Lab Project: Decimation and Interpolation, Multirate FIR filters [1]

Student group:	Professors comment:	Record editor:
Date:		Group members:
Professor:		

1 Objectives

The purpose of this lab is to understand decimation and interpolation using a constant factor M to implement multi-rate digital filters on the UniDAQ2 board. The UniDAQ2 board uses a Texas Instruments TMS320C6747 DSP. We will however use fixed-point representation with 16 bit only because most implementations are done with fixed-point arithmetic due to cost reasons.

After this laboratory you should be able to

1. design an FIR **high-pass/low-pass** branching filter according to a given specification
2. design a branching filter consisting of a linear phase FIR **high -pass or low-pass** implemented as multirate filter with factor L=M
3. find the optimum factor for M to achieve maximum data throughput
4. design a delay cascade consisting of K delays -|K·T|- where the output of the **high -pass or low-pass** is subtracted from the delay cascade, resulting in the desired branching filter
5. simulate the filters in MATLAB
6. implement the filters on a DSP (in combination with an assembly-coded FIR module)

2.1 Abbreviations

ADC	Analogue-to-Digital converter
CCS	Code Composer Studio (currently ver. 5.5 is used)
CODEC	A chip containing an ADC and a DAC
DAC	Digital-to-Analogue converter
R&S UPV	An audio spectrum analyzer made by Rohde & Schwarz
TI	Texas Instruments
UniDAQ2 board	A versatile data acquisition and signal processing system for industrial applications, scientific research and education from the company d.signT

2.2 Material for the lab

1. You do not have to set up a new project for the implementation. As known from the tutorial, a UniDAQ2 project is available in **d:\ti_work**.
2. Some helpful M-files like inputs(..), write_coeff(..) can be found in MOODLE, see MES_MATLAB_Lecture_Examples
3. Use Getting Started again for handling UniDAQ2 board and CCS.

3 Description of the lab tasks

3.1 “Conventional” FIR Filter design using MATLAB

A **high-pass/low-pass** FIR branching filter shall be designed using REMEZ MATLAB functions firpmord(…) and firpm(…) with

- $F_{s,in} = F_{s,out} =$ 50 kHz
- stop-band edge frequency of high-pass : 3100 Hz
- pass-band edge frequency of high-pass : 3350 Hz,
- (pass-band edge frequency of low-pass : 3100 Hz, see high-pass above)
- (stop-band edge frequency of low-pass : 3350 Hz, see high-pass above)
- pass-band ripple : 0.01 (absolute scale, NOT in dB)
- minimum stop-band attenuation : 40 dB

Use the MATLAB script “**`dec_kernel_int.m`**” and extend the script according to the assignments below. The amplitude response for the “normal” design is shown in **Fig. 3.1a,b**.

