Lab 2: FIR Filtering

Objective

The purpose of this lab is to compare the implementation of a finite impulse-response (FIR) filter using different structures.

Background

FIR filtering can be performed using various filtering structures. Among these are the direct form I and lattice structures. Each has advantages and disadvantages when considering number of arithmetic operations, memory requirements, and round-off error.

(Remember to include all of your C code in the write-up.)

Procedure

1. A 32 weight FIR filter using floating-point weights is given in the file:

   `firfilt32.dat`

   The sampling frequency is 48 kHz. Check the frequency response of the filter using **Matlab** and graph it in dB using **Matlab** or another graphing program and print out for future reference.

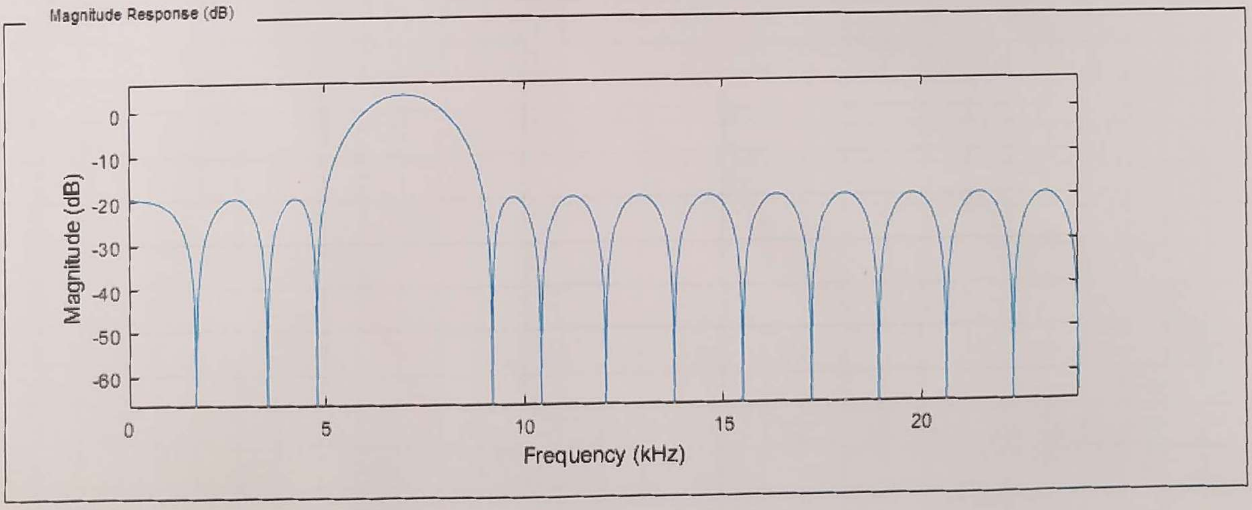


Figure 1 Matlab 32 tap FIR filter magnitude

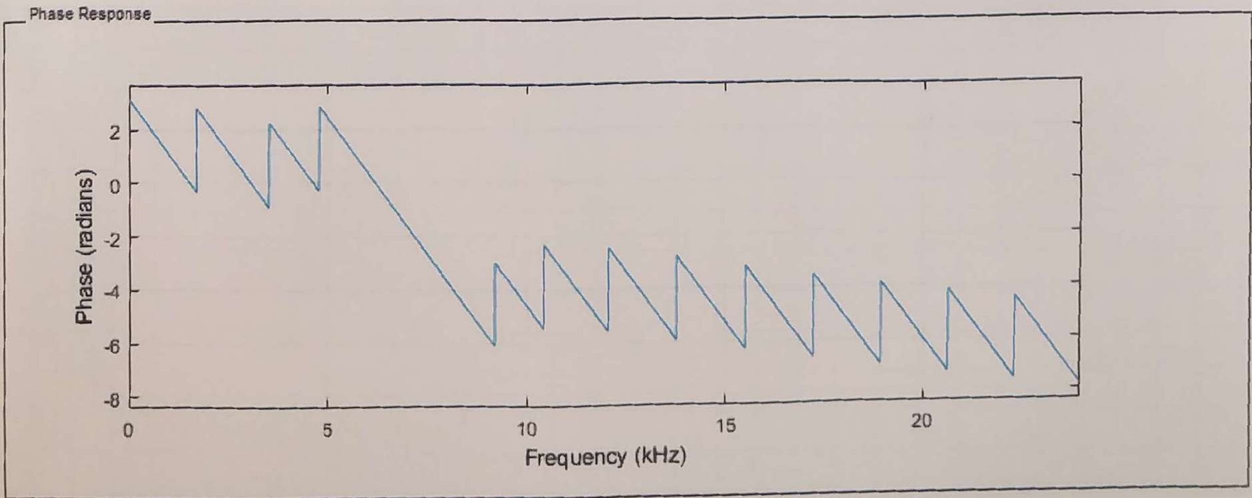


Figure 2 FIR Matlab 32 tap FIR phase response

1/23/19

2. Write a C program that implements a direct form I FIR filter. Your program should be able to filter with the filter given in procedure 1. For example, the following code could be included in the file:

```
.
.
.
/* FIR filter coefficients */
float coeffs[] = {coeff1, coeff2, ... , coeff32};
.
.
.
main()
{
.
.
.
```

