FIR-FILTER 2: IMPLEMENTATION ON GENERAL-PURPOSE COMPUTER

Introduction

The objective of this lab was to implement the FIR filter designed and documented in the last lab report on a general-purpose computer with a sound card. The filter was to be implemented with minimal delay in a C program using the JACK plugin for Linux systems. The filter specifications are documented in the previous lab. The program had to have a single input and be able to output to two channels: the filtered output on one, and a passthrough output on the other. It needed to read in the coefficients of the file created in Lab 1 and in other files. It also had to be able to run for an unlimited amount of time.

Program Description

The most important part of the program is located in the process_samples function. This is a base function called by the JACK API. However, we are able to implement the filter in it. This is done through the convolution of the coefficients and a delay line array. This is done through the inner product of the arrays. The delay line has the current input inserted at the beginning and is shifted.

```
delay_line[0] = x;
y = 0; //---- Reset Y

for(j=0;j<NUM_COEFF;j++){
      y += h[j] * delay_line[j]; // convolution loop
}

for(j=NUM_COEFF-1;j>0;j--){
     delay_line[j]=delay_line[j-1];  // shift loop
}

out_filter[i] = y; //----- output the sum
```

The passthrough output is set to the current output if on and 0 if off.

The other addition I did to the code was to implement the ability to toggle the passthrough while the program was running. To do this, the program waits for the character ‘f’ to be entered and then toggles the passthrough Boolean.

Test Procedure

The first test tested the filter's impulse response by observing the oscilloscope output. A function generator was connected to the input channel of the computer. The stereo output was connected to a set of stereo speakers and the oscilloscope. The code was then run, and measurements were taken comparing the filtered output to the passthrough output as the frequency of the generated sine wave increased.

The second test was done using a file with many coefficients in order to test if our program gracefully exited if overloaded with more than our defined max coefficients. This just required running the code and observing if it gracefully exited the program.

The third test had the same setup as the first. It used a file with coefficients that when used with the right input, would generate a signal that was morse code for TTU. The input had to be set as a 1 kHz square wave on burst mode. This was done in order to test if our program did the convolution in the right order.

The fourth test was to measure the filter’s impulse response using a network analyzer. The network analyzer was calibrated

using a wire as the DUT (Device Under Test) since it has relatively no impulse response. This was then replaced with the sound card running our program. We then measured the impulse response of the filtered output as a whole and of the passband. We also measured the impulse response of the passthrough output – essentially the impulse response of the sound card – as a whole and in the passband.

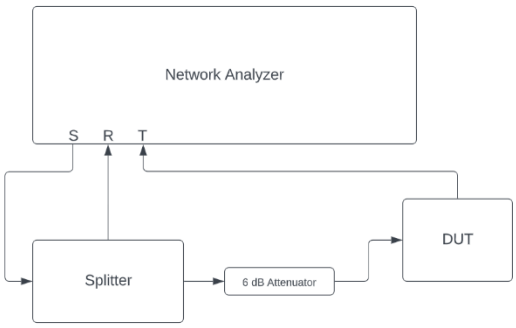

Figure 1 shows the Network Analyzer Connections

Test Results

The first test was demonstrated to Dr. Austen where it was shown that the filtered output behaved as expected, that is it showed a gain in the passband frequencies and attenuated in the stop band frequencies.

The program also was able to compile without warnings. During test 2, it was successfully able to exit when presented with a nonexistent file or when max coefficients were exceeded. This was also demonstrated to Dr. Austen. Included is also the result from an execution in Windows, which also shows the graceful exit.

```
Cannot connect to named pipe after wait = \\.\pipe\server_jack_default_0 err = 2
Cannot connect to server request channel
JACK server started
engine sample rate: 44100
File Opened
maximum coefficients reached
PS C:\Users\cdpop\Documents\DSP>
```

Figure 2 shows the output terminal if exceeded coefficients

Test 3's expected output would be the morse code for "TTU" on the oscilloscope. This would be - - ..- in morse code.

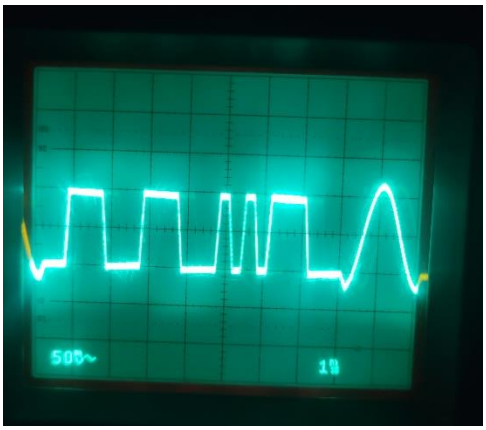

Figure 3 shows the output for test 3

For test 4, we expect the full filter impulse response to show a small passband ripple with an attenuation of about 50 dB in the stopband.