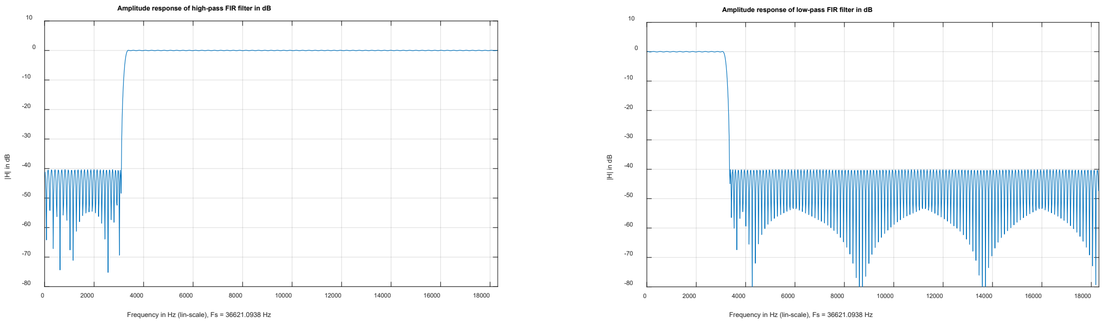


Fig. 3.1a, b Amplitude response of **desired FIR high-pass/low-pass filters**

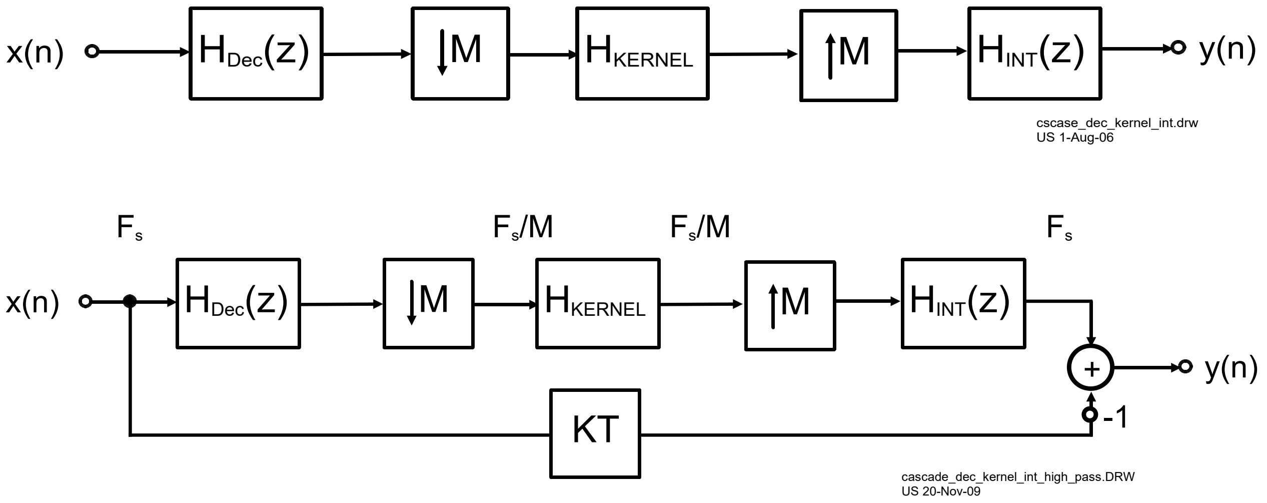
Answer in ***Attachment A:***

- How many cycles does the DSP allow per sampling period ? ($F_{c,DSP}$ = 456 MHz)
- What is the filter degree of the FIR filter? (is displayed by the MATLAB script)

This FIR filter can probably not be implemented on the TI DSP if no optimization is applied (using the ANSI C module and/or the assembler module for the FIR filter, as used in the tutorial). **We therefore use the Multi-rate filter approach, see next chapter.**

3.2 Multi-rate FIR Filter design using MATLAB

Fig. 3.2a, b shows the signal-flow diagram for a multi-rate filter, which consists of a decimation filter H_{DEC}, a KERNEL filter H_{KERNEL} and an interpolation filter H_{INT}. Fig. 3.2a is used for the low-pass filter type while Fig. 3.2b is used for the high-pass filter type. “K” is an integer, thus KT acts like a delay line. It has to be determined during this lab, how large K has to be.
The decimation filter and the interpolation filter have the same specification and are thus identical with respect to their coefficients. However, the delays of these filters require different variables/memory !



cscase_dec_kernel_int.drw
US 1-Aug-06

cascade_dec_kernel_int_high_pass.DRW
US 20-Nov-09

Fig. 3.2a, b Multi-rate filter arrangements with down-sampling factor/up-sampling factor M=L

Extend the MATLAB script "**`dec_kernel_int.m`**":
From the specification given above compute the optimum value for M_{min} . Using M_{min}, select for M a value which is a power of 2 and which is closest to M_{min} . Write down the exact value for M_{min} determined by MATLAB in ***Attachment C***.