The files to create a project in *Code Composer Studio* (CCS) are provided in the `.zip` archive `lab-files.zip` containing `template1.c`, `template1.h`, `aic3106.c`, `mcasp.c`, and `BoardConfig.ccxml`. The file `template1.c` chould be modified to uncomment the statments which read the samples from the `line in` port.

To make sure you optimize the computations necessary between samples, try to implement a circular buffer in software *without* using the "C" modulo operator. (Hint: try masking the indexing variable.) Do *not* unroll the loops of the filter code but optimize the "C" code enough to allow the filter to run as fast as possible without moving the input samples in the delay line each time a new sample is received.

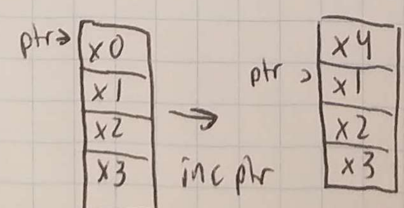
After you have debugged for correct functionality, you may need to turn on the compiler optimizer to get your code to run fast enough. This is easy to do; just select the `release` build configuration in CCS and be sure to configure like you did in the "Initial Code Composer Studio Setup" section of Lab 1.

Save and print out the assembly code created by the compiler.

masking example uint 32 value
value=0x01020304
byte1 = (value >> 24) //0x01
byte2 = (value >>16) & 0xFF or (value & 0xff0000) >>16 //0x0102
//& 0x00FF
//= 0x0002

to increment ~~value += ... 0x1~~ act like value % 32
value = (value + 0x1) & 0x1F