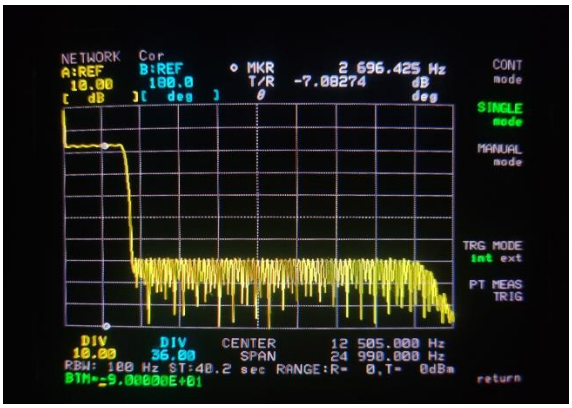

Figure 4 shows the impulse response of the filter

Figure 4 shows a correct attenuation at 5 divisions below the passband and divisions being 10 dB each.

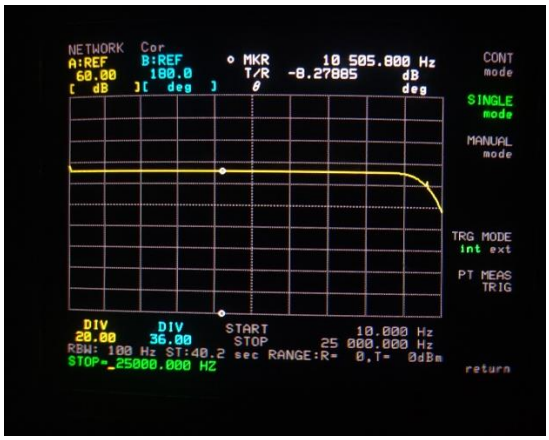

Figure 5 shows the impulse response of the sound card

Figure 5 shows an expected output as it explains the unwanted attenuation at the highest frequencies in figure 4.

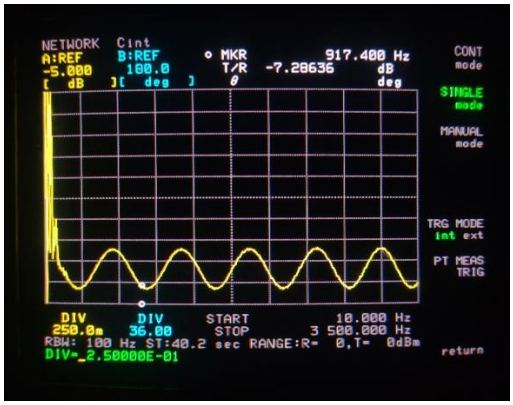

Figure 6 shows the passband for the filter

Figure 6 shows an expected result of 0.5 dB ripple in the passband. It is about 2 divisions high with each division being 0.25 dB.

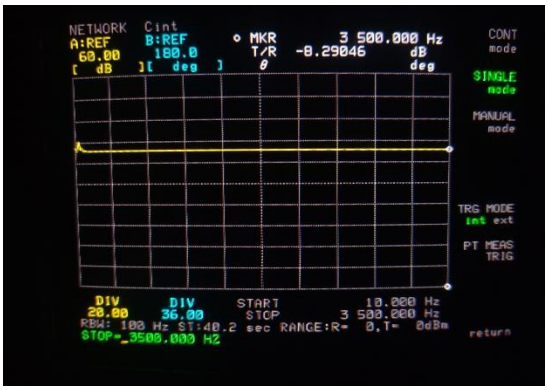

Figure 7 shows the passband for the sound card

Figure 7 shows an expected result as we would expect the sound card to be designed to show very little response in the frequencies people hear since otherwise, all audio would have to be changed in order to account for the impulse response.

Summary

I was able to successfully demonstrate the functionality of my FIR filter implementation. I was also able to successfully measure and compare its impulse response. My program ran and followed all specifications as well as received the bonus.

