EEL4511C

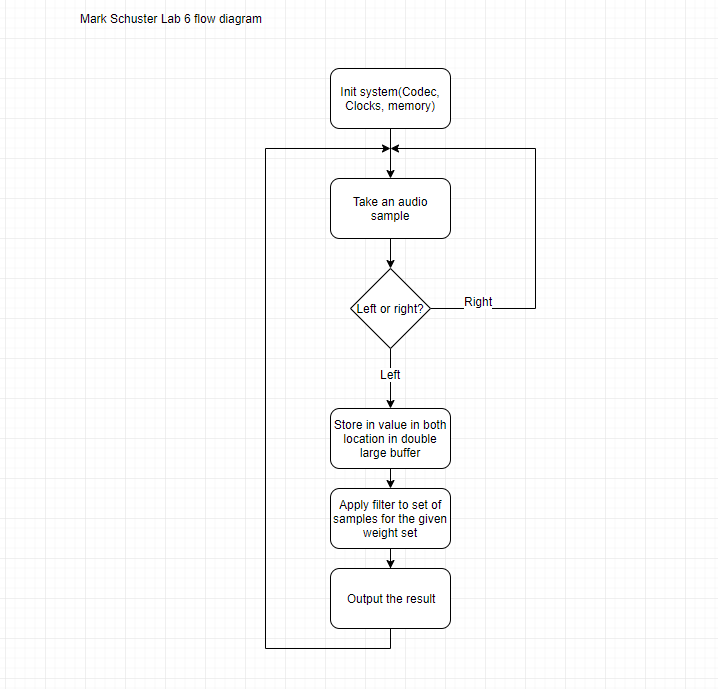
Schuster, Mark

Lab 6

03/28/2018

**1.1 Flowcharts:**

Flow chart for the LPF, BPF, and HPF flowchart:



**1.2 Schematics/Decoding Logic:**

None for this lab.

**1.3 Problems Encountered:**

In this lab, I had several errors that proved frustrating. First, I used some older code that incorrectly initialized the phase lock loop to 10 and 2 instead of the proper 10 and 3. This threw off the filter weights significantly until I corrected this. Upon fixing these initialization parameters, I ran into more problems. Again, my weights were not even close to producing the desired output. This is when I realized I was no delineating between the left and right channels. I then handled this by only outputting data to the left channel. Once these corrections were made, my filtering lab fell into place.

**1.4 Program Code:**

Headers:

**msLib.h**

// File: msLib.h

// Date: 03/08/2018

// Name: Mark Schuster

// Class: EEE4511C (DSP)

**#ifndef** MSLIB\_H\_

**#define** MSLIB\_H\_

**#include** <DSP2833x\_Device.h>

**#include** "DSP28x\_Project.h"

**#include** "ADC\_Utils.h"

**#include** "Codec\_Utils.h"

**#include** "extSram.h"

**#include** "I2C\_LCD\_Utils.h"

**#include** "Interrupt\_ISRs.h"

**#include** "Interrupt\_Utils.h"

**#include** "LEDs\_switches\_Utils.h"

**#include** "OneToOneI2CDriver.h"

**#include** "Timer1\_Utils.h"

**#include** "filters.h"

**#endif**

**ADC\_Utils.h**

// File: I2C\_LCD\_Utils.h

// Date: 02/24/2018

// Name: Mark Schuster

// Class: EEE4511C (DSP)

**#ifndef** ADCUTILS\_H\_

**#define** ADCUTILS\_H\_

**void** **initADC**(**void**);

**#endif**

**Codec\_Utils.h:**

// File: Codec\_Utils.h

// Date: 02/24/2018

// Name: Mark Schuster

// Class: EEE4511C (DSP)

**#ifndef** CODECUTILS\_H\_

**#define** CODECUTILS\_H\_

**extern** **interrupt** **void** codecIsr();

**void** **initCodec**(**void**);

**#endif**

**filters.h**

// File: filters.h

// Date: 03/25/2018

// Name: Mark Schuster

// Class: EEE4511C (DSP)

**#ifndef** FILTERS\_H\_

**#define** FILTERS\_H\_

**void** **initWeights**(**float**\* weights, Uint16 len, **float** scaler);

**float** **firC**(**float**\* weights, **float**\* samples, Uint16 weightLen);

**float** **firASM**(**float**\* weights, **float**\* samples, Uint16 weightLen);

**#define** FILTER\_TAP\_NUM\_LPF 43

**#define** FILTER\_TAP\_NUM\_BPF 207

**#define** FILTER\_TAP\_NUM\_HPF 33

**extern** **float** lpfWeights[FILTER\_TAP\_NUM\_LPF];

**extern** **float** bpfWeights[FILTER\_TAP\_NUM\_BPF];

**extern** **float** hpfWeights[FILTER\_TAP\_NUM\_HPF];

**#endif**

**I2C\_LCD\_Utils.h**

// File: I2C\_LCD\_Utils.h

// Date: 02/24/2018

// Name: Mark Schuster

// Class: EEE4511C (DSP)

**#ifndef** I2CLCDUTILS\_H\_

**#define** I2CLCDUTILS\_H\_

// Initializes the LCD and clears the display.

**void** **initLCD**(**void**);

// Sends a command to the LCD over I2C.

**void** **sendCmdListLCD**(Uint16 \*, Uint16);

// Writes a character to the LCD after it has been

// initialized with "initLCD".

**void** **sendCharLCD**(**char**);

// Writes a string to the LCD using "sendCharLCD".

**void** **sendStringLCD**(**char**\*);

// Send the clear screen to the LCD,

// wiping the display.

**void** **clearLCD**(**void**);

// Set the LCD's cursor the first

// position in the top right corner.

**void** **cursorHomeLCD**(**void**);

**#endif**

**Interrupt\_ISRs.h**

// File: Interrupt\_ISRs.h

// Date: 02/24/2018

// Name: Mark Schuster

// Class: EEE4511C (DSP)

**#ifndef** INTERRUPTISRS\_H\_

**#define** INTERRUPTISRS\_H\_

**#define** SAMPLE\_BUFFER\_SIZE 512

**#define** SAMPLE\_BUFFER\_OFFSET SAMPLE\_BUFFER\_SIZE/2

**#define** SAMPLE\_BUFFER\_MASK\_ON 0xFF00

**#define** SAMPLE\_BUFFER\_MASK\_OFF 0x00FF

**interrupt** **void** **timer1Isr**(**void**);

**interrupt** **void** **audioIsr**(**void**);

**extern** **float** sampleBuffer[SAMPLE\_BUFFER\_SIZE];

**extern** Uint16 dataReady;

**#endif**

**Interrupt\_Utils.h**

// File: Interrupt\_Utils.h

// Date: 02/24/2018

// Name: Mark Schuster

// Class: EEE4511C (DSP)

