT_OUTPUT_WIDTH*2) -1:0] mem [`NFFT -1:0];

	always_ff @(posedge clk) begin
		if (wa) begin
			mem[aa] <= da;
			qa <= da;
		end
		else begin
			qa <= mem[aa];
		end
	end

	always_ff @(posedge clk) begin
		if (wb) begin
			mem[ab] <= db;
			qb <= db;
		end
```

```
            else begin
                qb <= mem[ab];
            end
        end

endmodule
```

global_variables.sv

```
//`define RUNNING_SIMULATION  //define this to change ROM file locations to absolute paths fo vsim


// FFT Macros
`define NFFT 512 // if change this, change FREQ_WIDTH. Must be power of 2
`define nFFT 9  //log2(NFFT)

`define FREQS (`NFFT / 2)
`define FREQ_WIDTH 9 // if change NFFT, change this

`define FINAL_AMPL_WIDTH 32 // Must be less than or equal to INPUT_AMPL_WIDTH
`define INPUT_AMPL_WIDTH 32
`define TIME_COUNTER_WIDTH 32

`define PEAKS 6 // Changing this requires many changes in code

`define SFFT_INPUT_WIDTH 24
`define SFFT_OUTPUT_WIDTH `INPUT_AMPL_WIDTH
`define SFFT_FIXEDPOINT_INPUTSCALING  //define this macro if you want to scale adc inputs to match FixedPoint magnitudes. Increases accuracy, but could lead to overflow
`define SFFT_FIXED_POINT_ACCURACY 7
`define SFFT_STAGECOUNTER_WIDTH 5  //>= log2(nFFT)

//`define SFFT_DOWNSAMPLE_PRE  //define this macro if you want to downsample the incoming audio BEFORE the FFT calculation
`define SFFT_DOWNSAMPLE_PRE_FACTOR 2
`define nDOWNSAMPLE_PRE 1  // >= log2(SFFT_DOWNSAMPLE_PRE_FACTOR)

`define SFFT_DOWNSAMPLE_POST  //define this macro if you want to downsample the outgoing FFT calculation (will skip calculations). Will determine window-shift ammount
`define SFFT_DOWNSAMPLE_POST_FACTOR 256
`define nDOWNSAMPLE_POST 8  // >= log2(SFFT_DOWNSAMPLE_POST_FACTOR)
```

```
// Audio Codec Macros
`define AUDIO_IN_GAIN 9'h010
`define AUDIO_OUT_GAIN 9'h061

`define SAMPLE_RATE_CNTRL 9'd0  //No particularly helpful, but see page 44 of datasheet of more info: https://statics.cirrus.com/pubs/proDatasheet/WM8731_v4.9.pdf


//NOTE: Below bin values are no longer in use, since peaks are no longer found in hardware

// BINS NFFT=256
`define BIN_1 1
`define BIN_2 4
`define BIN_3 13
`define BIN_4 24
`define BIN_5 37
`define BIN_6 116
```

SfftPipeline_SingleStage.sv

```
/*
 * This module takes in samples of amplitudes, and outputs the N point FFT
 */

`include "global_variables.sv"
`include "bram.sv"


 /*
  * Top level SFFT pipeline module.
  *
  * Samples the input signal <SampleAmplitudeIn> at the rising edge of <advanceSignal>. Begins processing the FFT immediately.
  * Only outputs the real components of the FFT result. Will raise <OutputValid> high for 1 cycle when the output is finished.
  *
  * Output port provides access to an internal BRAM module where the results are stored. The reader must provide the address of the result they wish to read.
  *
```

```
 * Max sampling frequency ~=
(CLK_FREQ*DOWNSAMPLE_PRE_FACTOR*DOWNSAMPLE_POST_FACTOR) /
(log2(NFFT)*NFFT+2). Output indeterminate if exceeded.
 */
module SFFT_Pipeline(
        input clk,
        input reset,

        //Inputs
        input [`SFFT_INPUT_WIDTH -1:0] SampleAmplitudeIn,
        input advanceSignal,

        //Output BRAM IO
        input logic OutputBeingRead,
        output logic outputReadError,
        input logic [`nFFT -1:0] output_address,
        output reg [`SFFT_OUTPUT_WIDTH -1:0] SFFT_OutReal,
        output logic OutputValid,
        output reg [`SFFT_OUTPUT_WIDTH -1:0] Output_Why
        );


        //_____________________________
        //
        // ROM for static parameters
        //_____________________________

        reg [`nFFT -1:0] shuffledInputIndexes [`NFFT -1:0];

        reg [`nFFT -1:0] kValues [`nFFT*(`NFFT / 2) -1:0];

        reg [`nFFT -1:0] aIndexes [`nFFT*(`NFFT / 2) -1:0];
        reg [`nFFT -1:0] bIndexes [`nFFT*(`NFFT / 2) -1:0];

        reg [`SFFT_FIXED_POINT_ACCURACY:0] realCoefficents [(`NFFT / 2) -1:0];
        reg [`SFFT_FIXED_POINT_ACCURACY:0] imagCoefficents [(`NFFT / 2) -1:0];

        //Load values into ROM from generated text files
        initial begin
`ifdef RUNNING_SIMULATION
                //NOTE: These filepaths must be changed to their absolute local paths if
simulating with Vsim. Otherwise they should be relative to Hardware directory
```

```
		//NOTE: If simulating with Vsim, make sure to run the Matlab script GenerateRomFiles.m if you change any global variables

$readmemh("/user3/fall16/jer2201/notShazam/Hardware/GeneratedParameters/InputShuffledIndexes.txt", shuffledInputIndexes, 0);

$readmemh("/user3/fall16/jer2201/notShazam/Hardware/GeneratedParameters/Ks.txt", kValues, 0);

$readmemh("/user3/fall16/jer2201/notShazam/Hardware/GeneratedParameters/aIndexes.txt", aIndexes, 0);

$readmemh("/user3/fall16/jer2201/notShazam/Hardware/GeneratedParameters/bIndexes.txt", bIndexes, 0);

$readmemh("/user3/fall16/jer2201/notShazam/Hardware/GeneratedParameters/realCoefficients.txt", realCoefficents, 0);

$readmemh("/user3/fall16/jer2201/notShazam/Hardware/GeneratedParameters/imaginaryCoefficients.txt", imagCoefficents, 0);
`else
		$readmemh("GeneratedParameters/InputShuffledIndexes.txt", shuffledInputIndexes, 0);

		$readmemh("GeneratedParameters/Ks.txt", kValues, 0);

		$readmemh("GeneratedParameters/aIndexes.txt", aIndexes, 0);
		$readmemh("GeneratedParameters/bIndexes.txt", bIndexes, 0);

		$readmemh("GeneratedParameters/realCoefficients.txt", realCoefficents, 0);
		$readmemh("GeneratedParameters/imaginaryCoefficients.txt", imagCoefficents, 0);
`endif
	end

	//Map 2D ROM arrays into 3D
	wire [`nFFT -1:0] kValues_Mapped [`nFFT -1:0] [(`NFFT / 2) -1:0];
	wire [`nFFT -1:0] aIndexes_Mapped [`nFFT -1:0] [(`NFFT / 2) -1:0];
	wire [`nFFT -1:0] bIndexes_Mapped [`nFFT -1:0] [(`NFFT / 2) -1:0];
```

```
	genvar stage;
	generate
		for (stage=0; stage<`nFFT; stage=stage+1) begin : ROM_mapping
			assign kValues_Mapped[stage] = kValues[(stage+1)*(`NFFT / 2)-1 :
stage*(`NFFT / 2)];
			assign aIndexes_Mapped[stage] = aIndexes[(stage+1)*(`NFFT / 2)-1 :
stage*(`NFFT / 2)];
			assign bIndexes_Mapped[stage] = bIndexes[(stage+1)*(`NFFT / 2)-1 :
stage*(`NFFT / 2)];
		end
	endgenerate

	//_____________________________
	//
	// Input Sampling
	//_____________________________

	wire [`SFFT_INPUT_WIDTH -1:0] SampleAmplitudeIn_Processed;
	reg advanceSignal_Intermediate;
	reg advanceSignal_Processed;

	/*
	 * Implement downsampling if specified
	 */

	//Pre downsampling
`ifdef SFFT_DOWNSAMPLE_PRE
	//Shift buffer to hold SFFT_DOWNSAMPLE_PRE_FACTOR most recent raw samples
	reg [`SFFT_INPUT_WIDTH -1:0] WindowBuffers
[`SFFT_DOWNSAMPLE_PRE_FACTOR -1:0] = '{default:0};
	integer m;
	always @ (posedge advanceSignal) begin
		for (m=0; m<`SFFT_DOWNSAMPLE_PRE_FACTOR; m=m+1) begin
			if (m==0) begin
				//load most recent raw sample into buffer 0
				WindowBuffers[m] <= SampleAmplitudeIn;
			end
			else begin
				//Shift buffer contents down by 1
				WindowBuffers[m] <= WindowBuffers[m-1];
			end
		end
```

```
        end

        //Take moving average of window. Acts as lowpass filter
        logic [`SFFT_INPUT_WIDTH + `nDOWNSAMPLE_PRE -1:0] movingSum = 0;
        always @(posedge advanceSignal) begin
                movingSum = movingSum + SampleAmplitudeIn -
WindowBuffers[`SFFT_DOWNSAMPLE_PRE_FACTOR -1];
        end

        logic [`SFFT_INPUT_WIDTH + `nDOWNSAMPLE_PRE -1:0] movingAverage;
        always @(*) begin
                movingAverage = movingSum/`SFFT_DOWNSAMPLE_PRE_FACTOR;
        end

        assign SampleAmplitudeIn_Processed = movingAverage[`SFFT_INPUT_WIDTH -1:0];
//right shift by nDOWNSAMPLE_PRE to divide sum into average

        //Counter for input downsampling
        reg [`nDOWNSAMPLE_PRE -1:0] downsamplePRECounter = 0;
        always @ (posedge advanceSignal) begin
                if (downsamplePRECounter == `SFFT_DOWNSAMPLE_PRE_FACTOR -1)
begin
                        downsamplePRECounter <= 0;
                end
                else begin
                        downsamplePRECounter <= downsamplePRECounter + 1;
                end
        end

        always @ (posedge clk) begin
                advanceSignal_Intermediate <= (downsamplePRECounter ==
`SFFT_DOWNSAMPLE_PRE_FACTOR -1) && advanceSignal;
        end
`else
        assign SampleAmplitudeIn_Processed = SampleAmplitudeIn;

        always @(*) begin
                advanceSignal_Intermediate = advanceSignal;
        end
`endif

        //Post downsampling
`ifdef SFFT_DOWNSAMPLE_POST
```

```
	reg [`nDOWNSAMPLE_POST -1:0] downsamplePOSTCounter = 0;
	always @ (posedge advanceSignal_Intermediate) begin
		if (downsamplePOSTCounter == `SFFT_DOWNSAMPLE_POST_FACTOR -1)
begin
			downsamplePOSTCounter <= 0;
		end
		else begin
			downsamplePOSTCounter <= downsamplePOSTCounter + 1;
		end
	end

	always @ (posedge clk) begin
		advanceSignal_Processed <= (downsamplePOSTCounter ==
`SFFT_DOWNSAMPLE_POST_FACTOR -1) && advanceSignal_Intermediate;
	end
`else
	always @(*) begin
		advanceSignal_Processed = advanceSignal_Intermediate;
	end
`endif


	//Shift buffer to hold N most recent samples
	reg [`SFFT_INPUT_WIDTH -1:0] SampleBuffers [`NFFT -1:0] = '{default:0};

	integer i;
	always @ (posedge advanceSignal_Processed) begin
		for (i=0; i<`NFFT; i=i+1) begin
			if (i==0) begin
				//load most recent sample into buffer 0
				SampleBuffers[i] <= SampleAmplitudeIn_Processed;
			end
			else begin
				//Shift buffer contents down by 1
				SampleBuffers[i] <= SampleBuffers[i-1];
			end
		end
	end


	//Shuffle input buffer
	logic [`SFFT_OUTPUT_WIDTH -1:0] shuffledSamples [`NFFT -1:0];
```

```
	integer j;

`ifdef SFFT_FIXEDPOINT_INPUTSCALING
	parameter extensionBits = `SFFT_OUTPUT_WIDTH -
`SFFT_FIXED_POINT_ACCURACY - `SFFT_INPUT_WIDTH - 1;
	always @ (*) begin
		for (j=0; j<`NFFT; j=j+1) begin
			shuffledSamples[j] =
{{extensionBits{SampleBuffers[shuffledInputIndexes[j]][`SFFT_INPUT_WIDTH -1]}},
SampleBuffers[shuffledInputIndexes[j]] << `SFFT_FIXED_POINT_ACCURACY};  //Left shift
input by fixed-point accuracy, and sign extend to match output width
		end
	end

`else
	parameter extensionBits = `SFFT_OUTPUT_WIDTH - `SFFT_INPUT_WIDTH - 1;
	always @ (*) begin
		for (j=0; j<`NFFT; j=j+1) begin
			shuffledSamples[j] =
{{extensionBits{SampleBuffers[shuffledInputIndexes[j]][`SFFT_INPUT_WIDTH -1]}},
SampleBuffers[shuffledInputIndexes[j]]};  //Sign extend to match output width
		end
	end
`endif

	//Notify pipeline of new input
	reg newSampleReady;
	wire inputReceived;
	always @ (negedge clk) begin  //negedge to avoid race condition with
advanceSignal_Processed
		if (reset) begin
			newSampleReady <= 0;
		end

		else if ((inputReceived==1) && (newSampleReady==1)) begin
			newSampleReady <= 0;
		end

		else if ((advanceSignal_Processed==1) && (newSampleReady==0) &&
(inputReceived==0)) begin
			newSampleReady <= 1;
		end
	end
```

```
//_____________________________________
//
// Generate pipeline structure
//_____________________________________

/*
 * Copier instance
 */

//Input bus
wire [`SFFT_OUTPUT_WIDTH -1:0] StageInImag [`NFFT -1:0];
assign StageInImag = '{default:0};

//Output bus
wire [`nFFT -1:0] ramCopier_address_A;
wire ramCopier_writeEnable_A;
wire [`nFFT -1:0] ramCopier_address_B;
wire ramCopier_writeEnable_B;

wire [`SFFT_OUTPUT_WIDTH -1:0] ramCopier_dataInReal_A;
wire [`SFFT_OUTPUT_WIDTH -1:0] ramCopier_dataInImag_A;

wire [`SFFT_OUTPUT_WIDTH -1:0] ramCopier_dataInReal_B;
wire [`SFFT_OUTPUT_WIDTH -1:0] ramCopier_dataInImag_B;

//State control bus
wire copying;
assign inputReceived = copying;
wire copier_outputReady;
wire [1:0] copier_access_pointer;

copyToRamStage copier(
        .clk(clk),
        .reset(reset),

        .StageInReal(shuffledSamples),
        .StageInImag(StageInImag),
        .copySignal(newSampleReady),

        .address_A(ramCopier_address_A),
        .writeEnable_A(ramCopier_writeEnable_A),
```

```
        .address_B(ramCopier_address_B),
        .writeEnable_B(ramCopier_writeEnable_B),
        .dataInReal_A(ramCopier_dataInReal_A),
        .dataInImag_A(ramCopier_dataInImag_A),
        .dataInReal_B(ramCopier_dataInReal_B),
        .dataInImag_B(ramCopier_dataInImag_B),

        .copying(copying),
        .outputReady(copier_outputReady),
        .ram_access_pointer(copier_access_pointer)
        );

/*
 * Stage instance
 */

//Input bus
logic [`SFFT_OUTPUT_WIDTH -1:0] ramStage_dataOutReal_A;
logic [`SFFT_OUTPUT_WIDTH -1:0] ramStage_dataOutImag_A;

logic [`SFFT_OUTPUT_WIDTH -1:0] ramStage_dataOutReal_B;
logic [`SFFT_OUTPUT_WIDTH -1:0] ramStage_dataOutImag_B;

//Output bus
wire [`nFFT -1:0] ramStage_address_A;
wire ramStage_writeEnable_A;
wire [`nFFT -1:0] ramStage_address_B;
wire ramStage_writeEnable_B;

wire [`SFFT_OUTPUT_WIDTH -1:0] ramStage_dataInReal_A;
wire [`SFFT_OUTPUT_WIDTH -1:0] ramStage_dataInImag_A;

wire [`SFFT_OUTPUT_WIDTH -1:0] ramStage_dataInReal_B;
wire [`SFFT_OUTPUT_WIDTH -1:0] ramStage_dataInImag_B;

wire [1:0] pipelineStage_access_pointer;

//State control bus
wire idle;
wire [`SFFT_STAGECOUNTER_WIDTH -1:0] virtualStageCounter;

//ROM inputs
reg [`nFFT -1:0] kValues_In [(`NFFT / 2) -1:0];
```

```
reg [`nFFT -1:0] aIndexes_In [(`NFFT / 2) -1:0];
reg [`nFFT -1:0] bIndexes_In [(`NFFT / 2) -1:0];

//MUX for ROM inputs
always @(*) begin
        kValues_In = kValues_Mapped[virtualStageCounter];
        aIndexes_In = aIndexes_Mapped[virtualStageCounter];
        bIndexes_In = bIndexes_Mapped[virtualStageCounter];
end

pipelineStage Stage(
        .clk(clk),
        .reset(reset),

        .realCoefficents(realCoefficents),
        .imagCoefficents(imagCoefficents),
        .kValues(kValues_In),
        .aIndexes(aIndexes_In),
        .bIndexes(bIndexes_In),

        .ram_address_A(ramStage_address_A),
        .ram_writeEnable_A(ramStage_writeEnable_A),
        .ram_dataInReal_A(ramStage_dataInReal_A),
        .ram_dataInImag_A(ramStage_dataInImag_A),
        .ram_dataOutReal_A(ramStage_dataOutReal_A),
        .ram_dataOutImag_A(ramStage_dataOutImag_A),
        .ram_address_B(ramStage_address_B),
        .ram_writeEnable_B(ramStage_writeEnable_B),
        .ram_dataInReal_B(ramStage_dataInReal_B),
        .ram_dataInImag_B(ramStage_dataInImag_B),
        .ram_dataOutReal_B(ramStage_dataOutReal_B),
        .ram_dataOutImag_B(ramStage_dataOutImag_B),
        .ram_access_pointer(pipelineStage_access_pointer),

        .idle(idle),
        .virtualStageCounter(virtualStageCounter),
        .inputReady(copier_outputReady),
        .outputReady(OutputValid)
        );


/*
 * Output access handling
```

```
	 */

	logic [1:0] nextOutput_access_pointer = 3;  //Points to the most recent output of the pipeline

	always @(posedge OutputValid) begin
		if (reset) begin
			nextOutput_access_pointer <= 3;
		end

		else begin
			nextOutput_access_pointer <= nextOutput_access_pointer + 1;
		end
	end

	logic [1:0] output_access_pointer;  //Points to the buffer we're currently reading from the software

	always @(posedge clk) begin
		if (OutputBeingRead == 0) begin
			output_access_pointer <= nextOutput_access_pointer; //Only update output_access_pointer when we are not reading from software
			outputReadError <= 0;
		end
		else begin
			if (output_access_pointer == copier_access_pointer) begin
				//The copy stage has caught up with where the driver is reading from. Set error flag high
				outputReadError <= 1;
			end
		end
	end


	//_____________________________________
	//
	// Generate BRAM buffers
	//_____________________________________

	/*
	 * Buffer 0
	 */
	logic ramBuffer0_readClock;
```

```
        //Input bus
        logic [`nFFT -1:0] ramBuffer0_address_A;
        logic ramBuffer0_writeEnable_A;
        logic [`nFFT -1:0] ramBuffer0_address_B;
        logic ramBuffer0_writeEnable_B;

        logic [`SFFT_OUTPUT_WIDTH -1:0] ramBuffer0_dataInReal_A;
        logic [`SFFT_OUTPUT_WIDTH -1:0] ramBuffer0_dataInImag_A;

        logic [`SFFT_OUTPUT_WIDTH -1:0] ramBuffer0_dataInReal_B;
        logic [`SFFT_OUTPUT_WIDTH -1:0] ramBuffer0_dataInImag_B;

        //Output bus
        wire [`SFFT_OUTPUT_WIDTH -1:0] ramBuffer0_dataOutReal_A;
        wire [`SFFT_OUTPUT_WIDTH -1:0] ramBuffer0_dataOutImag_A;

        wire [`SFFT_OUTPUT_WIDTH -1:0] ramBuffer0_dataOutReal_B;
        wire [`SFFT_OUTPUT_WIDTH -1:0] ramBuffer0_dataOutImag_B;

