1 Software Listings

1.1 Buffer

arm_main.cpp

```
1  // Code that runs on the Arm MCU buffer. Reads data from ADCs using SPI, then
2  // formats it into a serial packet and sends it to the back-end over serial.
3  // Maintains sampling rate of >50kHz.
4  // Written by Matthew Johns (mrj1g17@soton.ac.uk)
5  #include "mbed.h"
6
7  // Constants for pin numbers
8  #define ADC0_PIN A0
9  #define ADC1_PIN A1
10 #define ADC2_PIN A2
11 #define ADC3_PIN A3
12 #define CS_PIN D5
13 #define CLK_PIN D6
14
15 #define BUFFER_SIZE 1024
16 #define DATA_BITS 9
17 #define NUM_MICS 4
18 #define CLK_DELAY 8
19 #define BAUD 460800
20 #define START_BYTE 0xFF
21 DigitalOut cs(CS_PIN);
22 DigitalOut clk(CLK_PIN);
23 Serial serial(USBTX, USBRX);
24
25 char serial_buffer[NUM_MICS*BUFFER_SIZE];
26 uint16_t current_sample[NUM_MICS];
27 uint16_t samples_buffer[BUFFER_SIZE*NUM_MICS];
28 uint16_t top = 0;
29 uint8_t stall=0;
30
31 DigitalIn a0(ADC0_PIN);
32 DigitalIn a1(ADC1_PIN);
33 DigitalIn a2(ADC2_PIN);
34 DigitalIn a3(ADC3_PIN);
35
36 // Uses a bit-bashing method similar to SPI to read the values from all four
37 // ADCs. Then the values are manipulated into the correct format to be
38 // represented by a single int variable each.// These can then be sent over
39 // serial.
40 void read_samples()
41 {
42     // Pulse clock once to get ADC sample going
43     cs = 0;
44     clk = 1;
45     clk = 0;
46
47     for(uint8_t i = 0; i < DATA_BITS; i++)
48     {
49         clk = 1;
50
51     // Introduce delay to maintain square clock pulse and keep SPI clock below
52     // 1MHz. #pragmas are to try and stop it being optimised out by compiler
53 #pragma GCC push_options
54 #pragma GCC optimze ("no-unroll-loops")
55         for(uint8_t d=0; d <= CLK_DELAY; d++) {stall=d; __asm volatile("NOP");}
56 #pragma GCC pop_options
```

```
57
58          clk = 0;
59
60          current_sample[0] += a0 << (DATA_BITS-(i+1));
61          current_sample[1] += a1 << (DATA_BITS-(i+1));
62          current_sample[2] += a2 << (DATA_BITS-(i+1));
63          current_sample[3] += a3 << (DATA_BITS-(i+1));
64
65          // Above logic produces enough of a delay to not need an extra one
66      }
67      // The MSB is a sign bit, and should always be 0. If it isn't, the bit may
68      // have been corrupted and the sample should be set to 0.
69      for(uint8_t i = 0; i < NUM_MICS; i++)
70      {
71          if(current_sample[i] >> (DATA_BITS-1))
72              current_sample[i] = 0;
73      }
74
75      cs = 1;
76  }
77
78  // Takes data in samples_buffer and makes it suitable for transmission, then
79  // sends it
80  void send_serial()
81  {
82      serial.putc(START_BYTE);
83      for(uint16_t i = 0; i < BUFFER_SIZE*NUM_MICS; i++)
84      {
85          // Truncating sample to so the serial only sends the necessary 8 bits
86          serial_buffer[i] = (uint8_t)(samples_buffer[i]);
87
88          // Start byte is 0xff (255). If the sample == 255 it must be made 254
89          // to avoid confusion. It's a small error so shouldn't cause issues
90          if(serial_buffer[i] == 255)
91              serial_buffer[i] = 254;
92
93          // Sending the value of the sample
94          serial.putc(serial_buffer[i]);
95      }
96
97      // Don't need to reset the samples buffer, as it will be overwritten. Just
98      // say the top is the first element and new samples will be stored there
99      top = 0;
100 }
101
102 int main()
103 {
104     // Initial setup to keep the ADCs happy
105     cs = 1;
106     clk = 0;
107
108     serial.baud(BAUD);
109
110     // mbed OS scheduler thread suspected to be messing with timings. Make
111     // everything critical except serial tranmissions to ensure the sampling
112     // rate is maintained
113     CriticalSectionLock::enable();
114
115     for(;;)
116     {
```

```
        read_samples();

        // Adding the current sample to the buffer once it is retrieved
        for(uint8_t i = 0; i < NUM_MICS; i++)
        {
            samples_buffer[top] = current_sample[i];
            top++;
        }

        // Checks to see if the buffer is full. If so, sends serial.
        // Have to disable CriticalSectionLock, as serial uses interrupts which
        // cannot work when locked (crashes OS). Locked straight after though
        if(top == BUFFER_SIZE*4)
        {
            CriticalSectionLock::disable();
            send_serial();
            CriticalSectionLock::enable();
        }
    }

    return 0;
}
```

logic_test_program.cpp

```
// Simple command line program to test the bit-shifting logic of the buffer
// code. Data on the ADC inputs is simulated using the a0-a3 arrays. Prints
// outputted char at the same time as it would send over serial.
// Structure is generally very similar to MCU code to be more comparable. More
// information on code function can be found in comments of arm_main.cpp.
// Written by Matthew Johns (mrj1g17@soton.ac.uk)
#include <iostream>
using namespace std;

#define BUFFER_SIZE 1
#define DATA_BITS 9
#define NUM_MICS 4

uint8_t serial_buffer[NUM_MICS*BUFFER_SIZE];
uint16_t current_sample[NUM_MICS];
uint16_t samples_buffer[BUFFER_SIZE*NUM_MICS];
uint16_t top = 0;

// These test the important cases:
uint8_t a0[DATA_BITS] = {0,0,0,1,0,0,1,1,1}; // Standard number
uint8_t a1[DATA_BITS] = {1,1,1,0,1,1,0,1,0}; // Negative reading
uint8_t a2[DATA_BITS] = {0,1,1,1,1,1,1,1,1}; // 255 value
uint8_t a3[DATA_BITS] = {0,0,0,0,0,0,0,0,0}; // 0 value


void sample()
{
    for(uint8_t i = 0; i < DATA_BITS; i++)
    {
        current_sample[0] += a0[i] << (DATA_BITS-(i+1));
        current_sample[1] += a1[i] << (DATA_BITS-(i+1));
        current_sample[2] += a2[i] << (DATA_BITS-(i+1));
        current_sample[3] += a3[i] << (DATA_BITS-(i+1));
    }

    for(uint8_t i = 0; i < NUM_MICS; i++)
```

```
    {
        if(current_sample[i] >> (DATA_BITS-1))
            current_sample[i] = 0;
    }
}

void serial()
{
    for(uint16_t i = 0; i < BUFFER_SIZE*NUM_MICS; i++)
    {
        // cout << "samples_buffer: " << samples_buffer[i] << endl;
        serial_buffer[i] = (uint8_t)(samples_buffer[i]);

        if(serial_buffer[i] == 255)
            serial_buffer[i] = 254;

        // Have to cast serial_buffer[] else it tries to print like a char.
        // (Gives nonsense/unhelpful output)
        cout << "Sample " << i << ": " << (int)serial_buffer[i] << endl;

    }
    top = 0;
}

int main()
{
    sample();

    for(uint8_t i = 0; i < NUM_MICS; i++)
    {
        // cout << "top: " << top << endl;
        samples_buffer[top] = current_sample[i];
        top++;
    }

    if(top == BUFFER_SIZE*4)
        serial();

    return 0;
}
```