**#ifndef** INTERRUPTUTILS\_H\_

**#define** INTERRUPTUTILS\_H\_

**void** **initInterrupts**(**void**);

**#endif**

**OneToOneI2CDriver.h**

/\*

\* OneToOneI2CDriver.h

\*

\* Created on: Sep 24, 2016

\* Author: Raz Aloni

\*/

**#ifndef** ONETOONEI2CDRIVER\_H\_

**#define** ONETOONEI2CDRIVER\_H\_

/\*

\* <summary>

\* Initializes the I2C to run in Master Mode for a One-To-One connection

\* </summary>

\* <param="slaveAddress">Address of the slave device to write to</param>

\* <param="sysClkMhz">System Clock Frequency in Mhz</param>

\* <param="I2CClkKHz">Desired I2C Clock Frequency in KHz</param>

\*/

**void** **I2C\_O2O\_Master\_Init**(Uint16 slaveAddress, float32 sysClkMhz, float32 I2CClkKHz);

/\*

\* <summary>

\* Sends bytes via I2C

\* </summary>

\* <param="values">Pointer to array of bytes to send</param>

\* <param-"length">Length of array</param>

\*/

**void** **I2C\_O2O\_SendBytes**(Uint16 \* **const** values, Uint16 length);

**#endif** /\* ONETOONEI2CDRIVER\_H\_ \*/

**Timer1\_Utils.h**

// File: Timer1\_Utils.h

// Date: 02/24/2018

// Name: Mark Schuster

// Class: EEE4511C (DSP)

**#ifndef** TIMER1UTILS\_H\_

**#define** TIMER1UTILS\_H\_

**extern** **interrupt** **void** timer1Isr();

**void** **initTimer1**(**void**);

**#endif**

**ADC\_Utils.c**

// File: ADC\_Utils.c

// Date: 02/24/2018

// Name: Mark Schuster

// Class: EEE4511C (DSP)

**#include** "msLib.h"

**void** **initADC**(**void**){

**InitAdc**();

EALLOW;

PieCtrlRegs.PIEIER1.bit.INTx6 = 1;

AdcRegs.ADCTRL1.all = 0x0170;

AdcRegs.ADCCHSELSEQ1.bit.CONV00 = 0;

AdcRegs.ADCREFSEL.bit.REF\_SEL = 0x0;

AdcRegs.ADCMAXCONV.all = 0;

AdcRegs.ADCTRL1.bit.CPS = 1;

AdcRegs.ADCTRL2.bit.SOC\_SEQ1 = 1;

**return**;

}

**Codec\_Utils.c**

// File: Codec\_Utils.c

// Date: 02/24/2018

// Name: Mark Schuster

// Class: EEE4511C (DSP)

**#include** "msLib.h"

**void** **initCodec**(**void**){

EALLOW;

PieVectTable.MRINTB = audioIsr; //link it to my interrupt

PieCtrlRegs.PIEIER6.bit.INTx3 = 1;

IER |= M\_INT6;

**EnableInterrupts**();

McbspbRegs.SPCR1.bit.RRST = 0;

McbspbRegs.SPCR1.bit.RRST = 1;

EDIS;

}

**extSram.c**

// File: extSram.c

// Date: 03/08/2018

// Name: Mark Schuster

// Class: EEE4511C (DSP)

**#include** "msLib.h"

**#pragma** DATA\_SECTION(sram, ".extsram")

Uint16 sram[0x40000];

**void** **initExtSRAM**(**void**){

EALLOW;

SysCtrlRegs.PCLKCR3.bit.XINTFENCLK = 1;

GpioCtrlRegs.GPAMUX2.all |= 0xFF000000;

GpioCtrlRegs.GPBMUX1.all |= 0xFFFFF000;

GpioCtrlRegs.GPCMUX1.all |= 0xFFFFFFFF;

GpioCtrlRegs.GPCMUX2.all |= 0xFFFF;

EDIS;

}

**filters.c**

// File: Timer1\_Utils.c

// Date: 02/24/2018

// Name: Mark Schuster

// Class: EEE4511C (DSP)

**#include** "msLib.h"

/\*

FIR filter designed with

http://t-filter.appspot.com

sampling frequency: 48000 Hz

\* 0 Hz - 1490 Hz

gain = 1

desired ripple = 3 dB

actual ripple = 2.0057410958006363 dB

\* 3100 Hz - 24000 Hz

gain = 0

desired attenuation = -40 dB

actual attenuation = -41.42777662856003 dB

\*/

**float** lpfWeights[FILTER\_TAP\_NUM\_LPF] = {

-0.006458890599120238,

-0.0052277234367167115,

-0.006802500304306202,

-0.008193107806323893,

-0.009163750472661164,

-0.00945852694971267,

-0.008816196248689984,

-0.007009096032907269,

-0.00383359020603197,

0.0008093761425741296,

0.0069640516710003895,

0.01454950392877843,

0.02337091925327902,

0.033140681462845246,

0.043467422746496225,

0.053886032275704335,

0.06388431627817463,

0.0729551021795933,

0.08060480653931533,

0.08640199887081489,

0.09001978297922335,

0.09124846729592041,

0.09001978297922335,

0.08640199887081489,

0.08060480653931533,

0.0729551021795933,

0.06388431627817463,

0.053886032275704335,

0.043467422746496225,

0.033140681462845246,

0.02337091925327902,

0.01454950392877843,

0.0069640516710003895,

0.0008093761425741296,

-0.00383359020603197,

-0.007009096032907269,

-0.008816196248689984,

-0.00945852694971267,

-0.009163750472661164,

-0.008193107806323893,

-0.006802500304306202,

-0.0052277234367167115,

-0.006458890599120238

};

