ece420_main.cpp

```
//
// Created by daran on 1/12/2017 to be used in ECE420 Sp17 for the first time.
// Modified by dwang49 on 1/1/2018 to adapt to Android 7.0 and Shield Tablet updates.
//

#include "ece420_main.h"

// Student Variables
#define FRAME_SIZE 128

// FIR Filter Function Defined here located at the bottom
int16_t firFilter(int16_t sample);

void ece420ProcessFrame(sample_buf *dataBuf) {
    // Keep in mind, we only have a small amount of time to process each buffer!
    struct timeval start;
    gettimeofday(&start, NULL);

    // Using {} initializes all values in the array to zero
    int16_t bufferIn[FRAME_SIZE] = {};
    int16_t bufferOut[FRAME_SIZE] = {};

    // Your buffer conversion (unpacking) here
    // Fetch data sample from dataBuf->buf_[], unpack and put into bufferIn[]
    // ******************** START YOUR CODE HERE ******************** //
    for (int i = 0; i < FRAME_SIZE; i++) {
        // PCM-16 data is stored as 2 bytes per sample (little-endian)
        int16_t sample = (int16_t)((dataBuf->buf_[2*i+1] << 8) | dataBuf->buf_[2*i]);
        bufferIn[i] = sample;
    }



    // ********************** END YOUR CODE HERE ********************** //

    // Loop code provided as a suggestion. This loop simulates sample-by-sample processing.
    for (int sampleIdx = 0; sampleIdx < FRAME_SIZE; sampleIdx++) {
        // Grab one sample from bufferIn[]
        int16_t sample = bufferIn[sampleIdx];
        // Call your filFilter funcion
```

```
        int16_t output = firFilter(sample);
        // Grab result and put into bufferOut[]
        bufferOut[sampleIdx] = output;
    }

    // Your buffer conversion (packing) here
    // Fetch data sample from bufferOut[], pack them and put back into dataBuf->buf_[]
    // ******************** START YOUR CODE HERE ******************** //
    for (int i = 0; i < FRAME_SIZE; i++) {
        // Convert int16_t back to PCM-16 (little-endian, 2 bytes per sample)
        dataBuf->buf_[2*i] = (uint8_t)(bufferOut[i] & 0xFF);        // Low byte
        dataBuf->buf_[2*i+1] = (uint8_t)((bufferOut[i] >> 8) & 0xFF); // High byte
    }




    // ********************** END YOUR CODE HERE ********************** //

    // Log the processing time to Android Monitor or Logcat window at the bottom
    struct timeval end;
    gettimeofday(&end, NULL);
    LOGD("Loop timer: %ld us",  ((end.tv_sec * 1000000 + end.tv_usec) - (start.tv_sec * 1000000
+ start.tv_usec)));

}

// TODO: Change N_TAPS to match your filter design
#define N_TAPS 201
// TODO: Change myfilter to contain the coefficients of your designed filter.
static const float myfilter[N_TAPS] = {
    0.0007161076f, 0.0006855612f, 0.0005778299f, 0.0004070146f, 0.0001940047f,
-0.0000350645f, -0.0002509304f, -0.0004237416f,
    -0.0005258082f, -0.0005343220f, -0.0004337959f, -0.0002179783f, 0.0001089653f,
0.0005321568f, 0.0010266771f, 0.0015591997f,
    0.0020904938f, 0.0025786161f, 0.0029825392f, 0.0032659066f, 0.0034005773f,
0.0033696204f, 0.0031694543f, 0.0028108817f,
    0.0023188647f, 0.0017309846f, 0.0010946544f, 0.0004632652f, -0.0001084407f,
-0.0005693924f, -0.0008771715f, -0.0010024938f,
    -0.0009327545f, -0.0006742553f, -0.0002528345f, 0.0002872551f, 0.0008862226f,
0.0014732184f, 0.0019721081f, 0.0023081022f,
    0.0024146992f, 0.0022403411f, 0.0017541718f, 0.0009503323f, -0.0001496750f,
-0.0014968775f, -0.0030171242f, -0.0046157195f,
```

```
    -0.0061841328f, -0.0076083208f, -0.0087780325f, -0.0095963465f, -0.0099886264f,
-0.0099100863f, -0.0093512311f, -0.0083405785f,
    -0.0069442699f, -0.0052624204f, -0.0034223298f, -0.0015689476f, 0.0001467567f,
0.0015806552f, 0.0026085388f, 0.0031383177f,
    0.0031202653f, 0.0025543255f, 0.0014936926f, 0.0000441332f, -0.0016411582f,
-0.0033710307f, -0.0049296744f, -0.0060929075f,
    -0.0066463343f, -0.0064040422f, -0.0052263955f, -0.0030354755f, 0.0001731726f,
0.0043236087f, 0.0092581923f, 0.0147423192f,