1.2 Signal Processing

conf.h

```
#define CONF_ROOT "/tmp/"

#define CONF_INPUT "/dev/ttyACM0"
#define CONF_CTL CONF_ROOT "chinchilla-backend-ctl"
#define CONF_SOUND CONF_ROOT "chinchilla-sounds"
#define CONF_FFT CONF_ROOT "chinchilla-fft"
```

main.c

```
#include <unistd.h>
#include <stdio.h>
#include <sys/stat.h>
#include "sample.h"
#include "xcorr.h"
#include "errno.h"
#include "string.h"
```

```
8  #include "conf.h"
9
10 /* Make and return a stream pointed to the backend control file */
11 FILE *ctl_file(void)
12 {
13     FILE *f;
14
15     if (mkfifo(CONF_CTL, 0666) == -1)
16     {
17         if (errno != EEXIST)
18             printf("Error, cannot make backend-ctl fifo: %s\n", strerror(errno));
19     }
20
21     f = fopen(CONF_CTL, "r");
22
23     return f;
24 }
25
26 /* Cleanup temporary files and fifos once I close */
27 void clean_files(void)
28 {
29     FILE *f;
30
31     /* Delete the control file */
32     unlink(CONF_CTL);
33
34     /* Just empty the CONF_SOUND file */
35     f = fopen(CONF_SOUND, "w");
36     fwrite("", 1, 0, f);
37 }
38
39 void main(void)
40 {
41     int running;
42     FILE *ctlf;
43     xcorr_manager_s manager;
44     /* Make child threads */
45     xcorr_manager_init(&manager);
46
47     running = 1;
48     /* Open a control file input */
49     ctlf = ctl_file();
50
51     while (running)
52     {
53         char line[16];
54         char *chr, *end;
55         chr = &line[0];
56         end = &line[sizeof(line) - 1];
57         memset(line, 0, sizeof(line));
58
59         /* Read a line from the control file */
60         while (chr < end)
61         {
62             int cint;
63             cint = fgetc(ctlf);
64
65             if (cint == -1)
66             {
67                 /* Clear any errors so we don't get stuck re-reading */
```

```
68              clearerr(ctlf);
69              usleep(100000);
70              break;
71          }
72
73          *(chr++) = (unsigned char)cint;
74      }
75
76      /* If there is a stop command, stop running */
77      if (memcmp("stop", line, 4) == 0)
78          running = 0;
79
80      /* Run a calibration routine if needed */
81      if (memcmp("calibrate", line, 4) == 0)
82      {
83          manager.calibrating = 1;
84          printf("CALIBRATING\n");
85          sleep(5);
86          printf("DONE\n");
87          manager.calibrating = 0;
88      }
89
90  }
91  fclose(ctlf);
92
93  /* Kill our child thread(s) */
94  xcorr_manager_kill(&manager);
95
96  /* Cleanup */
97  clean_files();
98 }
```

sample.h

```
1  #if !defined(SAMPLE_H)
2  # define SAMPLE_H
3  # include <stdio.h>
4
5  # define SAMPLE_SIZE 1024
6  # define XCORR_LEN 151
7  # define NUM_MICS 4
8  # define NUM_XCORR (NUM_MICS - 1)
9  # define MAX_PEAKS 4
10 # define SAMPLE_RATE 60000
11
12 // This is big, so avoid storing it on stack memory as much as possible :)
13 typedef struct packet packet_s;
14
15 struct packet
16 {
17     int data[NUM_MICS][SAMPLE_SIZE];
18     int xcorr[NUM_XCORR][XCORR_LEN];
19 };
20
21 int sample_packet_recv(packet_s *pkt, FILE *stream);
22
23 int sample_match_peaks(packet_s *pkt);
24
25 #endif
```

sample.c

```c
#include "sample.h"
#include "sound.h"
#include <string.h>
#include <errno.h>
#include <unistd.h>

/* Use select to wait until a stream is readable. *
 * There is a 1 second timeout on this function. *
 * It returns 1 if the stream has become readable, *
 * and 0 otherwise. */
int wait_for_file(FILE *stream)
{
    int fn;
    /* Timeout */
    struct timeval tout = { .tv_sec = 0, .tv_usec = 1000000 };
    fd_set waitfor;
    fn = fileno(stream);

    /* Set the appropriate bits in the fd_set */
    FD_ZERO(&waitfor);
    FD_SET(fn, &waitfor);

    /* Wait for the fd */
    if (select(fn + 1, &waitfor, NULL, NULL, &tout) == 1)
    {
        return 1;
    }

    return 0;
}

int sample_packet_recv(packet_s *pkt, FILE *stream)
{
    int c, n;
    size_t micnum, samplenum;
    micnum = 0;
    samplenum = 0;
    n = 0;

    // If there's clearly bullshit, run away
    while ((++n) < (100 * SAMPLE_SIZE))
    {
        // This is an experimental optimization, kill it if you want <3 - francis
        if (!wait_for_file(stream))
        {
            puts("Timed out waiting for input");
            return -1;
        }

        /* Get the next character */
        c = fgetc(stream);

        if (feof(stream))
        {
            /* Clear EOF or we'll continually read EOF chars */
            clearerr(stream);
            return -1;
        }
        else if (c == EOF)
        {
```

```
61            /* Other errors */
62            clearerr(stream);
63            printf("Error reading: %s\n", strerror(errno));
64            return -1;
65        }
66
67        /* The starting character */
68        if (c == 0xff)
69        {
70            /* It is expected as the first character */
71            if (micnum == 0 && samplenum == 0)
72                continue;
73            /* But not in other positions */
74            else
75            {
76                printf("Unexpected 0xff\n");
77                return -1;
78            }
79        }
80
81        /* Read the data point */
82        pkt->data[micnum][samplenum] = (int)c;
83        micnum += 1;
84
85        /* If we're done with a group of four mic readings, *
86         * increment the sample position. */
87        if (micnum == NUM_MICS)
88        {
89            micnum = 0;
90            samplenum += 1;
91        }
92
93        /* If we're out of samples to read, we're done with *
94         * a packet! */
95        if (samplenum == SAMPLE_SIZE)
96        {
97            return 0;
98        }
99    }
100
101   printf("No 0xff byte\n");
102   return -1;
103}
104
105int sample_match_peaks(packet_s *pkt)
106{
107   sound_s sound;
108   /* Vectors to store peaks and their amplitudes in */
109   double peaks[NUM_XCORR][MAX_PEAKS]; /* They are stored as times in seconds here */
110   int peakv[NUM_XCORR][MAX_PEAKS]; /* Amplitudes are stored here */
111   int numpeaks[NUM_XCORR];
112   int peak, xc;
113
114   /* For each cross correlation */
115   for (xc = 0; xc < NUM_XCORR; ++xc)
116   {
117       numpeaks[xc] = 0;
118       peak = -1;
119       while (numpeaks[xc] < MAX_PEAKS)
120       {
```

```
            /* Get the next peak */
            peak = xcorr_next_peak(pkt->xcorr[xc], peak);
            if (peak == -1)
                break;

            /* Convert the peak offset to a time delta */
            double dt = peak;
            dt -= XCORR_LEN / 2;
            dt /= SAMPLE_RATE;

            /* Set the peak position and value */
            peaks[xc][numpeaks[xc]] = dt;
            peakv[xc][numpeaks[xc]] = pkt->xcorr[xc][peak];

            numpeaks[xc] += 1;
        }
    }

    /* Match the sets of peaks to sounds */
    sound_match_peaks(&sound,
        peaks[0], numpeaks[0], peakv[0],
        peaks[1], numpeaks[1], peakv[1],
        peaks[2], numpeaks[2], peakv[2]
    );
}
```

sound.h

```
#if !defined(SOUND_H)
# define SOUND_H
# include "conf.h"
# include "sample.h"
# include "xcorr.h"
# include <math.h>
# include <stdint.h>
# include <sys/time.h>
# include <stdbool.h>

typedef struct sound sound_s;

struct sound
{
    double angle;
    double amplitude;
    double dt[NUM_XCORR];
};

/* 1 o----o 2
 * | |
 * 3 o----o 4
 *
 * |--> x
 * y v
 *
 * dt[0] is xcorr of 1 and 2,
 * dt[1] is xcorr of 1 and 3,
 * dt[2] is xcorr of 1 and 4
 */

#define SOUND_DT_X1(s) (s->dt[0])
#define SOUND_DT_X2(s) (s->dt[2] - s->dt[1])
```

```
34 #define SOUND_DT_Y1(s) (s->dt[1])
35 #define SOUND_DT_Y2(s) (s->dt[2] - s->dt[0])
36
37
38 /* Get the average delay of the sound in the y direction as it passes *
39  * the mics. */
40 static inline float get_sound_dy(sound_s *sound)
41 {
42     return (SOUND_DT_Y1(sound) + SOUND_DT_Y2(sound)) / 2.0;
43 }
44
45 /* Get the average delay of the sound in the x direction as it passes *
46  * the mics. */
47 static inline float get_sound_dx(sound_s *sound)
48 {
49     return (SOUND_DT_X1(sound) + SOUND_DT_X2(sound)) / 2.0;
50 }
51
52 /* Get the error in the sound. This is how far the sound deviates *
53  * from the expected uniform x velocity and uniform y velocity. *
54  * Large values mean either the sound is close, or that this is *
55  * not a sound. */
56 static inline float get_sound_error(sound_s *sound)
57 {
58     double x1, x2, xerr;
59     double y1, y2, yerr;
60
61     x1 = SOUND_DT_X1(sound);
62     x2 = SOUND_DT_X2(sound);
63     y1 = SOUND_DT_Y1(sound);
64     y2 = SOUND_DT_Y2(sound);
65
66     xerr = fabs((x1 - x2));
67     yerr = fabs((y1 - y2));
68
69     return xerr + yerr;
70 }
71 /* Get the angle of the sound from -pi to +pi */
72 static inline float get_sound_angle(sound_s *sound)
73 {
74     return atan2(get_sound_dy(sound), get_sound_dx(sound));
75 }
76
77 /* Estimate the speed of the sound in m/s */
78 static inline float get_sound_speed(sound_s *sound)
79 {
80     /* The distance between the pairs of mics */
81     float mic_dist = 0.2;
82     return mic_dist/sqrt(pow(get_sound_dx(sound), 2) + pow(get_sound_dy(sound), 2));
83 }
84
85 void sound_print(sound_s *sound, FILE *stream);
86
87 bool sound_verify(sound_s *sound);
88
89 bool sound_init(sound_s *sound, double dt0, double dt1, double dt2, int v );
90
91 bool sound_match_peaks(
92     sound_s *sound,
93     double *dt0, int ndt0, int *v0,
```

```
    double *dt1, int ndt1, int *v1,
    double *dt2, int ndt2, int *v2);

#endif
```