/\*

FIR filter designed with

http://t-filter.appspot.com

sampling frequency: 48000 Hz

\* 0 Hz - 700 Hz

gain = 0

desired attenuation = -40 dB

actual attenuation = -40.16027932496705 dB

\* 1000 Hz - 1800 Hz

gain = 1

desired ripple = 3 dB

actual ripple = 2.3243783057628526 dB

\* 2400 Hz - 24000 Hz

gain = 0

desired attenuation = -40 dB

actual attenuation = -40.16027932496705 dB

\*/

**float** bpfWeights[FILTER\_TAP\_NUM\_BPF] = {

-0.005550607155108627,

-0.0013056523576555079,

-0.0013342155644092896,

-0.001269101310882767,

-0.001101421082460746,

-0.0008273684084292744,

-0.0004491957439684645,

0.000024603430435381685,

0.0005792828460565788,

0.001194656709923292,

0.001845498464130876,

0.0025025189675394266,

0.0031343997760353358,

0.003708870767678973,

0.004195412370955356,

0.004565599392029179,

0.004795983474838851,

0.004867516274284377,

0.004769615960081084,

0.004497442595380547,

0.004054523482725583,

0.0034513935424908735,

0.002705389176418765,

0.001840601972125689,

0.0008856080403549967,

-0.0001279910715565196,

-0.0011655273606744083,

-0.0021915786055739478,

-0.003172637659481723,

-0.004075027784659611,

-0.004870725656342098,

-0.005533516807498927,

-0.00604329211033132,

-0.006384750558594912,

-0.006548896807853559,

-0.006532152252981715,

-0.006337043433827243,

-0.005971711665147881,

-0.005449896415970832,

-0.004790529990802946,

-0.004016818114311403,

-0.003156062425311335,

-0.0022387554259162517,

-0.001299008433040772,

-0.0003650969957501046,

0.000512687705217782,

0.0013286874506353132,

0.002031959164414224,

0.002601597098910619,

0.003025455138443938,

0.0032935671517588163,

0.0034022432252419003,

0.003358794628523227,

0.0031826920719666245,

0.0029045727065859475,

0.0025637172799632046,

0.002205591276143405,

0.0018806660955073108,

0.0016409612114559905,

0.0015379155922844096,

0.001617013336456975,

0.0019149591319081018,

0.00245403020260472,

0.003240645625434443,

0.004260320967111584,

0.005477027216898309,

0.006832751731288787,

0.008249063756204659,

0.009630419549910002,

0.010867124966547324,

0.01184062865328758,

0.012432442858158097,

0.012531030341076367,

0.01203602370121854,

0.010873898244569915,

0.008996957757385482,

0.00639647016306678,

0.003103148704083669,

-0.0008086209505012681,

-0.0052216644355174745,

-0.009978009769410935,

-0.014884827406725944,

-0.01972220950652443,

-0.024253356612257155,

-0.028236448919145174,

-0.031437139509148455,

-0.03364243456035483,

-0.0346731859284739,

-0.034397609583139456,

-0.032735985094257215,

-0.029679341889910622,

-0.025275350498777993,

-0.01964904819252152,

-0.012986430480041965,

-0.005528939977722972,

0.0024329162069420527,

0.010573660763370355,

0.018550192463854295,

0.026018333704934817,

0.03264960794675452,

0.03814772569637046,

0.04226378302615264,

0.044811024405979415,

0.045673080421838506,

0.044811024405979415,

0.04226378302615264,

0.03814772569637046,

0.03264960794675452,

0.026018333704934817,

0.018550192463854295,

0.010573660763370355,

0.0024329162069420527,

-0.005528939977722972,

-0.012986430480041965,

-0.01964904819252152,

-0.025275350498777993,

-0.029679341889910622,

-0.032735985094257215,

-0.034397609583139456,

-0.0346731859284739,

-0.03364243456035483,

-0.031437139509148455,

-0.028236448919145174,

-0.024253356612257155,

-0.01972220950652443,

-0.014884827406725944,

-0.009978009769410935,

-0.0052216644355174745,

-0.0008086209505012681,

0.003103148704083669,

0.00639647016306678,

0.008996957757385482,

0.010873898244569915,

0.01203602370121854,

0.012531030341076367,

0.012432442858158097,

0.01184062865328758,

0.010867124966547324,

0.009630419549910002,

0.008249063756204659,

0.006832751731288787,

0.005477027216898309,

0.004260320967111584,

0.003240645625434443,

0.00245403020260472,

0.0019149591319081018,

0.001617013336456975,

0.0015379155922844096,

0.0016409612114559905,

0.0018806660955073108,

0.002205591276143405,

0.0025637172799632046,

0.0029045727065859475,

0.0031826920719666245,

0.003358794628523227,

0.0034022432252419003,

0.0032935671517588163,

0.003025455138443938,

0.002601597098910619,

0.002031959164414224,

0.0013286874506353132,

0.000512687705217782,

-0.0003650969957501046,

-0.001299008433040772,

-0.0022387554259162517,

-0.003156062425311335,

-0.004016818114311403,

-0.004790529990802946,

-0.005449896415970832,

-0.005971711665147881,

-0.006337043433827243,

-0.006532152252981715,

-0.006548896807853559,

-0.006384750558594912,

-0.00604329211033132,

-0.005533516807498927,

-0.004870725656342098,

-0.004075027784659611,

-0.003172637659481723,

-0.0021915786055739478,

-0.0011655273606744083,

-0.0001279910715565196,

0.0008856080403549967,

0.001840601972125689,

0.002705389176418765,

0.0034513935424908735,

0.004054523482725583,

0.004497442595380547,

0.004769615960081084,

0.004867516274284377,

0.004795983474838851,

0.004565599392029179,

0.004195412370955356,

0.003708870767678973,

0.0031343997760353358,

0.0025025189675394266,

0.001845498464130876,

0.001194656709923292,

0.0005792828460565788,

0.000024603430435381685,

-0.0004491957439684645,

-0.0008273684084292744,

-0.001101421082460746,

-0.001269101310882767,

-0.0013342155644092896,

-0.0013056523576555079,

-0.005550607155108627

};

/\*

FIR filter designed with

http://t-filter.appspot.com

sampling frequency: 48000 Hz

\* 0 Hz - 2000 Hz

gain = 0

desired attenuation = -30 dB

actual attenuation = -35.883659487624854 dB

\* 4000 Hz - 24000 Hz

gain = 1

desired ripple = 3 dB

actual ripple = 1.1974127997293782 dB

\*/

**float** hpfWeights[FILTER\_TAP\_NUM\_HPF] = {

-0.031838401643728594,

0.0064122218895261655,

0.00976495299703675,

0.01440699048722218,

0.018932398840236647,

0.02174798406862183,

0.021296223390531832,

0.01630845892105471,

0.006051323386266427,

-0.009482725050514238,

-0.029506993109006438,

-0.05247080828847111,

-0.07611474810950448,

-0.09810777362216545,

-0.1159565445779877,

-0.12759377120503504,

0.8683650669381691,

-0.12759377120503504,

-0.1159565445779877,

-0.09810777362216545,

-0.07611474810950448,

-0.05247080828847111,

-0.029506993109006438,

-0.009482725050514238,

0.006051323386266427,

0.01630845892105471,

0.021296223390531832,

0.02174798406862183,

0.018932398840236647,

0.01440699048722218,

0.00976495299703675,

0.0064122218895261655,

-0.031838401643728594

};

**void** **initWeights**(**float**\* weights, Uint16 len, **float** scaler)

{

**for**(Uint16 i=0;i<len;i++)

weights[i] = scaler\*weights[i];

}

**float** **firC**(**float**\* weights, **float**\* samples, Uint16 weightLen)

{

**float** res = 0;

**for**(**int** i=0;i<weightLen;i++)

res+=weights[i]\*samples[-i];

**return** res;

}

**fir.asm:**

.cdecls C, NOLIST, "msLib.h"

.def \_firASM

.ASG "R0H", result

.ASG "AR1", itter

.ASG "AR4", weightPtrLower16

.ASG "AR5", samplePtrLower16

.ASG "R1H", currentSample

.ASG "R2H", currentWeight

.ASG "R3H", currentProduct

.define "\*XAR4", weightPtr

.define "\*XAR5", samplePtr

**\_firASM:**

**ZERO** result

**MOV** itter, AL

**FIR\_LOOP:**

**CMP** itter, #0

**B** LPF\_DONE, EQ

**MOV32** currentSample, samplePtr

**MOV32** currentWeight, weightPtr

**MPYF32** currentProduct, currentSample, currentWeight

**DEC** itter

**INC** weightPtrLower16

**INC** weightPtrLower16

**DEC** samplePtrLower16

**DEC** samplePtrLower16

**ADDF32** result, result, currentProduct

**B** FIR\_LOOP, UNC

**LPF\_DONE:**

**LRETR**

**I2C\_LCD\_Utils.c**

// File: I2C\_LCD\_Utils.c

// Date: 02/24/2018

// Name: Mark Schuster

// Class: EEE4511C (DSP)

**#include** "msLib.h"

// Initializes the LCD and clears the display.