        //Concatenate dataIn bus
        wire [(2*`SFFT_OUTPUT_WIDTH) -1:0] ramBuffer0_dataInConcatenated_A;
        assign ramBuffer0_dataInConcatenated_A = {ramBuffer0_dataInReal_A,
ramBuffer0_dataInImag_A};

        wire [(2*`SFFT_OUTPUT_WIDTH) -1:0] ramBuffer0_dataInConcatenated_B;
        assign ramBuffer0_dataInConcatenated_B = {ramBuffer0_dataInReal_B,
ramBuffer0_dataInImag_B};

        //Concatenate dataOut bus
        wire [(2*`SFFT_OUTPUT_WIDTH) -1:0] ramBuffer0_dataOutConcatenated_A;
        assign ramBuffer0_dataOutImag_A =
ramBuffer0_dataOutConcatenated_A[`SFFT_OUTPUT_WIDTH -1:0];
        assign ramBuffer0_dataOutReal_A =
ramBuffer0_dataOutConcatenated_A[(2*`SFFT_OUTPUT_WIDTH) -1
:`SFFT_OUTPUT_WIDTH ];

        wire [(2*`SFFT_OUTPUT_WIDTH) -1:0] ramBuffer0_dataOutConcatenated_B;
        assign ramBuffer0_dataOutImag_B =
ramBuffer0_dataOutConcatenated_B[`SFFT_OUTPUT_WIDTH -1:0];
        assign ramBuffer0_dataOutReal_B =
ramBuffer0_dataOutConcatenated_B[(2*`SFFT_OUTPUT_WIDTH) -1
:`SFFT_OUTPUT_WIDTH ];
```

```
//Instantiate Buffer 0
myNewerBram BRAM_0(
	.clk(clk),
	.aa(ramBuffer0_address_A),
	.ab(ramBuffer0_address_B),
	.da(ramBuffer0_dataInConcatenated_A),
	.db(ramBuffer0_dataInConcatenated_B),
	.wa(ramBuffer0_writeEnable_A),
	.wb(ramBuffer0_writeEnable_B),
	.qa(ramBuffer0_dataOutConcatenated_A),
	.qb(ramBuffer0_dataOutConcatenated_B)
	);

//Buffer 0 write access control
always @(*) begin
	if (copier_access_pointer == 0) begin
		//Give access to copier stage
		ramBuffer0_address_A = ramCopier_address_A;
		ramBuffer0_writeEnable_A = ramCopier_writeEnable_A;
		ramBuffer0_dataInReal_A = ramCopier_dataInReal_A;
		ramBuffer0_dataInImag_A = ramCopier_dataInImag_A;

		ramBuffer0_address_B = ramCopier_address_B;
		ramBuffer0_writeEnable_B = ramCopier_writeEnable_B;
		ramBuffer0_dataInReal_B = ramCopier_dataInReal_B;
		ramBuffer0_dataInImag_B = ramCopier_dataInImag_B;

		ramBuffer0_readClock = ~clk;
	end

	else if (pipelineStage_access_pointer == 0) begin
		//Give access to pipeline stage
		ramBuffer0_address_A = ramStage_address_A;
		ramBuffer0_writeEnable_A = ramStage_writeEnable_A;
		ramBuffer0_dataInReal_A = ramStage_dataInReal_A;
		ramBuffer0_dataInImag_A = ramStage_dataInImag_A;

		ramBuffer0_address_B = ramStage_address_B;
		ramBuffer0_writeEnable_B = ramStage_writeEnable_B;
		ramBuffer0_dataInReal_B = ramStage_dataInReal_B;
		ramBuffer0_dataInImag_B = ramStage_dataInImag_B;
```

```
                ramBuffer0_readClock = ~clk;
        end

        else if (output_access_pointer == 0) begin
                //Give access to output port
                ramBuffer0_address_A = output_address;
                ramBuffer0_writeEnable_A = 0;
                ramBuffer0_dataInReal_A = 0;
                ramBuffer0_dataInImag_A = 0;

                ramBuffer0_address_B = 0;
                ramBuffer0_writeEnable_B = 0;
                ramBuffer0_dataInReal_B = 0;
                ramBuffer0_dataInImag_B = 0;

                ramBuffer0_readClock = clk;
        end
end

/*
 * Buffer 1
 */
logic ramBuffer1_readClock;

//Input bus
logic [`nFFT -1:0] ramBuffer1_address_A;
logic ramBuffer1_writeEnable_A;
logic [`nFFT -1:0] ramBuffer1_address_B;
logic ramBuffer1_writeEnable_B;

logic [`SFFT_OUTPUT_WIDTH -1:0] ramBuffer1_dataInReal_A;
logic [`SFFT_OUTPUT_WIDTH -1:0] ramBuffer1_dataInImag_A;

logic [`SFFT_OUTPUT_WIDTH -1:0] ramBuffer1_dataInReal_B;
logic [`SFFT_OUTPUT_WIDTH -1:0] ramBuffer1_dataInImag_B;

//Output bus
wire [`SFFT_OUTPUT_WIDTH -1:0] ramBuffer1_dataOutReal_A;
wire [`SFFT_OUTPUT_WIDTH -1:0] ramBuffer1_dataOutImag_A;

wire [`SFFT_OUTPUT_WIDTH -1:0] ramBuffer1_dataOutReal_B;
wire [`SFFT_OUTPUT_WIDTH -1:0] ramBuffer1_dataOutImag_B;
```

```
	//Concatenate dataIn bus
	wire [(2*`SFFT_OUTPUT_WIDTH) -1:0] ramBuffer1_dataInConcatenated_A;
	assign ramBuffer1_dataInConcatenated_A = {ramBuffer1_dataInReal_A,
ramBuffer1_dataInImag_A};

	wire [(2*`SFFT_OUTPUT_WIDTH) -1:0] ramBuffer1_dataInConcatenated_B;
	assign ramBuffer1_dataInConcatenated_B = {ramBuffer1_dataInReal_B,
ramBuffer1_dataInImag_B};

	//Concatenate dataOut bus
	wire [(2*`SFFT_OUTPUT_WIDTH) -1:0] ramBuffer1_dataOutConcatenated_A;
	assign ramBuffer1_dataOutImag_A =
ramBuffer1_dataOutConcatenated_A[`SFFT_OUTPUT_WIDTH -1:0];
	assign ramBuffer1_dataOutReal_A =
ramBuffer1_dataOutConcatenated_A[(2*`SFFT_OUTPUT_WIDTH) -1
:`SFFT_OUTPUT_WIDTH ];

	wire [(2*`SFFT_OUTPUT_WIDTH) -1:0] ramBuffer1_dataOutConcatenated_B;
	assign ramBuffer1_dataOutImag_B =
ramBuffer1_dataOutConcatenated_B[`SFFT_OUTPUT_WIDTH -1:0];
	assign ramBuffer1_dataOutReal_B =
ramBuffer1_dataOutConcatenated_B[(2*`SFFT_OUTPUT_WIDTH) -1
:`SFFT_OUTPUT_WIDTH ];

	//Instantiate Buffer 1
	myNewerBram BRAM_1(
		.clk(clk),
		.aa(ramBuffer1_address_A),
		.ab(ramBuffer1_address_B),
		.da(ramBuffer1_dataInConcatenated_A),
		.db(ramBuffer1_dataInConcatenated_B),
		.wa(ramBuffer1_writeEnable_A),
		.wb(ramBuffer1_writeEnable_B),
		.qa(ramBuffer1_dataOutConcatenated_A),
		.qb(ramBuffer1_dataOutConcatenated_B)
		);

	//Buffer 1 write access control
	always @(*) begin
		if (copier_access_pointer == 1) begin
			//Give access to copier stage
			ramBuffer1_address_A = ramCopier_address_A;
			ramBuffer1_writeEnable_A = ramCopier_writeEnable_A;
```

```
                ramBuffer1_dataInReal_A = ramCopier_dataInReal_A;
                ramBuffer1_dataInImag_A = ramCopier_dataInImag_A;

                ramBuffer1_address_B = ramCopier_address_B;
                ramBuffer1_writeEnable_B = ramCopier_writeEnable_B;
                ramBuffer1_dataInReal_B = ramCopier_dataInReal_B;
                ramBuffer1_dataInImag_B = ramCopier_dataInImag_B;

                ramBuffer1_readClock = ~clk;
        end

        else if (pipelineStage_access_pointer == 1) begin
                //Give access to pipeline stage
                ramBuffer1_address_A = ramStage_address_A;
                ramBuffer1_writeEnable_A = ramStage_writeEnable_A;
                ramBuffer1_dataInReal_A = ramStage_dataInReal_A;
                ramBuffer1_dataInImag_A = ramStage_dataInImag_A;

                ramBuffer1_address_B = ramStage_address_B;
                ramBuffer1_writeEnable_B = ramStage_writeEnable_B;
                ramBuffer1_dataInReal_B = ramStage_dataInReal_B;
                ramBuffer1_dataInImag_B = ramStage_dataInImag_B;

                ramBuffer1_readClock = ~clk;
        end

        else if (output_access_pointer == 1) begin
                //Give access to output port
                ramBuffer1_address_A = output_address;
                ramBuffer1_writeEnable_A = 0;
                ramBuffer1_dataInReal_A = 0;
                ramBuffer1_dataInImag_A = 0;

                ramBuffer1_address_B = 0;
                ramBuffer1_writeEnable_B = 0;
                ramBuffer1_dataInReal_B = 0;
                ramBuffer1_dataInImag_B = 0;

                ramBuffer1_readClock = clk;
        end
end

/*
```

```
         * Buffer 2
         */
        logic ramBuffer2_readClock;

        //Input bus
        logic [`nFFT -1:0] ramBuffer2_address_A;
        logic ramBuffer2_writeEnable_A;
        logic [`nFFT -1:0] ramBuffer2_address_B;
        logic ramBuffer2_writeEnable_B;

        logic [`SFFT_OUTPUT_WIDTH -1:0] ramBuffer2_dataInReal_A;
        logic [`SFFT_OUTPUT_WIDTH -1:0] ramBuffer2_dataInImag_A;

        logic [`SFFT_OUTPUT_WIDTH -1:0] ramBuffer2_dataInReal_B;
        logic [`SFFT_OUTPUT_WIDTH -1:0] ramBuffer2_dataInImag_B;

        //Output bus
        wire [`SFFT_OUTPUT_WIDTH -1:0] ramBuffer2_dataOutReal_A;
        wire [`SFFT_OUTPUT_WIDTH -1:0] ramBuffer2_dataOutImag_A;

        wire [`SFFT_OUTPUT_WIDTH -1:0] ramBuffer2_dataOutReal_B;
        wire [`SFFT_OUTPUT_WIDTH -1:0] ramBuffer2_dataOutImag_B;

        //Concatenate dataIn bus
        wire [(2*`SFFT_OUTPUT_WIDTH) -1:0] ramBuffer2_dataInConcatenated_A;
        assign ramBuffer2_dataInConcatenated_A = {ramBuffer2_dataInReal_A,
ramBuffer2_dataInImag_A};

        wire [(2*`SFFT_OUTPUT_WIDTH) -1:0] ramBuffer2_dataInConcatenated_B;
        assign ramBuffer2_dataInConcatenated_B = {ramBuffer2_dataInReal_B,
ramBuffer2_dataInImag_B};



        //Concatenate dataOut bus
        wire [(2*`SFFT_OUTPUT_WIDTH) -1:0] ramBuffer2_dataOutConcatenated_A;
        assign ramBuffer2_dataOutImag_A =
ramBuffer2_dataOutConcatenated_A[`SFFT_OUTPUT_WIDTH -1:0];
        assign ramBuffer2_dataOutReal_A =
ramBuffer2_dataOutConcatenated_A[(2*`SFFT_OUTPUT_WIDTH) -1
:`SFFT_OUTPUT_WIDTH ];

        wire [(2*`SFFT_OUTPUT_WIDTH) -1:0] ramBuffer2_dataOutConcatenated_B;
```

```
	assign ramBuffer2_dataOutImag_B =
ramBuffer2_dataOutConcatenated_B[`SFFT_OUTPUT_WIDTH -1:0];
	assign ramBuffer2_dataOutReal_B =
ramBuffer2_dataOutConcatenated_B[(2*`SFFT_OUTPUT_WIDTH) -1
:`SFFT_OUTPUT_WIDTH ];

	//Instantiate Buffer 2
	myNewerBram BRAM_2(
		.clk(clk),
		.aa(ramBuffer2_address_A),
		.ab(ramBuffer2_address_B),
		.da(ramBuffer2_dataInConcatenated_A),
		.db(ramBuffer2_dataInConcatenated_B),
		.wa(ramBuffer2_writeEnable_A),
		.wb(ramBuffer2_writeEnable_B),
		.qa(ramBuffer2_dataOutConcatenated_A),
		.qb(ramBuffer2_dataOutConcatenated_B)
		);

	//Buffer 2 write access control
	always @(*) begin
		if (copier_access_pointer == 2) begin
			//Give access to copier stage
			ramBuffer2_address_A = ramCopier_address_A;
			ramBuffer2_writeEnable_A = ramCopier_writeEnable_A;
			ramBuffer2_dataInReal_A = ramCopier_dataInReal_A;
			ramBuffer2_dataInImag_A = ramCopier_dataInImag_A;

			ramBuffer2_address_B = ramCopier_address_B;
			ramBuffer2_writeEnable_B = ramCopier_writeEnable_B;
			ramBuffer2_dataInReal_B = ramCopier_dataInReal_B;
			ramBuffer2_dataInImag_B = ramCopier_dataInImag_B;

			ramBuffer2_readClock = ~clk;
		end

		else if (pipelineStage_access_pointer == 2) begin
			//Give access to pipeline stage
			ramBuffer2_address_A = ramStage_address_A;
			ramBuffer2_writeEnable_A = ramStage_writeEnable_A;
			ramBuffer2_dataInReal_A = ramStage_dataInReal_A;
			ramBuffer2_dataInImag_A = ramStage_dataInImag_A;
```

```
                ramBuffer2_address_B = ramStage_address_B;
                ramBuffer2_writeEnable_B = ramStage_writeEnable_B;
                ramBuffer2_dataInReal_B = ramStage_dataInReal_B;
                ramBuffer2_dataInImag_B = ramStage_dataInImag_B;

                ramBuffer2_readClock = ~clk;
        end

        else if (output_access_pointer == 2) begin
                //Give access to output port
                ramBuffer2_address_A = output_address;
                ramBuffer2_writeEnable_A = 0;
                ramBuffer2_dataInReal_A = 0;
                ramBuffer2_dataInImag_A = 0;

                ramBuffer2_address_B = 0;
                ramBuffer2_writeEnable_B = 0;
                ramBuffer2_dataInReal_B = 0;
                ramBuffer2_dataInImag_B = 0;

                ramBuffer2_readClock = clk;
        end
end

/*
 * Buffer 3
 */
logic ramBuffer3_readClock;

//Input bus
logic [`nFFT -1:0] ramBuffer3_address_A;
logic ramBuffer3_writeEnable_A;
logic [`nFFT -1:0] ramBuffer3_address_B;
logic ramBuffer3_writeEnable_B;

logic [`SFFT_OUTPUT_WIDTH -1:0] ramBuffer3_dataInReal_A;
logic [`SFFT_OUTPUT_WIDTH -1:0] ramBuffer3_dataInImag_A;

logic [`SFFT_OUTPUT_WIDTH -1:0] ramBuffer3_dataInReal_B;
logic [`SFFT_OUTPUT_WIDTH -1:0] ramBuffer3_dataInImag_B;

//Output bus
wire [`SFFT_OUTPUT_WIDTH -1:0] ramBuffer3_dataOutReal_A;
```

```
        wire [`SFFT_OUTPUT_WIDTH -1:0] ramBuffer3_dataOutImag_A;

        wire [`SFFT_OUTPUT_WIDTH -1:0] ramBuffer3_dataOutReal_B;
        wire [`SFFT_OUTPUT_WIDTH -1:0] ramBuffer3_dataOutImag_B;

        //Concatenate dataIn bus
        wire [(2*`SFFT_OUTPUT_WIDTH) -1:0] ramBuffer3_dataInConcatenated_A;
        assign ramBuffer3_dataInConcatenated_A = {ramBuffer3_dataInReal_A,
ramBuffer3_dataInImag_A};

        wire [(2*`SFFT_OUTPUT_WIDTH) -1:0] ramBuffer3_dataInConcatenated_B;
        assign ramBuffer3_dataInConcatenated_B = {ramBuffer3_dataInReal_B,
ramBuffer3_dataInImag_B};

        //Concatenate dataOut bus
        wire [(2*`SFFT_OUTPUT_WIDTH) -1:0] ramBuffer3_dataOutConcatenated_A;
        assign ramBuffer3_dataOutImag_A =
ramBuffer3_dataOutConcatenated_A[`SFFT_OUTPUT_WIDTH -1:0];
        assign ramBuffer3_dataOutReal_A =
ramBuffer3_dataOutConcatenated_A[(2*`SFFT_OUTPUT_WIDTH) -1
:`SFFT_OUTPUT_WIDTH ];

        wire [(2*`SFFT_OUTPUT_WIDTH) -1:0] ramBuffer3_dataOutConcatenated_B;
        assign ramBuffer3_dataOutImag_B =
ramBuffer3_dataOutConcatenated_B[`SFFT_OUTPUT_WIDTH -1:0];
        assign ramBuffer3_dataOutReal_B =
ramBuffer3_dataOutConcatenated_B[(2*`SFFT_OUTPUT_WIDTH) -1
:`SFFT_OUTPUT_WIDTH ];

        //Instantiate Buffer 3
        myNewerBram BRAM_3(
                .clk(clk),
                .aa(ramBuffer3_address_A),
                .ab(ramBuffer3_address_B),
                .da(ramBuffer3_dataInConcatenated_A),
                .db(ramBuffer3_dataInConcatenated_B),
                .wa(ramBuffer3_writeEnable_A),
                .wb(ramBuffer3_writeEnable_B),
                .qa(ramBuffer3_dataOutConcatenated_A),
                .qb(ramBuffer3_dataOutConcatenated_B)
                );

        //Buffer 3 write access control
```

```
always @(*) begin
	if (copier_access_pointer == 3) begin
		//Give access to copier stage
		ramBuffer3_address_A = ramCopier_address_A;
		ramBuffer3_writeEnable_A = ramCopier_writeEnable_A;
		ramBuffer3_dataInReal_A = ramCopier_dataInReal_A;
		ramBuffer3_dataInImag_A = ramCopier_dataInImag_A;

		ramBuffer3_address_B = ramCopier_address_B;
		ramBuffer3_writeEnable_B = ramCopier_writeEnable_B;
		ramBuffer3_dataInReal_B = ramCopier_dataInReal_B;
		ramBuffer3_dataInImag_B = ramCopier_dataInImag_B;

		ramBuffer3_readClock = ~clk;
	end

	else if (pipelineStage_access_pointer == 3) begin
		//Give access to pipeline stage
		ramBuffer3_address_A = ramStage_address_A;
		ramBuffer3_writeEnable_A = ramStage_writeEnable_A;
		ramBuffer3_dataInReal_A = ramStage_dataInReal_A;
		ramBuffer3_dataInImag_A = ramStage_dataInImag_A;

		ramBuffer3_address_B = ramStage_address_B;
		ramBuffer3_writeEnable_B = ramStage_writeEnable_B;
		ramBuffer3_dataInReal_B = ramStage_dataInReal_B;
		ramBuffer3_dataInImag_B = ramStage_dataInImag_B;

		ramBuffer3_readClock = ~clk;
	end

	else if (output_access_pointer == 3) begin
		//Give access to output port
		ramBuffer3_address_A = output_address;
		ramBuffer3_writeEnable_A = 0;
		ramBuffer3_dataInReal_A = 0;
		ramBuffer3_dataInImag_A = 0;

		ramBuffer3_address_B = 0;
		ramBuffer3_writeEnable_B = 0;
		ramBuffer3_dataInReal_B = 0;
		ramBuffer3_dataInImag_B = 0;
```

```
                ramBuffer3_readClock = clk;
        end
end

/*
 * Read access control
 */

//pipelineStage buffer read control
always @(*) begin
        if (pipelineStage_access_pointer == 0) begin
                //Read from buffer 0
                ramStage_dataOutReal_A = ramBuffer0_dataOutReal_A;
                ramStage_dataOutImag_A = ramBuffer0_dataOutImag_A;

                ramStage_dataOutReal_B = ramBuffer0_dataOutReal_B;
                ramStage_dataOutImag_B = ramBuffer0_dataOutImag_B;
        end

        else if (pipelineStage_access_pointer == 1) begin
                //Read from buffer 1
                ramStage_dataOutReal_A = ramBuffer1_dataOutReal_A;
                ramStage_dataOutImag_A = ramBuffer1_dataOutImag_A;

                ramStage_dataOutReal_B = ramBuffer1_dataOutReal_B;
                ramStage_dataOutImag_B = ramBuffer1_dataOutImag_B;
        end

        else if (pipelineStage_access_pointer == 2) begin
                //Read from buffer 2
                ramStage_dataOutReal_A = ramBuffer2_dataOutReal_A;
                ramStage_dataOutImag_A = ramBuffer2_dataOutImag_A;

                ramStage_dataOutReal_B = ramBuffer2_dataOutReal_B;
                ramStage_dataOutImag_B = ramBuffer2_dataOutImag_B;
        end

        else if (pipelineStage_access_pointer == 3) begin
                //Read from buffer 3
                ramStage_dataOutReal_A = ramBuffer3_dataOutReal_A;
                ramStage_dataOutImag_A = ramBuffer3_dataOutImag_A;

                ramStage_dataOutReal_B = ramBuffer3_dataOutReal_B;
```

```
                ramStage_dataOutImag_B = ramBuffer3_dataOutImag_B;
            end
        end