Appendix

References

Lab handout

Lab 1 – FIR Filter design

Code

Lab1_for_linux_fixed.c

```
/* CARSON POPE FINAL CODE FOR LAB */
/** @file lab1_for_linux_fixed.c
 *
 * @brief This client reads a .txt file for an FIR filter coefficients.
 *          It then convolutes them with the input signal to create a filter.
 *          It also has the option to toggle a pass through signal.
 *
 * This is a modified version of the original simple_client.c program.
 * The original was downloaded on 2012-01-14 from:
 *   http://trac.jackaudio.org/wiki/WalkThrough/Dev/SimpleAudioClient
 *   http://trac.jackaudio.org/browser/trunk/jack/example-clients/simple_client.c
 *   (previous versions/downloads: Oct. 2009)
 *
 * This program initializes the sound card and processes the audio samples from the input to the output.
 * It is modified to create a filter or other DSP application which uses the sound card.
 * For most applications, if used as a template, the only changes needed are in the marked areas;
 * the rest of the program should not need to be modified.
 * Certain lines will be marked with "//-----". This means the referenced line was not added but
 * instead replaced a line of the original code.
 * This version has stereo output. The input is on the left channel and output is directed to
 * left channel and right channel respectively, depending on the sound card driver configuration.
 *
 * On Linux using gcc compile with the '-ljack' option to link with the jack library.
 * If you use math functions, e.g. cos, the '-lm' option is also needed to link with the math library.
 *   E.g., gcc -Wall myprog.c -ljack -lm -o myprog
 * The -Wall enables most warnings and the -o specifies the executable file name.
 * The order matters on some systems: place the link library list (-l...) after the source file name.
 *
 * Updated: Feb. 2023
 */

#include <stdio.h>
#include <errno.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>

#include <jack/jack.h>

#define MAX_COEFF 50
#define MAX_LEN 100

jack_port_t *input_port;
jack_port_t *output_port_filtered; //----
```

```
jack_port_t *output_port_pass;     //----
jack_client_t *client;

/***** GLOBAL VARIABLE DECLARATIONS HERE *****/
    static float h[MAX_COEFF];
    static int NUM_COEFF;


    bool passthrough = 0;

/**
* This function checks for any input and compares it to the passed character.
* If it matches it will flip the passthrough boolean
**/
void checkInput(int ch){
    if(getchar() == ch){  //the getchar() function in the if statement will wait until a newline
        passthrough = !passthrough;         //before checking if it matches
        fprintf(stderr,"\nPASSTHROUGH %s . . .   \n", passthrough?"ON":"OFF"); // "ON" if passthrough is a
1, "OFF" if passthrough is a 0
        fprintf(stderr,"Please enter a lower case \"f\" to toggle:   ");
    }
}

/**
 * The process callback for this JACK application is called in a
 * special realtime thread once for each audio cycle.
 *
 * It will exit when stopped by
 * the user (e.g. using Ctrl-C on a unix-ish operating system)
 */
int process_samples (jack_nframes_t nframes, void *arg)
{
    jack_default_audio_sample_t *in;
    jack_default_audio_sample_t *out_filter; //----
    jack_default_audio_sample_t *out_pass;   //----
    int i;
    float x;
    float y;

/***** LOCAL VARIABLE DECLARATIONS HERE *****/
    //---------------------------------------------
        static float delay_line[MAX_COEFF] = { 0 };

        int j;
    //---------------------------------------------
```

```
    in = jack_port_get_buffer (input_port, nframes);
    out_filter = jack_port_get_buffer (output_port_filtered, nframes); //----
    out_pass = jack_port_get_buffer (output_port_pass, nframes);  //----

    for (i=0;i<(int)nframes;i++) {
        x=in[i];

/***** MY ALGORITHM *****/

        delay_line[0] = x;
        y = 0; //---- Reset Y

        for(j=0;j<NUM_COEFF;j++){
            y += h[j] * delay_line[j]; // convolution loop
        }

        for(j=NUM_COEFF-1;j>0;j--){
            delay_line[j]=delay_line[j-1];  // shift loop
        }

        out_filter[i] = y; //----- output the sum

        if(passthrough == 1){
            out_pass[i] = x;   //----- passthrough on
        } else {
            out_pass[i] = 0;   //----- passthrough off
        }

    }

    return 0;
}


/**
 * JACK calls this shutdown_callback if the server ever shuts down or
 * decides to disconnect the client.
 */
void jack_shutdown (void *arg)
{
    exit (EXIT_FAILURE);
}


int main (int argc, char *argv[])
```

```
{
    const char **ports;
    const char *client_name = "simple-client";
    const char *server_name = NULL;
    jack_options_t options = JackNullOption;
    jack_status_t status;

        int sample_rate;

/***** LOCAL VARIABLE DECLARATIONS HERE *****/
        FILE * coeff_file;
        char line[MAX_LEN], *firstLine;

    /* open a client connection to the JACK server */