**void** **initLCD**(**void**)

{

Uint16 LCDCommandList[] = {0x33,0x32,0x28,0x0F,0x01};

I2C\_O2O\_Master\_Init(0x3F, 150, 400);

sendCmdListLCD(LCDCommandList, **sizeof**(LCDCommandList)/**sizeof**(Uint16));

cursorHomeLCD();

**return**;

}

// Sends a command to the LCD over I2C.

**void** **sendCmdListLCD**(Uint16 \*cmdList, Uint16 cmdListLen)

{

**for**(Uint16 itter=0; itter<cmdListLen; itter++)

{

Uint16 upperNibble, lowerNibble;

lowerNibble = ((cmdList[itter] & 0x000F) << 4) | 0x8;

upperNibble = (cmdList[itter] & 0x00F0) | 0x8;

Uint16 nibbleEnableCmds[] = { (upperNibble | 0x4), upperNibble,

(lowerNibble | 0x4), lowerNibble};

I2C\_O2O\_SendBytes(nibbleEnableCmds, **sizeof**(nibbleEnableCmds)/**sizeof**(Uint16));

}

DELAY\_US(1000);

**return**;

}

// Writes a character to the LCD after it has been

// initialized with "initLCD".

**void** **sendCharLCD**(**char** c){

Uint16 upperNibble, lowerNibble;

lowerNibble = (((Uint16)c & 0x000F) << 4) | 0x9;

upperNibble = ((Uint16)c & 0x00F0) | 0x9;

Uint16 nibbleEnableCmds[] = { (upperNibble | 0x4), upperNibble,

(lowerNibble | 0x4), lowerNibble};

I2C\_O2O\_SendBytes(nibbleEnableCmds, **sizeof**(nibbleEnableCmds)/**sizeof**(Uint16));

**return**;

}

// Writes a string to the LCD using "sendCharLCD".

**void** **sendStringLCD**(**char**\* str){

**for**(Uint16 itter=0; str[itter]!='\0'; itter++)

sendCharLCD(str[itter]);

**return**;

}

// Send the clear screen to the LCD,

// wiping the display.

**void** **clearLCD**(**void**)

{

Uint16 clearVal[] = {0x01};

sendCmdListLCD(clearVal, **sizeof**(clearVal)/**sizeof**(Uint16));

**return**;

}

// Set the LCD's cursor the first

// position in the top right corner.

**void** **cursorHomeLCD**(**void**){

Uint16 homeVal[] = {0x02};

sendCmdListLCD(homeVal, **sizeof**(homeVal)/**sizeof**(Uint16));

**return**;

}

**Interrupt\_ISRs.c**

// File: Interrupt\_ISRs.c

// Date: 02/24/2018

// Name: Mark Schuster

// Class: EEE4511C (DSP)

**#include** "msLib.h"

**float** sampleBuffer[SAMPLE\_BUFFER\_SIZE];

**interrupt** **void** **timer1Isr**()

{

CpuTimer1Regs.TCR.bit.TIF = 1;

**return**;

}

**enum**

{

*LEFT*,

*RIGHT*,

};

**interrupt** **void** **audioIsr**(**void**)

{

**static** Uint16 currentBufIndex = 0xFF00;

**static** Uint16 state = *LEFT*;

**if**(state == *LEFT*)

{

GpioDataRegs.GPATOGGLE.bit.GPIO14 = 1;

**float** sample = (**float**)(int16)McbspbRegs.DRR2.all;

sampleBuffer[currentBufIndex&SAMPLE\_BUFFER\_MASK\_OFF] = sample;

sampleBuffer[SAMPLE\_BUFFER\_OFFSET + (currentBufIndex&SAMPLE\_BUFFER\_MASK\_OFF)] = sample;

// McbspbRegs.DXR2.all = (int16)(10.0f\*firC(lpfWeights, (sampleBuffer+(SAMPLE\_BUFFER\_OFFSET + (currentBufIndex&SAMPLE\_BUFFER\_MASK\_OFF))), FILTER\_TAP\_NUM\_LPF));

// McbspbRegs.DXR2.all = (int16)(10.0f\*firASM(lpfWeights, (sampleBuffer+(SAMPLE\_BUFFER\_OFFSET + (currentBufIndex&SAMPLE\_BUFFER\_MASK\_OFF))), FILTER\_TAP\_NUM\_LPF));

// McbspbRegs.DXR2.all = (int16)(10.0f\*firC(bpfWeights, (sampleBuffer+(SAMPLE\_BUFFER\_OFFSET + (currentBufIndex&SAMPLE\_BUFFER\_MASK\_OFF))), FILTER\_TAP\_NUM\_BPF));

// McbspbRegs.DXR2.all = (int16)(firASM(bpfWeights, (sampleBuffer+(SAMPLE\_BUFFER\_OFFSET + (currentBufIndex&SAMPLE\_BUFFER\_MASK\_OFF))), FILTER\_TAP\_NUM\_BPF));

// McbspbRegs.DXR2.all = (int16)(10.0f\*firC(hpfWeights, (sampleBuffer+(SAMPLE\_BUFFER\_OFFSET + (currentBufIndex&SAMPLE\_BUFFER\_MASK\_OFF))), FILTER\_TAP\_NUM\_HPF));

McbspbRegs.DXR2.all = (int16)(firASM(hpfWeights, (sampleBuffer+(SAMPLE\_BUFFER\_OFFSET + (currentBufIndex&SAMPLE\_BUFFER\_MASK\_OFF))), FILTER\_TAP\_NUM\_HPF));

currentBufIndex = (currentBufIndex|SAMPLE\_BUFFER\_MASK\_ON)+1;

GpioDataRegs.GPATOGGLE.bit.GPIO14 = 1;

}

Uint16 dummy = McbspbRegs.DRR2.all;

dummy = McbspbRegs.DRR1.all;

McbspbRegs.DXR1.all = 0;

PieCtrlRegs.PIEACK.all = PIEACK\_GROUP6;

state ^= 1;

}

**Interrupt\_Utils.c**

// File: Interrupt\_Utils.c

// Date: 02/24/2018

// Name: Mark Schuster

// Class: EEE4511C (DSP)

**#include** "msLib.h"

**void** **initInterrupts**(**void**){

**InitPieCtrl**();

**InitPieVectTable**();

**EnableInterrupts**();

}

**LEDs\_switches\_Utils.c**

// File: LEDs\_switches\_Utils.c

// Date: 02/24/2018

// Name: Mark Schuster

// Class: EEE4511C (DSP)

**#include** "msLib.h"

**void** **initLEDSAndSwitches**(**void**){

// Allow write to protected regs.

