```
//
// 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);
// IIR Filter Function Definition
int16_t iirFilter(int16_t sample);

/* global variable holding a mask */
int16_t lower_mask = 0x00FF;

        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 ******************** //

    /* unpack two 8 bit values as one 16 bit samples */
```

```cpp
    for (int i = 0; i < FRAME_SIZE*2; i+=2) {
        bufferIn[i/2] = dataBuf->buf_[i + 1] << 8 | dataBuf->buf_[i];
    }

    // ********************* 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);
//          int16_t output = iirFilter(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 ******************** //

    /* Note that buffer is little endian */
    for (int i = 0; i < FRAME_SIZE*2; i+= 2) {
        /* get the lowest 8 bits and store them */
        dataBuf->buf_[i] = lower_mask & bufferOut[i/2];
        /* right shift to get upper 8 bits */
        dataBuf->buf_[i+1] = lower_mask & (bufferOut[i/2] >> 8);
    }

    // ********************* 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 41
// TODO: Change myfilter to contain the coefficients of your designed filter.
double myfilter[N_TAPS] = {0.01754934379515457, 0.02315431249063755, 0.028552314229625165, 0.03331634014026463, 0.03701803027148863, 0.03926002999427127, 0.03970808680477073, 0.03812032996498682, 0.03437129044705291, 0.028468539988971742, 0.020560335168906853, 0.010933308207510043, -4.081721798621698e-18, -0.011723177600623504, -0.02364619876172064, -0.035141874793005956, -0.045585948767361426, -0.054397951293508454, -0.06107982731466831, -0.06524947165805227, 0.9333333333333331, -0.06524947165805227, -0.06107982731466831, -0.054397951293508454, -0.045585948767361426, -0.035141874793005956, -0.02364619876172064, -0.011723177600623504, -4.081721798621698e-18, 0.010933308207510043, 0.020560335168906853, 0.028468539988971742, 0.03437129044705291, 0.03812032996498682, 0.03970808680477073, 0.03926002999427127, 0.03701803027148863, 0.03331634014026463, 0.028552314229625165, 0.02315431249063755, 0.01754934379515457};

// Circular Buffer
int16_t circBuf[N_TAPS] = {};
int16_t circBufIdx = 0;

// 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 output;
    double inter = 0.0; // to preserve accuracy during the calculation

    /* insert sample into buffer */
    circBuf[circBufIdx] = sample;

    /* perform convolution */
    for (int i = 0; i < N_TAPS; i++) {
        inter += myfilter[i] * circBuf[ ((( circBufIdx - i) % N_TAPS) + N_TAPS) % N_TAPS];
    }
    output = int16_t(inter);
    /* update pointer */
    circBufIdx = (circBufIdx + 1) % N_TAPS;
//    ((index % n) + n) % n;
    // ********************* END YOUR CODE HERE ********************* //
    return output;
}

#define IIR_TAPS 18
double iir_b[IIR_TAPS] = {0.8817074369702885, -8.221571296895403, 32.82701927500731, -70.6038301419953, 80.2220458836989, -26.07181320392112, -42.75031874538362, 37.50230613229456 4, 24.
```

```
838122160824486, -38.44852592869242, -13.516064148386493, 39.67415178177526, -2.374959354353843, -39.12717733380474, 40.39164433441818, -19.67916418092505, 4.989601930099065, -0.5331746004406787};
double iir_a[IIR_TAPS] = {1.0, -9.076122288724601, 35.18349678281452, -73.0524471080759, 78.75942632201394, -20.385555127984734, -45.64541317237927, 33.49043536983728, 28.770390247301478, -35.8215620859818, -17.61311180325776, 38.56606641438201, 1.4482849430617926, -40.41107762985706, 39.08826203712684, -18.34762506215212, 4.516759946687295, -0.4702101818272066};
int16_t prevOutputs[IIR_TAPS - 1] = {};
int16_t prevOutputsIdx = 0;

// Circular Buffer
int16_t iirCircBuf[IIR_TAPS] = {};
int16_t iirCircBufIdx = 0;

int16_t iirFilter(int16_t sample) {

    int16_t output;
    int16_t output_p = 0;
    int16_t output_q = 0;

    iirCircBuf[iirCircBufIdx] = sample;

    /* calculation of the difference equation found from https://en.wikipedia.org/wiki/Infinite_impulse_response */
    for (int i = 0; i < IIR_TAPS; i++) {
        output_p += iir_b[i] * iirCircBuf[(((iirCircBufIdx - i) % IIR_TAPS) + IIR_TAPS) % IIR_TAPS];
        if (i > 0)
            output_q += iir_a[i] * prevOutputs[(((prevOutputsIdx - i) % IIR_TAPS) + IIR_TAPS) % IIR_TAPS];
    }

    output = (1 / iir_a[0]) * (output_p - output_q);
    iirCircBufIdx = (iirCircBufIdx + 1) % IIR_TAPS;
    prevOutputsIdx = (prevOutputsIdx + 1) % IIR_TAPS;

    return output;
}

```