    client = jack_client_open (client_name, options, &status, server_name);
    if (client == NULL) {
        fprintf (stderr, "jack_client_open() failed, "
             "status = 0x%2.0x\n", status);
        if (status & JackServerFailed) {
            fprintf (stderr, "Unable to connect to JACK server\n");
        }
        exit (EXIT_FAILURE);
    }
    if (status & JackServerStarted) {
        fprintf (stderr, "JACK server started\n");
    }
    if (status & JackNameNotUnique) {
        client_name = jack_get_client_name(client);
        fprintf (stderr, "unique name `%s' assigned\n", client_name);
    }

    /* tell the JACK server to call `process_samples()' whenever
     * there is work to be done.
     */

    jack_set_process_callback (client, process_samples, 0);

    /* tell the JACK server to call `jack_shutdown()' if
     * it ever shuts down, either entirely, or if it
     * just decides to stop calling us.
     */

    jack_on_shutdown (client, jack_shutdown, 0);
```

```
    /* get current sample rate */

        sample_rate = jack_get_sample_rate (client);

        printf ("engine sample rate: %d\n", sample_rate);

    /* create two ports */

    input_port = jack_port_register (client, "input",
                    JACK_DEFAULT_AUDIO_TYPE,
                    JackPortIsInput, 0);
    output_port_filtered = jack_port_register (client, "FilterOutput", //----
                     JACK_DEFAULT_AUDIO_TYPE,
                     JackPortIsOutput, 0);
    output_port_pass = jack_port_register (client, "PassThroughOutput", //----
                     JACK_DEFAULT_AUDIO_TYPE,
                     JackPortIsOutput, 0);


    if ((input_port == NULL) || (output_port_filtered == NULL) || (output_port_pass == NULL)){ //-----
        fprintf(stderr, "no more JACK ports available\n");
        exit (EXIT_FAILURE);
    }

/***** INSERT INITIALIZATION CODE HERE *****/
    /***** THIS STATEMENT OPENS THE FILE AND GRACEFULLY EXITS IF IT CANT OPEN ******/
    /***** COMMENTED OUT LINES ALLOW EASY SWITCH BETWEEN INPUT FILES          ******/

    if((coeff_file = fopen("pope_fir_coefficients_rev5.txt","r")) == NULL){
//  if((coeff_file = fopen("filter_test_TTU_morse_code_h.txt","r")) == NULL){
//  if((coeff_file = fopen("filter_test_HUGE_h.txt","r")) == NULL){
        fprintf(stderr, "cannot open file\n");
        exit(EXIT_FAILURE);
    }

    //GET FIRST LINE AND PRINT TO COMMAND LINE
    if((firstLine = fgets(line,MAX_LEN,coeff_file)) != NULL){
        printf("File Opened\n");
    }

    //LOOP THROUGH AND SAVE COEFFICIENTS IN h array
    NUM_COEFF = 0;
    while(fscanf(coeff_file,"%f",&h[NUM_COEFF]) != EOF){
        NUM_COEFF++;
        if(NUM_COEFF >= MAX_COEFF-1){
```

```
        fprintf(stderr,"maximum coefficients reached\n");
        exit (EXIT_FAILURE);
    }
}

//  printf("%i\n",NUM_COEFF);                   THIS CAN BE USED FOR
//  printf("%f %f\n",h[0],h[NUM_COEFF-1]);           DEBUGGING

fclose(coeff_file);

/* Tell the JACK server that we are ready to begin.
 * Our process_samples() callback will start running now.
 */
if (jack_activate (client)) {
    fprintf (stderr, "cannot activate client");
    exit (EXIT_FAILURE);
}

/* Connect the ports.  You can't do this before the client is
 * activated, because we can't make connections to clients
 * that aren't running.  Note the confusing (but necessary)
 * orientation of the driver backend ports: playback ports are
 * "input" to the backend, and capture ports are "output" from
 * it.
 */
ports = jack_get_ports (client, NULL, NULL,
            JackPortIsPhysical|JackPortIsOutput);
if (ports == NULL) {
    fprintf(stderr, "no physical capture ports\n");
    exit (EXIT_FAILURE);
}

if (jack_connect (client, ports[0], jack_port_name (input_port))) {
    fprintf (stderr, "cannot connect input ports\n");
}

free (ports);

ports = jack_get_ports (client, NULL, NULL,
            JackPortIsPhysical|JackPortIsInput);
if (ports == NULL) {
    fprintf(stderr, "no physical playback ports\n");
    exit (EXIT_FAILURE);
}
```

```
    if(jack_connect (client, jack_port_name (output_port_filtered), ports[0])){ //----
        fprintf (stderr, "cannot connect output ports\n");
    }
    if(jack_connect (client, jack_port_name (output_port_pass), ports[1])){ //----
        fprintf (stderr, "cannot connect output ports\n");
    }

    free (ports);

    /* it is now running....*/

/***** ACTIONS TO DO WHILE ALGORITHM IS RUNNING *****/
    //Start up messages
    fprintf(stderr,"Enter f to toggle passthrough ...  \n");
    fprintf(stderr,"Running ... press CTRL-C to exit ...   ");
    while (1) {
        //constantly check for new inputs in the console.
        checkInput('f'); //-----
    }

    /* This is may or may not be reached, depending on what is directly before it,
     * but if the program had some other way to exit besides being killed,
     * they would be important to call.
     */

    jack_client_close (client);
    exit (EXIT_SUCCESS);
}
```