The decimation filter and the interpolation filter are identical. Thus, only the decimation filter and the kernel filter have to be designed. The pass-band ripple of each filter is 1/3 of the total ripple (which is quite pessimistic).

Regardless of what you have determined for M_{min} above, use $M=M_{min} = 4$ throughout the remainder of this lab assignment.

Answer also in ***Attachment B:***

- What is the pass-band edge frequency of the decimation filter ?
- What is the stop-band edge frequency of the decimation filter ?
- What is the minimum stop-band attenuation of the decimation filter ?

- What is the pass-band edge frequency of the KERNEL filter ?
- What is the stop-band edge frequency of the KERNEL filter ?
- What is the minimum stop-band attenuation of the KERNEL filter ?

Design the filter in MATLAB using REMEZ algorithm via firpm(…) and plot the amplitude responses of the decimation filter and of the KERNEL filter. Write the coefficient values using "write_coeff.m" to the header file "dec_kernel_int.h". Some preparations have already been done in "**`dec_kernel_int.m`**".

Show in ***Attachment C:***

- your final MATLAB code for "**`dec_kernel_int.m`**".
- the amplitude response of the decimation filter determined with MATLAB

- the amplitude response of the KERNEL filter (MATLAB)
- the amplitude response of the cascade of the decimation filter, KERNEL filter and interpolation filter. Note that the KERNEL filter runs on Fs/M !

Answer in ***Attachment D:***

- What is the degree of the decimation filter /interpolation filter?
- What is the degree of the KERNEL filter?
- How much effort $E_{tot,cascade}$ is required per second for the cascade of the three filters if $\mathbf{M_{min} = 4}$ is used?
- Determine the ratio between the $E_{tot,cascade}$ in ***Attachment D*** and E_{tot} in ***Attachment A, question 3*** in order to find out which implementation is more efficient (Remark : The result is quite optimistic. The overhead for the control of the decimation/interpolation has not been considered).
- Still, what do you think: Can the three FIR filters be implemented on that DSP if no optimization is applied (using the ANSI C module and/or the assembler module for the FIR filter, as used in the tutorial)?

3.3 Multi-rate FIR Filter implementation on UniDAQ2 board

The code snippet **`decimate_by_5_FIR_1_stage_LAB.txt`** (see Annex A) can serve as starter for this lab task. The KERNEL filter and interpolation filter are of course not implemented yet. The decimation filter is the modified version (for **M=4** of the decimation filter with M=5 discussed before and shown in Annex A. The interpolation filter can be derived from the modified decimation filter with **M=4**. The KERNEL filter is a “normal” FIR filter, however running at Fs/M.

The decimation filter, KERNEL filter and interpolation filter have to be called in a certain sequence in order to work properly.

Answer in ***Attachment E:***
Develop the structure for the polyphase representation of the decimation and interpolation filter. Note that for the filters the same coefficients may be used. Be aware that different variable names / different memory for the delays is **required** ! (use e.g. the variable names “H_polyphase_filt_0_delays_**DEC**”, “H_polyphase_filt_0_delays_**INT**”). It is sufficient to draw polyphase component “0” in detail and to use “block representations” for components 1,…, 3 .
Answer in ***Attachment F:***

- How many samples of the input signal must be read from the ADC before **one** output sample of the decimation filter can be produced?
- What is therefore the delay between input and output of the multi-rate filter?
- When does the KERNEL filter have to be computed?
  What are the consequences for the balancing of the computations over $\mathbf{M_{min} = 4}$ samples?

Answer in ***Attachment G:***

- Fill in the following table which show what has to happen during a sequence of n=0, 1, 2, 3 input samples read by the ADC.

Note: The table is available as a WORD document in the ZIP files in EMIL.