xbuff[32] ~~xptr = &xbuff~~



during FIR
decrement ptr

1/23/19

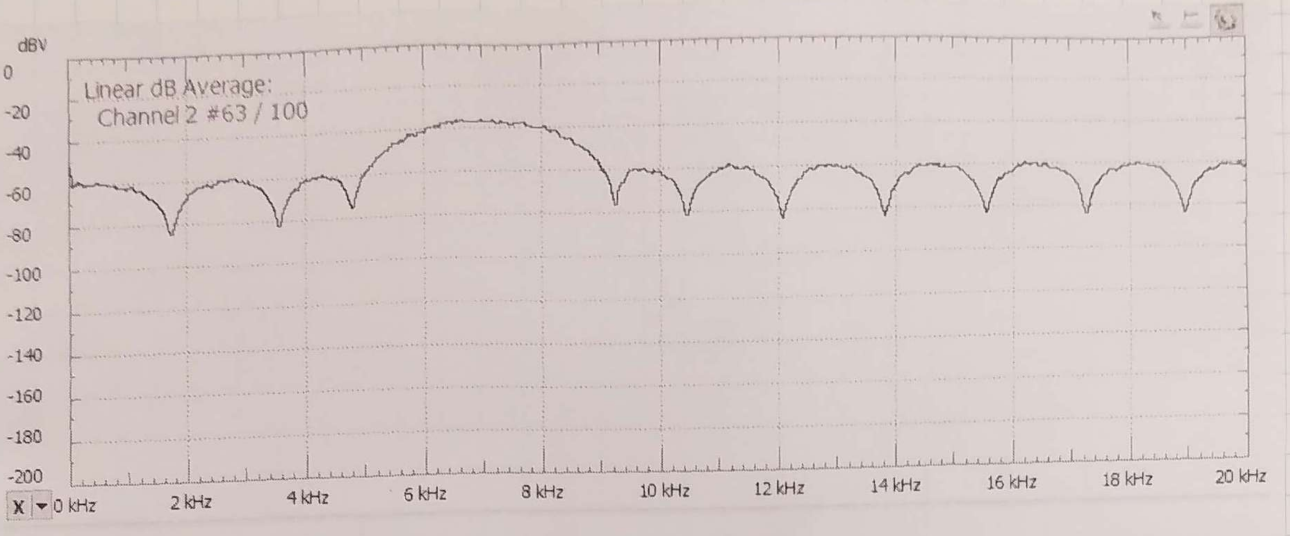


Figure 3 Ladder FIR FFT

Had some trouble getting the signal to ~~output~~ read/output. Had to look at data sheet to find out I needed to use RBUF and RRDY instead of XBUF and XRDY. The read buffer was also buffer 12 with transmit being buffer 11.

Code found on next page.

2/7/19

10

```
;******************************************************************************
;* FUNCTION NAME: FIR                                                         *
;*                                                                            *
;*   Regs Modified     : A0,A3,A4,A5,B0,B4,B5,B6,B7,SP,B31                    *
;*   Regs Used         : A0,A3,A4,A5,B0,B3,B4,B5,B6,B7,SP,B31                 *
;*   Local Frame Size  : 0 Args + 24 Auto + 0 Save = 24 byte                  *
;******************************************************************************
_FIR:
;** --------------------------------------------------------------------------*
        .dwcfi  cfa_offset, 0
        .dwcfi  save_reg_to_reg, 228, 19
           SUB     .D2     SP,24,SP          ; |136|
        .dwcfi  cfa_offset, 24
$C$DW$46    .dwtag  DW_TAG_variable, DW_AT_name("index")
        .dwattr $C$DW$46, DW_AT_TI_symbol_name("_index")
        .dwattr $C$DW$46, DW_AT_type(*$C$DW$T$42)
        .dwattr $C$DW$46, DW_AT_location[DW_OP_breg31 4]
$C$DW$47    .dwtag  DW_TAG_variable, DW_AT_name("xBuff")
        .dwattr $C$DW$47, DW_AT_TI_symbol_name("_xBuff")
        .dwattr $C$DW$47, DW_AT_type(*$C$DW$T$45)
        .dwattr $C$DW$47, DW_AT_location[DW_OP_breg31 8]
$C$DW$48    .dwtag  DW_TAG_variable, DW_AT_name("yVal")
        .dwattr $C$DW$48, DW_AT_TI_symbol_name("_yVal")
        .dwattr $C$DW$48, DW_AT_type(*$C$DW$T$17)
        .dwattr $C$DW$48, DW_AT_location[DW_OP_breg31 16]
$C$DW$49    .dwtag  DW_TAG_variable, DW_AT_name("i")
        .dwattr $C$DW$49, DW_AT_TI_symbol_name("_i")
        .dwattr $C$DW$49, DW_AT_type(*$C$DW$T$10)
        .dwattr $C$DW$49, DW_AT_location[DW_OP_breg31 24]

           MV      .L1X    B4,A3             ; |136|
||         STB     .D2T1   A4,*+SP(4)        ; |136|

           STW     .D2T1   A3,*+SP(8)        ; |136|
        .dwpsn  file "../lab_files/lab2.c",line 137,column 12,is_stmt
           ZERO    .L1     A5:A4             ; |137|
           STDW    .D2T1   A5:A4,*+SP(16)    ; |137|
        .dwpsn  file "../lab_files/lab2.c",line 139,column 9,is_stmt
           ZERO    .L2     B4                ; |139|
           STW     .D2T2   B4,*+SP(24)       ; |139|
        .dwpsn  file "../lab_files/lab2.c",line 139,column 14,is_stmt
           MVK     .S2     32,B5             ; |139|
           CMPLT   .L2     B4,B5,B0          ; |139|
   [!B0]   BNOP    .S1     $C$L11,5          ; |139|
           ; BRANCHCC OCCURS {$C$L11}        ; |139|
;*----------------------------------------------------------------------------*
;*   SOFTWARE PIPELINE INFORMATION
;*      Disqualified loop: Software pipelining disabled
;*----------------------------------------------------------------------------*
$C$L10:
$C$DW$L$_FIR$2$B:
        .dwpsn  file "../lab_files/lab2.c",line 141,column 9,is_stmt
           LDB     .D2T2   *+SP(4),B5        ; |141|
           LDW     .D2T2   *+SP(8),B6        ; |141|
           MVKL    .S2     _filterCoeff,B31
```

