The Buffer so far:

arm_main.cpp

```
#include "mbed.h"
#include <queue>
#include <vector>

#define ADC0_PIN D9
#define ADC1_PIN D10
#define ADC2_PIN D11
#define ADC3_PIN D12
#define CS_PIN D8
#define CLK_PIN D7

#define BUFFER_MAX 1024
#define DATA_BITS 13

DigitalOut cs(CS_PIN);
DigitalOut clk(D7);
BusIn adcs(D9, D10, D11, D12);
Serial serial(USBTX, USBRX);

//int sample_buffer[BUFFER_SIZE];
char serial_buffer[5]; // Includes 4 chars for sending samples, and a start packet
//int current_sample[4];
vector<int> current_sample;
queue< vector<int> > samples_buffer;

// Uses a bit-bashing method similar to SPI to read the values from all four ADCs. Then the
// values are manipulated into the correct format to be represented by a single int variable each.
// These can then be sent over serial.
// wait_us(1) is the smallest possible delay. If it turns out to be too large, this can be replaced
// with idle iterations through a for loop. This would be less exact and may need analysis on an
// oscilloscope to find frequencies.
void read_samples()
{
        int temp; // Stores ADC sample values, sample by sample, before they can be processed

        // Bit-bashing the ADCs as if SPI
        // The data only starts on the third falling edge, so put two clocks outside the loop
        cs = 0;
        //wait_us(1);
        clk = 1;
        wait_us(1);
        clk = 0;
        wait_us(1);
        clk = 1;
        wait_us(1);
        clk = 0;

        for(uint8_t i = 0; i < DATA_BITS; i++)
        {
            clk = 1;
            wait_us(1);
            clk = 0;
            temp = adcs.read();

            current_sample[0] += (temp&1) << (DATA_BITS-(i+1));
            current_sample[1] += (temp&2) << (DATA_BITS-(i+2));
            current_sample[2] += (temp&4) << (DATA_BITS-(i+3));
            current_sample[3] += (temp&8) << (DATA_BITS-(i+4));
            // Debateable need for a delay in here to maintain clock pulse - hoping
            // the above logic is slow enough to generate a delay of about 1us anyway
        }

        // The MSB is a sign bit, and should always be 0. If it isn't, the bit
        // may have been corrupted and the sample should be set to 0.
        for(uint8_t i = 0; j < 4; i++)
        {
            if(current_sample[i] >> DATA_BITS-1)
                current_sample[i] = 0;
        }

        cs = 1;
}

int main()
{
    cs = 1;
    clk = 0;

    serial.baud(115200);    // THIS NEEDS TO BE SET THE SAME ON THE PI

    // Creates four elements in the vector to put samples in
    for(uint8_t i = 0; i < 4; i++)
    {
        current_sample.push_back(0);
    }

    for(;;)
    {
        read_samples();

        samples_buffer.push(current_sample);
        if(samples_buffer.size() == BUFFER_MAX)
            send_serial();

        // current_samples needs to be reset ready for the next iteration
        for(uint8_t i = 0; i < 4; i++)
        {
            current_sample[i] = 0;
        }


    }

    return 0;
}
```

The back-end so far:

xcorr.h

```
#if !defined(XCORR_H)
# define XCORR_H
# include <pthread.h>
# include "sample.h"

# define XCORR_NUM
```

```c
typedef struct xcorr_job     xcorr_job_s;
typedef struct xcorr_manager xcorr_manager_s;

struct xcorr_job
{
    pthread_t thread;

    int *a, *b;
    int *res;
};

struct xcorr_manager
{
    int running;
    pthread_t thread;

    xcorr_job_s workers[NUM_XCORR];
    packet_s *packet;
};

void  xcorr_manager_init(xcorr_manager_s *manager);
void  xcorr_manager_kill(xcorr_manager_s *manager);

#endif
```