        //output buffer read control

        always @(*) begin
            if (output_access_pointer == 0) begin
                //Read from buffer 0
                SFFT_OutReal = ramBuffer0_dataOutReal_A;
                Output_Why = ramBuffer0_dataOutReal_A;
            end

            else if (output_access_pointer == 1) begin
                //Read from buffer 1
                SFFT_OutReal = ramBuffer1_dataOutReal_A;
                Output_Why = ramBuffer1_dataOutReal_A;
            end

            else if (output_access_pointer == 2) begin
                //Read from buffer 2
                SFFT_OutReal = ramBuffer2_dataOutReal_A;
                Output_Why = ramBuffer2_dataOutReal_A;
            end

            else if (output_access_pointer == 3) begin
                //Read from buffer 3
                SFFT_OutReal = ramBuffer3_dataOutReal_A;
                Output_Why = ramBuffer3_dataOutReal_A;
            end
        end

        //_____________________________________
        //
        // Simulation Probes
        //_____________________________________
`ifdef RUNNING_SIMULATION
        wire [`nFFT -1:0] PROBE_shuffledInputIndexes [`NFFT -1:0];
        assign PROBE_shuffledInputIndexes = shuffledInputIndexes;

        wire [`SFFT_INPUT_WIDTH -1:0] PROBE_SampleBuffers [`NFFT -1:0];
        assign PROBE_SampleBuffers = SampleBuffers;
```

```
        wire [`SFFT_OUTPUT_WIDTH -1:0] PROBE_shuffledSamples [`NFFT -1:0];
        assign PROBE_shuffledSamples = shuffledSamples;

        wire PROBE_newSampleReady;
        assign PROBE_newSampleReady = newSampleReady;

`ifdef SFFT_DOWNSAMPLE_PRE
        wire [`SFFT_INPUT_WIDTH -1:0] PROBE_WindowBuffers
[`SFFT_DOWNSAMPLE_PRE_FACTOR -1:0];
        assign PROBE_WindowBuffers = WindowBuffers;
`endif  //SFFT_DOWNSAMPLE_PRE
`endif  //RUNNING_SIMULATION


 endmodule  //SFFT_Pipeline


 /*
  * Performs a single stage of the FFT butterfly calculation. Buffers inputs and outputs.
  */
 module pipelineStage(
        input clk,
        input reset,

        //Coefficient ROM
        input logic [`SFFT_FIXED_POINT_ACCURACY:0] realCoefficents [(`NFFT / 2) -1:0],
        input logic [`SFFT_FIXED_POINT_ACCURACY:0] imagCoefficents [(`NFFT / 2) -1:0],
        //K values for stage ROM
        input logic [`nFFT -1:0] kValues [(`NFFT / 2) -1:0],
        //Butterfly Indexes
        input logic [`nFFT -1:0] aIndexes [(`NFFT / 2) -1:0],
        input logic [`nFFT -1:0] bIndexes [(`NFFT / 2) -1:0],

        //BRAM IO
        output logic [`nFFT -1:0] ram_address_A,
        output logic ram_writeEnable_A,

        output wire [`SFFT_OUTPUT_WIDTH -1:0] ram_dataInReal_A,
        output wire [`SFFT_OUTPUT_WIDTH -1:0] ram_dataInImag_A,

        input logic [`SFFT_OUTPUT_WIDTH -1:0] ram_dataOutReal_A,
        input logic [`SFFT_OUTPUT_WIDTH -1:0] ram_dataOutImag_A,
```

```
output logic [`nFFT -1:0] ram_address_B,
output logic ram_writeEnable_B,

output wire [`SFFT_OUTPUT_WIDTH -1:0] ram_dataInReal_B,
output wire [`SFFT_OUTPUT_WIDTH -1:0] ram_dataInImag_B,

input logic [`SFFT_OUTPUT_WIDTH -1:0] ram_dataOutReal_B,
input logic [`SFFT_OUTPUT_WIDTH -1:0] ram_dataOutImag_B,

output logic [1:0] ram_access_pointer,

//State control
output reg idle,
output reg [`SFFT_STAGECOUNTER_WIDTH -1:0] virtualStageCounter,
input inputReady,
output reg outputReady
);


//Counter for iterating through butterflies
parameter bCounterWidth = `nFFT - 1;
reg [bCounterWidth -1:0] btflyCounter;


//_____________________________________
//
// Instantiate butterfly module
//_____________________________________

//Inputs
reg [`SFFT_FIXED_POINT_ACCURACY:0] wInReal;
reg [`SFFT_FIXED_POINT_ACCURACY:0] wInImag;

//Instantiate B
butterfly B(
        .aReal(ram_dataOutReal_A),
        .aImag(ram_dataOutImag_A),
        .bReal(ram_dataOutReal_B),
        .bImag(ram_dataOutImag_B),
        .wReal(wInReal),
        .wImag(wInImag),

        //Connect outputs directly to BRAM buffer outside of this module
```

```
        .AReal(ram_dataInReal_A),
        .AImag(ram_dataInImag_A),
        .BReal(ram_dataInReal_B),
        .BImag(ram_dataInImag_B)
        );

//MUX for selecting butterfly inputs
always @ (*) begin
        wInReal = realCoefficents[kValues[btflyCounter]];
        wInImag = imagCoefficents[kValues[btflyCounter]];
end

//Mux for BRAM buffer addresses
always @(*) begin
        ram_address_A = aIndexes[btflyCounter];
        ram_address_B = bIndexes[btflyCounter];
end

//_____________________________________
//
// Pipeline stage behaviour
//_____________________________________

parameter pipelineWidth = `NFFT /2;
integer i;
integer j;

reg clockDivider = 0;
reg processing;

assign ram_writeEnable_A = processing && clockDivider;
assign ram_writeEnable_B = processing && clockDivider;

always @ (posedge clk) begin
        if (reset) begin
                idle <= 1;

                outputReady <= 0;
                btflyCounter <= 0;
                virtualStageCounter <= 0;

                processing <= 0;
                clockDivider <= 0;
```

```
                ram_access_pointer <= 0;
            end

            else begin
                if ((idle==1) && (inputReady==1) && (outputReady==0)) begin
                    //Start processing
                    idle <= 0;

                    processing <= 1;
                    btflyCounter <= 0;
                end

                else if (idle==0) begin
                    //Write outputs
                        //NOTE: This operation is now taken care of by the BRAM
buffer outside of this module

                    //Toggle clockDivider
                    clockDivider <= ~clockDivider;


                    if (clockDivider) begin
                        //Increment counter
                        btflyCounter <= btflyCounter + 1;

                        if (btflyCounter == (pipelineWidth-1)) begin
                            //We've reached the last butterfly calculation in this
virtual stage

                            if (virtualStageCounter == `nFFT-1) begin
                                //We've reached the last stage
                                outputReady <= 1;
                                idle <= 1;

                                virtualStageCounter <= 0;

                                processing <= 0;

                                //Select which BRAM buffer to use next
                                ram_access_pointer <=
ram_access_pointer + 1;
                            end
```

```
                                        else begin
                                                //Move onto next virtual stage
                                                virtualStageCounter <= virtualStageCounter
+ 1;
                                        end
                                    end
                                end
                            end

                            else if (outputReady) begin
                                //Next stage has recieved our outputs. Set flag to 0
                                outputReady <= 0;
                            end
                        end
                end

                //_____________________________________
                //
                // Simulation Probes
                //_____________________________________
`ifdef RUNNING_SIMULATION
                wire [bCounterWidth -1:0] PROBE_btflyCounter;
                assign PROBE_btflyCounter = btflyCounter;

                wire [`SFFT_OUTPUT_WIDTH -1:0] PROBE_StageReal [`NFFT -1:0];
                assign PROBE_StageReal = StageReal;

                wire [`SFFT_OUTPUT_WIDTH -1:0] PROBE_StageImag [`NFFT -1:0];
                assign PROBE_StageImag = StageImag;

                wire [`SFFT_OUTPUT_WIDTH -1:0] PROBE_StageReal_Buffer [`NFFT -1:0];
                assign PROBE_StageReal_Buffer = StageReal_Buffer;

                wire [`SFFT_OUTPUT_WIDTH -1:0] PROBE_StageImag_Buffer [`NFFT -1:0];
                assign PROBE_StageImag_Buffer = StageImag_Buffer;

                wire [`SFFT_OUTPUT_WIDTH -1:0] PROBE_StageOutReal [`NFFT -1:0];
                assign PROBE_StageOutReal = StageOutReal;

                wire [`SFFT_OUTPUT_WIDTH -1:0] PROBE_StageOutImag [`NFFT -1:0];
                assign PROBE_StageOutImag = StageOutImag;

                //Coefficient ROM
```

```
        wire [`SFFT_FIXED_POINT_ACCURACY:0] PROBE_realCoefficents [(`NFFT / 2) -1:0];
        assign PROBE_realCoefficents = realCoefficents;
        wire [`SFFT_FIXED_POINT_ACCURACY:0] PROBE_imagCoefficents [(`NFFT / 2) -1:0];
        assign PROBE_imagCoefficents = imagCoefficents;
        //K values for stage ROM
        wire [`nFFT -1:0] PROBE_kValues [(`NFFT / 2) -1:0];
        assign PROBE_kValues = kValues;
        //Butterfly Indexes
        wire [`nFFT -1:0] PROBE_aIndexes [(`NFFT / 2) -1:0];
        assign PROBE_aIndexes = aIndexes;
        wire [`nFFT -1:0] PROBE_bIndexes [(`NFFT / 2) -1:0];

        assign PROBE_bIndexes = bIndexes;
`endif

 endmodule  //pipelineStage


 /*
  * Performs a single 2-radix FFT. Performed continuously and asynchrounously. Does not
buffer input or output
  */
module butterfly(
        //Inputs
        input [`SFFT_OUTPUT_WIDTH -1:0] aReal,
        input [`SFFT_OUTPUT_WIDTH -1:0] aImag,

        input [`SFFT_OUTPUT_WIDTH -1:0] bReal,
        input [`SFFT_OUTPUT_WIDTH -1:0] bImag,

        input [`SFFT_FIXED_POINT_ACCURACY:0] wReal,
        input [`SFFT_FIXED_POINT_ACCURACY:0] wImag,

        //Outputs
        output reg [`SFFT_OUTPUT_WIDTH -1:0] AReal,
        output reg [`SFFT_OUTPUT_WIDTH -1:0] AImag,

        output reg [`SFFT_OUTPUT_WIDTH -1:0] BReal,
        output reg [`SFFT_OUTPUT_WIDTH -1:0] BImag
        );

        //Sign extend coefficient to match bit width
```

```
        reg [`SFFT_OUTPUT_WIDTH + `SFFT_FIXED_POINT_ACCURACY -1:0]
wReal_Extended;
        reg [`SFFT_OUTPUT_WIDTH + `SFFT_FIXED_POINT_ACCURACY -1:0]
wImag_Extended;

        parameter extensionBitsCoeff = `SFFT_OUTPUT_WIDTH -1;

        always @ (*) begin
                wReal_Extended = {
{extensionBitsCoeff{wReal[`SFFT_FIXED_POINT_ACCURACY]}}, wReal};
                wImag_Extended = {
{extensionBitsCoeff{wImag[`SFFT_FIXED_POINT_ACCURACY]}}, wImag};
        end

        //Increase a bitwidth. Multiply a by 2^FIXEDPOINT
        reg [`SFFT_OUTPUT_WIDTH + `SFFT_FIXED_POINT_ACCURACY -1:0]
aReal_Extended;
        reg [`SFFT_OUTPUT_WIDTH + `SFFT_FIXED_POINT_ACCURACY -1:0]
aImag_Extended;

        parameter extensionBitsA = `SFFT_FIXED_POINT_ACCURACY;

        always @ (*) begin
                //Leftshift to multiply
                aReal_Extended = {aReal, {extensionBitsA{1'b0}}};
                aImag_Extended = {aImag, {extensionBitsA{1'b0}}};
        end

        //Increase b bitwidth
        reg [`SFFT_OUTPUT_WIDTH + `SFFT_FIXED_POINT_ACCURACY -1:0]
bReal_Extended;
        reg [`SFFT_OUTPUT_WIDTH + `SFFT_FIXED_POINT_ACCURACY -1:0]
bImag_Extended;

        parameter extensionBitsB = `SFFT_FIXED_POINT_ACCURACY;

        always @ (*) begin
                //Sign extension
                bReal_Extended = { {extensionBitsB{bReal[`SFFT_OUTPUT_WIDTH -1]}},
bReal};
                bImag_Extended = { {extensionBitsB{bImag[`SFFT_OUTPUT_WIDTH -1]}},
bImag};
        end
```

```
        //Do butterfly calculation
        reg [`SFFT_OUTPUT_WIDTH + `SFFT_FIXED_POINT_ACCURACY -1:0]
AReal_Intermediate;  //Intermediate values with wider bitwidths
        reg [`SFFT_OUTPUT_WIDTH + `SFFT_FIXED_POINT_ACCURACY -1:0]
AImag_Intermediate;

        reg [`SFFT_OUTPUT_WIDTH + `SFFT_FIXED_POINT_ACCURACY -1:0]
BReal_Intermediate;
        reg [`SFFT_OUTPUT_WIDTH + `SFFT_FIXED_POINT_ACCURACY -1:0]
BImag_Intermediate;

        always @ (*) begin
                //A = a + wb
                AReal_Intermediate = aReal_Extended + (wReal_Extended*bReal_Extended) -
(wImag_Extended*bImag_Extended);
                AImag_Intermediate = aImag_Extended + (wReal_Extended*bImag_Extended) +
(wImag_Extended*bReal_Extended);

                //B = a - wb
                BReal_Intermediate = aReal_Extended - (wReal_Extended*bReal_Extended) +
(wImag_Extended*bImag_Extended);
                BImag_Intermediate = aImag_Extended - (wReal_Extended*bImag_Extended) -
(wImag_Extended*bReal_Extended);

                //Decrease magnitude of outputs by 2^7
                AReal = AReal_Intermediate[`SFFT_OUTPUT_WIDTH +
`SFFT_FIXED_POINT_ACCURACY -1:`SFFT_FIXED_POINT_ACCURACY];
                AImag = AImag_Intermediate[`SFFT_OUTPUT_WIDTH +
`SFFT_FIXED_POINT_ACCURACY -1:`SFFT_FIXED_POINT_ACCURACY];

                BReal = BReal_Intermediate[`SFFT_OUTPUT_WIDTH +
`SFFT_FIXED_POINT_ACCURACY -1:`SFFT_FIXED_POINT_ACCURACY];
                BImag = BImag_Intermediate[`SFFT_OUTPUT_WIDTH +
`SFFT_FIXED_POINT_ACCURACY -1:`SFFT_FIXED_POINT_ACCURACY];
        end
endmodule  //butterfly


/*
 * Copies values from buffer array into a given BRAM module
 */
module copyToRamStage(
```

```
input clk,
input reset,

//Buffer array in
input logic [`SFFT_OUTPUT_WIDTH -1:0] StageInReal [`NFFT -1:0],
input logic [`SFFT_OUTPUT_WIDTH -1:0] StageInImag [`NFFT -1:0],
input copySignal,

//BRAM IO
output wire [`nFFT -1:0] address_A,
output logic writeEnable_A,
output wire [`nFFT -1:0] address_B,
output logic writeEnable_B,

output logic [`SFFT_OUTPUT_WIDTH -1:0] dataInReal_A,
output logic [`SFFT_OUTPUT_WIDTH -1:0] dataInImag_A,

output logic [`SFFT_OUTPUT_WIDTH -1:0] dataInReal_B,
output logic [`SFFT_OUTPUT_WIDTH -1:0] dataInImag_B,

//State control
output reg copying,
output reg outputReady,
output logic [1:0] ram_access_pointer
);


reg [`nFFT -1:0] addressCounter = 0;

assign address_A = addressCounter;
assign address_B = addressCounter + 1;

//Mux for dataIn values
always @(*) begin
        dataInReal_A = StageInReal[address_A];
        dataInImag_A = StageInImag[address_A];

        dataInReal_B = StageInReal[address_B];
        dataInImag_B = StageInImag[address_B];
end

always @ (posedge clk) begin
        if (reset) begin
```

```
                addressCounter <= 0;
                copying <= 0;
                outputReady <= 0;

                writeEnable_A <= 0;
                writeEnable_B <= 0;

                ram_access_pointer <= 0;
            end

            else begin
                if ((copying == 0) && (copySignal == 1)) begin
                    //start copying operation
                    copying <= 1;

                    addressCounter <= 0;
                    writeEnable_A <= 1;
                    writeEnable_B <= 1;
                end
                else if (copying) begin
                    addressCounter <= addressCounter + 1;
                    if (addressCounter == `NFFT-2) begin
                        //We're done copying
                        writeEnable_A <= 0;
                        writeEnable_B <= 0;

                        copying <= 0;
                        outputReady <= 1;

                        //Select which BRAM buffer to use next
                        ram_access_pointer <= ram_access_pointer + 1;
                    end
                end

                else if (outputReady) begin
                    outputReady <= 0;
                end
            end

        end

endmodule  //copyToRamStage
```

TEST_bram.sv

```
`include "global_variables.sv"
`include "bramNewer.v"

module BRAM_Testbench();

	reg clk = 0;

	//Input bus
	logic [`nFFT -1:0] ramBuffer0_address_A = 0;
	logic ramBuffer0_writeEnable_A = 0;
	logic [`nFFT -1:0] ramBuffer0_address_B = 0;
	logic ramBuffer0_writeEnable_B = 0;

	logic [`SFFT_OUTPUT_WIDTH -1:0] ramBuffer0_dataInReal_A = 0;
	logic [`SFFT_OUTPUT_WIDTH -1:0] ramBuffer0_dataInImag_A = 0;

	logic [`SFFT_OUTPUT_WIDTH -1:0] ramBuffer0_dataInReal_B = 0;
	logic [`SFFT_OUTPUT_WIDTH -1:0] ramBuffer0_dataInImag_B = 0;

	//Output bus
	wire [`SFFT_OUTPUT_WIDTH -1:0] ramBuffer0_dataOutReal_A;
	wire [`SFFT_OUTPUT_WIDTH -1:0] ramBuffer0_dataOutImag_A;

	wire [`SFFT_OUTPUT_WIDTH -1:0] ramBuffer0_dataOutReal_B;
	wire [`SFFT_OUTPUT_WIDTH -1:0] ramBuffer0_dataOutImag_B;

	//Concatenate dataIn bus
	wire [(2*`SFFT_OUTPUT_WIDTH) -1:0] ramBuffer0_dataInConcatenated_A;
	assign ramBuffer0_dataInConcatenated = {ramBuffer0_dataInReal_A,
ramBuffer0_dataInImag_A};

	wire [(2*`SFFT_OUTPUT_WIDTH) -1:0] ramBuffer0_dataInConcatenated_B;
	assign ramBuffer0_dataInConcatenated = {ramBuffer0_dataInReal_B,
ramBuffer0_dataInImag_B};

	//Concatenate dataOut bus
	wire [(2*`SFFT_OUTPUT_WIDTH) -1:0] ramBuffer0_dataOutConcatenated_A;
	assign ramBuffer0_dataOutConcatenated = {ramBuffer0_dataOutReal_A,
ramBuffer0_dataOutImag_A};

	wire [(2*`SFFT_OUTPUT_WIDTH) -1:0] ramBuffer0_dataOutConcatenated_B;
```

```
	assign ramBuffer0_dataOutConcatenated = {ramBuffer0_dataOutReal_B,
ramBuffer0_dataOutImag_B};

	bramNewer BRAM_0(
		.address_a ( ramBuffer0_address_A ),
		.address_b ( ramBuffer0_address_B ),
		.clock ( clk ),
		.data_a ( ramBuffer0_dataInConcatenated_A ),
		.data_b ( ramBuffer0_dataInConcatenated_B ),
		.wren_a ( ramBuffer0_writeEnable_A ),
		.wren_b ( ramBuffer0_writeEnable_B ),
		.q_a ( ramBuffer0_dataOutConcatenated_A ),
		.q_b ( ramBuffer0_dataOutConcatenated_B )
		);

	initial begin
		ramBuffer0_writeEnable_A = 1;
		ramBuffer0_writeEnable_B = 1;

		ramBuffer0_address_A = 1;
		ramBuffer0_address_B = 2;

		ramBuffer0_dataInReal_A = 11;
		ramBuffer0_dataInImag_A = 12;

		ramBuffer0_dataInReal_B = 21;
		ramBuffer0_dataInImag_B = 22;