2/7/19
M

```
        MVKH    .S2     _filterCoeff,B31
        LDW     .D2T2   *+B31[B4],B4        ; |141|
        NOP             1
        LDH     .D2T2   *+B6[B5],B6         ; |141|
        NOP             4
        INTSP   .L2     B6,B5               ; |141|
        LDDW    .D2T2   *+SP(16),B7:B6      ; |141|
        NOP             2
        MPYSP   .M2     B4,B5,B4            ; |141|
        NOP             3
        SPDP    .S2     B4,B5:B4            ; |141|
        NOP             1
        ADDDP   .L2     B5:B4,B7:B6,B5:B4 ; |141|
        NOP             6
        STDW    .D2T2   B5:B4,*+SP(16)      ; |141|
    .dwpsn file "../lab_files/lab2.c",line 139,column 29,is_stmt
        LDW     .D2T2   *+SP(24),B5         ; |139|
        LDB     .D2T2   *+SP(4),B4          ; |139|
        NOP             4

        ADD     .L2     1,B5,B4             ; |139|
||      SUB     .S2     B4,1,B5             ; |139|

        STW     .D2T2   B4,*+SP(24)         ; |139|
||      EXTU    .S2     B5,27,27,B4         ; |139|

        STB     .D2T2   B4,*+SP(4)          ; |139|
    .dwpsn file "../lab_files/lab2.c",line 139,column 14,is_stmt
        LDW     .D2T2   *+SP(24),B4         ; |139|
        MVK     .S1     32,A3               ; |139|
        NOP             3
        CMPLT   .L1X    B4,A3,A0            ; |139|
   [ A0] BNOP   .S1     $C$L10,5            ; |139|
        ; BRANCHCC OCCURS {$C$L10}          ; |139|
$C$DW$L$_FIR$2$E:
;** --------------------------------------------------------------------------*
$C$L11:
    .dwpsn file "../lab_files/lab2.c",line 143,column 5,is_stmt
        LDDW    .D2T1   *+SP(16),A5:A4      ; |143|
        NOP             4
        DPSP    .L1     A5:A4,A4            ; |143|
        NOP             3
    .dwpsn file "../lab_files/lab2.c",line 144,column 1,is_stmt
        ADDK    .S2     24,SP               ; |144|
    .dwcfi cfa_offset, 0
    .dwcfi cfa_offset, 0
$C$DW$50    .dwtag  DW_TAG_TI_branch
    .dwattr $C$DW$50, DW_AT_low_pc(0x00)
    .dwattr $C$DW$50, DW_AT_TI_return
        RETNOP  .S2     B3,5                ; |144|
        ; BRANCH OCCURS {B3}                ; |144|
```

2/7/19

NJ

12

4. Design a lattice filter using reflection coefficients computed from the filter weights from procedure 1. The computation of the reflection coefficients can be done in a program on the host (do *not* use `Matlab` to compute the reflection coefficients) or in the 'C67 program itself. You can check that the coefficients are correct by comparing your computed values to coefficients computed by `Matlab`. Implement the lattice filter on the 'C67. (Note that a circular buffer is *not* needed for the lattice filter.) Print the frequency response.