xcorr.c

```c
#include "xcorr.h"
#include "sample.h"
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>

static void  xcorr_job_init(xcorr_job_s *job, int *a, int *b, int *res);
static void  xcorr_job_kill(xcorr_job_s *job);
static void *xcorr_job_main(void *arg);

static void *xcorr_manager_main(void *arg);

static void xcorr_job_init(xcorr_job_s *job, int *a, int *b, int *res)
{
    job->a   = a;
    job->b   = b;
    job->res = res;

    pthread_create(&(job->thread), NULL, xcorr_job_main, job);
}

static void xcorr_job_kill(xcorr_job_s *job)
{
    pthread_join(job->thread, NULL);
}

static void *xcorr_job_main(void *arg)
{
    xcorr_job_s *job;

    job = arg;
    xcorr(job->a, job->b, job->res);

    return NULL;
}
```

```c
void xcorr_manager_init(xcorr_manager_s *job)
{
    job->running = 1;
    job->packet = malloc(sizeof(packet_s));

    pthread_create(&(job->thread), NULL, xcorr_manager_main, job);
}

void xcorr_manager_kill(xcorr_manager_s *job)
{
    job->running = 0;

    pthread_join(job->thread, NULL);
}

static void *xcorr_manager_main(void *arg)
{
    packet_s *pkt;
    xcorr_manager_s *job;
    xcorr_job_s     *workers;

    job     = arg;
    workers = job->workers;
    pkt     = job->packet;

    while (job->running)
    {
        int njob;

        if (sample_packet_recv(pkt, NULL) != 0)
        {
            usleep(100000);
            continue;
        }

        for (njob = 0; njob < NUM_XCORR; ++njob)
            xcorr_job_init(
                &(workers[njob]),
                pkt->data[0], pkt->data[1 + njob], pkt->xcorr[njob]
            );

        for (njob = 0; njob < NUM_XCORR; ++njob)
            xcorr_job_kill(&(workers[njob]));
    }

    return NULL;
}

int main(void)
{
    xcorr_manager_s m;
    xcorr_manager_init(&m);
    sleep(100);
    xcorr_manager_kill(&m);
    return 0;
}
```

sample.h

```c
#if !defined(SAMPLE_H)
# define SAMPLE_H
# include <stdio.h>

# define SAMPLE_SIZE 1024
# define XCORR_LEN   50
# define NUM_MICS    4
# define NUM_XCORR   (NUM_MICS - 1)

// This is big, so avoid storing it on stack memory as much as possible :)
typedef struct packet packet_s;
struct packet
{
    int data[NUM_MICS][SAMPLE_SIZE];
    int xcorr[NUM_XCORR][XCORR_LEN];
};

int sample_packet_recv(packet_s *pkt, FILE *stream);

#endif
```

sample.c

```c
#include "sample.h"

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)
    {
        c = fgetc(stream);
        if (c == -1)
            return -1;

        if (c != '\xff')
        {
            if (micnum == 0 && samplenum == 0)
                continue;

            else
                return -1;
        }

        pkt->data[micnum][samplenum] = (int)c;
        samplenum += 1;

        if (samplenum == SAMPLE_SIZE)
        {
            samplenum = 0;
            micnum   += 1;
        }

        if (micnum == NUM_MICS)
        {
            return 0;
        }
    }

    return -1;
}
```

sound.h

```
#if !defined(SOUND_H)
# define SOUND_H
# include "sample.h"
# include "xcorr.h"
# include <math.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])
#define SOUND_DT_Y1(s) (s->dt[1])
#define SOUND_DT_Y2(s) (s->dt[2] - s->dt[0])


/* Get the average delay of the sound in the y direction as it passes *
 * the mics.                                                          */
float get_sound_dy(sound_s *sound)
{
    return (SOUND_DT_Y1(sound) + SOUND_DT_Y2(sound)) / 2.0;
}

/* Get the average delay of the sound in the x direction as it passes *
 * the mics.                                                          */
float get_sound_dx(sound_s *sound)
{
    return (SOUND_DT_X1(sound) + SOUND_DT_X2(sound)) / 2.0;
}

/* Get the error in the sound. This is how far the sound deviates *
 * from the expected uniform x velocity and uniform y velocity.   *
 * Large values mean either the sound is close, or that this is   *
 * not a sound.                                                   */
float get_sound_error(sound_s *sound)
{
    double x1, x2, xerr;
    double y1, y2, yerr;
    x1 = SOUND_DT_X1(sound);
    x2 = SOUND_DT_X2(sound);
    y1 = SOUND_DT_Y1(sound);
    y2 = SOUND_DT_Y2(sound);

    xerr = 2.0 * fabs((x1 - x2)/get_sound_dx(sound));
    yerr = 2.0 * fabs((y1 - y2)/get_sound_dx(sound));

    return xerr + yerr;
}
/* Get the angle of the sound from -pi to +pi */
float get_sound_angle(sound_s *sound)
{
    return atan2(get_sound_dy(sound), get_sound_dx(sound));
}

/* Estimate the speed of the sound in m/s */
float get_sound_speed(sound_s *sound)
{
    /* The distance between the pairs of mics */
    float mic_dist = 0.2;
    return mic_dist/sqrt(pow(get_sound_dx(sound), 2) + pow(get_sound_dy(sound), 2));
}

#endif
```