Note also that the cascade of decimation filter, KERNEL filter and interpolation filter is actually now a single-rate filter implementation. The output signal of the decimation filter is generated only every Fs/M samples but this output is not sent to the DAC.

sample n	**Tasks carried out during this sample n**
0	compute polyphase component #0 of decimation filter using fir_filter( ) function, input is from ADC, output is ... compute polyphase component #0 of interpolation filter using fir_filter( ) function, input is from …, output is sent to DAC
1	compute polyphase component #... of decimation filter using fir_filter( ) function, input is from ADC, output is ... compute polyphase component #... of interpolation filter using fir_filter( ) function, input is from …, output is sent to DAC
2	
3	

Tab. 3.3.1 Timing chart for multi-rate filter for one sequence of samples **n=0,…, 3**

From the timing chart in Tab. 3.3.1, the C-code for the multi-rate filter can be derived. Indicate **clearly** in your modified Tab. 3.3.1 where the KERNEL filter has to be computed and where the interpolation filter has to be computed.

Use "no optimization" for the multi-rate filter. Measure the amplitude response of the multi-rate filter using the UPV analyzer (UPV set file **`dec_kernel_int.set`**) . Use ADC 0 input and DAC 0 (low-pass) and DAC 1 (high-pass) output for the branching filter!
Show the plot of the amplitude response of the multi-rate branching filter in ***Attachment H***.

Show in ***Attachment I***

- the content of the file FIR_normal.h (for the FIR filter according to 3.1)
- the content of the file dec_kernel_int.h (for the three filters, Decimation, Kernel and Interpolation filter according to 3.2/3.3)

Show in ***Attachment K*** the **documented** ANSI C source code for the complete dec_kernel_int.c file with Decimation, Kernel and Interpolation filter.

3.4 Multi-rate FIR Filter using arrays of pointers

For a larger value of M, the ADC INT routine is very long. It consists of an if-else-if sequence with M branches, for **M=4** with **4** branches. This makes programming error-prone because the correct index has to be used in every branch.

A much easier way is to use four additional arrays of pointers, two for the delays and two for the coefficient sets of the polyphase branches.

```
#define MM 4
short *p2p_H_polyphase_filt_DEC[MM]; // is an array of MM pointers
// H_polyphase_filt_0_delays_DEC contains the start address for the delays
// for the first polyphase branch in Fig. 3.2, which belongs to coefficient
// set {b_0, b_5, b_10}
// store this address in array of pointers, element [0]
p2p_H_polyphase_filt_DEC[0] = H_polyphase_filt_0_delays_DEC;
p2p_H_polyphase_filt_DEC[1] = H_polyphase_filt_3_delays_DEC;
…

short *p2p_H_polyphase_filt_INT[MM]; // is an array of MM pointers
// H_polyphase_filt_0_delays_INT as above for Interpolation Filter
p2p_H_polyphase_filt_INT[0] = H_polyphase_filt_0_delays_INT;
p2p_H_polyphase_filt_INT[1] = H_polyphase_filt_1_delays_INT;
p2p_H_polyphase_filt_INT[2] = H_polyphase_filt_2_delays_INT;
…

short *p2p_coe_polyphase_filt_DEC[MM]; // is an array of MM pointers
// p2p_coe_polyphase_filt_DEC[0] contains the start address of the coefficient
// set {b_0, b_5, b_10} for the first polyphase branch in Fig. 3.2
p2p_coe_polyphase_filt_DEC[0] = coe_polyphase_filt_b0;
p2p_coe_polyphase_filt_DEC[1] = coe_polyphase_filt_b3;
…

short *p2p_coe_polyphase_filt_INT[MM]; // is an array of MM pointers
// p2p_coe_polyphase_filt_INT[0] as above for Interpolation filter
p2p_coe_polyphase_filt_INT[0] = coe_polyphase_filt_b0;
p2p_coe_polyphase_filt_INT[1] = coe_polyphase_filt_b1;
…
```