   Remember that the derivation of the lattice filter in your book assumes the first filter coefficient, α_0, is 1. Since your filter (probably) does not meet this condition, you can add 1.0 to the beginning of the filter coefficients, and subtract the input from the output and produce a 1-sample delayed FIR output.

   Again, save and print out the assembly code created by the compiler.

```
int main(void)
{
    int16_t xBuff[FILTER_SIZE] = { 0 };
    int16_t newVal=0;

    float gDelayed[FILTER_SIZE] = {0};
    float g[FILTER_SIZE]={0};
//    int8_t xIndex = 0x0;
    float yVal = 0;                // Sine Sample
    float k,gDel;
    int16_t temp;
    int8_t i=0;

#ifdef LATTICE
    float alpha[FILTER_SIZE], beta[FILTER_SIZE];
    float kVal[FILTER_SIZE] = { 0 };
    memcpy(alpha,filterCoeff,sizeof(filterCoeff));
    ComputeK(alpha, FILTER_SIZE-1,beta, kVal);
#endif

    Init();


    // Infinite loop:    Each loop reads/writes one sample to the left and right
channels.
    while (1)
    {
        //write out y value
#ifndef LATTICE
        xBuff[x0Index] = newVal;
        yVal = FIR(x0Index, xBuff);
        x0Index = (x0Index + 0x1) & 0x1F;
#else

        g[0]=newVal;
        yVal=newVal;
//
        for(i=1; i<FILTER_SIZE/2; i++){
```

2/7/19
MV

```
            k=kVal[i];
            gDel=gDelayed[i-1];
            g[i]=k*yVal+gDel;
            yVal=yVal+k*gDel;
        }

#endif
        while (!CHKBIT(MCASP->SRCTL12, RRDY)) { }
            temp = MCASP->RBUF12;  // read next value from left channel
        while (!CHKBIT(MCASP->SRCTL11, XRDY)) { }
            MCASP->XBUF11 = 0;
            // write output to left channel
#ifdef LATTICE
            for(i=FILTER_SIZE/2; i<FILTER_SIZE; i++){
                        k=kVal[i];
                        gDel=gDelayed[i-1];
                        g[i]=k*yVal+gDel;
                        yVal=yVal+k*gDel;
                    }
            memcpy(gDelayed,g,sizeof(g));
            yVal=yVal-newVal;
#endif
        while (!CHKBIT(MCASP->SRCTL11, XRDY)) { }
          MCASP->XBUF11 =  (int16_t) yVal; //(int16_t) xBuff[xIndex];
        while (!CHKBIT(MCASP->SRCTL12, RRDY)) { }
            newVal=MCASP->RBUF12;
                // write 0 to right channel


    }
}

void ComputeK(float *alpha, int m, float *beta, float *k)
{
    int i;

    k[m] = alpha[m];
    if (m == 0)
    {
        return;
    }
    //reverse coeff for beta
    for(i=0;i<=m;i++){
        beta[i]=alpha[m-i];
    }
//    beta=memcpy(&beta[0],&alpha[0],FILTER_SIZE*sizeof(float));

    //computes z^i coefficients of A_(m-1)
    for (i = 0; i < m; i++)
    {
        alpha[i] = (alpha[i] - k[m] * beta[i])
                / (1 - k[m] * k[m]);
    }
    //compute K_(m-1)
    ComputeK(alpha, m - 1,beta,k);
}
```

Had some issues with the lattice Filter. I originally over thought how to compute $y_{m-1}(n)$ and $g_{m-1}(n)$. Made a Recursive function that called itself ~~too~~ twice. This was too slow with too many recursive steps. The new version of the lattice filter (used in the above code) works much better and simpler. While the logic for the lattice filter was correct, I ~~could~~ could not get any output. It was found that when ComputeK was called, The output buffer would stop functioning

2/7/19

AV

~~Various~~: It was found that when I tried accessing an array inside my function, memory got overwritten somewhere (I think). When only doing iteration 2-3 times, ~~the~~ I would get output, but any higher and the output would be zero. I found that moving my ComputeK function before I initiallized the board fixxed the issue.