EALLOW;

GpioCtrlRegs.GPAMUX1.all &= 0x0003FFF0;

GpioCtrlRegs.GPADIR.all |= 0x00007F80;

GpioCtrlRegs.GPAMUX1.all &= 0xFFFFF30F;

GpioCtrlRegs.GPAPUD.all &= 0xFFFFFF8B;

GpioCtrlRegs.GPADIR.all &= 0xFFFFFF8B;

EDIS;

**return**;

};

Uint16 **getSwitches**(**void**){

**volatile** Uint16 s = ((GpioDataRegs.GPADAT.all & 0x00000070) >> 3| GpioDataRegs.GPADAT.bit.GPIO2) & 0x000F;

**return** s;

}

**void** **setLEDS**(Uint16 newLEDVal){

newLEDVal = ~newLEDVal;

**volatile** Uint32 led = (GpioDataRegs.GPADAT.all & 0xFFFF807F) | (newLEDVal << 7);

GpioDataRegs.GPADAT.all = led;

**return**;

}

**OneToOneI2CDriver.c**

/\*

\* OneToOneI2CDriver.c

\*

\* Created on: Sep 24, 2016

\* Author: Raz Aloni

\*/

**#include** <DSP2833x\_Device.h>

**#include** "OneToOneI2CDriver.h"

/\* Ideal module clock frequency for I2C \*/

**static** **const** Uint16 IdealModClockFrqMHz = 12;

/\*

\* <summary>

\* Initializes the GPIO for the I2C

\* </summary>

\*/

**static** **void** **InitI2CGpio**();

/\*

\* <summary>

\* Calculates and sets the ClockDivides for the I2C Module

\* </summary>

\* <param="sysClkMhz">System Clock Frequency in Mhz</param>

\* <param="I2CClkKHz">Desired I2C Clock Frequency in KHz</param>

\*/

**static** **inline** **void** **SetClockDivides**(float32 sysClkMHz, float32 I2CClkKHz);

/\*

\* <summary>

\* Initializes the I2C to run in Master Mode for a One-To-One connection

\* </summary>

\* <param="slaveAddress">Address of the slave device to write to</param>

\* <param="sysClkMhz">System Clock Frequency in Mhz</param>

\* <param="I2CClkKHz">Desired I2C Clock Frequency in KHz</param>

\*/

**void** **I2C\_O2O\_Master\_Init**(Uint16 slaveAddress, float32 sysClkMhz, float32 I2CClkKHz)

{

// Init GPIO

InitI2CGpio();

EALLOW;

// Enable Clock for I2C

SysCtrlRegs.PCLKCR0.bit.I2CAENCLK = 1;

// Put I2C into Reset Mode

I2caRegs.I2CMDR.bit.IRS = 0;

// Set Slave Address

I2caRegs.I2CSAR = slaveAddress;

// Set Clocks

SetClockDivides(sysClkMhz, I2CClkKHz);

// Release from Reset Mode

I2caRegs.I2CMDR.bit.IRS = 1;

EDIS;

}

/\*

\* <summary>

\* Sends bytes via I2C

\* </summary>

\* <param="values">Pointer to array of bytes to send</param>

\* <param-"length">Length of array</param>

\*/

**void** **I2C\_O2O\_SendBytes**(Uint16 \* **const** values, Uint16 length)

{

// Set to Master, Repeat Mode, TRX, FREE, Start

I2caRegs.I2CMDR.all = 0x66A0;

// Write values to I2C

**for** (Uint16 i = 0; i < length; i++)

{

// Wait if Transmit is not ready

**while**(!I2caRegs.I2CSTR.bit.ARDY);

I2caRegs.I2CDXR = values[i];

}

// Stop Bit

I2caRegs.I2CMDR.bit.STP = 1;

}

/\*

\* <summary>

\* Calculates and sets the ClockDivides for the I2C Module

\* </summary>

\* <param="sysClkMhz">System Clock Frequency in Mhz</param>

\* <param="I2CClkKHz">Desired I2C Clock Frequency in KHz</param>

\*/

**static** **inline** **void** **SetClockDivides**(float32 sysClkMHz, float32 I2CClkKHz)

{

/\* Calculate Module Clock Frequency - Must be between 7-12 MHz

\* Module Clock Frequency = sysClkMhz/(IPSC + 1)

\*/

Uint16 IPSC = (Uint16)(sysClkMHz/IdealModClockFrqMHz);

/\* Calculate Divide Downs for SCL

\* FreqMClk = sysClkMHz/((IPSC + 1)[(ICCL + d) + (ICCH + d)])

\*

\* Assume an even clock size -> ICCH == ICCL

\* ICCL = ICCH = sysclkMHz/(2000 \* I2CClkKHz \* (IPSC + 1)) - d

\*/

// Find value for d

Uint16 d = 5;

**if** (IPSC < 2)

{

d++;

**if** (IPSC < 1)

{

d++;

}

}

Uint16 ICCLH = (Uint16)(1000 \* sysClkMHz/(2 \* I2CClkKHz \* (IPSC + 1)) - d);

// Set values

I2caRegs.I2CPSC.all = IPSC;

I2caRegs.I2CCLKL = ICCLH;

I2caRegs.I2CCLKH = ICCLH;

}

/\*

\* <summary>

\* Initializes the GPIO for the I2C

\* </summary>

\*/

**static** **void** **InitI2CGpio**()

{

EALLOW;

/\* Enable internal pull-up for the selected pins \*/

// Pull-ups can be enabled or disabled disabled by the user.

// This will enable the pullups for the specified pins.

// Comment out other unwanted lines.

GpioCtrlRegs.GPBPUD.bit.GPIO32 = 0; // Enable pull-up for GPIO32 (SDAA)

GpioCtrlRegs.GPBPUD.bit.GPIO33 = 0; // Enable pull-up for GPIO33 (SCLA)

/\* Set qualification for selected pins to asynch only \*/

// This will select asynch (no qualification) for the selected pins.

// Comment out other unwanted lines.

GpioCtrlRegs.GPBQSEL1.bit.GPIO32 = 3; // Asynch input GPIO32 (SDAA)

GpioCtrlRegs.GPBQSEL1.bit.GPIO33 = 3; // Asynch input GPIO33 (SCLA)

/\* Configure SCI pins using GPIO regs\*/

// This specifies which of the possible GPIO pins will be I2C functional pins.

// Comment out other unwanted lines.

GpioCtrlRegs.GPBMUX1.bit.GPIO32 = 1; // Configure GPIO32 for SDAA operation

GpioCtrlRegs.GPBMUX1.bit.GPIO33 = 1; // Configure GPIO33 for SCLA operation

EDIS;

}

**Timer1\_Utils.c**

// File: Timer1\_Utils.c

// Date: 02/24/2018

// Name: Mark Schuster

// Class: EEE4511C (DSP)

**#include** "msLib.h"

**void** **initTimer1**(**void**){

**InitCpuTimers**();

**ConfigCpuTimer**(&CpuTimer1, 150, 1E5);

EALLOW;

PieVectTable.XINT13 = timer1Isr;

IER |= M\_INT13;

EDIS;

CpuTimer1.RegsAddr->TCR.bit.TSS = 0;

}

**main.c**

// File: main.c

// Date: 03/25/201

// Name: Mark Schuster

// Class: EEE4511C (DSP)

**#include** <DSP2833x\_Device.h>

**#include** "DSP28x\_Project.h"

**#include** <AIC23.h>

**#include** <InitAIC23.h>

**#include** "msLib.h"

**extern** Uint16 audioState;

// Simple enum for bool values.