sound.c

```
#include <stdio.h>
#include <unistd.h>
#include <sys/stat.h>
#include <sys/time.h>
#include <sys/types.h>
#include <sys/select.h>
#include "sound.h"

/* Get the current time since the epoch */
static uint64_t get_time_ms()
{
    uint64_t rtn;
    struct timeval tv;

    gettimeofday(&tv, NULL);

    rtn = 1000 * tv.tv_sec;
    rtn += tv.tv_usec / 1000;

    return rtn;
}

/* Dump the JSON representing a sound to a file */
void sound_print(sound_s *sound, FILE *stream)
{
    static int id = 1;
    int nchrs = 1024;
    char buf[nchrs];
    char *ptr = &buf[0];
    char *end = &buf[nchrs];
    uint64_t time;

    ptr += snprintf(ptr, end - ptr, "{\"id\": %d, ", id++);
    ptr += snprintf(ptr, end - ptr, "\"angle\": %f, ", sound->angle);
    ptr += snprintf(ptr, end - ptr, "\"amplitude\": %f, ", sound->amplitude);
    ptr += snprintf(ptr, end - ptr, "\"freq\": null, ");
    ptr += snprintf(ptr, end - ptr, "\"speed\": %f, ", get_sound_speed(sound));
    ptr += snprintf(ptr, end - ptr, "\"error\": %f, ", get_sound_error(sound));
    ptr += snprintf(ptr, end - ptr, "\"time\": %ld }\n", get_time_ms());

    if (ptr >= end)
        return;

    fwrite(buf, 1, ptr - buf, stream);
}

/* Truncate a long file down to size when it gets too long */
FILE *sound_trim_file(const char *fname)
{
    FILE *filein;
    FILE *fileout;

    const int maxsize = 4096, trimsize = 1024;

```

```
55      char file[trimsize];
56      struct stat status;
57      char *end, *iter;
58
59      /* If the file doesn't exist, return null */
60      if (access(fname, F_OK))
61          return NULL;
62
63      /* If stat doesn't run, return null */
64      if (stat(fname, &status))
65          return NULL;
66
67      /* If the file isn't long enough to truncate, return null */
68      if (status.st_size <= maxsize)
69          return NULL;
70
71      /* Get the file to truncate and seek to the earliest byte *
72       * that might be preserved. */
73      filein = fopen(fname, "r");
74      fseek(filein, status.st_size - trimsize -1, SEEK_SET);
75      /* Read the remainder of the file to a buffer */
76      fread(file, 1, trimsize, filein);
77      fclose(filein);
78
79      /* Iterate along the buffer until the first newline *
80       * (we need to truncate along newlines, which is why *
81       * we write back the final 1024 bytes, at the first *
82       * newline. */
83      iter = &file[0];
84      end = &file[trimsize - 1];
85      while (iter < end)
86      {
87          if (*(++iter) == '\n')
88          {
89              fileout = fopen(fname, "w");
90              fwrite(iter + 1, 1, iter - end, fileout);
91              return fileout;
92          }
93      }
94
95      return NULL;
96  }
97
98  /* Open the file where we write sounds */
99  FILE *sound_get_file(void)
100 {
101     FILE *rtn;
102
103     rtn = sound_trim_file(CONF_SOUND);
104     if (!rtn)
105         return fopen(CONF_SOUND, "a");
106
107     return rtn;
108 }
109
110 /* Verify whether a sound could exist. This is used to *
111  * ignore sounds which aren't legitimate. */
112 bool sound_verify(sound_s *sound)
113 {
114     double speed = get_sound_speed(sound);
```

```
115     double error = get_sound_error(sound);
116 
117     return (error < 0.2e-3) && (speed > 300.0) && (speed < 450.0);
118 }
119 
120 /* Initialize a sound from a set of delta times between microphones *
121  * if the sound is verified, true is returned. Otherwise, false. */
122 bool sound_init(sound_s *sound, double dt0, double dt1, double dt2, int v)
123 {
124     sound->dt[0] = dt0;
125     sound->dt[1] = dt1;
126     sound->dt[2] = dt2;
127 
128     if (!sound_verify(sound))
129         return false;
130 
131     sound->angle = get_sound_angle(sound);
132     sound->amplitude = v;
133 
134     return true;
135 }
136 
137 bool sound_match_peaks(
138     sound_s *sound,
139     double *dt0, int ndt0, int *v0,
140     double *dt1, int ndt1, int *v1,
141     double *dt2, int ndt2, int *v2)
142 {
143     int i0, i1, i2;
144     FILE *f;
145 
146     f = sound_get_file();
147 
148     for (i0 = 0; i0 < ndt0; ++i0)
149     for (i1 = 0; i1 < ndt1; ++i1)
150     for (i2 = 0; i2 < ndt2; ++i2)
151     {
152         sound_s sound;
153         if (sound_init(
154             &sound,
155             dt0[i0], dt1[i1], dt2[i2],
156             v0[i0] + v1[i1] + v2[i2]))
157         {
158             sound_print(&sound, f);
159         }
160     }
161 
162     fclose(f);
163 }
```

xcorr.h

```
1  #if !defined(XCORR_H)
2  # define XCORR_H
3  # include <pthread.h>
4  # include "sample.h"
5  # include "conf.h"
6  
7  #define PEAK_X_THRESHOLD 20
8  #define PEAK_Y_THRESHOLD 0
9  
```

```
10 typedef struct xcorr_job xcorr_job_s;
11 typedef struct xcorr_manager xcorr_manager_s;
12
13 struct xcorr_job
14 {
15     pthread_t thread;
16
17     int running;
18     pthread_cond_t launch;
19     pthread_mutex_t launch_mtx;
20     pthread_cond_t done;
21     pthread_mutex_t done_mtx;
22
23     int *a, *b;
24     int *res;
25 };
26
27 struct xcorr_manager
28 {
29     int running;
30     pthread_t thread;
31     int calibrating;
32     int calibratingstarted;
33
34     int calib[NUM_XCORR][XCORR_LEN];
35     int ncalib;
36     xcorr_job_s workers[NUM_XCORR];
37     packet_s *packet;
38 };
39
40 void xcorr_manager_init(xcorr_manager_s *manager);
41 void xcorr_manager_kill(xcorr_manager_s *manager);
42 int xcorr_next_peak(int *vals, int prev);
43
44 #endif
```