		#1
		ramBuffer0_address_A = 3;
		ramBuffer0_address_B = 4;

		ramBuffer0_dataInReal_A = 31;
		ramBuffer0_dataInImag_A = 32;

		ramBuffer0_dataInReal_B = 41;
		ramBuffer0_dataInImag_B = 42;

		#2
		ramBuffer0_address_A = 5;
		ramBuffer0_address_B = 6;

		ramBuffer0_dataInReal_A = 51;
```

```
                ramBuffer0_dataInImag_A = 52;

                ramBuffer0_dataInReal_B = 61;
                ramBuffer0_dataInImag_B = 62;
                #1
                ramBuffer0_writeEnable_A = 0;
                ramBuffer0_writeEnable_B = 0;

                #1
                ramBuffer0_address_A = 1;
                ramBuffer0_address_B = 2;

                #2
                ramBuffer0_address_A = 3;
                ramBuffer0_address_B = 4;

                #2
                ramBuffer0_address_A = 5;
                ramBuffer0_address_B = 6;

        end

        //Clock toggling
        always begin
                #1  //2-step period
                clk <= ~clk;
        end

endmodule
```

TEST_SfftPipeline.sv

```
/*
 * Testbench for SFFT pipeline
 *
 * Tested using the following macors:
        // FFT Macros
        `define NFFT 32 // if change this, change FREQ_WIDTH. Must be power of 2
        `define nFFT 5  //log2(NFFT)

        `define FREQS (`NFFT / 2)
        `define FREQ_WIDTH 8 // if change NFFT, change this
```

```
	`define FINAL_AMPL_WIDTH 32 // Must be less than or equal to INPUT_AMPL_WIDTH
	`define INPUT_AMPL_WIDTH 32
	`define TIME_COUNTER_WIDTH 32

	`define PEAKS 6 // Changing this requires many changes in code

	`define RUNNING_SIMULATION  //define this to change ROM file locations to absolute paths fo vsim
	`define SFFT_INPUT_WIDTH 24
	`define SFFT_OUTPUT_WIDTH `INPUT_AMPL_WIDTH
	`define SFFT_FIXEDPOINT_INPUTSCALING  //define this macro if you want to scale adc inputs to match FixedPoint magnitudes. Increases accuracy, but could lead to overflow
	`define SFFT_FIXED_POINT_ACCURACY 7
	`define SFFT_STAGECOUNTER_WIDTH 5  //>= log2(nFFT)

	//`define SFFT_DOWNSAMPLE_PRE  //define this macro if you want to downsample the incoming audio BEFORE the FFT calculation
	`define SFFT_DOWNSAMPLE_PRE_FACTOR 3
	`define nDOWNSAMPLE_PRE 2  // >= log2(SFFT_DOWNSAMPLE_PRE_FACTOR)

	//`define SFFT_DOWNSAMPLE_POST  //define this macro if you want to downsample the outgoing FFT calculation (will skip calculations)
	`define SFFT_DOWNSAMPLE_POST_FACTOR 5
	`define nDOWNSAMPLE_POST 3  // >= log2(SFFT_DOWNSAMPLE_POST_FACTOR)

	// Audio Codec Macros
	`define AUDIO_IN_GAIN 9'h010
	`define AUDIO_OUT_GAIN 9'h061
 */

`include "global_variables.sv"
//`include "SfftPipeline.sv"
`include "SfftPipeline_SingleStage.sv"

`define CALCULATION_DELAY #80000

 module Sfft_Testbench();
	reg reset = 0;
	reg clk = 0;

	//Inputs
	reg [`SFFT_INPUT_WIDTH -1:0] SampleAmplitudeIn = 0;
```

```
reg advanceSignal =0;
reg OutputBeingRead = 0;

//Outputs
logic [`nFFT -1:0] output_address = 0;
wire outputReadError;
wire [`SFFT_OUTPUT_WIDTH -1:0] SFFT_OutReal;
wire [`SFFT_OUTPUT_WIDTH -1:0] Output_Why;
wire OutputValid;

SFFT_Pipeline sfft(
	.clk(clk),
	.reset(reset),

	.SampleAmplitudeIn(SampleAmplitudeIn),
	.advanceSignal(advanceSignal),

	.OutputBeingRead(OutputBeingRead),
	.outputReadError(outputReadError),
	.output_address(output_address),
	.SFFT_OutReal(SFFT_OutReal),
	.OutputValid(OutputValid),
	.Output_Why(Output_Why)
	);

initial
begin
	clk <= 0;
	reset <= 1; //Reset all modules

	#1 //posedge
	#1 //negedge

	#1 //posedge
	#1 //negedge

	reset <= 0;

	//Load in samples 100, 53, 29, 47, 30, 91, 69, 64, 50, 28, 8, 4, 45, 59, 30, 10, 74,
31, 24, 46, 71, 81, 92, 24, 93, 34, 52, 47, 5, 96, 81, 70
	SampleAmplitudeIn <= 70;
	advanceSignal <= 0;
	`CALCULATION_DELAY  //Wait for calculation to complete
```

```
#1 //posedge
advanceSignal <= 1;
#1 //negedge

SampleAmplitudeIn <= 81;
advanceSignal <= 0;
`CALCULATION_DELAY  //Wait for calculation to complete
#1 //posedge
advanceSignal <= 1;
#1 //negedge

SampleAmplitudeIn <= 96;
advanceSignal <= 0;
`CALCULATION_DELAY  //Wait for calculation to complete
#1 //posedge
advanceSignal <= 1;
#1 //negedge

SampleAmplitudeIn <= 5;
advanceSignal <= 0;
`CALCULATION_DELAY  //Wait for calculation to complete
#1 //posedge
advanceSignal <= 1;
#1 //negedge

SampleAmplitudeIn <= 47;
advanceSignal <= 0;
`CALCULATION_DELAY  //Wait for calculation to complete
#1 //posedge
advanceSignal <= 1;
#1 //negedge

SampleAmplitudeIn <= 52;
advanceSignal <= 0;
`CALCULATION_DELAY  //Wait for calculation to complete
#1 //posedge
advanceSignal <= 1;
#1 //negedge

SampleAmplitudeIn <= 34;
advanceSignal <= 0;
`CALCULATION_DELAY  //Wait for calculation to complete
#1 //posedge
```

```
                advanceSignal <= 1;
                #1 //negedge

                SampleAmplitudeIn <= 93;
                advanceSignal <= 0;
                `CALCULATION_DELAY  //Wait for calculation to complete
                #1 //posedge
                advanceSignal <= 1;
                #1 //negedge

                //Load in samples 100, 53, 29, 47, 30, 91, 69, 64, 50, 28, 8, 4, 45, 59, 30, 10, 74,
31, 24, 46, 71, 81, 92, 24, 93, 34, 52, 47, 5, 96, 81, 70
                SampleAmplitudeIn <= 24;
                advanceSignal <= 0;
                `CALCULATION_DELAY  //Wait for calculation to complete
                #1 //posedge
                advanceSignal <= 1;
                #1 //negedge

                SampleAmplitudeIn <= 92;
                advanceSignal <= 0;
                `CALCULATION_DELAY  //Wait for calculation to complete
                #1 //posedge
                advanceSignal <= 1;
                #1 //negedge

                SampleAmplitudeIn <= 81;
                advanceSignal <= 0;
                `CALCULATION_DELAY  //Wait for calculation to complete
                #1 //posedge
                advanceSignal <= 1;
                #1 //negedge

                SampleAmplitudeIn <= 71;
                advanceSignal <= 0;
                `CALCULATION_DELAY  //Wait for calculation to complete
                #1 //posedge
                advanceSignal <= 1;
                #1 //negedge

                SampleAmplitudeIn <= 46;
                advanceSignal <= 0;
                `CALCULATION_DELAY  //Wait for calculation to complete
```

```
                #1 //posedge
                advanceSignal <= 1;
                #1 //negedge

                SampleAmplitudeIn <= 24;
                advanceSignal <= 0;
                `CALCULATION_DELAY  //Wait for calculation to complete
                #1 //posedge
                advanceSignal <= 1;
                #1 //negedge

                SampleAmplitudeIn <= 31;
                advanceSignal <= 0;
                `CALCULATION_DELAY  //Wait for calculation to complete
                #1 //posedge
                advanceSignal <= 1;
                #1 //negedge

                SampleAmplitudeIn <= 74;
                advanceSignal <= 0;
                `CALCULATION_DELAY  //Wait for calculation to complete
                #1 //posedge
                advanceSignal <= 1;
                #1 //negedge

                //Load in samples 100, 53, 29, 47, 30, 91, 69, 64, 50, 28, 8, 4, 45, 59, 30, 10, 74,
31, 24, 46, 71, 81, 92, 24, 93, 34, 52, 47, 5, 96, 81, 70
                SampleAmplitudeIn <= 10;
                advanceSignal <= 0;
                `CALCULATION_DELAY  //Wait for calculation to complete
                #1 //posedge
                advanceSignal <= 1;
                #1 //negedge

                SampleAmplitudeIn <= 30;
                advanceSignal <= 0;
                `CALCULATION_DELAY  //Wait for calculation to complete
                #1 //posedge
                advanceSignal <= 1;
                #1 //negedge

                SampleAmplitudeIn <= 59;
                advanceSignal <= 0;
```

```
        `CALCULATION_DELAY  //Wait for calculation to complete
        #1 //posedge
        advanceSignal <= 1;
        #1 //negedge

        SampleAmplitudeIn <= 45;
        advanceSignal <= 0;
        `CALCULATION_DELAY  //Wait for calculation to complete
        #1 //posedge
        advanceSignal <= 1;
        #1 //negedge

        SampleAmplitudeIn <= 4;
        advanceSignal <= 0;
        `CALCULATION_DELAY  //Wait for calculation to complete
        #1 //posedge
        advanceSignal <= 1;
        #1 //negedge

        SampleAmplitudeIn <= 8;
        advanceSignal <= 0;
        `CALCULATION_DELAY  //Wait for calculation to complete
        #1 //posedge
        advanceSignal <= 1;
        #1 //negedge

        SampleAmplitudeIn <= 28;
        advanceSignal <= 0;
        `CALCULATION_DELAY  //Wait for calculation to complete
        #1 //posedge
        advanceSignal <= 1;
        #1 //negedge

        SampleAmplitudeIn <= 50;
        advanceSignal <= 0;
        `CALCULATION_DELAY  //Wait for calculation to complete
        #1 //posedge
        advanceSignal <= 1;
        #1 //negedge

        //Load in samples 100, 53, 29, 47, 30, 91, 69, 64, 50, 28, 8, 4, 45, 59, 30, 10, 74,
31, 24, 46, 71, 81, 92, 24, 93, 34, 52, 47, 5, 96, 81, 70
        SampleAmplitudeIn <= 64;
```

```
advanceSignal <= 0;
`CALCULATION_DELAY  //Wait for calculation to complete
#1 //posedge
advanceSignal <= 1;
#1 //negedge

SampleAmplitudeIn <= 69;
advanceSignal <= 0;
`CALCULATION_DELAY  //Wait for calculation to complete
#1 //posedge
advanceSignal <= 1;
#1 //negedge

SampleAmplitudeIn <= 91;
advanceSignal <= 0;
`CALCULATION_DELAY  //Wait for calculation to complete
#1 //posedge
advanceSignal <= 1;
#1 //negedge

SampleAmplitudeIn <= 30;
advanceSignal <= 0;
`CALCULATION_DELAY  //Wait for calculation to complete
#1 //posedge
advanceSignal <= 1;
#1 //negedge

SampleAmplitudeIn <= 47;
advanceSignal <= 0;
`CALCULATION_DELAY  //Wait for calculation to complete
#1 //posedge
advanceSignal <= 1;
#1 //negedge

SampleAmplitudeIn <= 29;
advanceSignal <= 0;
`CALCULATION_DELAY  //Wait for calculation to complete
#1 //posedge
advanceSignal <= 1;
#1 //negedge

SampleAmplitudeIn <= 53;
advanceSignal <= 0;
```

```
                `CALCULATION_DELAY  //Wait for calculation to complete
                #1 //posedge
                advanceSignal <= 1;
                #1 //negedge

                SampleAmplitudeIn <= 100;
                advanceSignal <= 0;
                `CALCULATION_DELAY  //Wait for calculation to complete
                #1 //posedge
                advanceSignal <= 1;
                #1 //negedge

                advanceSignal <= 0;

                `CALCULATION_DELAY
                `CALCULATION_DELAY
                $stop;
        end

        //Clock toggling
        always begin
                #1  //2-step period
                clk <= ~clk;
        end

        always @(posedge clk) begin
                output_address <= output_address + 1;
        end

 endmodule
```

soc_system.tcl

```
# Generate Quartus project files for the DE1-SoC board
#
# Stephen A. Edwards, Columbia University

# Invoke as
#
# quartus_sh -t soc_system.tcl

set project "soc_system"

# Top-level SystemVerilog file should be <project>_top.sv, with Verilog module
```

```
# <project>_top in it

set systemVerilogSource "${project}_top.sv"
set qip "${project}/synthesis/${project}.qip"

project_new $project -overwrite

foreach {name value} {
   FAMILY "Cyclone V"
   DEVICE 5CSEMA5F31C6

   PROJECT_OUTPUT_DIRECTORY output_files

   CYCLONEII_RESERVE_NCEO_AFTER_CONFIGURATION "USE AS REGULAR IO"

   NUM_PARALLEL_PROCESSORS 4

} { set_global_assignment -name $name $value }

set_global_assignment -name TOP_LEVEL_ENTITY "${project}_top"

foreach filename $systemVerilogSource {
   set_global_assignment -name SYSTEMVERILOG_FILE $filename
}

foreach filename $qip {
   set_global_assignment -name QIP_FILE $filename
}

# FPGA pin assignments

foreach {pin port} {
   PIN_AJ4 ADC_CS_N
   PIN_AK4 ADC_DIN
   PIN_AK3 ADC_DOUT
   PIN_AK2 ADC_SCLK

   PIN_K7 AUD_ADCDAT
   PIN_K8 AUD_ADCLRCK
   PIN_H7 AUD_BCLK
   PIN_J7 AUD_DACDAT
   PIN_H8 AUD_DACLRCK
   PIN_G7 AUD_XCK
```

PIN_AA16 CLOCK2_50
PIN_Y26 CLOCK3_50
PIN_K14 CLOCK4_50
PIN_AF14 CLOCK_50

PIN_AK14 DRAM_ADDR[0]
PIN_AH14 DRAM_ADDR[1]
PIN_AG15 DRAM_ADDR[2]
PIN_AE14 DRAM_ADDR[3]
PIN_AB15 DRAM_ADDR[4]
PIN_AC14 DRAM_ADDR[5]
PIN_AD14 DRAM_ADDR[6]
PIN_AF15 DRAM_ADDR[7]
PIN_AH15 DRAM_ADDR[8]
PIN_AG13 DRAM_ADDR[9]
PIN_AG12 DRAM_ADDR[10]
PIN_AH13 DRAM_ADDR[11]
PIN_AJ14 DRAM_ADDR[12]
PIN_AF13 DRAM_BA[0]
PIN_AJ12 DRAM_BA[1]
PIN_AF11 DRAM_CAS_N
PIN_AK13 DRAM_CKE
PIN_AH12 DRAM_CLK
PIN_AG11 DRAM_CS_N
PIN_AK6 DRAM_DQ[0]
PIN_AJ7 DRAM_DQ[1]
PIN_AK7 DRAM_DQ[2]
PIN_AK8 DRAM_DQ[3]
PIN_AK9 DRAM_DQ[4]
PIN_AG10 DRAM_DQ[5]
PIN_AK11 DRAM_DQ[6]
PIN_AJ11 DRAM_DQ[7]
PIN_AH10 DRAM_DQ[8]
PIN_AJ10 DRAM_DQ[9]
PIN_AJ9 DRAM_DQ[10]
PIN_AH9 DRAM_DQ[11]
PIN_AH8 DRAM_DQ[12]
PIN_AH7 DRAM_DQ[13]
PIN_AJ6 DRAM_DQ[14]
PIN_AJ5 DRAM_DQ[15]
PIN_AB13 DRAM_LDQM
PIN_AE13 DRAM_RAS_N

PIN_AK12 DRAM_UDQM
PIN_AA13 DRAM_WE_N

PIN_AA12 FAN_CTRL

PIN_J12 FPGA_I2C_SCLK
PIN_K12 FPGA_I2C_SDAT

PIN_AC18 GPIO_0[0]
PIN_Y17 GPIO_0[1]
PIN_AD17 GPIO_0[2]
PIN_Y18 GPIO_0[3]
PIN_AK16 GPIO_0[4]
PIN_AK18 GPIO_0[5]
PIN_AK19 GPIO_0[6]
PIN_AJ19 GPIO_0[7]
PIN_AJ17 GPIO_0[8]
PIN_AJ16 GPIO_0[9]
PIN_AH18 GPIO_0[10]
PIN_AH17 GPIO_0[11]
PIN_AG16 GPIO_0[12]
PIN_AE16 GPIO_0[13]
PIN_AF16 GPIO_0[14]
PIN_AG17 GPIO_0[15]
PIN_AA18 GPIO_0[16]
PIN_AA19 GPIO_0[17]
PIN_AE17 GPIO_0[18]
PIN_AC20 GPIO_0[19]
PIN_AH19 GPIO_0[20]
PIN_AJ20 GPIO_0[21]
PIN_AH20 GPIO_0[22]
PIN_AK21 GPIO_0[23]
PIN_AD19 GPIO_0[24]
PIN_AD20 GPIO_0[25]
PIN_AE18 GPIO_0[26]
PIN_AE19 GPIO_0[27]
PIN_AF20 GPIO_0[28]
PIN_AF21 GPIO_0[29]
PIN_AF19 GPIO_0[30]
PIN_AG21 GPIO_0[31]
PIN_AF18 GPIO_0[32]
PIN_AG20 GPIO_0[33]
PIN_AG18 GPIO_0[34]

PIN_AJ21 GPIO_0[35]

PIN_AB17 GPIO_1[0]
PIN_AA21 GPIO_1[1]
PIN_AB21 GPIO_1[2]
PIN_AC23 GPIO_1[3]
PIN_AD24 GPIO_1[4]
PIN_AE23 GPIO_1[5]
PIN_AE24 GPIO_1[6]
PIN_AF25 GPIO_1[7]
PIN_AF26 GPIO_1[8]
PIN_AG25 GPIO_1[9]
PIN_AG26 GPIO_1[10]
PIN_AH24 GPIO_1[11]
PIN_AH27 GPIO_1[12]
PIN_AJ27 GPIO_1[13]
PIN_AK29 GPIO_1[14]
PIN_AK28 GPIO_1[15]
PIN_AK27 GPIO_1[16]
PIN_AJ26 GPIO_1[17]
PIN_AK26 GPIO_1[18]
PIN_AH25 GPIO_1[19]
PIN_AJ25 GPIO_1[20]
PIN_AJ24 GPIO_1[21]
PIN_AK24 GPIO_1[22]
PIN_AG23 GPIO_1[23]
PIN_AK23 GPIO_1[24]
PIN_AH23 GPIO_1[25]
PIN_AK22 GPIO_1[26]
PIN_AJ22 GPIO_1[27]
PIN_AH22 GPIO_1[28]
PIN_AG22 GPIO_1[29]
PIN_AF24 GPIO_1[30]
PIN_AF23 GPIO_1[31]
PIN_AE22 GPIO_1[32]
PIN_AD21 GPIO_1[33]
PIN_AA20 GPIO_1[34]
PIN_AC22 GPIO_1[35]

PIN_AE26 HEX0[0]
PIN_AE27 HEX0[1]
PIN_AE28 HEX0[2]
PIN_AG27 HEX0[3]

PIN_AF28 HEX0[4]
PIN_AG28 HEX0[5]
PIN_AH28 HEX0[6]

PIN_AJ29 HEX1[0]
PIN_AH29 HEX1[1]
PIN_AH30 HEX1[2]
PIN_AG30 HEX1[3]
PIN_AF29 HEX1[4]
PIN_AF30 HEX1[5]
PIN_AD27 HEX1[6]

PIN_AB23 HEX2[0]
PIN_AE29 HEX2[1]
PIN_AD29 HEX2[2]
PIN_AC28 HEX2[3]
PIN_AD30 HEX2[4]
PIN_AC29 HEX2[5]
PIN_AC30 HEX2[6]

PIN_AD26 HEX3[0]
PIN_AC27 HEX3[1]
PIN_AD25 HEX3[2]
PIN_AC25 HEX3[3]
PIN_AB28 HEX3[4]
PIN_AB25 HEX3[5]
PIN_AB22 HEX3[6]

PIN_AA24 HEX4[0]
PIN_Y23 HEX4[1]
PIN_Y24 HEX4[2]
PIN_W22 HEX4[3]
PIN_W24 HEX4[4]
PIN_V23 HEX4[5]
PIN_W25 HEX4[6]

PIN_V25 HEX5[0]
PIN_AA28 HEX5[1]
PIN_Y27 HEX5[2]
PIN_AB27 HEX5[3]
PIN_AB26 HEX5[4]
PIN_AA26 HEX5[5]
PIN_AA25 HEX5[6]

PIN_AA30 IRDA_RXD
PIN_AB30 IRDA_TXD

PIN_AA14 KEY[0]
PIN_AA15 KEY[1]
PIN_W15 KEY[2]
PIN_Y16 KEY[3]

PIN_V16 LEDR[0]
PIN_W16 LEDR[1]
PIN_V17 LEDR[2]
PIN_V18 LEDR[3]
PIN_W17 LEDR[4]
PIN_W19 LEDR[5]
PIN_Y19 LEDR[6]
PIN_W20 LEDR[7]
PIN_W21 LEDR[8]
PIN_Y21 LEDR[9]

PIN_AD7 PS2_CLK
PIN_AD9 PS2_CLK2
PIN_AE7 PS2_DAT
PIN_AE9 PS2_DAT2

PIN_AB12 SW[0]
PIN_AC12 SW[1]
PIN_AF9 SW[2]
PIN_AF10 SW[3]
PIN_AD11 SW[4]
PIN_AD12 SW[5]
PIN_AE11 SW[6]
PIN_AC9 SW[7]
PIN_AD10 SW[8]
PIN_AE12 SW[9]

PIN_H15 TD_CLK27
PIN_D2 TD_DATA[0]
PIN_B1 TD_DATA[1]
PIN_E2 TD_DATA[2]
PIN_B2 TD_DATA[3]
PIN_D1 TD_DATA[4]
PIN_E1 TD_DATA[5]