Inside ADC INT routine, it is now possible to replace **all** if-branches by a single instruction sequence like

```
//interrupt service routine ADC
interrupt void adcInt (void)
{
    for (idx=0; idx<16; idx++)
        {
          sData[idx] = PRU_addaRegs->adc[idx];
        }
…
        y1_DEC_add += FIR_filter(
            p2p_H_polyphase_filt_DEC[count_ADC_INT],
            p2p_coe_polyphase_filt_DEC[count_ADC_INT],
            N_COE_PER_BRANCH,
            delays[count_ADC_INT]);
        delays[count_ADC_INT] = sData[0];

        y1_INT_out = FIR_filter(
            p2p_H_polyphase_filt_INT[count_ADC_INT],
            p2p_coe_polyphase_filt_INT[count_ADC_INT],
            N_COE_PER_BRANCH,
            T_KERNEL_out);
…
}
```

Copy your file **dec_kernet_int.c** to **dec_kernel_int_M_flex.c**, add this one instead to the project and make the required modifications in your code in order to be able to use the software construct shown above using arrays of pointers.
Show in ***Attachment M***

- The **documented** ANSI C source code for the complete **dec_kernel_int_M_flex.c** file using arrays to pointers
- Show that the programs with and without arrays of pointers behave the same way.

ANNEX A C-Code of the decimation filter for M = 5

This part shows the ADC routine only. Note that this version **still contains delays** in all except the first branch of the polyphase components.

```
interrupt void adcInt (void)   //interrupt service routine ADC
{
       for (idx=0; idx<16; idx++)
          {
             sData[idx] = PRU_addaRegs->adc[idx];
          }
       inL = sData[0];                              // use channel 0 only

// reset input counter for ADC INT
       if (count_ADC_INT >= MM)
               count_ADC_INT = 0;

// Note, we have a decimator :
// ==> The SWITCH rotates anti-Clockwise, sequence 0, 4, 3, 2, 1
//
// distribute computations across 5 output INTs
//----------------------------------  sample 0 ---------------------------------
       if (count_ADC_INT == 0) {
// for INT handler
               count_ADC_INT++;
               y1_part1 = FIR_filter_sc( H_filt_50_delays, // compute polyphase FIR out and save
                              b_filt_b0_b5_usw, N_delays_H_filt_50_delays, inL, 15);
       }

//----------------------------------  sample 1 ---------------------------------
       else if (count_ADC_INT == 1) {
// for INT handler
               count_ADC_INT++;
               y1_part2 = FIR_filter_sc ( H_filt_54_delays, // compute polyphase FIR out and save
                              b_filt_b4_b9_usw, N_delays_H_filt_54_delays, delay4, 15);
               delay4 = inL;                                       // now update delay
       }

//----------------------------------  sample 2 ---------------------------------
       else if (count_ADC_INT == 2) {
// for INT handler
               count_ADC_INT++;
               y1_part3 = FIR_filter_sc ( H_filt_53_delays, // compute polyphase FIR out and save
                              b_filt_b3_b8_usw, N_delays_H_filt_53_delays, delay3, 15);
               delay3 = inL;                                       // now update delay
       }

//----------------------------------  sample 3 ---------------------------------
       else if (count_ADC_INT == 3) {
// for INT handler
               count_ADC_INT++;
               y1_part4 = FIR_filter_sc ( H_filt_52_delays, // compute polyphase FIR out and save
                              b_filt_b2_b7_usw, N_delays_H_filt_52_delays, delay2, 15);
               delay2 = inL;                                       // now update delay
       }

//----------------------------------  sample 4 ---------------------------------
       else if (count_ADC_INT == 4) {
// for INT handler
               count_ADC_INT++;
               y1_part5 = FIR_filter_sc ( H_filt_51_delays, // compute polyphase FIR out and save
                              b_filt_b1_b6_usw, N_delays_H_filt_51_delays,delay1, 15);
               delay1 = inL;                                       // now update delay

// add all partial results up
               y1 = y1_part1 + y1_part2 + y1_part3 + y1_part4 + y1_part5;
       }
}
```