xcorr.c

```
1  #include <string.h>
2  #include "fft/wrap.h"
3  #include "xcorr.h"
4  #include "sample.h"
5  #include <stdlib.h>
6  #include <stdio.h>
7  #include <unistd.h>
8  #include <sys/stat.h>
9  #include <math.h>
10
11 /* This is the file where our program spends most of its time *
12  * it is where all the management of threads takes place */
13
14 /* The threading model is simple, three child threads are launched *
15  * by one manager thread, and associated with a pair of microphones *
16  * to crosscorrelate. Condition locks are used by all the threads *
17  * to wait for their manager to signal a go, and then are used by *
18  * the manager to wait for each thread to finish. The manager *
19  * meanwhile does an FFT of the data. */
20 static void xcorr_job_init(xcorr_job_s *job, int *a, int *b, int *res);
21 static void xcorr_job_kill(xcorr_job_s *job);
22 static void *xcorr_job_main(void *arg);
23 static void xcorr_job_wait(xcorr_job_s *job);
```

```
24
25 static void *xcorr_manager_main(void *arg);
26
27 /* Wait for a job to finish */
28 static void xcorr_job_wait(xcorr_job_s *job)
29 {
30     pthread_cond_wait(&(job->done), &(job->done_mtx));
31 }
32
33 /* Initialize a job */
34 static void xcorr_job_init(xcorr_job_s *job, int *a, int *b, int *res)
35 {
36     job->a = a;
37     job->b = b;
38     job->res = res;
39     job->running = 1;
40
41     /* Start the pair of condition locks */
42     pthread_mutex_init(&(job->launch_mtx), NULL);
43     pthread_cond_init(&(job->launch), NULL);
44     pthread_mutex_init(&(job->done_mtx), NULL);
45     pthread_cond_init(&(job->done), NULL);
46
47     /* Lock the done mutex before setting the thread, that way *
48      * we can wait for it to send a done condition when it is *
49      * initialized. */
50     pthread_mutex_lock(&(job->done_mtx));
51
52     pthread_create(&(job->thread), NULL, xcorr_job_main, job);
53
54     xcorr_job_wait(job);
55 }
56
57 #define MAX(a, b) ((a > b) ? a : b)
58 #define MIN(a, b) ((a < b) ? a : b)
59
60 /* Normalize a sample to be zero average */
61 static void xcorr_norm(int *a)
62 {
63     int ind;
64     double avg;
65     avg = 0.0;
66     for (ind = 0; ind < SAMPLE_SIZE; ++ind)
67     {
68         avg += a[ind];
69     }
70
71     avg /= SAMPLE_SIZE;
72
73     for (ind = 0; ind < SAMPLE_SIZE; ++ind)
74     {
75         a[ind] -= avg;
76     }
77 }
78
79 /* Do a cross-correlation */
80 static void xcorr(int *a, int *b, int *res)
81 {
82     int offset, offind, ind;
83
```

```
84      /* To keep the cross correlation flat, we do each offset with the *
85       * same number of samples. */
86      for (offind = 0; offind < XCORR_LEN; ++offind)
87      {
88          int sum;
89          sum = 0;
90          offset = offind - (XCORR_LEN / 2);
91
92          for (ind = (XCORR_LEN / 2) - offset; ind < SAMPLE_SIZE - (XCORR_LEN / 2) - offset;
                ++ind)
93          {
94              sum += a[ind] * b[ind + offset];
95          }
96          res[offind] = sum;
97      }
98  }
99
100 /* Launch a job */
101 static void xcorr_job_launch(xcorr_job_s *job)
102 {
103     pthread_mutex_lock(&(job->launch_mtx));
104     pthread_cond_signal(&(job->launch));
105     pthread_mutex_unlock(&(job->launch_mtx));
106 }
107
108 /* Kill a job */
109 static void xcorr_job_kill(xcorr_job_s *job)
110 {
111     job->running = 0;
112     /* It must be launched first, so that it isn't blocked on its condition */
113     xcorr_job_launch(job);
114     pthread_join(job->thread, NULL);
115 }
116
117 static void *xcorr_job_main(void *arg)
118 {
119     xcorr_job_s *job;
120     job = arg;
121
122     /* Start by locking the launch mutex */
123     pthread_mutex_lock(&(job->launch_mtx));
124
125     /* Signal we are initialized. We must do this after locking launch, *
126      * to avoid race conditions! */
127     pthread_mutex_lock(&(job->done_mtx));
128     pthread_cond_signal(&(job->done));
129     pthread_mutex_unlock(&(job->done_mtx));
130
131     while (job->running)
132     {
133         /* Wait to be launched */
134         pthread_cond_wait(&(job->launch), &(job->launch_mtx));
135
136         /* If we're no longer alive, die */
137         if (!job->running)
138             break;
139
140         /* Do our job */
141         xcorr(job->a, job->b, job->res);
142
```

```
143         /* Signal that we are done now! */
144         pthread_mutex_lock(&(job->done_mtx));
145         pthread_cond_signal(&(job->done));
146         pthread_mutex_unlock(&(job->done_mtx));
147     }
148     puts("DONE");
149     return NULL;
150 }
151 
152 /* Initialize the manager thread */
153 void xcorr_manager_init(xcorr_manager_s *job)
154 {
155     job->running = 1;
156     job->packet = malloc(sizeof(packet_s));
157 
158     pthread_create(&(job->thread), NULL, xcorr_manager_main, job);
159 }
160 
161 /* Kill the manager thread */
162 void xcorr_manager_kill(xcorr_manager_s *job)
163 {
164     job->running = 0;
165 
166     pthread_join(job->thread, NULL);
167 }
168 
169 /* Calculate an fft and send it to file */
170 void dft_to_file(int *in)
171 {
172     int i;
173     FILE *stream;
174 
175     double reals[DFT_OUT_LEN];
176     double imags[DFT_OUT_LEN];
177 
178     dft_wrap(in, reals, imags);
179 
180     stream = fopen(CONF_FFT, "w");
181     fprintf(stream, "{\"fft\": {\n");
182 
183     for (i = 0; i < DFT_OUT_LEN; ++i)
184     {
185         if (i) fprintf(stream, ",\n");
186         fprintf(stream, "    %.2f: %.2f",
187             (i + 1) * (DFT_MAX_FREQ/DFT_OUT_LEN),
188             sqrt(reals[i] * reals[i] + imags[i] * imags[i])
189         );
190     }
191 
192     fprintf(stream, "\n}}\n");
193     fclose(stream);
194 }
195 
196 /* The main thread for the manager thread */
197 static void *xcorr_manager_main(void *arg)
198 {
199     packet_s *pkt;
200     FILE *f;
201     xcorr_manager_s *job;
202     xcorr_job_s *workers;
```

```
203     int njob;
204
205     job = arg;
206     workers = job->workers;
207     pkt = job->packet;
208
209     /* This is set when the last xcorr was part of a calibration */
210     job->calibratingstarted = 0;
211     /* This is set by the main thread */
212     job->calibrating = 0;
213
214     memset(job->calib, 0, sizeof(job->calib));
215     job->ncalib = 1;
216
217     // When this is working fully, we don't need to mkfifo!
218     //mkfifo("/tmp/chinchilla-serial", 0666);
219     f = fopen(, "r");
220
221     /* Initialize the xcorrelation workers */
222     for (njob = 0; njob < NUM_XCORR; ++njob)
223         xcorr_job_init(
224             &(workers[njob]),
225             pkt->data[0], pkt->data[1 + njob], pkt->xcorr[njob]
226         );
227
228     while (job->running)
229     {
230         int ind;
231
232         if (sample_packet_recv(pkt, f) != 0)
233         {
234             usleep(100000);
235             continue;
236         }
237
238         for (ind = 0; ind < NUM_MICS; ++ind)
239             xcorr_norm(pkt->data[ind]);
240
241         for (njob = 0; njob < NUM_XCORR; ++njob)
242             xcorr_job_launch(&(workers[njob]));
243
244         dft_to_file(pkt->data[0]);
245
246         for (njob = 0; njob < NUM_XCORR; ++njob)
247             xcorr_job_wait(&(workers[njob]));
248
249         // This is the case where calibrating has just been started.
250         if (job->calibrating && !job->calibratingstarted)
251         {
252             puts("Starting calibration");
253             job->calibratingstarted = 1;
254             job->ncalib = 0;
255             memset(job->calib, 0, sizeof(job->calib));
256             // This the ongoing case is run
257         }
258
259         // This is the case where calibrating is ongoing
260         if (job->calibrating && job->calibratingstarted)
261         {
262             int xc;
```

```
263             for (xc = 0; xc < NUM_XCORR; ++xc)
264             {
265                 int ind;
266                 for (ind = 0; ind < XCORR_LEN; ++ind)
267                 {
268                     job->calib[xc][ind] += pkt->xcorr[xc][ind];
269                 }
270             }
271             job->ncalib += 1;
272         }
273         // This is the case where calibrating has just stopped
274         else if (!job->calibrating && job->calibratingstarted)
275         {
276             if (job->ncalib)
277             {
278                 puts("Ending calibration");
279                 int xc;
280                 for (xc = 0; xc < NUM_XCORR; ++xc)
281                 {
282                     int ind;
283                     for (ind = 0; ind < XCORR_LEN; ++ind)
284                     {
285                         job->calib[xc][ind] /= job->ncalib;
286                     }
287                 }
288                 printf("%d NCALIB\n", job->ncalib);
289             }
290             else
291             {
292                 puts("Empty calibration :( I was told to calibrate but got no data");
293             }
294             job->calibratingstarted = 0;
295             // Then the normal case is run
296         }
297 
298         // This is the normal case
299         if (!job->calibrating)
300         {
301             int xc;
302             for (xc = 0; xc < NUM_XCORR; ++xc)
303             {
304                 int ind;
305                 for (ind = 0; ind < XCORR_LEN; ++ind)
306                 {
307                     printf("%d %d %d %d\n", xc, ind, job->calib[xc][ind],
308 pkt->xcorr[xc][ind]);
309                     pkt->xcorr[xc][ind] -= job->calib[xc][ind];
310                 }
311             }
312             sample_match_peaks(pkt);
313         }
314     }
315 
316     for (njob = 0; njob < NUM_XCORR; ++njob)
317         xcorr_job_kill(&(workers[njob]));
318 
319     fclose(f);
320 
321     return NULL;
322 }
```

```
323
324 int xcorr_next_peak(int *vals, int prev)
325 {
326     int peak, off;
327     if (prev != -1)
328         peak = prev + PEAK_X_THRESHOLD;
329     else
330         peak = PEAK_X_THRESHOLD;
331
332     while (peak < XCORR_LEN - PEAK_X_THRESHOLD)
333     {
334         // Iterate forward and see if the current peak is
335         // a maximum forwards
336         for (off = 0; off < PEAK_X_THRESHOLD; ++off)
337         {
338             if (vals[peak] < vals[peak + off])
339                 break;
340         }
341
342         // If there is no larger peak forwards
343         if (off == PEAK_X_THRESHOLD)
344         {
345             // Iterate backwards and see if the peak is the maximum
346             // looking backwards
347             for (off = 0; off > -PEAK_X_THRESHOLD; --off)
348             {
349                 if (vals[peak] < vals[peak + off])
350                     break;
351             }
352
353             // If it is, and it is over the Y threshold, it is a peak
354             if (off == -PEAK_X_THRESHOLD && vals[peak] > PEAK_Y_THRESHOLD)
355                 return peak;
356             else
357                 peak += PEAK_X_THRESHOLD;
358         }
359         else
360         {
361             peak += off;
362         }
363     }
364     return -1;
365 }
```