The next problem was the Lattice code wasn't running fast enough. If I ran the board at 24kHz sampling rate, I would get the correct output, but all the freq were cut in half. The code needed to be optimized to run faster. The loop for computing k and g were split in half with transmits in between

Lattice 0 - length/2

~~out~~ left output = 0

Lattice length/2 - length

right output = yval

by spitting up the computations, the filter was able to function at the desired 48 kHz

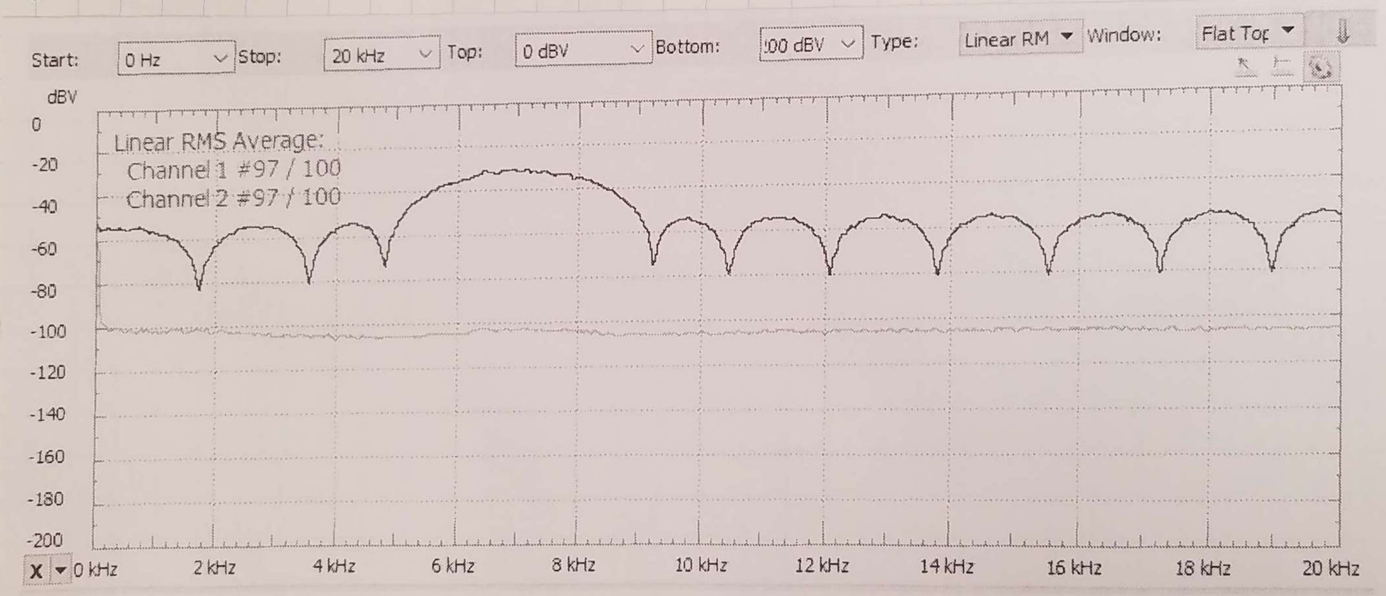


Figure 4 Lattice FIR FFT

Questions

1. What are the memory requirements and computational requirements of the two implementations in number of locations and number of computations?

Ladder N = filter size

memory: N adds: N-1 mult: N

Lattice

memory: 2·N adds: ~~2·N~~ 2(N-1) mult: 2(N-1)

G, G(n-1)

2/7/19

Ny

2. How do the two frequency responses compare?

The frequency Responses appear to be ~~the~~ identical. This is expected as the K values were designed to match the FIR Filter

3. Which do you think is the "best" implementation. Why?

Unless there are specific numerical ~~reson~~ reasons for Lattice. The ladder filter is faster. there are less computations and it is easier to implement.

4. Look at the assembly code. Do you think you could optimize it to run more efficiently? How?

There are some NOP between some instructions where additional computations could be done. If it is possible to perform some steps in parallel, it could be faster, F and G could be computed at the same time in this case.

2/7/19

M