```
    PIN_C2  TD_DATA[6]
    PIN_B3  TD_DATA[7]
    PIN_A5  TD_HS
    PIN_F6  TD_RESET_N
    PIN_A3  TD_VS

    PIN_A13 VGA_R[0]
    PIN_C13 VGA_R[1]
    PIN_E13 VGA_R[2]
    PIN_B12 VGA_R[3]
    PIN_C12 VGA_R[4]
    PIN_D12 VGA_R[5]
    PIN_E12 VGA_R[6]
    PIN_F13 VGA_R[7]

    PIN_J9  VGA_G[0]
    PIN_J10 VGA_G[1]
    PIN_H12 VGA_G[2]
    PIN_G10 VGA_G[3]
    PIN_G11 VGA_G[4]
    PIN_G12 VGA_G[5]
    PIN_F11 VGA_G[6]
    PIN_E11 VGA_G[7]

    PIN_B13 VGA_B[0]
    PIN_G13 VGA_B[1]
    PIN_H13 VGA_B[2]
    PIN_F14 VGA_B[3]
    PIN_H14 VGA_B[4]
    PIN_F15 VGA_B[5]
    PIN_G15 VGA_B[6]
    PIN_J14 VGA_B[7]

    PIN_A11 VGA_CLK
    PIN_B11 VGA_HS
    PIN_D11 VGA_VS
    PIN_F10 VGA_BLANK_N
    PIN_C10 VGA_SYNC_N
} {
    set_location_assignment $pin -to $port
    set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to $port
}
```

```
# HPS assignments

# 3.3-V LVTTL pins
foreach port {
    HPS_CONV_USB_N
    HPS_ENET_GTX_CLK
    HPS_ENET_INT_N
    HPS_ENET_MDC
    HPS_ENET_MDIO
    HPS_ENET_RX_CLK
    HPS_ENET_RX_DATA[0]
    HPS_ENET_RX_DATA[1]
    HPS_ENET_RX_DATA[2]
    HPS_ENET_RX_DATA[3]
    HPS_ENET_RX_DV
    HPS_ENET_TX_DATA[0]
    HPS_ENET_TX_DATA[1]
    HPS_ENET_TX_DATA[2]
    HPS_ENET_TX_DATA[3]
    HPS_ENET_TX_EN
    HPS_GSENSOR_INT
    HPS_I2C1_SCLK
    HPS_I2C1_SDAT
    HPS_I2C2_SCLK
    HPS_I2C2_SDAT
    HPS_I2C_CONTROL
    HPS_KEY
    HPS_LED
    HPS_LTC_GPIO
    HPS_SD_CLK
    HPS_SD_CMD
    HPS_SD_DATA[0]
    HPS_SD_DATA[1]
    HPS_SD_DATA[2]
    HPS_SD_DATA[3]
    HPS_SPIM_CLK
    HPS_SPIM_MISO
    HPS_SPIM_MOSI
    HPS_SPIM_SS
    HPS_UART_RX
    HPS_UART_TX
    HPS_USB_CLKOUT
    HPS_USB_DATA[0]
```

```
    HPS_USB_DATA[1]
    HPS_USB_DATA[2]
    HPS_USB_DATA[3]
    HPS_USB_DATA[4]
    HPS_USB_DATA[5]
    HPS_USB_DATA[6]
    HPS_USB_DATA[7]
    HPS_USB_DIR
    HPS_USB_NXT
    HPS_USB_STP
} {
    set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to $port
}

# There are a lot of settings for the HPS_DDR3 interface not listed here.
# Instead, the
#
# soc_system/synthesis/submodules/hps_sdram_p0_pin_assignments.tcl
#
# script generated by qsys adds that information.  However, quartus_map
# must be run before this .tcl script may run because the script
# relies on being able to look at the (HPS) netlist to determine which
# pins to constrain

set sdcFilename "${project}.sdc"

set_global_assignment -name SDC_FILE $sdcFilename

set sdcf [open $sdcFilename "w"]
puts $sdcf {
    foreach {clock port} {
        clock_50_1 CLOCK_50
        clock_50_2 CLOCK2_50
        clock_50_3 CLOCK3_50
        clock_50_4 CLOCK4_50
    } {
        create_clock -name $clock -period 20ns [get_ports $port]
    }

    create_clock -name clock_27_1 -period 37 [get_ports TD_CLK27]

    derive_pll_clocks -create_base_clocks
    derive_clock_uncertainty
```

```
}
close $sdcf

project_close
```

soc_system_top.sv

```
// ============================================================================
// Copyright (c) 2013 by Terasic Technologies Inc.
// ============================================================================
//
// Modified 2019 by Stephen A. Edwards
// Further modifies by Jose Rubianes(jer2202) & Tomin Perea-Chamblee(tep2116) & Eitan
Kaplan(ek2928)
//
// Permission:
//
//   Terasic grants permission to use and modify this code for use in
//   synthesis for all Terasic Development Boards and Altera
//   Development Kits made by Terasic.  Other use of this code,
//   including the selling ,duplication, or modification of any
//   portion is strictly prohibited.
//
// Disclaimer:
//
//   This VHDL/Verilog or C/C++ source code is intended as a design
//   reference which illustrates how these types of functions can be
//   implemented.  It is the user's responsibility to verify their
//   design for consistency and functionality through the use of
//   formal verification methods.  Terasic provides no warranty
//   regarding the use or functionality of this code.
//
// ============================================================================
//
//  Terasic Technologies Inc

//  9F., No.176, Sec.2, Gongdao 5th Rd, East Dist, Hsinchu City, 30070. Taiwan
//
//
//                     web: http://www.terasic.com/
//                     email: support@terasic.com
module soc_system_top(

 ///////// ADC /////////
```

```
inout         ADC_CS_N,
output        ADC_DIN,
input         ADC_DOUT,
output        ADC_SCLK,

///////// AUD /////////
input         AUD_ADCDAT,
inout         AUD_ADCLRCK,
inout         AUD_BCLK,
output        AUD_DACDAT,
inout         AUD_DACLRCK,
output        AUD_XCK,

///////// CLOCK2 /////////
input         CLOCK2_50,

///////// CLOCK3 /////////
input         CLOCK3_50,

///////// CLOCK4 /////////
input         CLOCK4_50,

///////// CLOCK /////////
input         CLOCK_50,

///////// DRAM /////////
output [12:0] DRAM_ADDR,
output [1:0]  DRAM_BA,
output        DRAM_CAS_N,
output        DRAM_CKE,
output        DRAM_CLK,
output        DRAM_CS_N,
inout [15:0]  DRAM_DQ,
output        DRAM_LDQM,
output        DRAM_RAS_N,
output        DRAM_UDQM,
output        DRAM_WE_N,

///////// FAN /////////
output        FAN_CTRL,

///////// FPGA /////////
output        FPGA_I2C_SCLK,
```

```
inout         FPGA_I2C_SDAT,

//////// GPIO ////////
inout [35:0]  GPIO_0,
inout [35:0]  GPIO_1,

//////// HEX0 ////////
output [6:0]  HEX0,

//////// HEX1 ////////
output [6:0]  HEX1,

//////// HEX2 ////////
output [6:0]  HEX2,

//////// HEX3 ////////
output [6:0]  HEX3,

//////// HEX4 ////////
output [6:0]  HEX4,

//////// HEX5 ////////
output [6:0]  HEX5,

//////// HPS ////////
inout         HPS_CONV_USB_N,
output [14:0] HPS_DDR3_ADDR,
output [2:0]  HPS_DDR3_BA,
output        HPS_DDR3_CAS_N,
output        HPS_DDR3_CKE,
output        HPS_DDR3_CK_N,
output        HPS_DDR3_CK_P,
output        HPS_DDR3_CS_N,
output [3:0]  HPS_DDR3_DM,
inout [31:0]  HPS_DDR3_DQ,
inout [3:0]   HPS_DDR3_DQS_N,
inout [3:0]   HPS_DDR3_DQS_P,
output        HPS_DDR3_ODT,
output        HPS_DDR3_RAS_N,
output        HPS_DDR3_RESET_N,
input         HPS_DDR3_RZQ,
output        HPS_DDR3_WE_N,
output        HPS_ENET_GTX_CLK,
```

```
inout         HPS_ENET_INT_N,
output        HPS_ENET_MDC,
inout         HPS_ENET_MDIO,
input         HPS_ENET_RX_CLK,
input [3:0]   HPS_ENET_RX_DATA,
input         HPS_ENET_RX_DV,
output [3:0]  HPS_ENET_TX_DATA,
output        HPS_ENET_TX_EN,
inout         HPS_GSENSOR_INT,
inout         HPS_I2C1_SCLK,
inout         HPS_I2C1_SDAT,
inout         HPS_I2C2_SCLK,
inout         HPS_I2C2_SDAT,
inout         HPS_I2C_CONTROL,
inout         HPS_KEY,
inout         HPS_LED,
inout         HPS_LTC_GPIO,
output        HPS_SD_CLK,
inout         HPS_SD_CMD,
inout [3:0]   HPS_SD_DATA,
output        HPS_SPIM_CLK,
input         HPS_SPIM_MISO,
output        HPS_SPIM_MOSI,
inout         HPS_SPIM_SS,
input         HPS_UART_RX,
output        HPS_UART_TX,
input         HPS_USB_CLKOUT,
inout [7:0]   HPS_USB_DATA,
input         HPS_USB_DIR,
input         HPS_USB_NXT,
output        HPS_USB_STP,

///////// IRDA /////////
input         IRDA_RXD,
output        IRDA_TXD,

///////// KEY /////////
input [3:0]   KEY,

///////// LEDR /////////
output [9:0]  LEDR,

///////// PS2 /////////
```

```
inout         PS2_CLK,
inout         PS2_CLK2,
inout         PS2_DAT,
inout         PS2_DAT2,

//////// SW ////////
input [9:0]   SW,

//////// TD ////////
input         TD_CLK27,
input [7:0]   TD_DATA,
input         TD_HS,
output        TD_RESET_N,
input         TD_VS,


//////// VGA ////////
output [7:0]  VGA_B,
output        VGA_BLANK_N,
output        VGA_CLK,
output [7:0]  VGA_G,
output        VGA_HS,
output [7:0]  VGA_R,
output        VGA_SYNC_N,
output        VGA_VS
);

  soc_system soc_system0(
    .clk_clk                    ( CLOCK_50 ),
    .reset_reset_n              ( 1'b1 ),

    .hps_ddr3_mem_a             ( HPS_DDR3_ADDR ),
    .hps_ddr3_mem_ba            ( HPS_DDR3_BA ),
    .hps_ddr3_mem_ck            ( HPS_DDR3_CK_P ),
    .hps_ddr3_mem_ck_n          ( HPS_DDR3_CK_N ),
    .hps_ddr3_mem_cke           ( HPS_DDR3_CKE ),
    .hps_ddr3_mem_cs_n          ( HPS_DDR3_CS_N ),
    .hps_ddr3_mem_ras_n         ( HPS_DDR3_RAS_N ),
    .hps_ddr3_mem_cas_n         ( HPS_DDR3_CAS_N ),
    .hps_ddr3_mem_we_n          ( HPS_DDR3_WE_N ),
    .hps_ddr3_mem_reset_n       ( HPS_DDR3_RESET_N ),
    .hps_ddr3_mem_dq            ( HPS_DDR3_DQ ),
    .hps_ddr3_mem_dqs           ( HPS_DDR3_DQS_P ),
```

```
.hps_ddr3_mem_dqs_n          ( HPS_DDR3_DQS_N ),
.hps_ddr3_mem_odt            ( HPS_DDR3_ODT ),
.hps_ddr3_mem_dm             ( HPS_DDR3_DM ),
.hps_ddr3_oct_rzqin          ( HPS_DDR3_RZQ ),

.hps_hps_io_emac1_inst_TX_CLK ( HPS_ENET_GTX_CLK ),
.hps_hps_io_emac1_inst_TXD0   ( HPS_ENET_TX_DATA[0] ),
.hps_hps_io_emac1_inst_TXD1   ( HPS_ENET_TX_DATA[1] ),
.hps_hps_io_emac1_inst_TXD2   ( HPS_ENET_TX_DATA[2] ),
.hps_hps_io_emac1_inst_TXD3   ( HPS_ENET_TX_DATA[3] ),
.hps_hps_io_emac1_inst_RXD0   ( HPS_ENET_RX_DATA[0] ),
.hps_hps_io_emac1_inst_MDIO   ( HPS_ENET_MDIO  ),
.hps_hps_io_emac1_inst_MDC    ( HPS_ENET_MDC   ),
.hps_hps_io_emac1_inst_RX_CTL ( HPS_ENET_RX_DV ),
.hps_hps_io_emac1_inst_TX_CTL ( HPS_ENET_TX_EN ),
.hps_hps_io_emac1_inst_RX_CLK ( HPS_ENET_RX_CLK ),
.hps_hps_io_emac1_inst_RXD1   ( HPS_ENET_RX_DATA[1]  ),
.hps_hps_io_emac1_inst_RXD2   ( HPS_ENET_RX_DATA[2]  ),
.hps_hps_io_emac1_inst_RXD3   ( HPS_ENET_RX_DATA[3]  ),

.hps_hps_io_sdio_inst_CMD     ( HPS_SD_CMD         ),
.hps_hps_io_sdio_inst_D0      ( HPS_SD_DATA[0]     ),
.hps_hps_io_sdio_inst_D1      ( HPS_SD_DATA[1]     ),
.hps_hps_io_sdio_inst_CLK     ( HPS_SD_CLK         ),
.hps_hps_io_sdio_inst_D2      ( HPS_SD_DATA[2]     ),
.hps_hps_io_sdio_inst_D3      ( HPS_SD_DATA[3]     ),

.hps_hps_io_usb1_inst_D0      ( HPS_USB_DATA[0]    ),
.hps_hps_io_usb1_inst_D1      ( HPS_USB_DATA[1]    ),
.hps_hps_io_usb1_inst_D2      ( HPS_USB_DATA[2]    ),
.hps_hps_io_usb1_inst_D3      ( HPS_USB_DATA[3]    ),
.hps_hps_io_usb1_inst_D4      ( HPS_USB_DATA[4]    ),
.hps_hps_io_usb1_inst_D5      ( HPS_USB_DATA[5]    ),
.hps_hps_io_usb1_inst_D6      ( HPS_USB_DATA[6]    ),
.hps_hps_io_usb1_inst_D7      ( HPS_USB_DATA[7]    ),
.hps_hps_io_usb1_inst_CLK     ( HPS_USB_CLKOUT     ),
.hps_hps_io_usb1_inst_STP     ( HPS_USB_STP        ),
.hps_hps_io_usb1_inst_DIR     ( HPS_USB_DIR        ),
.hps_hps_io_usb1_inst_NXT     ( HPS_USB_NXT        ),

.hps_hps_io_spim1_inst_CLK    ( HPS_SPIM_CLK  ),
.hps_hps_io_spim1_inst_MOSI   ( HPS_SPIM_MOSI ),
.hps_hps_io_spim1_inst_MISO   ( HPS_SPIM_MISO ),
```

```
    .hps_hps_io_spim1_inst_SS0    ( HPS_SPIM_SS   ),

    .hps_hps_io_uart0_inst_RX     ( HPS_UART_RX     ),
    .hps_hps_io_uart0_inst_TX     ( HPS_UART_TX     ),

    .hps_hps_io_i2c0_inst_SDA     ( HPS_I2C1_SDAT     ),
    .hps_hps_io_i2c0_inst_SCL     ( HPS_I2C1_SCLK     ),

    .hps_hps_io_i2c1_inst_SDA     ( HPS_I2C2_SDAT     ),
    .hps_hps_io_i2c1_inst_SCL     ( HPS_I2C2_SCLK     ),

    .hps_hps_io_gpio_inst_GPIO09  ( HPS_CONV_USB_N ),
    .hps_hps_io_gpio_inst_GPIO35  ( HPS_ENET_INT_N ),
    .hps_hps_io_gpio_inst_GPIO40  ( HPS_LTC_GPIO ),

    .hps_hps_io_gpio_inst_GPIO48  ( HPS_I2C_CONTROL ),
    .hps_hps_io_gpio_inst_GPIO53  ( HPS_LED ),
    .hps_hps_io_gpio_inst_GPIO54  ( HPS_KEY ),
    .hps_hps_io_gpio_inst_GPIO61  ( HPS_GSENSOR_INT ),

    .fft_keyinput(KEY),
    .fft_display0(HEX0),
    .fft_display1(HEX1),
    .fft_display2(HEX2),
    .fft_display3(HEX3),
    .fft_display4(HEX4),
    .fft_display5(HEX5),
    .fft_i2c_clk(FPGA_I2C_SCLK),
    .fft_i2c_data(FPGA_I2C_SDAT),
    .fft_aud_xck(AUD_XCK),
    .fft_aud_daclrck(AUD_DACLRCK),
    .fft_aud_adclrck(AUD_ADCLRCK),
    .fft_aud_bclk(AUD_BCLK),
    .fft_aud_adcdat(AUD_ADCDAT),
    .fft_aud_dacdat(AUD_DACDAT)
);

  // The following quiet the "no driver" warnings for output
  // pins and should be removed if you use any of these peripherals

  assign ADC_CS_N = SW[1] ? SW[0] : 1'bZ;
  assign ADC_DIN = SW[0];
  assign ADC_SCLK = SW[0];
```

```
//assign AUD_ADCLRCK = SW[1] ? SW[0] : 1'bZ;
//assign AUD_BCLK = SW[1] ? SW[0] : 1'bZ;
//assign AUD_DACDAT = SW[0];
//assign AUD_DACLRCK = SW[1] ? SW[0] : 1'bZ;
//assign AUD_XCK = SW[0];

assign DRAM_ADDR = { 13{ SW[0] } };
assign DRAM_BA = { 2{ SW[0] } };
assign DRAM_DQ = SW[1] ? { 16{ SW[0] } } : 16'bZ;
assign {DRAM_CAS_N, DRAM_CKE, DRAM_CLK, DRAM_CS_N,
        DRAM_LDQM, DRAM_RAS_N, DRAM_UDQM, DRAM_WE_N} = { 8{SW[0]} };

assign FAN_CTRL = SW[0];

//assign FPGA_I2C_SCLK = SW[0];
//assign FPGA_I2C_SDAT = SW[1] ? SW[0] : 1'bZ;

assign GPIO_0 = SW[1] ? { 36{ SW[0] } } : 36'bZ;
assign GPIO_1 = SW[1] ? { 36{ SW[0] } } : 36'bZ;