DFT.h

```
1  #include <iostream>
2  #include <cmath>
3  #include <vector>
4  #include <iomanip>
5
6  using namespace std;
7
8  vector<double> c_dft_re(const vector<double> &dec_in); //compute DTF real part
9
10 vector<double> c_dft_im(const vector<double> &dec_in); //compute DTF imaginary part
11
12 vector<double> i_dft(const vector<double> &re_freq, const vector<double> &im_freq); //
       compute inverse DTF
```

DFT.cpp

```
#include "DFT.h"

using namespace std;
constexpr double PI = 3.14159265358979323846;

vector<double> c_dft_re(const vector<double> &dec_in) //compute Discrete Fourier Transform, real part only, by using the vector that have been passed
{
	vector<double> re_freq_temp; //temp vector

	for (int i = 0; i < dec_in.size(); i++)
	{
		re_freq_temp.push_back(0); //allocate memory
	}

	//compute Discrete Fourier Transform
	for (int i = 0; i < dec_in.size(); i++)
	{
		for (int j = 0; j < dec_in.size(); j++)
		{
			re_freq_temp[i] += dec_in[j] * cos( (2 * PI*i*j) / dec_in.size());
		}

		cout << setprecision(6) << "re_freq_temp " << i << "is: " << re_freq_temp[i] << endl; //cout for display data and checking
	}

	cout << endl;

	return re_freq_temp; //return DFT_re
}

vector<double> c_dft_im(const vector<double> &dec_in) //compute Discrete Fourier Transform, imaginary part only, by using the vector that have been passed
{
	vector<double> im_freq_temp; //temp vector

	for (int i = 0; i < dec_in.size(); i++)
	{
		im_freq_temp.push_back(0); //allocate memory
	}

	//compute Discrete Fourier Transform
	for (int i = 0; i < dec_in.size(); i++)
	{
		for (int j = 0; j < dec_in.size(); j++)
		{
			im_freq_temp[i] += -dec_in[j] * sin( (2 * PI*i*j) / dec_in.size());
		}

		cout << setprecision(6) << "im_freq_temp " << i << "is: " << im_freq_temp[i] << "j" << endl; //cout for display data and checking
	}

	cout << endl;

	return im_freq_temp; //return DFT_im
}

vector<double> i_dft(const vector<double> &re_freq,const vector<double> &im_freq) //compute
```

```
     inverse Discrete Fourier Transform, by using the vector of DFT_re & DFT_im that have
    been passed
{

       int vec_size;

       //store the biggest size of vector
       if (re_freq.size() >= im_freq.size())
       {
              vec_size = re_freq.size();
       }
       else
       {
              vec_size = im_freq.size();
       }

       //temp vector
       vector<double> re_freq_temp;
       vector<double> im_freq_temp;
       vector<double> i_dft_temp;

       re_freq_temp = re_freq;
       im_freq_temp = im_freq;

       //allocate memory
       for (int i = 0; i < vec_size; i++)
       {
              i_dft_temp.push_back(0);
       }

       //compute inverse Discrete Fourier Transform
       for (int j = 0; j < vec_size; j++)
       {
              for (int i = 0; i < vec_size; i++)
              {
                     i_dft_temp[j] += re_freq_temp[i] * cos( (2 * PI*i*j) / vec_size);
                     i_dft_temp[j] += -im_freq_temp[i] * sin( (2 * PI*i*j) / vec_size);
              }

              i_dft_temp[j] /= vec_size;

              cout << setprecision(6) << "i_dft_temp " << j << " is: " << i_dft_temp[j] <<
                  endl; //cout for display data and checking
       }
       cout << endl;

       return i_dft_temp; //return the result of inverse DFT
}
```

backend_code.cpp

```

#include "DFT.h"
#include "x_corr.h"
#include "dw_iface.h"

using namespace std;

int main()
{
       int sample;
```

```
11
12          //test data
13          vector<double> mic_1 = { 47, 115, 87, 128, 38, 210, 35, 127, 63, 165, 61, 255, 245,
                144, 23, 80, 50, 17, 143, 156, 198, 39, 107, 82, 223, 105, 94, 199, 84, 226 };
14          vector<double> mic_2 = { 115, 87, 128, 38, 210, 35, 127, 63, 165, 61, 255, 245, 144,
                 23, 80, 50, 17, 143, 156, 198, 39, 107, 82, 223, 105, 94, 199, 84, 226, 132 };
15          vector<double> mic_3 = { 47, 115, 87, 128, 38, 210, 35, 127, 63, 165, 61, 255, 245,
                144, 23, 80, 50, 17, 143, 156, 198, 39, 107, 82, 223, 105, 94, 199, 84, 226 };
16          vector<double> mic_4 = { 63, 165, 61, 255, 245, 144, 23, 80, 50, 17, 143, 156, 198,
                39, 107, 82, 223, 105, 94, 199, 84, 226, 27, 55, 106, 111, 210, 92, 179, 243 };
17
18          //init vector
19          vector<double> dec_str_1;
20          vector<double> dec_str_2;
21
22          vector<double> dft_str_1_re;
23          vector<double> dft_str_1_im;
24          vector<double> dft_str_2_re;
25          vector<double> dft_str_2_im;
26
27          vector<double> idft_str_1;
28          vector<double> idft_str_2;
29
30          vector<double> x_corr_f;
31          vector<double> x_corr_s;
32
33          //init delay
34          int delay;
35
36          //select the data to compute
37          int mic_no1, mic_no2;
38
39          mic_no1 = select_mic();
40
41          switch (mic_no1)
42          {
43          case 1: dec_str_1 = mic_1;
44                  break;
45          case 2: dec_str_1 = mic_2;
46                  break;
47          case 3: dec_str_1 = mic_3;
48                  break;
49          case 4: dec_str_1 = mic_4;
50                  break;
51          }
52
53          mic_no2 = select_mic();
54
55          switch (mic_no2)
56          {
57          case 1: dec_str_2 = mic_1;
58                  break;
59          case 2: dec_str_2 = mic_2;
60                  break;
61          case 3: dec_str_2 = mic_3;
62                  break;
63          case 4: dec_str_2 = mic_4;
64                  break;
65          }
66
```

```
67
68          //not in used (cross_correlation using convolution)
69          //x_corr_s = x_corr(dec_str_1, dec_str_2);
70
71
72          //compute DFT
73          dft_str_1_re = c_dft_re(dec_str_1);
74          dft_str_1_im = c_dft_im(dec_str_1);
75
76          dft_str_2_re = c_dft_re(dec_str_2);
77          dft_str_2_im = c_dft_im(dec_str_2);
78
79          if (dft_str_1_re.size() >= dft_str_1_im.size())
80          {
81                  sample = dft_str_1_re.size();
82          }
83          else
84          {
85                  sample = dft_str_1_im.size();
86          }
87
88          //cout for display data and checking
89          for (int i = 0; i < sample; i++)
90          {
91                  if (dft_str_1_im[i] < 0)
92                  {
93                          cout << setprecision(6) << "dft_str " << i << " is: " << dft_str_1_re[i
                                ] << dft_str_1_im[i] << "i" << endl;
94                  }
95                  else
96                  {
97                          cout << setprecision(6) << "dft_str " << i << " is: " << dft_str_1_re[i
                                ] << "+" << dft_str_1_im[i] << "i" << endl;
98                  }
99          }
100
101         cout << endl;
102
103         //inverse DFT
104         idft_str_1 = i_dft(dft_str_1_re, dft_str_1_im);
105         idft_str_2 = i_dft(dft_str_2_re, dft_str_2_im);
106
107         //cross_correlation (using DFT)
108         x_corr_f = x_corr_dft(dec_str_1, dec_str_2);
109
110         //calculate the power of signal in dB
111         cal_amplitude(dec_str_1);
112
113         //find delay of 2 data
114         delay = delay_dft_func(x_corr_f, dec_str_1, dec_str_2);
115
116         system("pause");
117         return 0;
118 }
```