The web framework so far:

index.php

```
<!DOCTYPE html>
<html>
    <head>
        <title>D4 UI</title>
        <link rel="stylesheet" href="style.css">
        <script src="radar.js"></script>
        <script src="log.js"></script>
        <script>
            window.onload = function()
            {
                var radar = new Radar(document.getElementById("ui-radar"));
                radar.init(200);

                window.setInterval(function()
                {
                    radar.blip(Math.random() * 2 * Math.PI, Math.random(), Math.random());
                }, 1000);
            }
        </script>
    </head>
    <body>
        <?php ini_set('display_errors', 'On'); error_reporting(E_ALL | E_STRICT); ?>
    <form action="upload.php" method="post" enctype="multipart/form-data">
            Select firmware to upload:
            <input type="file" name="fileToUpload" id="fileToUpload">
            <input type="submit" value="Upload Firmware" name="submit">
    </form>
    <div id="ui">
            <div id="ui-radar" class="radar">
            </div>
            <div id="ui-log" class="log">
                <table id="ui-log-table" class="log-table">
                    <thead>
                        <tr>
                            <th>Angle</th>
                            <th>Amplitude</th>
                            <th>&Delta;t1</th>
                            <th>&Delta;t2</th>
                            <th>&Delta;t3</th>
                        </tr>
                    </thead>
                    <tbody>
                    </tbody>
                </table>
            </div>
        </div>
    </body>
</html>
```

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
        circle_style(elem, radius);

        for (var angle=0; angle < Math.PI - 0.01; angle += Math.PI / 6)
        {
            this.add_radial(angle);
        }

        for (var radius = 0.1; radius < 1; radius += 0.2)
        {
            this.add_circular(radius);
        }
    }

    /* Make a blip appear on the radar                                     *
     *  - angle is the angular position of the blip in radians             *
     *  - radius is the distance from the origin of the blip (0.0 to 1.0) *
     *  - size is the size of the blip (0.0 to 1.0)                        */
    this.blip = function(angle, radius, size)
    {
        var blip = new Blip(this);
        blip.init(angle, radius, size);
    }

    /* Add a radial line to the radar display *
     *  - the angle of the line (rads)         */
    this.add_radial = function(angle)
    {
        var elem = document.createElement("div");
        elem.className = "radial";

        elem.style.top   = this.radius.toString() + "px";
        elem.style.width = (2 * this.radius.toString()) + "px";
        elem.style.transform = "rotate(" + angle.toString() + "rad)";

        this.elem.appendChild(elem);
    }

    /* Add a circular line to the radar display  *
     *  - the radius to add the line to (0 to 1) */
    this.add_circular = function(radius)
    {
        var elem = document.createElement("div");
        elem.className = "circular";

        circle_style(elem, radius * this.radius);
        elem.style.top  = ((1 - radius) * this.radius).toString() + "px";
        elem.style.left = ((1 - radius) * this.radius).toString() + "px";

        this.elem.appendChild(elem);
    }
}

/* A blip on the radar                            *
 *  - radar is the Radar() where we want the blip */
function Blip(radar)
{
    this.radar = radar;
    /* Initialize the blip *
     *  - 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);
    }
}
```

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 {
}
```