//assign HEX0 = { 7{ SW[1] } };
//assign HEX1 = { 7{ SW[2] } };
//assign HEX2 = { 7{ SW[3] } };
//assign HEX3 = { 7{ SW[4] } };
//assign HEX4 = { 7{ SW[5] } };
//assign HEX5 = { 7{ SW[6] } };

assign IRDA_TXD = SW[0];

assign LEDR = { 10{SW[7]} };

assign PS2_CLK = SW[1] ? SW[0] : 1'bZ;
assign PS2_CLK2 = SW[1] ? SW[0] : 1'bZ;
assign PS2_DAT = SW[1] ? SW[0] : 1'bZ;
assign PS2_DAT2 = SW[1] ? SW[0] : 1'bZ;

assign TD_RESET_N = SW[0];

assign {VGA_R, VGA_G, VGA_B} = { 24{ SW[0] } };
assign {VGA_BLANK_N, VGA_CLK,
        VGA_HS, VGA_SYNC_N, VGA_VS} = { 5{ SW[0] } };
```

```
endmodule
```

SOFTWARE MODEL

fft.py

```
#################################
'''
This script is for temporarily generating the
spectrogram for our recognition algorithm.
'''
#################################

import matplotlib.pyplot as plt
from scipy.io import wavfile
from scipy import signal
import numpy as np
import os

def generateFFT(audioDirPath, songName, fftResolution):

    # Read the wav file (mono)
    samplingFrequency, signalData = wavfile.read(audioDirPath+songName)
    # Generate spectogram
    #print(signalData[:8])
    signalData = np.mean(signalData[:(len(signalData)//2)*2].reshape(-1, 2), axis=1)

    print(songName)

    fmag = open(songName[0:-4]+".mag", 'w')
    freal = open(songName[0:-4]+".real", 'w')

    length = len(signalData) - (len(signalData) % fftResolution)
```

```
    i = 0
    while i < length - fftResolution/2:

        fft_temp = np.fft.fft(signalData[i:i+fftResolution])

        j = 0
        while j < fftResolution/2:
            #Magnitude/ Absolute Value
            fmag.write(str(np.abs(fft_temp[j])))
            fmag.write(" ")
            #Real Part of the FFT
            freal.write(str(np.abs(np.real(fft_temp[j]))))
            freal.write(" ")
            j = j + 1

        fmag.write("\n")
        freal.write("\n")

        i = i + fftResolution/2


    fmag.close()
    freal.close()


def main():
    '''
    Test our algorithm on the noisy sample tracks in the "InputFiles" folder.
    Uses the songs in the "SongFiles" folder as the library to search against.
    '''
    f = open("song_list.txt", 'w');
    songFileList = os.listdir("SongFiles")
    for songFile in songFileList:
        f.write(songFile[0:-4] + "\n");
        generateFFT("SongFiles/", songFile,  512)
    songFileList = os.listdir("InputFiles")
    for songFile in songFileList:
        generateFFT("InputFiles/", songFile, 512)
    f.close()

main()
```

generate_constellations.cpp

```
/*
 */

#include <iostream>
#include <fstream>
#include <string>
#include <cstring>
#include <sstream>
#include <list>
#include <set>
#include <vector>
#include <unordered_map>
#include <algorithm>
#include <cfloat>
#include <cmath>

/*
#define NFFT 256
#define NBINS 6
#define BIN0 0
#define BIN1 5
#define BIN2 10
#define BIN3 20
#define BIN4 40
#define BIN5 80
#define BIN6 120
*/
#define NFFT 512
#define NBINS 6
#define BIN0 0
#define BIN1 10
#define BIN2 20
#define BIN3 40
#define BIN4 80
#define BIN5 160
#define BIN6 240

#define PRUNING_COEF 1.4f
#define PRUNING_TIME_WINDOW 500
#define NORM_POW 1.0f
#define STD_DEV_COEF 1.25
#define T_ZONE 4
```

```
struct peak_raw {
	float ampl;
	uint16_t freq;
	uint16_t time;
};

struct peak {
	uint16_t freq;
	uint16_t time;
};

struct fingerprint {
	uint16_t anchor;
	uint16_t point;
	uint16_t delta;
};

struct song_data {
	std::string song_name;
	uint16_t time_pt;
	uint16_t song_ID;
};

struct hash_pair {
	uint64_t fingerprint;
	struct song_data value;
};

struct count_ID {
	std::string song;
	int count;
	int num_hashes;
};

struct database_info{
	std::string song_name;
	uint16_t song_ID;
	int hash_count;
};

std::vector<std::vector<float>> read_fft(std::string filename);
```

```
std::list<hash_pair> hash_create(std::string song_name, uint16_t song_ID);

std::vector<std::vector<float>> read_fft_noise(std::string filename);

std::list<hash_pair> hash_create_noise(std::string song_name, uint16_t song_ID);

std::list<peak> max_bins(std::vector<std::vector<float>> fft, int nfft);

std::list<peak_raw> get_peak_max_bin(std::vector<std::vector<float>> fft,
        int fft_res, int start, int end);

std::list<peak> prune(std::list<peak_raw> peaks, int max_time);

std::list<hash_pair> generate_fingerprints(std::list<peak> pruned,
        std::string song_name, uint16_t song_ID);

std::unordered_map<uint16_t, count_ID> identify_sample(
        const std::list<hash_pair> & sample_prints,
        const std::unordered_multimap<uint64_t, song_data> & database,
        std::list<database_info> song_list);

std::list<peak> generate_constellation_map(std::vector<std::vector<float>> fft, int nfft);

void write_constellation(std::list<peak> pruned, std::string filename);

std::list<peak> read_constellation(std::string filename);

float score(const struct count_ID &c) {
        return ((float) c.count)/std::pow(c.num_hashes, NORM_POW);
}
bool sortByScore(const struct count_ID &lhs, const struct count_ID &rhs) {
        return score(lhs) > score(rhs);
}

int main()
{
        /*
         * Assumes fft spectrogram files are availible at ./song_name, and that
         * song_list.txt exists and contains a list of the song names.
         */

        std::unordered_multimap<uint64_t, song_data> db;
        std::list<database_info> song_names;
```

```
std::unordered_map<uint16_t, count_ID> results;
struct hash_pair pair;
std::pair<uint64_t, song_data> temp_db;
struct database_info temp_db_info;
std::string temp_match;
std::string temp_s;

std::string output;
int hash_count;

std::fstream file;
std::string line;
std::vector<std::string> song_file_list;

uint16_t num_db = 0;

file.open("song_list.txt");
while(getline(file, line)){
  if(!line.empty()){

        num_db++;

        temp_s = line;
        hash_count = 0;

        std::list<hash_pair> identify;
        std::list<hash_pair> temp;
        temp = hash_create(temp_s+".mag", num_db);
        temp = hash_create(temp_s+".real", num_db);
        identify = hash_create_noise(temp_s + "_NOISY.mag", 0);
        identify = hash_create_noise(temp_s + "_NOISY.real", 0);
        /*
        for(std::list<hash_pair>::iterator it = temp.begin();
              it != temp.end(); ++it){

            temp_db.first = it->fingerprint;
            temp_db.second = it->value;
            db.insert(temp_db);

            hash_count++;
        }

        temp_db_info.song_name = temp_s;
```

```
            temp_db_info.hash_count = hash_count;
            temp_db_info.song_ID = num_db;
            song_names.push_back(temp_db_info);

            std::cout <<  "(" << num_db << ") ";
            std::cout << temp_s;
            std::cout << " databased.\n Number of hash table entries: ";
            std::cout << temp.size() << std::endl;
            std::cout << std::endl;
            std::cout << std::endl;*/
    }
}
file.close();

/*DEBUG
std::cout << "Full database completed \n\n" << std::endl;

std::cout << "Next is identifying: \n\n" << std::endl;

file.open("song_list.txt");
int correct = 0;
num_db = 0;
while(getline(file, line))
{
            num_db++;
            std::cout << "{" <<  num_db << "} ";
            temp_s = line;
            std::list<hash_pair> identify;
            // identify = hash_create_noise(temp_s, num_db);
            identify = hash_create_noise(temp_s + "_NOISY", num_db);

            std::cout << temp_s << + "_NOISY" << std::endl;
            // std::cout << temp_s << std::endl;

            results = identify_sample(identify, db, song_names);

            temp_match = "";
            std::vector<count_ID> sorted_results;
            for(auto iter = results.begin();
                        iter != results.end(); ++iter){
                        sorted_results.push_back(iter->second);
            }
            std::sort(sorted_results.begin(), sorted_results.end(), sortByScore);
```

```
                for (auto c = sorted_results.cbegin(); c != sorted_results.cend(); c++) {
                    std::cout << "-" << c->song << " /" << score(*c) << "/" << c->count << std::endl;
                }



                // float count_percent;
                // count_percent = (float) results[iter->song_ID].count;
                // count_percent = count_percent/std::pow(results[iter->song_ID].num_hashes,
NORM_POW);

                // std::cout << "-" << results[iter->song_ID].song <<
                //         " /" << count_percent << "/" << results[iter->song_ID].count << std::endl;

                //if(count_percent > max_count){
                //         temp_match = results[iter->song_ID].song;
                //         max_count = count_percent;
                //         }

                //}

                if(sorted_results[0].song == temp_s)
                {
                        correct++;
                }

                output = "Song Name: " + sorted_results[0].song;
                std::cout << "************************************"
                        << "*************************" << std::endl;
                std::cout << output << std::endl;
                std::cout << "Correctly matched: " << correct << "/" << num_db
                        << std::endl;
                std::cout << "************************************"
                        << "*************************" << std::endl;

        }
        file.close();
        */
        return 0;
}


std::unordered_map<uint16_t, count_ID> identify_sample(
```

```
        const std::list<hash_pair> & sample_prints,
        const std::unordered_multimap<uint64_t, song_data> & database,
        std::list<database_info> song_list)
{
        std::cout << "call to identify" << std::endl;

        std::unordered_map<uint16_t, count_ID> results;
        //new database, keys are songIDs concatenated with time anchor
        //values are number of appearances, if 5 we've matched
        std::unordered_map<uint64_t, uint8_t> db2;
        uint64_t new_key;
        uint16_t identity;

        for(std::list<database_info>::iterator iter = song_list.begin();
                iter != song_list.end(); ++iter){
                //scaling may no longer be necessary, but currently used
                results[iter->song_ID].num_hashes = iter->hash_count;
                results[iter->song_ID].song = iter->song_name;
                //set count to zero, will now be number of target zones matched
                results[iter->song_ID].count = 0;

        }

        //for fingerpint in sampleFingerprints
        for(auto iter = sample_prints.begin();
                iter != sample_prints.end(); ++iter){

           // get all the entries at this hash location
           const auto & ret = database.equal_range(iter->fingerprint);

           //lets insert the song_ID, time anchor pairs in our new database
           for(auto  it = ret.first; it != ret.second; ++it){

                   new_key = it->second.song_ID;
                   new_key = new_key << 16;
                   new_key |= it->second.time_pt;
                   new_key = new_key << 16;
                   new_key |= iter->value.time_pt;

                   db2[new_key]++;
           }

        }
```

```
	// second database is fully populated


	//adds to their count in the results structure, which is returned
	for(std::unordered_map<uint64_t,uint8_t>::iterator
			it = db2.begin(); it != db2.end(); ++it){

		//full target zone matched
		if(it->second >= T_ZONE)
		{
			//std::cout << it->second << std::endl;
			identity = it->first >> 32;
			results[identity].count += (int) (it->second);
		}
	}

	return results;

}

std::list<hash_pair> hash_create(std::string song_name, uint16_t song_ID)
{
	std::cout << "call to hash_create" << std::endl;
	std::cout << "Song ID = " << song_ID << std::endl;
	std::vector<std::vector<float>> fft;
	fft = read_fft(song_name);

	std::list<peak> pruned_peaks;
	pruned_peaks = generate_constellation_map(fft, NFFT);
	//pruned_peaks = read_constellation(song_name);

	write_constellation(pruned_peaks, song_name);
	//check that read constellation is the same
	//if(song_ID)
	//{
		std::list<peak> pruned_copy;
		pruned_copy = read_constellation(song_name);
		if(pruned_copy.front().freq == pruned_peaks.front().freq
			&& pruned_copy.front().time == pruned_peaks.front().time
			&& pruned_copy.back().freq == pruned_peaks.back().freq
			&& pruned_copy.back().time == pruned_peaks.back().time)
		{
			std::cout << "Proper constellation stored\n";
```

```
		}
		else
		{
			std::cout << "Error storing/reading constellation\n";
		}
	//}


	std::list<hash_pair> hash_entries;
	hash_entries = generate_fingerprints(pruned_peaks, song_name, song_ID);

	return hash_entries;
}

std::list<hash_pair> hash_create_noise(std::string song_name, uint16_t song_ID)
{
	std::cout << "call to hash_create_noise" << std::endl;
	std::vector<std::vector<float>> fft;
	fft = read_fft_noise(song_name);

	std::list<peak> pruned_peaks;
	pruned_peaks = generate_constellation_map(fft, NFFT);
	write_constellation(pruned_peaks, song_name);
	//check that read constellation is the same
	//if(song_ID)
	//{
		std::list<peak> pruned_copy;
		pruned_copy = read_constellation(song_name);
		if(pruned_copy.front().freq == pruned_peaks.front().freq
			&& pruned_copy.front().time == pruned_peaks.front().time
			&& pruned_copy.back().freq == pruned_peaks.back().freq
			&& pruned_copy.back().time == pruned_peaks.back().time)
		{
			std::cout << "Proper constellation stored\n";
		}
		else
		{
			std::cout << "Error storing/reading constellation\n";
		}
	//}

	std::list<hash_pair> hash_entries;
	hash_entries = generate_fingerprints(pruned_peaks, song_name, song_ID);
```

```
	return hash_entries;
}

/* get peak max bins, returns for one bin */
std::list<peak_raw> get_peak_max_bin(
	std::vector<std::vector<float>> fft,
	int fft_res, int start, int end)
{
	std::cout << "call to get_peak_max_bin" << std::endl;
	std::list<peak_raw> peaks;
	uint16_t columns;
	uint16_t sample;
	struct peak_raw current;

	columns = fft[0].size();
	sample = 1;
	// first bin
	// Assumes first bin has only the zero freq.
	if(!start && end){
	  for(uint16_t j = 1; j < columns-2; j++){
		if(fft[0][j] > fft[0][j-1] && //west
			fft[0][j] > fft[0][j+1] && //east
			fft[0][j] > fft[1][j]){ //south

		  current.freq = 0;
		  current.ampl = fft[0][j];
		  current.time = sample;
		  peaks.push_back(current);
		  sample++;
		}
	  }
	}
	// remaining bins
	else{
	 for(uint16_t i = start; i < end - 2; i++){
	  for(uint16_t j = 1; j < columns-2; j++){
		if(fft[i][j] > fft[i][j-1] && //west
			fft[i][j] > fft[i][j+1] && //east
			fft[i][j] > fft[i-1][j] && //north
			fft[i][j] > fft[i+1][j]){ //south

		  current.freq = i;
```

```
                current.ampl = fft[i][j];
                current.time = sample;
                peaks.push_back(current);
                sample++;
              }
            }
           }
          }
          return peaks;
}

/* prune a bin of peaks, returns processed std::list */
std::list<peak> prune(std::list<peak_raw> peaks, int max_time)
{
          std::cout << "call to prune" << std::endl;
          int time_bin_size;
          int time;
          float num;
          int den;
          float avg;
          std::list<peak_raw> current;
          std::list<peak> pruned;
          struct peak new_peak;
          std::set<uint16_t> sample_set;
          std::pair<std::set<uint16_t>::iterator,bool> ret;

          num = 0;
          den = 0;
          for(std::list<peak_raw>::iterator it = peaks.begin();
                    it != peaks.end(); ++it){
                num += it->ampl;
                den++;
          }

          if(den){
                avg = num/den;
                std::list<peak_raw>::iterator it = peaks.begin();

                while(it !=peaks.end()){
                      if(it->ampl <= .125*avg)
                            {peaks.erase(it++);}
                      else
```

```
                        {++it;}
        }
}

time = 0;
time_bin_size = 50;

while(time < max_time){

  num = 0;
  den = 0;

  for(std::list<peak_raw>::iterator it = peaks.begin();
                  it != peaks.end(); ++it){

        if(it->time > time && it->time < time + time_bin_size){

                ret = sample_set.insert(it->time);
                if(ret.second){
                  current.push_back(*it);
                  num += it->ampl;
                  den++;
                }
                else{
                  for(std::list<peak_raw>::iterator
                          iter = current.begin();
                          iter != current.end(); ++iter){

                          if(iter->time == it->time){
                                  //greater, update list
                                  if(it->ampl > iter->ampl){
                                          num -= iter->ampl;
                                          current.erase(iter);
                                          current.push_back(*it);
                                          num += it->ampl;
                                  }
                                  // there should only be one
                                  // so leave this inner loop
                                  break;
                          }
                  }
                }
        }
```

```
            }

            if(den){

                    avg = num/den;
                    for(std::list<peak_raw>::iterator it = current.begin();
                       it != current.end(); ++it){

                            if(it->ampl >= 1.85*avg)
                            {
                                    new_peak.freq =         it->freq;
                                    new_peak.time = it->time;
                                    pruned.push_back(new_peak);
                            }
                    }
            }

            time += time_bin_size;
            current = std::list<peak_raw>();
           }

           return pruned;
}

std::list<hash_pair> generate_fingerprints(std::list<peak> pruned,
           std::string song_name, uint16_t song_ID)
{
           std::list<hash_pair> fingerprints;
           struct fingerprint f;
           struct song_data sdata;
           struct hash_pair entry;
           uint16_t target_zone_t;
           uint64_t template_print;
           struct peak other_point;
           struct peak anchor_point;

           int target_offset = 2;

           target_zone_t = T_ZONE;


           for(std::list<peak>::iterator it = pruned.begin();
            std::next(it, target_zone_t + target_offset) != pruned.end(); it++){
```

```
                anchor_point= *it;

                for(uint16_t i = 1; i <= target_zone_t; i++){

                        other_point = *(std::next(it, i + target_offset));

                        f.anchor = anchor_point.freq;
                        f.point = other_point.freq;
                        f.delta  = other_point.time - anchor_point.time;

                        sdata.song_name = song_name;
                        sdata.time_pt = anchor_point.time;
                        sdata.song_ID = song_ID;

                        template_print = f.anchor;
                        template_print = template_print << 16;
                        template_print |= f.point;
                        template_print = template_print << 16;
                        template_print |= f.delta;

                        entry.fingerprint = template_print;
                        entry.value = sdata;

                        fingerprints.push_back(entry);
                }
        }

        return fingerprints;
}


/* Gets complete set of processed peaks */
std::list<peak> max_bins(std::vector<std::vector<float>> fft, int nfft)
{
        std::list<peak> peaks;
        std::list<peak_raw> temp_raw;
        std::list<peak> temp;

        temp_raw = get_peak_max_bin(fft, nfft/2, BIN0, BIN1);
        temp = prune(temp_raw, fft[0].size());
        peaks.splice(peaks.end(), temp);
```

```
	temp_raw = get_peak_max_bin(fft, nfft/2, BIN1, BIN2);
	temp = prune(temp_raw, fft[0].size());
	peaks.splice(peaks.end(), temp);

	temp_raw = get_peak_max_bin(fft, nfft/2, BIN2, BIN3);
	temp = prune(temp_raw, fft[0].size());
	peaks.splice(peaks.end(), temp);

	temp_raw = get_peak_max_bin(fft, nfft/2, BIN3, BIN4);
	temp = prune(temp_raw, fft[0].size());
	peaks.splice(peaks.end(), temp);

	temp_raw = get_peak_max_bin(fft, nfft/2, BIN4, BIN5);
	temp = prune(temp_raw, fft[0].size());
	peaks.splice(peaks.end(), temp);

	temp_raw = get_peak_max_bin(fft, nfft/2, BIN5, BIN6);
	temp = prune(temp_raw, fft[0].size());
	peaks.splice(peaks.end(), temp);

	return peaks;
}

std::vector<std::vector<float>> read_fft_noise(std::string filename)
{
	std::cout << "call to read_fft_noise" << std::endl;
	std::fstream file;
	std::string line;
	file.open(filename.c_str());

	int i;
	int length = 5000;

	std::vector<std::vector<float>> fft;
	fft.reserve(NFFT/2);
	for (int j = 0; j < NFFT/2; j++) {
		std::vector<float> vec;
		vec.reserve(length);
		fft.push_back(vec);
	}

	int offset = 2000;
	int counter = 0;
```

```
	while(getline(file, line) && counter < offset + length){
		if(!line.empty()){
			if (counter < offset) {
				counter++;
				continue;
			}
			std::istringstream ss(line);
			std::vector<float> line_vector;
			i = 0;
			do{
				std::string word;
				float temp;

				ss >> word;
				if(!word.empty()){
					temp = std::stof(word);
				}
				fft[i].push_back(temp);
				i++;