dw_iface.cpp

```
1  #include "dw_iface.h"
2
3  //select which microphone data to compute
4  int select_mic()
```

```
{
	bool check = 0;
	int mic_no = 0;
	while (check == 0)
	{
		cout << "Please enter mic number to compute: " << endl;
		cin >> mic_no;

		//check the input is valid or not, because only 4 microphones available
		if (mic_no <= 0 || mic_no >= 5)
		{
			cout << "error: selected number out of range (range:1 to 4)" << endl << endl;
		}
		else
		{
			check = 1;
		}
	}
	return mic_no; //return selected microphone number
}

//calculate the power of signal in dB
double cal_amplitude(const vector<double> &data)
{
	vector<double> data_temp = data;

	double data_avg = 0;
	double sum_sqre = 0;
	double amplitude = 0;

	//calculate the average of data
	for (int i = 0; i < data_temp.size(); i++)
	{
		data_avg += data_temp[i];
	}

	data_avg /= data_temp.size();

	//amplitude = the sum of data[i]^2 - average of data
	for (int i = 0; i < data_temp.size(); i++)
	{
		amplitude += (pow(data_temp[i], 2) - data_avg);
	}

	//calculate in dB
	amplitude = 10 * log(amplitude);

	cout << "signal power(in dB) is: " << amplitude << endl << endl;

	return amplitude;
}
```

1.3 Web

index.php

```
<!DOCTYPE html>
<html>
   <head>
      <title>D4 UI</title>
```

```
5         <link rel="stylesheet" href="style.css">
6         <?php
7             $dataJSON = file_get_contents("/tmp/chinchilla-fft");
8             $dataArray = json_decode($dataJSON,true);
9             print_r($dataArray);
10            echo $dataArray[0];
11        ?>
12        <script src="radar.js"></script>
13        <script src="log.js"></script>
14        <script type="text/javascript" src="canvasjs.min.js"></script>
15        <script type="text/javascript">
16            window.onload = function()
17            {
18                var chart = new CanvasJS.Chart("chartContainer", {
19                    interactivityEnabled: true,
20                    title: {
21                        text: "Amplitude Response"
22                    },
23                    axisX: {
24                        logarithmic: true,
25                        title: "Frequency (Hz)",
26                        minimum: 1,
27                        maximum: 10000
28                    },
29                    axisY: {
30                        title: "Magnitude (dB)"
31                    },
32                    data: [
33                        {
34                            type: "line",
35                            dataPoints: [
36                                {x: 1, y: 10},
37                                {x: 10, y: 1}
38                            ]
39                        }
40                    ]
41                });
42                chart.render();
43                var radar = new Radar(document.getElementById("ui-radar"));
44                radar.init(200);
45
46                window.setInterval(function()
47                {
48                    radar.blip(Math.random() * 2 * Math.PI, Math.random(), Math.random());
49                }, 1000);
50            }
51        </script>
52    </head>
53    <body>
54        <?php ini_set('display_errors', 'On'); error_reporting(E_ALL | E_STRICT); ?>
55            <!-- <?php phpinfo();?> -->
56        <?php
57            if($_SERVER['REQUEST_METHOD'] == "POST" and isset($_POST['restart']))
58            {
59                restartPi();
60            } else if($_SERVER['REQUEST_METHOD'] == "POST" and isset($_POST['calibrate']))
61            {
62                calibratePi();
63            }
64            function restartPi()
```

```
65          {
66              $filePath = fopen("chinchilla-reset", "w");
67              //echo $filePath;
68              //if(!$filePath) {echo "File Open failed";}
69              //echo "Writing";
70              fwrite($filePath, "reset\n");
71              //echo "Closing";
72              fclose($filePath);
73          }
74          function calibratePi() {
75              $filePath = fopen("chinchilla-reset", "w");
76              fwrite($filePath,"calibrate\n");
77              fclose($filePath);
78          }
79      ?>
80      <form action="upload.php" method="post" enctype="multipart/form-data">
81             Select firmware to upload:
82             <input type="file" name="fileToUpload" id="fileToUpload">
83             <input type="submit" value="Upload Firmware" name="submit">
84      </form>
85   <form action="index.php" method="post">
86      <input type="submit" name="restart" value="Restart Pi" />
87      <input type="submit" name="calibrate" value="Calibrate Device" />
88   </form>
89   <div id="chartContainer" style="height: 200px; width: 100%;"></div>
90      <div id="ui">
91          <div id="ui-radar" class="radar">
92          </div>
93          <div id="ui-log" class="log">
94              <table id="ui-log-table" class="log-table">
95                  <thead>
96                      <tr>
97                          <th>Angle</th>
98                          <th>Amplitude</th>
99                          <th>Speed</th>
100                         <th>Error</th>
101                     </tr>
102                 </thead>
103                 <tbody>
104                 </tbody>
105             </table>
106         </div>
107     </div>
108   </body>
109 </html>
```

upload.php

```
1  <?php
2  ini_set('display_errors', 'On');
3  error_reporting(E_ALL | E_STRICT);
4
5  $target_dir = "uploads/";
6  $target_file = $target_dir . "firmware.zip.gpg";
7
8  $goodFile = 1;
9
10 $fileType = strtolower(pathinfo($_FILES["fileToUpload"]["name"],PATHINFO_EXTENSION));
11
12 if($_FILES["fileToUpload"]["size"] > 1000000) {
13        echo nl2br("File too large, must be <1MB. \n");
```

```
14        $goodFile = 0;
15  }
16
17  if($fileType != "gpg") {
18        echo nl2br("Incorrect file type, please upload signed .zip.gpg only. \n" );
19        $goodFile = 0;
20  }
21
22  if($goodFile == 0) {
23        echo nl2br("File not uploaded. \n Redirecting...");
24  }else{
25        if(move_uploaded_file($_FILES["fileToUpload"]["tmp_name"],$target_file)) {
26              echo nl2br("The file " . basename($_FILES["fileToUpload"]["name"]). " has been
                    uploaded. \n Redirecting...");
27                //exec("fwExtract/installationScripts/install");
28        echo "Opening";
29        $filePath = fopen("chinchilla-reset","w");
30        echo $filePath;
31        //if(!$filePath) {echo "File Open failed";}
32        echo "Writing";
33        fwrite($filePath,"install\n");
34        echo "Closing";
35        fclose($filePath);
36
37        } else {
38              echo nl2br("Sorry, error uploading file, please try again. \n Redirecting...")
                  ;
39        }
40  }
41  header('refresh:5; url=index.php');
42  die();
43  ?>
```