upload.php

```
<?php
ini_set('display_errors', 'On');
error_reporting(E_ALL | E_STRICT);

$target_dir = "uploads/";
$target_file = $target_dir . "firmware.jpg";

$goodFile = 1;

$fileType = strtolower(pathinfo($_FILES["fileToUpload"]["name"],PATHINFO_EXTENSION));

if($_FILES["fileToUpload"]["size"] > 1000000) {
    echo nl2br("File too large, must be <1MB. \n");
    $goodFile = 0;
}
```

```
if($fileType != "jpg") {
    echo nl2br("Incorrect file type, please upload .jpg only. \n" );
    $goodFile = 0;
}

if($goodFile == 0) {
    echo nl2br("File not uploaded. \n Redirecting...");
}else{
    if(move_uploaded_file($_FILES["fileToUpload"]["tmp_name"],$target_file)) {
        echo nl2br("The file " . basename($_FILES["fileToUpload"]["name"]).
                    " has been uploaded. \n Redirecting...");
    } else {
        echo nl2br("Sorry, error uploading file, please try again. \n Redirecting...");
    }
}
header('refresh:5; url=index.php');
die();
?>
```

LED Controller so far:

Version1.py

```
# Based on example by adafruit

import time
import board
import neopixel
import json

# Neopixel setup
pixel_pin = board.D18
num_pixels = 25
ORDER = neopixel.GRB
pixels = neopixel.NeoPixel(pixel_pin, num_pixels, brightness=0.2, auto_write=False, pixel_order=ORDER)

# Define pi
pi = 3.14159265359

# Declare Colour memory
red = 1
green = 0
blue = 0

# Declare position and offset on LED ring and offset signedness
ring_pos = 0
ring_offset = 0
offset_sign = 0

# Led ring code
def led_ring(angle, amplitude):
    ring_pos = round((angle * num_pixels) / (2 * pi))
    ring_offset = ((angle * num_pixels) / (2 * pi)) - ring_pos

    if ring_offset < 0:
        offset_sign = -1
    elif ring_offset > 0:
        offset_sign = 1
    else:
        offset_sign = 0
    ring_offset = abs(ring_offset)

    for i in range(0, 256):
        pixles[ring_pos + (-2 * offset_sign)] = (( (round(i * amplitude * red * (1 - ring_offset) / 3 )),
                                                   (round(i * amplitude * green * (1 - ring_offset) / 3 )),
                                                   (round(i * amplitude * blue * (1 - ring_offset) / 3 )) ))
        pixles[ring_pos + (-1 * offset_sign)] = (( (round(i * amplitude * red * (2 - ring_offset) / 3 )),
                                                   (round(i * amplitude * green * (2 - ring_offset) / 3 )),
                                                   (round(i * amplitude * blue * (2 - ring_offset) / 3 )) ))
        pixels[ring_pos + ( 0 * offset_sign)] = (( (round(i * amplitude * red * (3 - ring_offset) / 3 )),
                                                   (round(i * amplitude * green * (3 - ring_offset) / 3 )),
                                                   (round(i * amplitude * blue * (3 - ring_offset) / 3 )) ))
        pixels[ring_pos + ( 1 * offset_sign)] = (( (round(i * amplitude * red * (2 + ring_offset) / 3 )),
                                                   (round(i * amplitude * green * (2 + ring_offset) / 3 )),
                                                   (round(i * amplitude * blue * (2 + ring_offset) / 3 )) ))
        pixles[ring_pos + ( 2 * offset_sign)] = (( (round(i * amplitude * red * (1 + ring_offset) / 3 )),
                                                   (round(i * amplitude * green * (1 + ring_offset) / 3 )),
                                                   (round(i * amplitude * blue * (1 + ring_offset) / 3 )) ))
        pixles[ring_pos + ( 3 * offset_sign)] = (( (round(i * amplitude * red * (0 + ring_offset) / 3 )),
                                                   (round(i * amplitude * green * (0 + ring_offset) / 3 )),
                                                   (round(i * amplitude * blue * (0 + ring_offset) / 3 )) ))

        pixels.show()
        time.sleep(0.002)

        if red == 1:
            red = 0
            green = 1
            blue = 0
        elif green == 1:
            red = 0
            green = 0
            blue = 1
        else:
            red = 1
            green = 0
            blue = 0

# Main loop
while True:
        with open('/tmp/chinchilla-backend.json', 'r') as json_file:
                data = json.load(json_file)
```