			} while(ss && i < NFFT/2);
			counter++;
		}
	}
	file.close();

	return fft;
}


std::vector<std::vector<float>> read_fft(std::string filename)
{
	std::cout << "call to read_fft" << std::endl;
	std::cout << filename << std::endl;
	std::fstream file;
	std::string line;
	file.open(filename.c_str());
	int i;

	std::vector<std::vector<float>> fft;
	fft.reserve(NFFT/2);
	for (int j = 0; j < NFFT/2; j++) {
```

```
                std::vector<float> vec;
                fft.push_back(vec);
        }

        while(getline(file, line)){
                if(!line.empty()){
                        i = 0;
                        std::istringstream ss(line);
                        std::vector<float> line_vector;
                        do{
                                std::string word;
                                float temp;

                                ss >> word;
                                if(!word.empty()){
                                        temp = std::stof(word);
                                }

                                fft[i].push_back(temp);
                                i++;

                        } while(ss && i < NFFT/2);
                }
        }
        file.close();

        return fft;
}



inline int freq_to_bin(uint16_t freq) {
        if (freq <  BIN1)
                return 1;
        if (freq <  BIN2)
                return 2;
        if (freq <  BIN3)
                return 3;
        if (freq <  BIN4)
                return 4;
        if (freq <  BIN5)
                return 5;
        if (freq <  BIN6)
```

```
                return 6;
        return 0;
}

std::list<peak_raw> get_raw_peaks(std::vector<std::vector<float>> fft, int nfft)
{
   std::list<peak_raw> peaks;
   uint16_t size_in_time;

   size_in_time = fft[0].size();
   for(uint16_t j = 1; j < size_in_time-2; j++){
        // WARNING not parametrized by NBINS
        float max_ampl_by_bin[NBINS + 1] = {FLT_MIN, FLT_MIN, FLT_MIN, FLT_MIN,
FLT_MIN, FLT_MIN, FLT_MIN};
        struct peak_raw max_peak_by_bin[NBINS + 1] = {};
      for(uint16_t i = 0; i < fft.size() - 1; i++){
        if(     fft[i][j] > fft[i][j-1]                 && //west
                fft[i][j] > fft[i][j+1]                     && //east
                (i < 1          || fft[i][j] > fft[i-1][j]) && //north
                (i >= fft.size() || fft[i][j] > fft[i+1][j])) { //south
                if (fft[i][j] > max_ampl_by_bin[freq_to_bin(i)]) {
                   max_ampl_by_bin[freq_to_bin(i)] = fft[i][j];
                   max_peak_by_bin[freq_to_bin(i)].freq = i;
                   max_peak_by_bin[freq_to_bin(i)].ampl = fft[i][j];
                   max_peak_by_bin[freq_to_bin(i)].time = j;
                }
        }
      }
        for (int k = 1; k <= NBINS; k++) {
           if (max_peak_by_bin[k].time != 0) {
           peaks.push_back(max_peak_by_bin[k]);
           }
        }
   }
   return peaks;
}

std::list<peak> prune_in_time(std::list<peak_raw> unpruned_peaks) {
        int time = 0;
        float num[NBINS + 1] = { };
        float den[NBINS + 1] = { };
        float dev[NBINS + 1] = { };
        int bin;
```

```
	unsigned int bin_counts[NBINS + 1] = { };
	unsigned int bin_prune_counts[NBINS + 1] = { };
	std::list<peak> pruned_peaks;
	auto add_iter = unpruned_peaks.cbegin();
	auto dev_iter = unpruned_peaks.cbegin();
	for(auto avg_iter = unpruned_peaks.cbegin(); add_iter != unpruned_peaks.cend(); ){

		if (avg_iter->time <= time + PRUNING_TIME_WINDOW && avg_iter !=
unpruned_peaks.cend()) {
			bin = freq_to_bin(avg_iter->freq);
			den[bin]++;
			num[bin] += avg_iter->ampl;
			avg_iter++;
		} else {

			while(dev_iter != avg_iter){
				if (dev_iter->time <= time + PRUNING_TIME_WINDOW
					&& dev_iter != unpruned_peaks.cend()) {

					bin = freq_to_bin(dev_iter->freq);
					if(den[bin]){
						dev[bin] += pow(dev_iter->ampl - num[bin]/den[bin],
2);
					}
					else{
						dev[bin] = den[bin];
					}
				}
				dev_iter++;
			}
			for (int i = 1; i <= NBINS; i++)
			{
				if(den[i]){
					dev[i] = sqrt(dev[i]/den[i]);
				}
				//std::cout << dev[i] << " ";
			}
			//std::cout << std::endl;
			while (add_iter != avg_iter) {
				bin = freq_to_bin(add_iter->freq);
				if (den[bin] && add_iter->ampl > STD_DEV_COEF*dev[bin] +
num[bin]/den[bin]  ) {
```

```
                                        pruned_peaks.push_back({add_iter->freq,
add_iter->time});
                                        bin_counts[freq_to_bin(add_iter->freq)]++;
                                } else {
                                        bin_prune_counts[freq_to_bin(add_iter->freq)]++;
                                }
                                add_iter++;
                        }
                        memset(num, 0, sizeof(num));
                        memset(den, 0, sizeof(den));
                        time += PRUNING_TIME_WINDOW;
                }
        }
        for (int i = 1; i <= NBINS; i++) {
                std::cout << "bin " << i << ": " << bin_counts[i] << "|  pruned: " <<
bin_prune_counts[i] << std::endl;
        }
        return pruned_peaks;
}



std::list<peak> generate_constellation_map(std::vector<std::vector<float>> fft, int nfft)
{
        std::list<peak_raw> unpruned_map;
        unpruned_map = get_raw_peaks(fft, nfft);
        return prune_in_time(unpruned_map);
}

void write_constellation(std::list<peak> pruned, std::string filename){

        std::ofstream fout;
        uint32_t peak_32;
        struct peak temp;

        fout.open(filename+"peak", std::ios::binary | std::ios::out);
        for(std::list<peak>::iterator it = pruned.begin();
                        it != pruned.end(); it++){

                temp = *it;

                peak_32 = temp.freq;
                peak_32 = peak_32 << 16;
```

```
                peak_32 |= temp.time;

                fout.write((char *)&peak_32,sizeof(peak_32));
        }

        fout.close();
}


std::list<peak> read_constellation(std::string filename){

        std::ifstream fin;
        std::list<peak> constellation;
        uint32_t peak_32;
        struct peak temp;
        std::streampos size;
        char * memblock;
        int i;

        fin.open(filename+"peak", std::ios::binary | std::ios::in
                        | std::ios::ate);

        i = 0;
        if (fin.is_open())
        {
            size = fin.tellg();
            memblock = new char [size];

            fin.seekg (0, std::ios::beg);
            fin.read (memblock, size);
            fin.close();

            while(i < size)
            {

                    peak_32 = *(uint32_t *)(memblock+i);
                    temp.time = peak_32;
                    temp.freq = peak_32 >> 16;
                    constellation.push_back(temp);
                    /* MUST incremet by this amount here*/
                    i += sizeof(peak_32);
            }
```

```
            delete[] memblock;
        }

        return constellation;

}


recognize.cpp
/*
 */

#include <iostream>
#include <fstream>
#include <string>
#include <cstring>
#include <sstream>
#include <list>
#include <set>
#include <vector>
#include <unordered_map>
#include <algorithm>
#include <cfloat>
#include <cmath>

#define NFFT 256
#define NBINS 6
#define BIN0 0
#define BIN1 5
#define BIN2 10
#define BIN3 20
#define BIN4 40
#define BIN5 80
#define BIN6 120
/*
#define NFFT 512
#define NBINS 6
#define BIN0 0
#define BIN1 10
#define BIN2 20
#define BIN3 40
#define BIN4 80
```

```
#define BIN5 160
#define BIN6 240
*/
#define PRUNING_COEF 1.4f
#define PRUNING_TIME_WINDOW 500
#define NORM_POW 1.0f
#define STD_DEV_COEF 1.25
#define T_ZONE 4

struct peak_raw {
	float ampl;
	uint16_t freq;
	uint16_t time;
};

struct peak {
	uint16_t freq;
	uint16_t time;
};

struct fingerprint {
	uint16_t anchor;
	uint16_t point;
	uint16_t delta;
};

struct song_data {
	std::string song_name;
	uint16_t time_pt;
	uint16_t song_ID;
};

struct hash_pair {
	uint64_t fingerprint;
	struct song_data value;
};

struct count_ID {
	std::string song;
	int count;
	int num_hashes;
};
```

```
struct database_info{
	std::string song_name;
	uint16_t song_ID;
	int hash_count;
};

std::vector<std::vector<float>> read_fft(std::string filename);

std::list<hash_pair> hash_create(std::string song_name, uint16_t song_ID);

std::vector<std::vector<float>> read_fft_noise(std::string filename);

std::list<hash_pair> hash_create_noise(std::string song_name, uint16_t song_ID);

std::list<peak> max_bins(std::vector<std::vector<float>> fft, int nfft);

std::list<peak_raw> get_peak_max_bin(std::vector<std::vector<float>> fft,
	int fft_res, int start, int end);

std::list<peak> prune(std::list<peak_raw> peaks, int max_time);

std::list<hash_pair> generate_fingerprints(std::list<peak> pruned,
	std::string song_name, uint16_t song_ID);

std::unordered_map<uint16_t, count_ID> identify_sample(
	const std::list<hash_pair> & sample_prints,
	const std::unordered_multimap<uint64_t, song_data> & database,
	std::list<database_info> song_list);

std::list<peak> generate_constellation_map(std::vector<std::vector<float>> fft, int nfft);

void write_constellation(std::list<peak> pruned, std::string filename);

std::list<peak> read_constellation(std::string filename);

float score(const struct count_ID &c) {
	return ((float) c.count)/std::pow(c.num_hashes, NORM_POW);
}
bool sortByScore(const struct count_ID &lhs, const struct count_ID &rhs) {
	return lhs.count == rhs.count ? score(lhs) > score(rhs) : lhs.count > rhs.count;
}

int main()
```

```
{
        /*
         * Assumes fft spectrogram files are availible at ./song_name, and that
         * song_list.txt exists and contains a list of the song names.
         */

        std::unordered_multimap<uint64_t, song_data> db;
        std::list<database_info> song_names;
        std::unordered_map<uint16_t, count_ID> results;
        struct hash_pair pair;
        std::pair<uint64_t, song_data> temp_db;
        struct database_info temp_db_info;
        std::string temp_match;
        std::string temp_s;

        std::string output;
        int hash_count;

        std::fstream file;
        std::string line;
        std::vector<std::string> song_file_list;

        uint16_t num_db = 0;

        file.open("song_list.txt");
        while(getline(file, line)){
          if(!line.empty()){

                num_db++;

                temp_s = "./"+line;
                hash_count = 0;

                std::list<hash_pair> temp;
                temp = hash_create(temp_s, num_db);

                for(std::list<hash_pair>::iterator it = temp.begin();
                        it != temp.end(); ++it){

                        temp_db.first = it->fingerprint;
                        temp_db.second = it->value;
                        db.insert(temp_db);
```

```
                hash_count++;
        }

        temp_db_info.song_name = temp_s;
        temp_db_info.hash_count = hash_count;
        temp_db_info.song_ID = num_db;
        song_names.push_back(temp_db_info);

        std::cout <<  "(" << num_db << ") ";
        std::cout << temp_s;
        std::cout << " databased.\n Number of hash table entries: ";
        std::cout << temp.size() << std::endl;
        std::cout << std::endl;
        std::cout << std::endl;
  }
}
file.close();

/*DEBUG*/
std::cout << "Full database completed \n\n" << std::endl;

std::cout << "Next is identifying: \n\n" << std::endl;

file.open("song_list.txt");
int correct = 0;
num_db = 0;
while(getline(file, line))
{
        num_db++;
        std::cout << "{" <<  num_db << "} ";
        temp_s = "./"+line;
        std::list<hash_pair> identify;
        // identify = hash_create_noise(temp_s, num_db);
        identify = hash_create_noise(temp_s + "_NOISY", num_db);

        std::cout << temp_s << + "_NOISY" << std::endl;
        // std::cout << temp_s << std::endl;

        results = identify_sample(identify, db, song_names);

        temp_match = "";
        std::vector<count_ID> sorted_results;
        for(auto iter = results.begin();
```

```
                iter != results.end(); ++iter){
                sorted_results.push_back(iter->second);
            }
            std::sort(sorted_results.begin(), sorted_results.end(), sortByScore);
            for (auto c = sorted_results.cbegin(); c != sorted_results.cend(); c++) {
               std::cout << "-" << c->song << " /" << score(*c) << "/" << c->count << std::endl;
            }



            // float count_percent;
            // count_percent = (float) results[iter->song_ID].count;
            // count_percent = count_percent/std::pow(results[iter->song_ID].num_hashes,
NORM_POW);

            // std::cout << "-" << results[iter->song_ID].song <<
            //      " /" << count_percent << "/" << results[iter->song_ID].count << std::endl;

            //if(count_percent > max_count){
            //      temp_match = results[iter->song_ID].song;
            //      max_count = count_percent;
            //      }

            //}

            if(sorted_results[0].song == temp_s)
            {
                correct++;
            }

            output = "Song Name: " + sorted_results[0].song;
            std::cout << "************************************"
                << "*************************" << std::endl;
            std::cout << output << std::endl;
            std::cout << "Correctly matched: " << correct << "/" << num_db
                << std::endl;
            std::cout << "************************************"
                << "*************************" << std::endl;

    }
    file.close();

    return 0;
```

```
}


std::unordered_map<uint16_t, count_ID> identify_sample(
        const std::list<hash_pair> & sample_prints,
        const std::unordered_multimap<uint64_t, song_data> & database,
        std::list<database_info> song_list)
{
        std::cout << "call to identify" << std::endl;

        std::unordered_map<uint16_t, count_ID> results;
        //new database, keys are songIDs concatenated with time anchor
        //values are number of appearances, if 5 we've matched
        std::unordered_map<uint64_t, uint8_t> db2;
        uint64_t new_key;
        uint16_t identity;

        for(std::list<database_info>::iterator iter = song_list.begin();
                iter != song_list.end(); ++iter){
                //scaling may no longer be necessary, but currently used
                results[iter->song_ID].num_hashes = iter->hash_count;
                results[iter->song_ID].song = iter->song_name;
                //set count to zero, will now be number of target zones matched
                results[iter->song_ID].count = 0;

        }

        //for fingerpint in sampleFingerprints
        for(auto iter = sample_prints.begin();
                iter != sample_prints.end(); ++iter){

           // get all the entries at this hash location
           const auto & ret = database.equal_range(iter->fingerprint);

           //lets insert the song_ID, time anchor pairs in our new database
           for(auto  it = ret.first; it != ret.second; ++it){

                   new_key = it->second.song_ID;
                   new_key = new_key << 16;
                   new_key |= it->second.time_pt;
                   new_key = new_key << 16;
                   new_key |= iter->value.time_pt;
```

```
                db2[new_key]++;
            }

        }
        // second database is fully populated


        //adds to their count in the results structure, which is returned
        for(std::unordered_map<uint64_t,uint8_t>::iterator
                        it = db2.begin(); it != db2.end(); ++it){

                //full target zone matched
                if(it->second >= T_ZONE)
                {
                        //std::cout << it->second << std::endl;
                        identity = it->first >> 32;
                        results[identity].count += (int) (it->second);
                }
        }

        return results;

}

std::list<hash_pair> hash_create(std::string song_name, uint16_t song_ID)
{
        std::cout << "call to hash_create" << std::endl;
        std::cout << "Song ID = " << song_ID << std::endl;
        //std::vector<std::vector<float>> fft;
        //fft = read_fft(song_name);

        std::list<peak> pruned_peaks;
        //pruned_peaks = generate_constellation_map(fft, NFFT);
        pruned_peaks = read_constellation(song_name);
        /*
        //write_constellation(pruned_peaks, song_name);
        //check that read constellation is the same
        if(song_ID)
        {
                std::list<peak> pruned_copy;
                pruned_copy = read_constellation(song_name);
                if(pruned_copy.front().freq == pruned_peaks.front().freq
                        && pruned_copy.front().time == pruned_peaks.front().time
```

```
                && pruned_copy.back().freq == pruned_peaks.back().freq
                && pruned_copy.back().time == pruned_peaks.back().time)
        {
                std::cout << "Proper constellation stored\n";
        }
        else
        {
                std::cout << "Error storing/reading constellation\n";
        }
    }
    */

    std::list<hash_pair> hash_entries;
    hash_entries = generate_fingerprints(pruned_peaks, song_name, song_ID);

    return hash_entries;
}

std::list<hash_pair> hash_create_noise(std::string song_name, uint16_t song_ID)
{
    std::cout << "call to hash_create_noise" << std::endl;
    std::vector<std::vector<float>> fft;
    fft = read_fft_noise(song_name);

    std::list<peak> pruned_peaks;
    //pruned_peaks = generate_constellation_map(fft, NFFT);
    pruned_peaks = read_constellation(song_name);
    std::list<hash_pair> hash_entries;
    hash_entries = generate_fingerprints(pruned_peaks, song_name, song_ID);

    return hash_entries;
}

/* get peak max bins, returns for one bin */
std::list<peak_raw> get_peak_max_bin(
    std::vector<std::vector<float>> fft,
    int fft_res, int start, int end)
{
    std::cout << "call to get_peak_max_bin" << std::endl;
    std::list<peak_raw> peaks;
    uint16_t columns;
    uint16_t sample;
    struct peak_raw current;
```

```
        columns = fft[0].size();
        sample = 1;
        // first bin
        // Assumes first bin has only the zero freq.
        if(!start && end){
          for(uint16_t j = 1; j < columns-2; j++){
                if(fft[0][j] > fft[0][j-1] && //west
                        fft[0][j] > fft[0][j+1] && //east
                        fft[0][j] > fft[1][j]){ //south

                  current.freq = 0;
                  current.ampl = fft[0][j];
                  current.time = sample;
                  peaks.push_back(current);
                  sample++;
                }
          }
        }
        // remaining bins
        else{
         for(uint16_t i = start; i < end - 2; i++){
          for(uint16_t j = 1; j < columns-2; j++){
                if(fft[i][j] > fft[i][j-1] && //west
                        fft[i][j] > fft[i][j+1] && //east
                        fft[i][j] > fft[i-1][j] && //north
                        fft[i][j] > fft[i+1][j]){ //south

                  current.freq = i;
                  current.ampl = fft[i][j];
                  current.time = sample;
                  peaks.push_back(current);
                  sample++;
                }
          }
         }
        }
        return peaks;
}

/* prune a bin of peaks, returns processed std::list */
std::list<peak> prune(std::list<peak_raw> peaks, int max_time)
{
```

```
std::cout << "call to prune" << std::endl;
int time_bin_size;
int time;
float num;
int den;
float avg;
std::list<peak_raw> current;
std::list<peak> pruned;
struct peak new_peak;
std::set<uint16_t> sample_set;
std::pair<std::set<uint16_t>::iterator,bool> ret;

num = 0;
den = 0;
for(std::list<peak_raw>::iterator it = peaks.begin();
            it != peaks.end(); ++it){
        num += it->ampl;
        den++;
}

if(den){
        avg = num/den;
        std::list<peak_raw>::iterator it = peaks.begin();

        while(it !=peaks.end()){
                if(it->ampl <= .125*avg)
                        {peaks.erase(it++);}
                else

                        {++it;}
        }
}

time = 0;
time_bin_size = 50;

while(time < max_time){

  num = 0;
  den = 0;

  for(std::list<peak_raw>::iterator it = peaks.begin();
                    it != peaks.end(); ++it){
```

```
        if(it->time > time && it->time < time + time_bin_size){

                ret = sample_set.insert(it->time);
                if(ret.second){
                  current.push_back(*it);
                  num += it->ampl;
                  den++;
                }
                else{
                  for(std::list<peak_raw>::iterator
                        iter = current.begin();
                        iter != current.end(); ++iter){

                        if(iter->time == it->time){
                                //greater, update list
                                if(it->ampl > iter->ampl){
                                        num -= iter->ampl;
                                        current.erase(iter);
                                        current.push_back(*it);
                                        num += it->ampl;
                                }
                                // there should only be one
                                // so leave this inner loop
                                break;
                        }
                  }
                }
        }
}

if(den){

        avg = num/den;
        for(std::list<peak_raw>::iterator it = current.begin();
           it != current.end(); ++it){

                if(it->ampl >= 1.85*avg)
                {
                        new_peak.freq =         it->freq;
                        new_peak.time = it->time;
                        pruned.push_back(new_peak);
                }
```