install-daemon.sh

```
1   #!/bin/bash
2
3   fname="/var/www/html/chinchilla-reset"
4   logfile="/tmp/chinchilla-log"
5   ctlfile="/tmp/chinchilla-backend-ctl"
6
7   [[ -p $fname ]] || mkfifo $fname
8   [[ -f $logfile ]] && rm $logfile
9
10  shutdown()
11  {
12      echo SHUTTING DOWN!
13      [[ -p $ctlfile ]] && echo stop > $ctlfile
14      # Here, put code to stop all current processes
15  }
16
17  start()
18  {
19      echo STARTING!
20      sleep 10
21      # Serial channel setup
22      stty -F /dev/ttyACM0 406:0:18b4:8a30:3:1c:7f:15:4:2:64:0:11:13:1a:0:12:f
            :17:16:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0
23      /var/www/backend > /tmp/chinchilla-backend-log &
24      # Here, put code to start a new set of processes
25  }
```

```
26
27 install()
28 {
29     echo INSTALLING!
30     # Here, put code to verify an install file and install it
31     lxterminal -e echo Hello!
32     rm /var/www/html/uploads/firmware.zip
33     echo $?
34     gpg -o /var/www/html/uploads/firmware.zip -d /var/www/html/uploads/firmware.zip.gpg
35     result=$?
36     echo $result
37     if [[ $result -eq "0" ]]; then
38             echo Unzipping!
39             rm /var/www/html/fwExtract/*
40         unzip /var/www/html/uploads/firmware.zip -d /var/www/html/fwExtract 2>file
41         echo Extracting!
42         instdir=$(echo /media/pi/NODE_L432KC* | cut -d " " -f 1)
43         echo "$instdir"
44         [[ -d "$instdir" ]] && cp /var/www/html/fwExtract/*.bin "$instdir"
45 ## cp /var/www/html/fwExtract/*.bin /media/pi/NODE_L432KC
46 # find /var/www/html/fwExtract -iname '*.bin' -exec cp {} /media/pi/NODE_L432K* \;
47     else
48         echo Incorrectly signed file!
49     fi
50 }
51
52 reset()
53 {
54     echo RESTARTING PI!
55     reboot
56 }
57
58 calibrate()
59 {
60     echo Initialising calibration!
61     [[ -p $ctlfile ]] && echo calibrate > $ctlfile
62 }
63
64 sleep 5
65 install
66 sleep 10
67 start
68 while true; do
69     if read -r line < $fname; then
70         echo $line
71         case $line in
72         restart)
73             shutdown
74             start
75             ;;
76         stop)
77             shutdown
78             ;;
79         start)
80             start
81             ;;
82         install)
83             shutdown
84             install
85             start
```

```
            ;;
        reset)
            shutdown
            reset
            ;;
        calibrate)
            calibrate
            ;;
        esac
    else
        echo Sleepy
        sleep 1
    fi 2>&1 >> $logfile
done
```

log.js

```
function Log(elem)
{
    this.elem = elem;
    this.body = elem.getElementsByTagName("tbody")[0];

    this.pop_row = function()
    {
        this.body.removeChild(this.body.firstChild);
    }

    this.push_row = function(data)
    {
        var row = document.createElement("tr");
        for (str of data)
        {
            var textnode = document.createTextNode(str);
            var cell = document.createElement("td");
            cell.appendChild(textnode);
            row.appendChild(cell);
        }
        this.body.appendChild(row);
    }
}
```

radar.js

```
/* Add the styles to an element to make it a circular shape */
function circle_style(elem, radius)
{
    elem.style.borderRadius = radius.toString() + "px";
    elem.style.width = (radius * 2).toString()+"px";
    elem.style.height = (radius * 2).toString()+"px";
}

/* A radar element of the UI *
 * - elem is the DOM node representing the radar element */
function Radar(elem)
{
    this.elem = elem;

    /* Initialize the radar element *
     * - radius is the radius of the radar in pixels */
    this.init = function(radius)
    {
        this.radius = radius
```

```
20          circle_style(elem, radius);
21
22          for (var angle=0; angle < Math.PI - 0.01; angle += Math.PI / 6)
23          {
24              this.add_radial(angle);
25          }
26
27          for (var radius = 0.1; radius < 1; radius += 0.2)
28          {
29              this.add_circular(radius);
30          }
31      }
32
33      /* Make a blip appear on the radar *
34       * - angle is the angular position of the blip in radians *
35       * - radius is the distance from the origin of the blip (0.0 to 1.0) *
36       * - size is the size of the blip (0.0 to 1.0) */
37      this.blip = function(angle, radius, size)
38      {
39          var blip = new Blip(this);
40          blip.init(angle, radius, size);
41      }
42
43      /* Add a radial line to the radar display *
44       * - the angle of the line (rads) */
45      this.add_radial = function(angle)
46      {
47          var elem = document.createElement("div");
48          elem.className = "radial";
49
50          elem.style.top = this.radius.toString() + "px";
51          elem.style.width = (2 * this.radius.toString()) + "px";
52          elem.style.transform = "rotate(" + angle.toString() + "rad)";
53
54          this.elem.appendChild(elem);
55      }
56
57      /* Add a circular line to the radar display *
58       * - the radius to add the line to (0 to 1) */
59      this.add_circular = function(radius)
60      {
61          var elem = document.createElement("div");
62          elem.className = "circular";
63
64          circle_style(elem, radius * this.radius);
65          elem.style.top = ((1 - radius) * this.radius).toString() + "px";
66          elem.style.left = ((1 - radius) * this.radius).toString() + "px";
67
68          this.elem.appendChild(elem);
69      }
70  }
71
72  /* A blip on the radar *
73   * - radar is the Radar() where we want the blip */
74  function Blip(radar)
75  {
76      this.radar = radar;
77
78      /* Initialize the blip *
79       * - angle is the angular position of the blip in radians *
```

```
     * - radius is the radial position of the blip (0 to 1) *
     * - size is the size of the blip (0 to 1) */
    this.init = function(angle, radius, size)
    {
        this.elem = document.createElement("div");
        this.elem.className = "blip";
        this.radius = this.radar.radius * size / 25;

        circle_style(this.elem, this.radius);
        this.elem.style.left = this.get_xpos(angle, radius, size).toString() + "px";
        this.elem.style.top = this.get_ypos(angle, radius, size).toString() + "px";

        this.radar.elem.appendChild(this.elem);
        var self = this;

        window.setTimeout(function () { self.fade() }, 1000);
        window.setTimeout(function () { self.kill() }, 4000);

        this.elem.style.display = "block";
    }

    /* Get the offset of the blip from its parent element in pixels */
    this.get_xpos = function(angle, radius, size)
    {
        var centre = Math.cos(angle) * radius * this.radar.radius;
        return (centre - this.radius) + this.radar.radius;
    }

    /* Get the offset of the blip from its parent element in pixels */
    this.get_ypos = function(angle, radius, size)
    {
        var centre = - Math.sin(angle) * radius * this.radar.radius;
        return (centre - this.radius) + this.radar.radius;
    }

    /* Cause this blip to start fading away to nothing */
    this.fade = function()
    {
        this.elem.style.transform = "scale(0)";
    }

    /* Cause this blip to stop existing */
    this.kill = function()
    {
        this.radar.elem.removeChild(this.elem);
    }
}
```

requestor.js

```
function Requestor()
{
    this.radar = new Radar(document.getElementById("ui-radar"));
    this.log = new Log(document.getElementById("ui-log"));
    this.lastid = 0;

    this.add_row = function(data)
    {
        if (data.id <= this.lastid)
        {
            return;
```

```
        }

        this.lastid = data.id;
        this.push_row([data.angle, data.amplitude, data.speed, data.error])
    }

    this.add_blip = function(data)
    {
        this.radar.blip(data.angle, 0.5, data.amplitude);
    }

    this.on_sounds = function()
    {
        var text = this.req.responseText;
        var lines = this.text.split("\n");

        for (line of lines)
        {
            var data = JSON.parse(line);
            this.add_row(data);
            this.add_blip(data);
        }
    }

    this.request_sounds = function()
    {
        this.req = new XMLHttpRequest();
        this.req.open("GET", "/chinchilla-sounds");
        this.req.onreadystatechange = this.on_sounds;
    }
}
```

style.css

```
.radar {
    background-color: #335;
    position: relative;
}

.blip {
    background-color: #ff7;
    transition: transform 2s;
    position: absolute;
    display: none;
}

/* The radial lines on the radar */
.radar > .radial {
    background-color: #fff;
    height: 1px;
    position: absolute;
}

/* The circular lines on the radar */
.radar > .circular {
    border: 1px solid #fff;
    position: absolute;
    background-color: #335;
}

/* The first circle has a solid background to block the middle, but the *
```

```
 * others are all transparent */
.radar > .circular ~ .circular { background-color: transparent; }

.log {
       display: inline-block;
       overflow: auto;
       height: 400px;
}

.log-table th {
       padding: 10px;
       background-color: #335;
       color: white;
       font-style: bold;
}

.log-table td {
       background-color: #559;
       color: white;
       padding: 2px;
       border: 1px solid #335;
}

.log-row {
}
```