    0.0204751811f, 0.0261060723f, 0.0312553329f, 0.0355386332f, 0.0385930041f,
0.0401028268f, 0.0398239292f, 0.0376040082f,
    0.0333977982f, 0.0272757339f, 0.0194252782f, 0.0101445821f, -0.0001713206f,
-0.0110510586f, -0.0219729910f, -0.0323951736f,
    -0.0417873011f, -0.0496625344f, -0.0556071025f, -0.0593056982f, 0.9394390433f,
-0.0593056982f, -0.0556071025f, -0.0496625344f,
    -0.0417873011f, -0.0323951736f, -0.0219729910f, -0.0110510586f, -0.0001713206f,
0.0101445821f, 0.0194252782f, 0.0272757339f,
    0.0333977982f, 0.0376040082f, 0.0398239292f, 0.0401028268f, 0.0385930041f,
0.0355386332f, 0.0312553329f, 0.0261060723f,
    0.0204751811f, 0.0147423192f, 0.0092581923f, 0.0043236087f, 0.0001731726f,
-0.0030354755f, -0.0052263955f, -0.0064040422f,
    -0.0066463343f, -0.0060929075f, -0.0049296744f, -0.0033710307f, -0.0016411582f,
0.0000441332f, 0.0014936926f, 0.0025543255f,
    0.0031202653f, 0.0031383177f, 0.0026085388f, 0.0015806552f, 0.0001467567f,
-0.0015689476f, -0.0034223298f, -0.0052624204f,
    -0.0069442699f, -0.0083405785f, -0.0093512311f, -0.0099100863f, -0.0099886264f,
-0.0095963465f, -0.0087780325f, -0.0076083208f,
    -0.0061841328f, -0.0046157195f, -0.0030171242f, -0.0014968775f, -0.0001496750f,
0.0009503323f, 0.0017541718f, 0.0022403411f,
    0.0024146992f, 0.0023081022f, 0.0019721081f, 0.0014732184f, 0.0008862226f,
0.0002872551f, -0.0002528345f, -0.0006742553f,
    -0.0009327545f, -0.0010024938f, -0.0008771715f, -0.0005693924f, -0.0001084407f,
0.0004632652f, 0.0010946544f, 0.0017309846f,
    0.0023188647f, 0.0028108817f, 0.0031694543f, 0.0033696204f, 0.0034005773f,
0.0032659066f, 0.0029825392f, 0.0025786161f,
    0.0020904938f, 0.0015591997f, 0.0010266771f, 0.0005321568f, 0.0001089653f,
-0.0002179783f, -0.0004337959f, -0.0005343220f,
    -0.0005258082f, -0.0004237416f, -0.0002509304f, -0.0000350645f, 0.0001940047f,
0.0004070146f, 0.0005778299f, 0.0006855612f,
    0.0007161076f
};

// FirFilter Function
//int16_t firFilter(int16_t sample) {
// This function simulates sample-by-sample processing. Here you will
```

```
// implement an FIR filter such as:
//
// y[n] = a x[n] + b x[n-1] + c x[n-2] + ...
//
// You will maintain a circular buffer to store your prior samples
// x[n-1], x[n-2], ..., x[n-k]. Suggested initializations circBuf
// and circBufIdx are given.
//
// Input 'sample' is the current sample x[n].
// ******************** START YOUR CODE HERE ******************** //


int16_t      circBuf[N_TAPS] = {0};
int          circIdx = 0;

int16_t firFilter(int16_t sample) {

    int16_t output = 0;

    // current sample in buffer
    circBuf[circIdx] = sample;

    //  y(n) = sum h(k)* x(n-k
    float acc = 0.0f;  // float fast, no int overflow
    for (int k = 0; k < N_TAPS; ++k) {
        // wrap using modulo
        int idx = (((int)circIdx - k + N_TAPS) % N_TAPS);
        acc += myfilter[k] * (float)circBuf[idx];
    }

    // ++circ index (modulo)
    circIdx = (int16_t)((((int)circIdx) + 1) % N_TAPS);


    if (acc >  32767.0f) acc =  32767.0f;
    if (acc < -32768.0f) acc = -32768.0f;
    output = (int16_t)acc;

    return output;
}


// ********************* END YOUR CODE HERE ********************* //
```

IIR Design

```
// 2nd-order IIR low-pass filter
static const float b0 = 0.0675f;
static const float b1 = 0.1349f;
static const float b2 = 0.0675f;
static const float a1 = -1.1429f;
static const float a2 = 0.4128f;

// Delay buffers for x[n-1], x[n-2], y[n-1], y[n-2]
static float x1 = 0.0f, x2 = 0.0f;
static float y1 = 0.0f, y2 = 0.0f;

int16_t iirFilter(int16_t sample) {
    // cast to float for accumulation
    float x0 = (float) sample;

    // IIR difference equation
    float y0 = b0*x0 + b1*x1 + b2*x2 - a1*y1 - a2*y2;

    // Shift states
    x2 = x1;
    x1 = x0;
    y2 = y1;
    y1 = y0;

    // Saturate back to int16
    if (y0 >  32767.0f) y0 =  32767.0f;
    if (y0 < -32768.0f) y0 = -32768.0f;
    return (int16_t) y0;
}
```