```
                }
            }

            time += time_bin_size;
            current = std::list<peak_raw>();
        }

        return pruned;
}

std::list<hash_pair> generate_fingerprints(std::list<peak> pruned,
        std::string song_name, uint16_t song_ID)
{
        std::list<hash_pair> fingerprints;
        struct fingerprint f;
        struct song_data sdata;
        struct hash_pair entry;
        uint16_t target_zone_t;
        uint64_t template_print;
        struct peak other_point;
        struct peak anchor_point;

        int target_offset = 2;

        target_zone_t = T_ZONE;


        for(std::list<peak>::iterator it = pruned.begin();
            std::next(it, target_zone_t + target_offset) != pruned.end(); it++){

                anchor_point= *it;

                for(uint16_t i = 1; i <= target_zone_t; i++){

                        other_point = *(std::next(it, i + target_offset));

                        f.anchor = anchor_point.freq;
                        f.point = other_point.freq;
                        f.delta  = other_point.time - anchor_point.time;

                        sdata.song_name = song_name;
                        sdata.time_pt = anchor_point.time;
                        sdata.song_ID = song_ID;
```

```
                        template_print = f.anchor;
                        template_print = template_print << 16;
                        template_print |= f.point;
                        template_print = template_print << 16;
                        template_print |= f.delta;

                        entry.fingerprint = template_print;
                        entry.value = sdata;

                        fingerprints.push_back(entry);
                }
        }

        return fingerprints;
}


/* Gets complete set of processed peaks */
std::list<peak> max_bins(std::vector<std::vector<float>> fft, int nfft)
{
        std::list<peak> peaks;
        std::list<peak_raw> temp_raw;
        std::list<peak> temp;

        temp_raw = get_peak_max_bin(fft, nfft/2, BIN0, BIN1);
        temp = prune(temp_raw, fft[0].size());
        peaks.splice(peaks.end(), temp);

        temp_raw = get_peak_max_bin(fft, nfft/2, BIN1, BIN2);
        temp = prune(temp_raw, fft[0].size());
        peaks.splice(peaks.end(), temp);

        temp_raw = get_peak_max_bin(fft, nfft/2, BIN2, BIN3);
        temp = prune(temp_raw, fft[0].size());
        peaks.splice(peaks.end(), temp);

        temp_raw = get_peak_max_bin(fft, nfft/2, BIN3, BIN4);
        temp = prune(temp_raw, fft[0].size());
        peaks.splice(peaks.end(), temp);

        temp_raw = get_peak_max_bin(fft, nfft/2, BIN4, BIN5);
        temp = prune(temp_raw, fft[0].size());
```

```
	peaks.splice(peaks.end(), temp);

	temp_raw = get_peak_max_bin(fft, nfft/2, BIN5, BIN6);
	temp = prune(temp_raw, fft[0].size());
	peaks.splice(peaks.end(), temp);

	return peaks;
}

std::vector<std::vector<float>> read_fft_noise(std::string filename)
{
	std::cout << "call to read_fft_noise" << std::endl;
	std::fstream file;
	std::string line;
	file.open(filename.c_str());

	int i;
	int length = 5000;

	std::vector<std::vector<float>> fft;
	fft.reserve(NFFT/2);
	for (int j = 0; j < NFFT/2; j++) {
		std::vector<float> vec;
		vec.reserve(length);
		fft.push_back(vec);
	}

	int offset = 2000;
	int counter = 0;

	while(getline(file, line) && counter < offset + length){
		if(!line.empty()){
			if (counter < offset) {
				counter++;
				continue;
			}
			std::istringstream ss(line);
			std::vector<float> line_vector;
			i = 0;
			do{
				std::string word;
				float temp;
```

```
                                ss >> word;
                                if(!word.empty()){
                                        temp = std::stof(word);
                                }
                                fft[i].push_back(temp);
                                i++;

                        } while(ss && i < NFFT/2);
                        counter++;
                }
        }
        file.close();

        return fft;
}

std::vector<std::vector<float>> read_fft(std::string filename)
{
        std::cout << "call to read_fft" << std::endl;
        std::cout << filename << std::endl;
        std::fstream file;
        std::string line;
        file.open(filename.c_str());
        int i;

        std::vector<std::vector<float>> fft;
        fft.reserve(NFFT/2);
        for (int j = 0; j < NFFT/2; j++) {
                std::vector<float> vec;
                fft.push_back(vec);
        }

        while(getline(file, line)){
                if(!line.empty()){
                        i = 0;
                        std::istringstream ss(line);
                        std::vector<float> line_vector;
                        do{
                                std::string word;
                                float temp;

                                ss >> word;
                                if(!word.empty()){
```

```
                                        temp = std::stof(word);
                                    }

                                    fft[i].push_back(temp);
                                    i++;

                                } while(ss && i < NFFT/2);
                            }
                        }
                        file.close();

                        return fft;
}


// Eitan's re-write:

inline int freq_to_bin(uint16_t freq) {
                        if (freq <  BIN1)
                                return 1;
                        if (freq <  BIN2)
                                return 2;
                        if (freq <  BIN3)
                                return 3;
                        if (freq <  BIN4)
                                return 4;
                        if (freq <  BIN5)
                                return 5;
                        if (freq <  BIN6)
                                return 6;
                        return 0;
}

std::list<peak_raw> get_raw_peaks(std::vector<std::vector<float>> fft, int nfft)
{
   std::list<peak_raw> peaks;
   uint16_t size_in_time;

   size_in_time = fft[0].size();
   for(uint16_t j = 1; j < size_in_time-2; j++){
                        // WARNING not parametrized by NBINS
                        float max_ampl_by_bin[NBINS + 1] = {FLT_MIN, FLT_MIN, FLT_MIN, FLT_MIN,
FLT_MIN, FLT_MIN, FLT_MIN};
```

```
        struct peak_raw max_peak_by_bin[NBINS + 1] = {};
      for(uint16_t i = 0; i < fft.size() - 1; i++){
        if(     fft[i][j] > fft[i][j-1]                    && //west
                fft[i][j] > fft[i][j+1]                       && //east
                (i < 1          || fft[i][j] > fft[i-1][j]) && //north
                (i >= fft.size() || fft[i][j] > fft[i+1][j])) { //south
                if (fft[i][j] > max_ampl_by_bin[freq_to_bin(i)]) {
                   max_ampl_by_bin[freq_to_bin(i)] = fft[i][j];
                   max_peak_by_bin[freq_to_bin(i)].freq = i;
                   max_peak_by_bin[freq_to_bin(i)].ampl = fft[i][j];
                   max_peak_by_bin[freq_to_bin(i)].time = j;
                }
        }
      }
        for (int k = 1; k <= NBINS; k++) {
           if (max_peak_by_bin[k].time != 0) {
           peaks.push_back(max_peak_by_bin[k]);
           }
        }
   }
   return peaks;
}

std::list<peak> prune_in_time(std::list<peak_raw> unpruned_peaks) {
        int time = 0;
        float num[NBINS + 1] = { };
        float den[NBINS + 1] = { };
        float dev[NBINS + 1] = { };
        int bin;
        unsigned int bin_counts[NBINS + 1] = { };
        unsigned int bin_prune_counts[NBINS + 1] = { };
        std::list<peak> pruned_peaks;
        auto add_iter = unpruned_peaks.cbegin();
        auto dev_iter = unpruned_peaks.cbegin();
        for(auto avg_iter = unpruned_peaks.cbegin(); add_iter != unpruned_peaks.cend(); ){

                if (avg_iter->time <= time + PRUNING_TIME_WINDOW && avg_iter !=
unpruned_peaks.cend()) {
                        bin = freq_to_bin(avg_iter->freq);
                        den[bin]++;
                        num[bin] += avg_iter->ampl;
                        avg_iter++;
                } else {
```

```
                        while(dev_iter != avg_iter){
                                if (dev_iter->time <= time + PRUNING_TIME_WINDOW
                                        && dev_iter != unpruned_peaks.cend()) {

                                        bin = freq_to_bin(dev_iter->freq);
                                        if(den[bin]){
                                                dev[bin] += pow(dev_iter->ampl - num[bin]/den[bin],
2);
                                        }
                                        else{
                                                dev[bin] = den[bin];
                                        }
                                }
                                dev_iter++;
                        }
                        for (int i = 1; i <= NBINS; i++)
                        {
                                if(den[i]){
                                        dev[i] = sqrt(dev[i]/den[i]);
                                }
                                //std::cout << dev[i] << " ";
                        }
                        //std::cout << std::endl;
                        while (add_iter != avg_iter) {
                                bin = freq_to_bin(add_iter->freq);
                                if (den[bin] && add_iter->ampl > STD_DEV_COEF*dev[bin] +
num[bin]/den[bin]  ) {
                                        pruned_peaks.push_back({add_iter->freq,
add_iter->time});
                                        bin_counts[freq_to_bin(add_iter->freq)]++;
                                } else {
                                        bin_prune_counts[freq_to_bin(add_iter->freq)]++;
                                }
                                add_iter++;
                        }
                        memset(num, 0, sizeof(num));
                        memset(den, 0, sizeof(den));
                        time += PRUNING_TIME_WINDOW;
                }
        }
        for (int i = 1; i <= NBINS; i++) {
```

```
                std::cout << "bin " << i << ": " << bin_counts[i] << "|  pruned: " <<
bin_prune_counts[i] << std::endl;
        }
        return pruned_peaks;
}



std::list<peak> generate_constellation_map(std::vector<std::vector<float>> fft, int nfft)
{
        std::list<peak_raw> unpruned_map;
        unpruned_map = get_raw_peaks(fft, nfft);
        return prune_in_time(unpruned_map);
}

void write_constellation(std::list<peak> pruned, std::string filename){

        std::ofstream fout;
        uint32_t peak_32;
        struct peak temp;

        fout.open(filename+".peak", std::ios::binary | std::ios::out);
        for(std::list<peak>::iterator it = pruned.begin();
                        it != pruned.end(); it++){

                temp = *it;

                peak_32 = temp.freq;
                peak_32 = peak_32 << 16;
                peak_32 |= temp.time;

                fout.write((char *)&peak_32,sizeof(peak_32));
        }

        fout.close();
}


std::list<peak> read_constellation(std::string filename){

        std::ifstream fin;
        std::list<peak> constellation;
        uint32_t peak_32;
```

```
        struct peak temp;
        std::streampos size;
        char * memblock;
        int i;

        fin.open(filename+"_48.magpeak", std::ios::binary | std::ios::in
                        | std::ios::ate);

        i = 0;
        if (fin.is_open())
        {
            size = fin.tellg();
            memblock = new char [size];

            fin.seekg (0, std::ios::beg);
            fin.read (memblock, size);
            fin.close();

            while(i < size)
            {

                    peak_32 = *(uint32_t *)(memblock+i);
                    temp.time = peak_32;
                    temp.freq = peak_32 >> 16;
                    constellation.push_back(temp);
                    /* MUST incremet by this amount here*/
                    i += sizeof(peak_32);
            }


            delete[] memblock;
        }

        return constellation;

}
```

recognizeSongs.py

```
##################################
'''
```

```
This script is for testing the functionality of our recognition algorithm.
Algorithm and implimentation is based on this article:
http://coding-geek.com/how-shazam-works/
'''
################################

import matplotlib.pyplot as plt
from scipy.io import wavfile
from scipy import signal
from skimage.feature import peak_local_max
import numpy as np
from statistics import mean
from statistics import mode
from collections import Counter, defaultdict, namedtuple
import os
import math
import heapq


def generateConstellationMap(audioFilePath,
                             fftResolution=256,
                             downsampleFactor=1,
                             sampleLength=False):
    '''
    Generates a constellation map for a given audio file.

    Parameters:
        <string> audioFilePath: path to the input audio file. File must be a
                 mono wave file with no metadata
        <int> fftResolution: number of frequencies to sample over (power of 2).
                 Defaults to 512
        <int> downsampleFactor: factor by which to downsample the timesteps of
                 the output. Defaults to 1
        <float> sampleLength: If defined, creates a constellation map only for
                 the first N seconds of the recording. Defaults to false.
                 Must be less than the length of the recording

    Returns:
        (times, peakFreaquencies)
        times: ndarray
        Array of time segments
        peakFrequencies: ndarray
        Array of peak frequencies (in Hz)
```

```
        '''

    # Read the wav file (mono)
    samplingFrequency, signalData = wavfile.read(audioFilePath)
    signalData = signalData[0:len(signalData):4]

    # Generate spectogram
    spectrogramData, frequencies, times, _ = plt.specgram(
        signalData, Fs=samplingFrequency, NFFT=fftResolution, noverlap=fftResolution/2,
scale_by_freq=False)


    #spectrogramData = 10. * np.log10(spectrogramData)

    plt.clf()
    plt.imshow(spectrogramData)

    #spectrogramData = np.transpose(spectrogramData)
    peaks = get_peaks_max_bins(spectrogramData, fftResolution)
    #peaks = get_peaks_skimage(spectrogramData)
    #plt.scatter(peaks[:,1], peaks[:,0])
    #plt.show()

    return peaks


def get_peaks_skimage(spectrogramData):
    peaks = peak_local_max(spectrogramData, min_distance=50)
    peaks = [tuple(row) for row in peaks]
    peaks = sorted(list(set(peaks)),key=lambda row: row[1])
    return peaks


def get_bins(n, nfft):
    bins_new = [0, 1, 4, 13, 24, 37, 116]
    bins = []
    j = 0
    for i in range(int(math.log2(nfft)) - n, int(math.log2(nfft))):
        bins.append(j)
        j += 2**i
    bins.append(int(nfft/2))
    print(bins)
    #print(bins_new)
```

```
    return bins


def prune_binned_peaks(peaks, NFFT, time_bin_size=50):
    bins = get_bins(6, NFFT)
    time = time_bin_size
    interval_peaks = defaultdict(list)
    pruned_peaks = []
    for peak in peaks:
        if peak.time > time:
            for binned_peaks in interval_peaks.values():
                if len(binned_peaks) != 0:
                    avg = mean(p.ampl for p in binned_peaks)
                    pruned_peaks += list(filter(lambda p: p.ampl > 1.5*avg, binned_peaks))
            interval_peaks.clear()
            time = time + time_bin_size

        interval_peaks[np.searchsorted(bins, peak.freq, side='right')].append(
            peak)
    print(len(pruned_peaks))
    return pruned_peaks


def get_peaks_max_bins(spectrogramData, NFFT, down=1):
    Peak = namedtuple('Peak', ['ampl', 'freq', 'time'])
    bins = get_bins(6, NFFT)
    sample = 0
    peaks = []
    for i in range(down, spectrogramData.shape[1] - down, down):
        fft_prev = spectrogramData[:, i-down]
        fft = spectrogramData[:, i]
        fft_next = spectrogramData[:, i+down]
        peak_bins = defaultdict(list)
        if fft[0] > fft[0+1] and fft[0] > fft_next[0] and fft[0] > fft_prev[0]:
            peak = Peak(ampl=fft[0], freq=0, time=sample)
            peak_bins[np.searchsorted(bins, peak.freq, side='right')].append(peak)
            pass
        for j in range(1, bins[-1]):
            if fft[j] > fft[j-1] and fft[j] > fft[j+1] \
                    and fft[j] > fft_next[j] and fft[j] > fft_prev[j]:
                peak = Peak(ampl=fft[j], freq=j, time=sample)
                peak_bins[np.searchsorted(bins, peak.freq, side='right')].append(peak)
        bin_peaks = [max(x, key=lambda p: p.ampl) for x in peak_bins.values()]
```

```
        if len(bin_peaks) == 0:
            continue
        avg_peak_ampl = mean([p.ampl for p in bin_peaks])
        for p in bin_peaks:
            if p.ampl >= .005*avg_peak_ampl:
                peaks.append(p)
        peak_bins.clear()
        bin_peaks.clear()
        sample += 1
    print(len(peaks))

    return [(p.freq, p.time) for p in prune_binned_peaks(peaks, NFFT)]


def get_peaks_all(spectrogramData, down=1):
    sample = 0
    peaks = []
    for i in range(down, spectrogramData.shape[1] - down, down):
        fft_prev = spectrogramData[:,i-down]
        fft = spectrogramData[:,i]
        fft_next = spectrogramData[:,i+down]
        for j in range(1, len(fft) - 1):
            if fft[j] > fft[j-1] and fft[j] > fft[j+1] \
                    and fft[j] > fft_next[j] and fft[j] > fft_prev[j]:
                peaks.append((j, sample))
        sample += 1
    return peaks


def get_peaks_avg(spectrogramData, down=1):
    learning = 0.2
    sample = 0
    peaks = []
    exp_avg = np.zeros(spectrogramData.shape[0])
    for i in range(down, spectrogramData.shape[1] - down, down):
        fft_prev = spectrogramData[:,i-down]
        fft = spectrogramData[:,i]
        fft_next = spectrogramData[:,i+down]
        for j in range(1, len(fft) - 1):
            if fft[j] > fft[j-1] and fft[j] > fft[j+1] \
                    and fft[j] > fft_next[j] and fft[j] > fft_prev[j] \
                    and fft[j] >= exp_avg[j]:
                peaks.append((j, sample))
```

```
            exp_avg[j] = fft[j] if exp_avg[j] == 0.0 \
                else exp_avg[j]*(1 - learning) + learning*fft[j]
        sample += 1
    print(len(peaks))
    return peaks

def generateFingerprints(points, songID):
    '''
    Generates fingerprints from a constellation map.
    Returns fingerprints as a list.
    '''
    # Create target zones by peaks
    targetZoneSize = 5  #number of points to include in each target zone
    targetZones = []
    for i in range(0, len(points)-targetZoneSize, 1):
        targetZones.append(points[i:i+targetZoneSize])

    # Create target zones by time
#   targetZoneDuration = 5
#   targetZones = []
#   for i in range(0, len(points), 1):
#       zone = []
#       for j in range(1, len(points) - i):
#           if points[i+j][1] - points[i][1] <= targetZoneDuration:
#               zone.append(points[i+j])
#           else:
#               break
#       targetZones.append(zone)

    # Generate fingerprints
    fingerprints = []
#    for t in range(0, len(points)): # CHANGE THIS
    for t in range(0, len(points)-targetZoneSize): # CHANGE THIS
        targetZone = targetZones[t]
        anchorPoint = points[t]
        anchorFrequency = anchorPoint[0]
        anchorTimepoint = anchorPoint[1]

        for point in targetZone:
            pointFrequency = point[0]
            timeDelta = point[1] - anchorTimepoint
            pointFingerprint = (anchorFrequency, pointFrequency, timeDelta)
            fingerprints.append((pointFingerprint, (songID, anchorTimepoint)))
```

```
    return fingerprints


def identifySample(sampleFingerprints, hashTable):
    '''
    Identify the sample based on the fingerprints in the hashtable.
    Returns the songID as a string.
    '''
    times = defaultdict(list)
    possibleMatches = []
    for fingerprint in sampleFingerprints:
        if fingerprint[0] in hashTable:
            for songData in hashTable[fingerprint[0]]:
                possibleMatches.append(songData[0])
                times[songData[0]].append((songData[1], fingerprint[1][1]))

    for song in times:
        x = []
        y=[]
        for point in times[song]:
            x.append(point[0])
            y.append(point[1])
        np_x = np.array(x)
        np_y = np.array(y)
        del x, y
        print(song + ": " + str(np.corrcoef(np_x, np_y)[0,1]))
        print(np.cov(np_x, np_y))
#       plt.clf()
#       plt.scatter(x,y)
#       plt.title(song)
#       plt.show()
    return Counter(possibleMatches).most_common()


def main():
    '''
    Test our algorithm on the noisy sample tracks in the "InputFiles" folder.
    Uses the songs in the "SongFiles" folder as the library to search against.
    '''

    # Generates a hash table (dictionary) for the song library, using fingerprints as the hash (key)
and the songID and timepoints as the data points
```

```
    print("_____")
    print("Generating fingerprints for song library...")
    hashTable = {}
    songFileList = os.listdir("SongFiles")
    for songFile in songFileList:
        print("   " + songFile)
        peaks = generateConstellationMap("SongFiles/"+songFile, downsampleFactor=4)
        fingerprints = generateFingerprints(peaks, songFile.split("_")[0])

        for fingperprint in fingerprints:
            hashAddress = fingperprint[0]
            songData = fingperprint[1]

            if hashAddress in hashTable:  # fingerprint already exists
                hashTable[hashAddress].append(songData)
            else:
                hashTable[hashAddress] = [songData]  # create new list of possible matches

    # Try to identify noisy samples
    print("_____")
    print("Identifying samples...")
    songFileList = os.listdir("InputFiles")
    for songFile in songFileList:
        peaks = generateConstellationMap("InputFiles/"+songFile, downsampleFactor=4,
sampleLength=20)  #try to identify on first 20 seconds of sample
        sampleFingerprints = generateFingerprints(peaks, songFile.split("_")[0])
        results = identifySample(sampleFingerprints, hashTable)
        if len(results) == 0:
            guess = "UNKNOWN"
            confidence = 0
        else:
            guess = results[0][0]
            if len(results) == 1:
                confidence = 1
            else:
                confidence = (results[0][1] - results[1][1])/results[0][1]
        print("   " + songFile + " => " + guess + " -\t confidence: " + str(confidence))


main()
```