1.4 LED Control

led_ctl.py

```
# Based on example by Adafruit and using Adafruit libraries

# imports
import time
import board
import neopixel
import json
import digitalio
import ast
from math import pi

# Definitions
FAN_OUT = 3
NUM_PIXELS = 46

# ------------------------------------------------------------------------------
# Setup
#
# Setup LEDs for Adafruit library
# Setup button and declare as pull up input
# Setup memory for remembering id, time and button state between loops
# ------------------------------------------------------------------------------

# LED setup
pixel_pin = board.D18
ORDER = neopixel.GRB
pixels = neopixel.NeoPixel(pixel_pin, NUM_PIXELS, brightness=1,
                           auto_write=False, pixel_order=ORDER)

# Button setup
button = digitalio.DigitalInOut(board.D23)
```

```
32 button.direction = digitalio.Direction.INPUT
33 button.pull = digitalio.Pull.UP
34
35 # Declare memory between angles
36 last_id = 0
37 last_ms = 0
38
39 # ------------------------------------------------------------------------
40 # Loading animation on LEDs
41 # ------------------------------------------------------------------------
42 def load_screen():
43
44     # Clear all LEDs
45     for n in range(0,NUM_PIXELS):
46         pixels[n] = ((0, 0, 0))
47     pixels.show()
48
49     # Make first and last LED white
50     pixels[0] = ((255, 255, 255))
51     pixels[NUM_PIXELS - 1] = ((255, 255, 255))
52
53     # Display LED values calculated
54     pixels.show()
55
56     # Send white LED lit up round ring
57     for n in range(1, int(NUM_PIXELS / 2)):
58         time.sleep(0.1)
59
60         pixels[n - 1] = ((0, 0, 0))
61         pixels[n] = ((255, 255, 255))
62
63         pixels[NUM_PIXELS - n] = ((0, 0, 0))
64         pixels[NUM_PIXELS -1 - n] = ((255, 255, 255))
65
66         # Display LED values calculated
67         pixels.show()
68
69     # Clear all LEDs
70     for n in range(0,NUM_PIXELS):
71         pixels[n] = ((0, 0, 0))
72     pixels.show()
73
74     # Light up all LEDs in sequence with colours that range from orange to purple
75     for n in range(int(NUM_PIXELS / 2) - 1, -1, -1):
76         time.sleep(0.1)
77         pixels[n] = ((round(n * 255 / (int(NUM_PIXELS / 2) - 1)), n, round(255 - (n * 255 /
               (int(NUM_PIXELS / 2) - 1))) ))
78         pixels[NUM_PIXELS -1 - n] = ((round(n * 255 / (int(NUM_PIXELS / 2) - 1)), n, round
               (255 - (n * 255 / (int(NUM_PIXELS / 2) - 1))) ))
79
80         # Display LED values calculated
81         pixels.show()
82
83     time.sleep(1)
84
85     # Clear all LEDs
86     for n in range(0,NUM_PIXELS):
87         pixels[n] = ((0, 0, 0))
88     pixels.show()
89
```

```
# ------------------------------------------------------------------------
# Calibration animation
# ------------------------------------------------------------------------
def calibrate():
    # Clear all LEDs
    for n in range(0,NUM_PIXELS):
        pixels[n] = ((0, 0, 0))
    pixels.show()

    # Make first LED white
    pixels[0] = ((255, 255, 255))

    # Display LED values calculated
    pixels.show()

    # Send white LED lit up round ring
    for n in range(1, NUM_PIXELS):
        time.sleep(0.1)

        pixels[n - 1] = ((0, 0, 0))
        pixels[n] = ((255, 255, 255))

        # Display LED values calculated
        pixels.show()

    # Clear all LEDs
    for n in range(0,NUM_PIXELS):
        pixels[n] = ((0, 0, 0))
    pixels.show()

# ------------------------------------------------------------------------
# Sign function
#
# This outputs only the sign of a number ignoring the value
# Possible outputs are 1 and -1
# ------------------------------------------------------------------------
def sign(num):
    if num <= 0:
        return -1
    else:
        return 1

# ------------------------------------------------------------------------
# LED ring code
#
# Calculates LED RGB values based on angle and amplitude
# Fades out as it goes round the ring up to the fanout value
# ------------------------------------------------------------------------
def led_ring(angle, amplitude, freq):

    # Translate angle to LED number
    ring_pos = round((angle * NUM_PIXELS) / (2 * pi))

    # Offset from Led number found above
    ring_offset = ((angle * NUM_PIXELS) / (2 * pi)) - ring_pos

    # For loop stepping through FAN_OUT
    for n in range(1 - FAN_OUT, 1 + FAN_OUT):

        # Calculate LED index based on angle
```

```
150         index = ring_pos + (n * sign(ring_offset)) + (NUM_PIXELS *
151                 sign(-ring_pos - (n * sign(ring_offset))))
152 
153         # Extract current value of LED in question
154         pixel = list(pixels[index])
155 
156         # Calculate RGB values for LEDs incorporating:
157         # past value, amplitude and angle
158 
159         pixels[index] = (( max(0, min(255, pixel[0] + round(amplitude * 255 *
160                         max(0, ((freq / 500) - 1)) * (FAN_OUT - abs(n) +
161                         (abs(ring_offset) * sign(n)) ) / FAN_OUT ))),
162 
163                         max(0, min(255, pixel[1] + round(amplitude * 255 *
164                         (1 - abs((freq / 500) - 1)) * (FAN_OUT - abs(n) +
165                         (abs(ring_offset) * sign(n)) ) / FAN_OUT ))),
166 
167                         max(0, min(255, pixel[2] + round(amplitude * 255 *
168                         max(0, ((-freq / 500) + 1)) * (FAN_OUT - abs(n) +
169                         (abs(ring_offset) * sign(n)) ) / FAN_OUT )) )))
170 
171     # Display LED values calculated
172     pixels.show()
173 
174 # ------------------------------------------------------------------------
175 # Main Code
176 #
177 # Call setup
178 # Imports angle and amplitude value from json file, and triggers LED ring code
179 # Fades the LED values by 1/3 each loop
180 # Reads button value
181 # Sends a calibrate command if short button press
182 # Sends a reset command if long button press
183 # ------------------------------------------------------------------------
184 
185 load_screen()
186 
187 while True:
188 
189     # Fade all LEDs out by 1/3 each time
190     for i in range(0, NUM_PIXELS):
191 
192         # Convert from tuple to list for current LED
193         pixel = list(pixels[i])
194         if pixel[0] > 0:
195             pixel[0] -= max(1, round(pixel[0]/3))
196         if pixel[1] > 0:
197             pixel[1] -= max(1, round(pixel[1]/3))
198         if pixel[2] > 0:
199             pixel[2] -= max(1, round(pixel[2]/3))
200 
201         # Convert back from list to tuple for current LED
202         pixels[i] = tuple(pixel)
203 
204         # Display LED values calculated
205         pixels.show()
206 
207     # Open Json file and read id, angle and amplitude
208     with open('/tmp/chinchilla-sounds', 'r') as json_file:
209         for line in json_file.readlines():
```

```
            object = json.loads(line)
            id = object['id']
            angle = object['angle']
            amplitude = object['amplitude'] / 2000000.0

    # Import FFT data
    try:
        fft = ast.literal_eval(open('/tmp/chinchilla-fft', 'r').read())
    except:
        fft = None

    # Check for frequency with largest amplitude and use that
    if fft != None:
        largest_amp = 0
        for currfreq, ampl in fft['fft'].items():
            if ampl > largest_amp:
                freq = currfreq
                largest_amp = ampl

    # Or default frequency of 500
    else:
        freq = 500

    # Check id has increased (don't repeat same sound)
    if id > last_id:

        # Call LED ring code
        led_ring(angle, amplitude, freq)

        # Update last id memory
        last_id = id

    # Get current time in milliseconds
    now_ms = int(time.time() * 1000)

    # Get current button value
    button_value = not button.value

    # Open file, write command and close if button press is greater than 50ms (debounce)
    f = open('/tmp/backend-ctl', 'w')
    if (not button_value):
        last_ms = now_ms
    if (button_value and now_ms > last_ms + 50):
        f.write('calibrate')
    f.close()

    # Trigger calibration animation
    if (button_value and now_ms > last_ms + 50):
        calibrate()
```