**enum**{

*FALSE*,

*TRUE*,

};

// Function prototypes:

Uint16 **main**(**void**)

{

// Disable the WDT, and init the phase lock loop.

**InitSysCtrl**();

initLEDSAndSwitches();

InitMcBSPb();

InitSPIA();

InitAIC23();

initCodec();

setLEDS(0x00);

SpiTransmit(CLKsampleratecontrol(SR48));

// initWeights(lpfWeights, FILTER\_TAP\_NUM\_LPF, 10.0);

// initWeights(bpfWeights, FILTER\_TAP\_NUM\_BPF, 20.0);

initWeights(hpfWeights, FILTER\_TAP\_NUM\_HPF, 10.0);

**for**(;;);

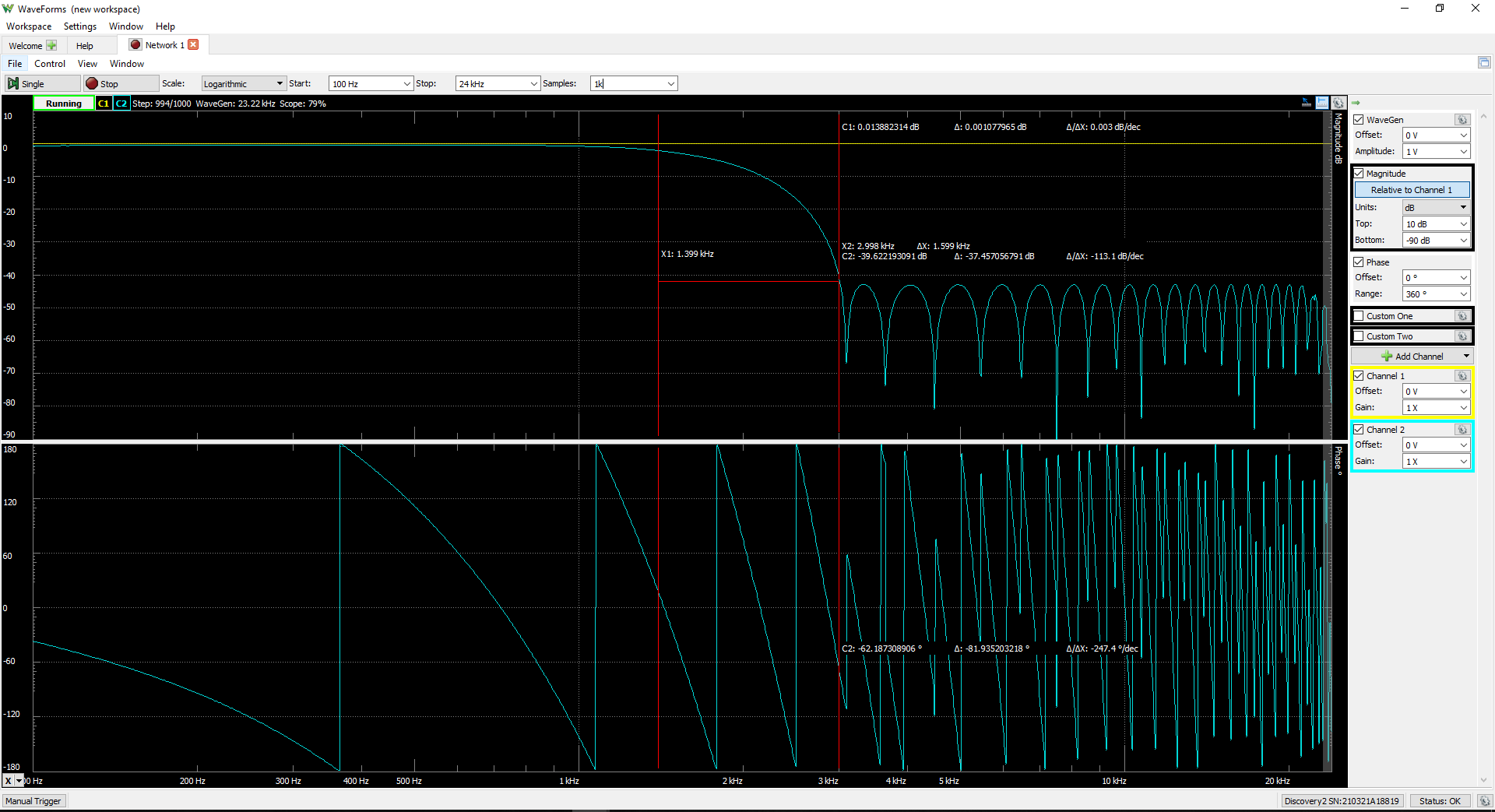
}

**1.5 Program Description:**

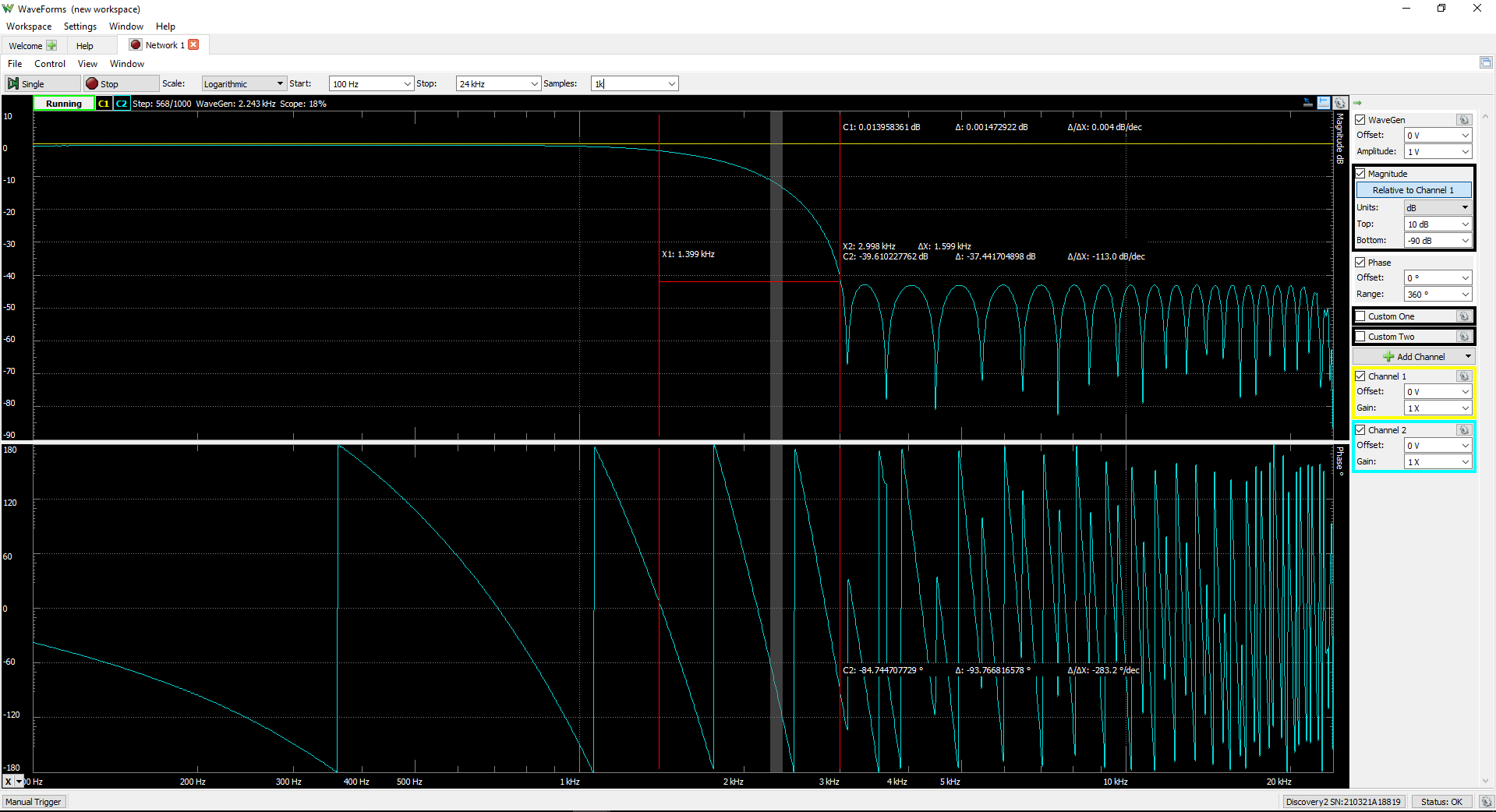
Take in audio and apply the respective weights for a given filter to the data, using a difference equation. This is done simple by multiplying each weight by its respective time delayed audio input.

**1.6 Question Answers:**

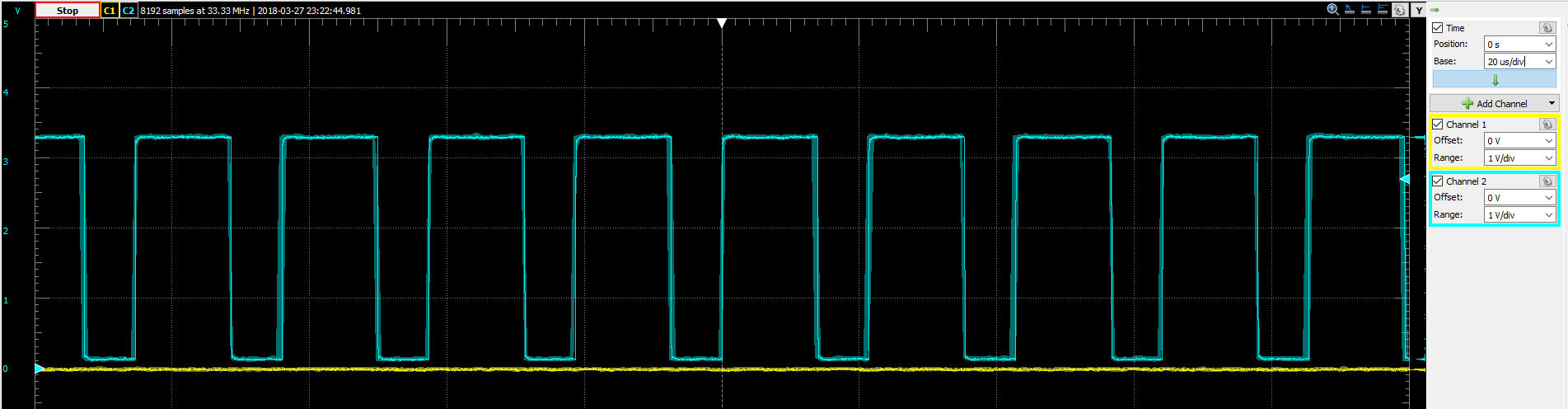
1.1 Screenshot of LPF in ASM:



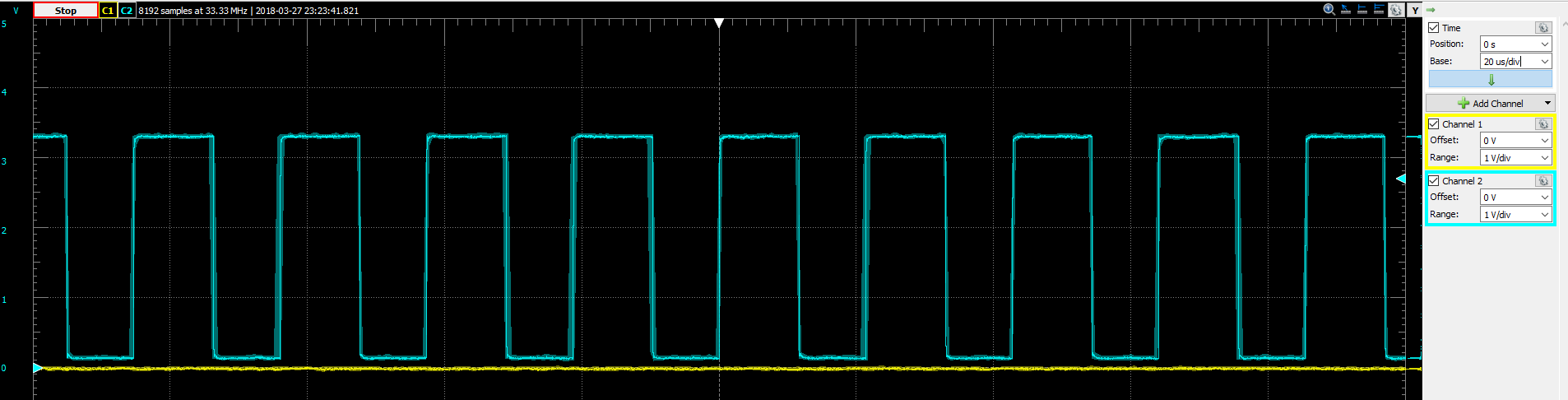
1.1 Screenshot of LPF in C:

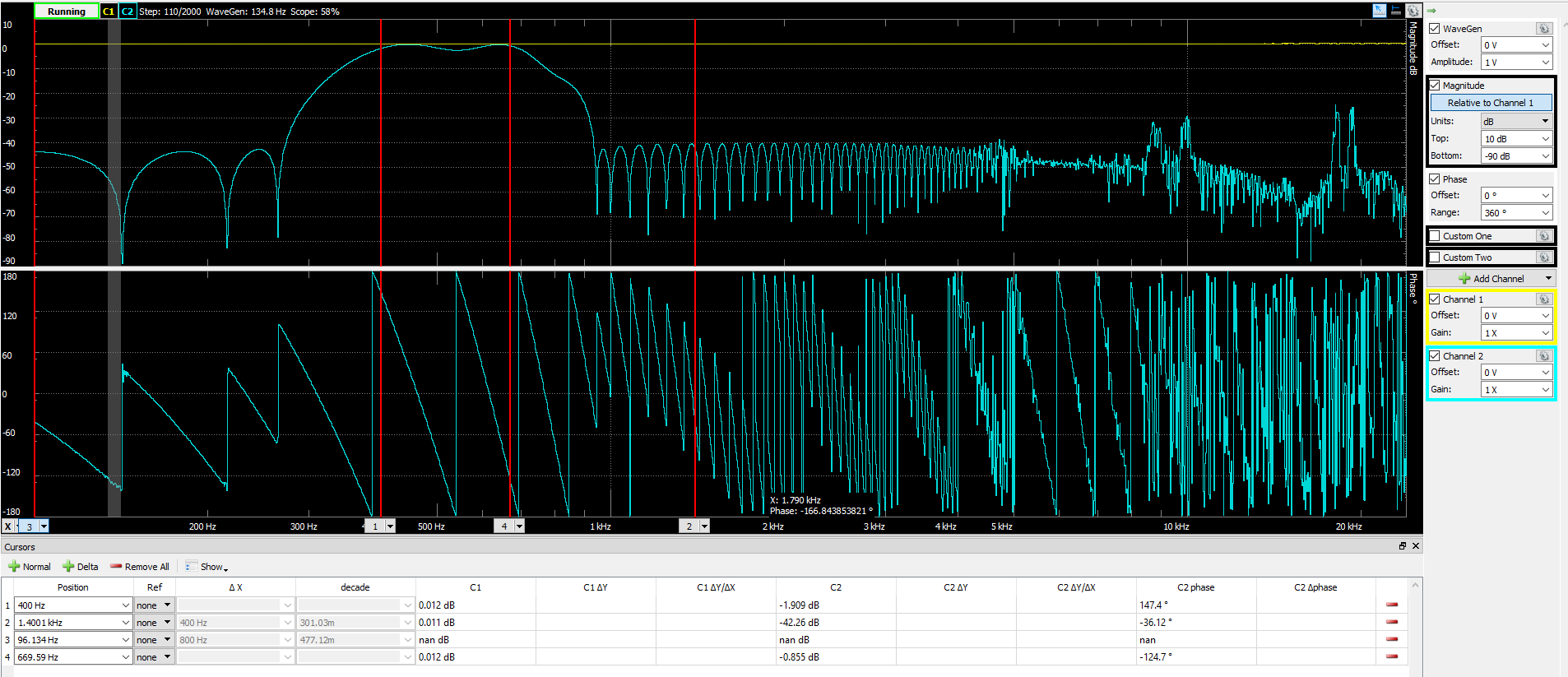


GPIO toggling LPF asm:

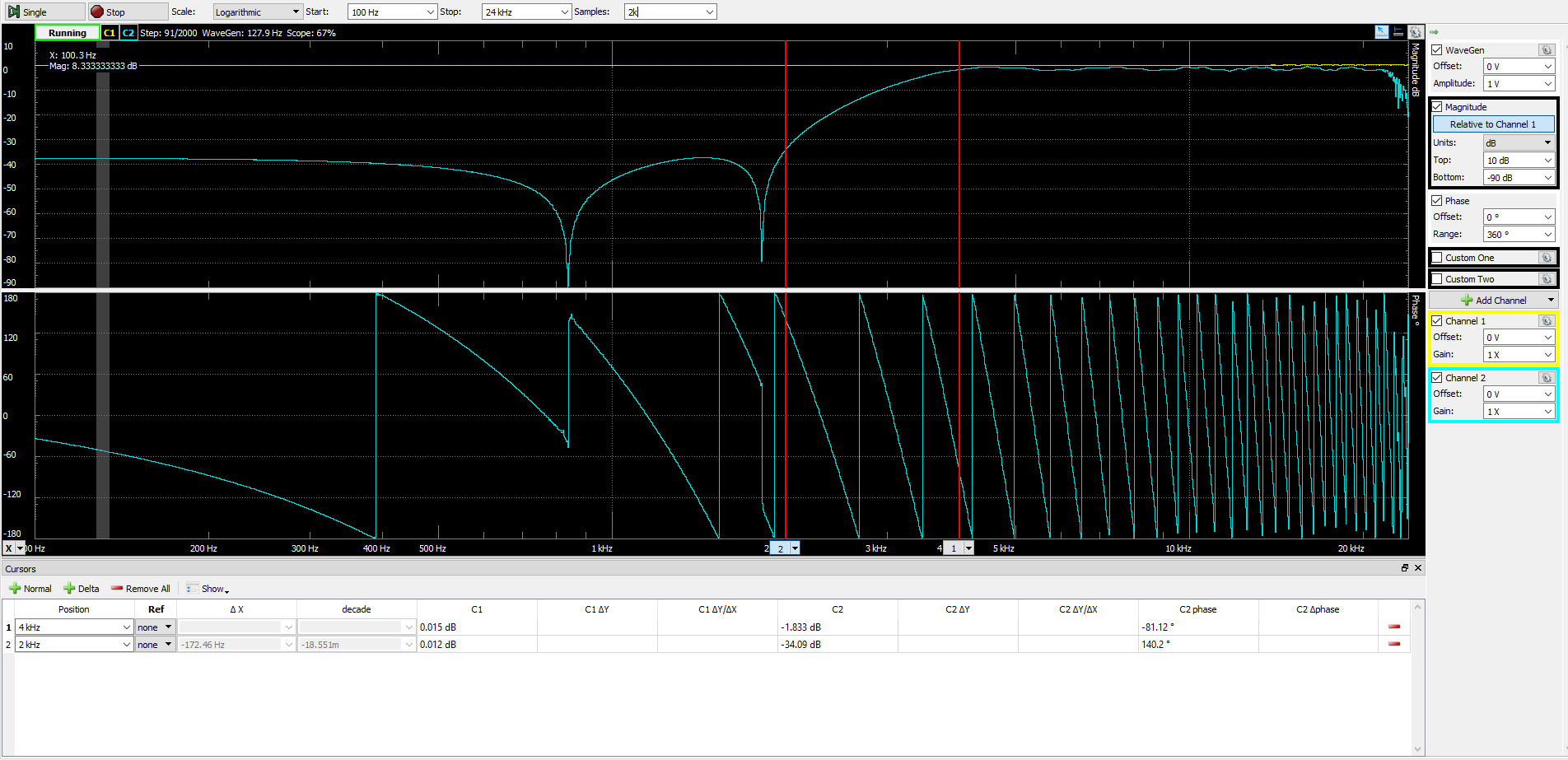


GPIO toggling LPF C



Screenshot of BPF in asm:

Screenshot of HPF in asm:



GPIO toggling HPF asm:

