Imperial College
London

Department of Electrical and Electronic Engineering Imperial College London

Real-time digital signal processing

Report on "*Real time implementation of FIR filters*" (laboratory 4)

37 pages total

Report written by:

Quentin McGaw, CID 00746622 Username QDM12

Alexandra Rouhana, CID 00736752 Username AR4412

1. Conceptual objective

In this laboratory, the final objective is to obtain a finite impulse response (FIR) filter satisfying the following diagram (figure 1.1), which is given to us in the assignment document.

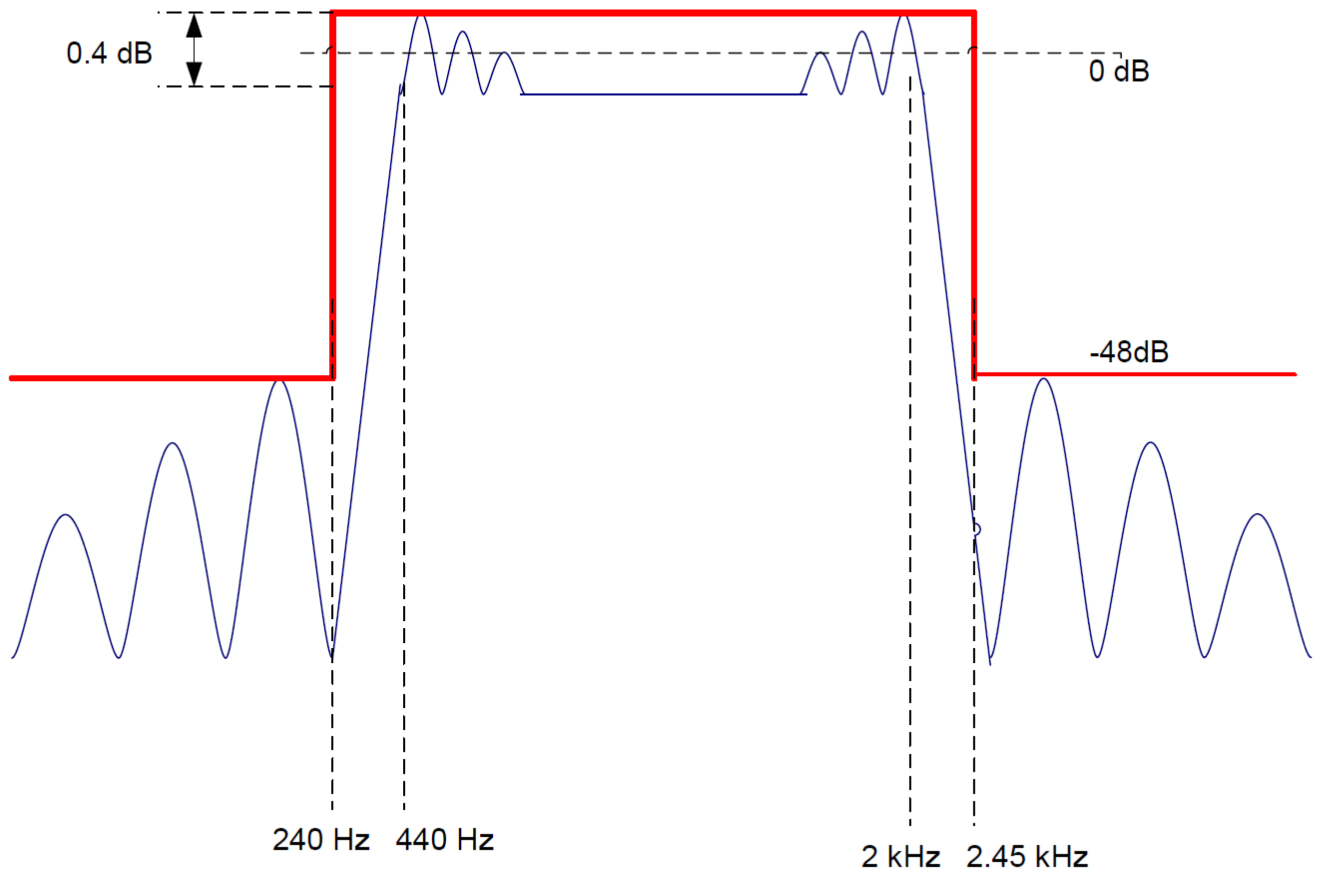

Figure 1.1 : Conceptual FIR filter specifications (*from PDF document given)*

Several parameters are deduced from this drawing in order to design the FIR filter. These are described in the tables below (*figure 1.2, figure 1.3).*

Parameter	Frequency range(s) (Hertz)
Stop bands	[0 - 240], [2450 - …]
Transition bands	[240 - 440], [2000 - 2450]
Pass band	[440 - 2000]

Figure 1.2 : Frequency parameters

Parameter	Gain (decibels)	Gain (linear)
Stop band gain	*-infinity*	0
Stop band ripple	-48	0.004
Pass band gain	0	1
Pass band ripple	0.4	0.023

Figure 1.3 : Gain parameters

The stop band ripple corresponds to the minimum attenuation of the stopband, which is -48 dB. The pass band ripple is the difference between the maximum and minimum allowable gain. In this case, its value is given as 0.4 dB. The pass band deviation corresponds to the deviation from the pass band gain (0 dB here). It can be calculated using the pass band ripple with this formula:

Pass band ripple *(db)* = 20log(1+DEV) – 20log(1-DEV), where *DEV* is the pass band deviation.
The method of calculation of this deviation is further described in the explanation of the MatLab script below.

2. Designing the FIR filter with MatLab

MatLab will be used to determine the necessary coefficients of the FIR filter. The Parks-McClelland algorithm will help the design of the FIR filter through the Matlab built-in functions *firpm* and *firpmod*. The full MatLab script used is in appendix 1. The script will be explained below.

```
% ~~~~~~~~~~~~~~~~ F U N C T I O N S ~~~~~~~~~~~~~~~~

% ~~~ db_2_gain function
db_2_gain = @(db_input) 10^(db_input/20);
    %Note: This is the definition of an anonymous function.
    %Note: This functions converts decibels into linear form.
```

Figure 2.1: MatLab script, Functions part

First of all, an anonymous function is defined in order to convert a gain in decibels units to a linear gain. It takes db_input as its argument and returns the linear converted value.

```
% ~~~~~~~~~~~~~~~ P A R A M E T E R S ~~~~~~~~~~~~~~~

% ~~~ Debug parameters ~~~
display_firpmod_dev = false;
display_firpmod = false;
% ~~~ Frequency parameters ~~~
Fsampling = 8000; %sampling frequency (C6713 sampling)
Fcutoff = [240 440 2000 2450]; %Cut off frequencies
    %Note: size of array must be 2*sizeof(amp_cutoff) - 2
    %Note: asymmetric transition frequencies gives the "bump"
% ~~~ Gain parameters ~~~
PB_gain = db_2_gain(0); %Pass band gain
SB_gain = 0; %Stop band gain
Acutoff = [SB_gain PB_gain SB_gain]; %Cut off amplitudes
% ~~~ Ripple parameters ~~~
PB_ripple=db_2_gain(0.4); %pass band ripple
SB_ripple=db_2_gain(-48); %stop band ripple
PB_dev=(PB_ripple - 1)/(PB_ripple + 1));
dev = [SB_ripple PB_dev SB_ripple]; %deviation
    %Note: size of array must be sizeof(Acutoff)
if (display_firpmod_dev == true)
    dev
end
```

Figure 2.2: MatLab script, Parameters part

All the parameters previously deduced from the conceptual drawing are inputted in this section of the script. The debug parameters are used to display results for debugging purposes. The frequency parameters include the sampling frequency of the C6713 DSK board (8 KHz), as well as the cut off frequencies of the FIR filter. The gain parameters contain the linear gains of the pass band and the stop bands. These two values are then concatenated in the variable *Acutoff* for future use. The ripple parameters are the pass band and stop band ripples. The pass band deviation *PB_dev* is then calculated from the pass band ripple with `PB_dev=(PB_ripple - 1)/(PB_ripple + 1));` where

`PB_ripple=db_2_gain(0.4);` . Lastly, the row vector *dev* is stored using the values inputted and its content can be displayed if the variable `display_firpmod_dev` is set to `true`.

```
% ~~~~~~~~~~~~~~ C A L C U L A T I O N S ~~~~~~~~~~~~~~

% ~~~ Calculation of firpmord
[N,normFBEdges,FBAmplitudes,weights]=firpmord(Fcutoff,Acutoff,dev,Fsampling);
    %Note: N is the order of the filter
if (display_firpmod == true)
    N, normFBEdges, FBAmplitudes, weights
end
% ~~~ Calculation of firpm
b = firpm(N, normFBEdges, FBAmplitudes, weights);
    %Note: FIRcoefficients is a N+1 row vector
```

Figure 2.3: MatLab script, Calculations part

The parameters previously stored are now used in the calculations of the FIR filter's coefficients. The built-in function *firpmod* takes as arguments the cut off frequencies, the cut off amplitudes (stop band and pass band associated gains), the deviation and the sampling frequency. It produces the order of the filter *N,* the normalized frequency band edges *normFBEdges*, the frequency band amplitudes *FBAmplitudes*, and the *weights* corresponding to the FIR filter.

Again, these 4 resulting variables can be displayed in the console if the debugging parameter *display_firpmod* is set to true.

The final calculation uses the built-in *firpm* function to calculate the FIR filter's coefficients from the results previously obtained. These are then stored in the N+1 row vector denoted by *b*.

```
% ~~~~~~~~~~~~~~~~~ R E S U L T S ~~~~~~~~~~~~~~~~~

freqz(b);
title('FIR filter designed to specifications')
save fir_coef.txt b -ASCII -DOUBLE -TABS;
%Note: In order to make the file fir_coef.txt readable
%      by the C code, the tabulations separating each
%      coefficient of the array of doubles b are replaced
%      with ", " using Notepad++. Some other modifications
%      are done as shown in the fir_coef.txt (appendices).
```

Figure 2.4: MatLab script, Results part

The last section of the script plots the magnitude and phase frequency responses with the built-in function *freqz* and then save the FIR filter coefficients in the file *fir_coef.txt* in the ASCII format. The plots obtained with MatLab are shown in *figure 1.8* below.

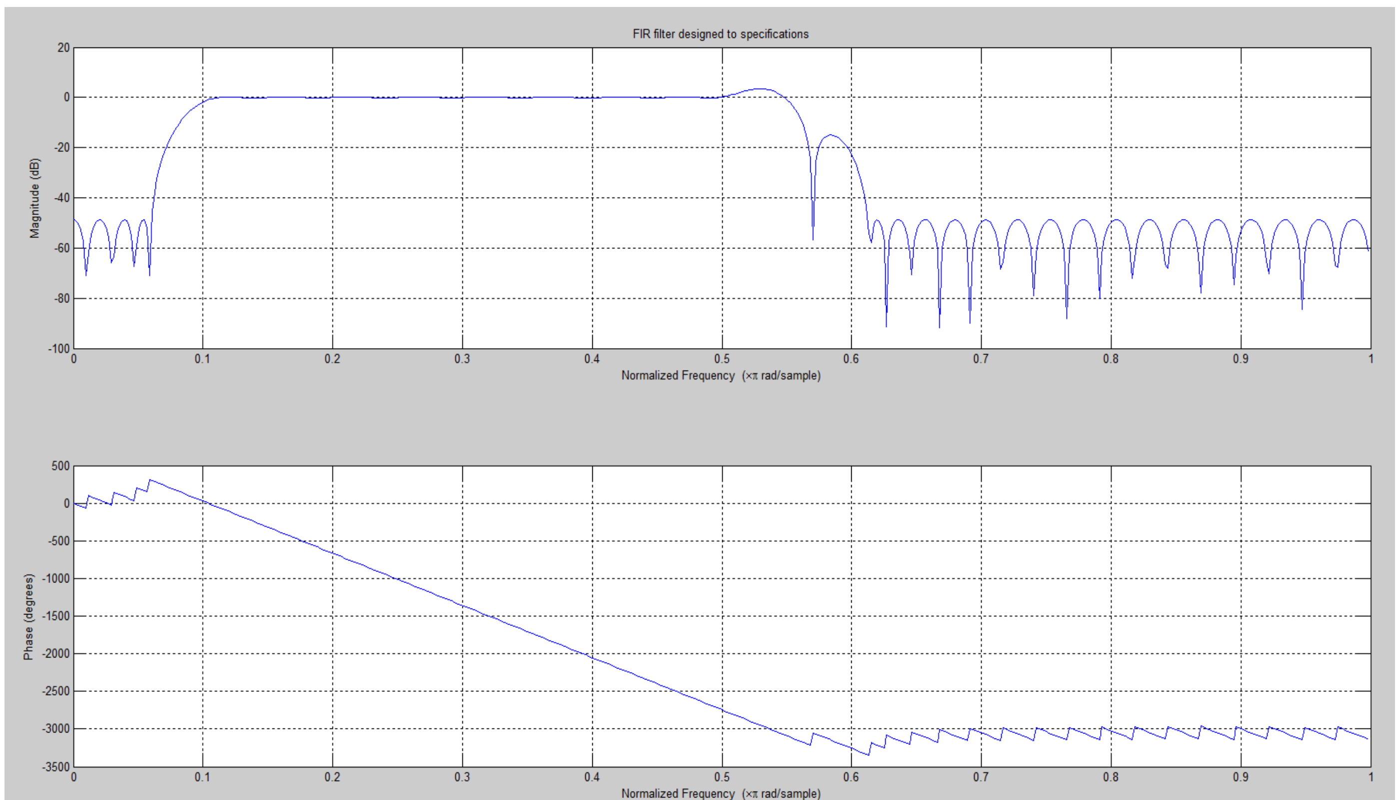

Figure 2.5: Magnitude and Phase frequency response obtained with the MatLab script

As the X axis is the normalized frequency, note that 1 PI rad/sample corresponds to the Nyquist rate of 4 KHz, so 0.5 PI rad/sample corresponds to a physical frequency of 2 KHz for example. The magnitude stays under -48 dB in the stop bands and is at 0 dB in the pass band as requested. The increase in magnitude in the second transition band is due to the allowed ripple at this frequency range.

The phase varies linearly in the pass band as shown in the second plot. Indeed, from 0.06 PI rad/sample to 0.56 PI rad/sample (or from 240 Hz to 2.24 KHz), the phase varies as a straight line. Its negative derivative, also called the group delay, will then be constant. From the second plot, the constant group delay can be calculated as:

- (0.1523 - 0.5)/(- 335.7 – (-2745)) = 0.3477 / 2409.3 = 1.4431e-4 = 0.14431 milliseconds.

As we wanted to design an FIR filter, the phase response is meant to be linear so this result is as expected.

There are 78 coefficients obtained are shown in the table below.

(x)= [0]	0.002154134578117116	(x)= [26]	0.007406391749957123	(x)= [52]	-0.02661781185635281
(x)= [1]	0.009800535789610711	(x)= [27]	0.04303939327371469	(x)= [53]	0.02631953002098781
(x)= [2]	0.005437151557883184	(x)= [28]	-0.02103178601593447	(x)= [54]	0.0475252511262115
(x)= [3]	-0.005114249471376731	(x)= [29]	-0.05646886351853166	(x)= [55]	-0.004688336751128472
(x)= [4]	-0.001513506920649701	(x)= [30]	0.006371277469153339	(x)= [56]	-0.008458029372103371
(x)= [5]	0.008162617675536773	(x)= [31]	-0.0001533998344406505	(x)= [57]	0.03766252815272615
(x)= [6]	-0.002253258295526966	(x)= [32]	-0.09762144584917198	(x)= [58]	0.02265256757921339
(x)= [7]	-0.01388025934009781	(x)= [33]	-0.07914980818719607	(x)= [59]	-0.0204801437861095
(x)= [8]	0.0002647629650941496	(x)= [34]	0.01156993090636228	(x)= [60]	-0.0007921538320172176
(x)= [9]	0.00704851562375174	(x)= [35]	-0.0793554777589599	(x)= [61]	0.0252695285004713
(x)= [10]	-0.01547843764650902	(x)= [36]	-0.2085232611327358	(x)= [62]	-0.005873969818268656
(x)= [11]	-0.0182176577153744	(x)= [37]	0.01103154097020508	(x)= [63]	-0.024949016057599
(x)= [12]	0.008917057932837851	(x)= [38]	0.4083753987878134	(x)= [64]	0.00343878766500461
(x)= [13]	0.00343878766500461	(x)= [39]	0.4083753987878134	(x)= [65]	0.008917057932837851
(x)= [14]	-0.024949016057599	(x)= [40]	0.01103154097020508	(x)= [66]	-0.0182176577153744
(x)= [15]	-0.005873969818268656	(x)= [41]	-0.2085232611327358	(x)= [67]	-0.01547843764650902
(x)= [16]	0.0252695285004713	(x)= [42]	-0.0793554777589599	(x)= [68]	0.00704851562375174
(x)= [17]	-0.0007921538320172176	(x)= [43]	0.01156993090636228	(x)= [69]	0.0002647629650941496
(x)= [18]	-0.0204801437861095	(x)= [44]	-0.07914980818719607	(x)= [70]	-0.01388025934009781
(x)= [19]	0.02265256757921339	(x)= [45]	-0.09762144584917198	(x)= [71]	-0.002253258295526966
(x)= [20]	0.03766252815272615	(x)= [46]	-0.0001533998344406505	(x)= [72]	0.008162617675536773
(x)= [21]	-0.008458029372103371	(x)= [47]	0.006371277469153339	(x)= [73]	-0.001513506920649701
(x)= [22]	-0.004688336751128472	(x)= [48]	-0.05646886351853166	(x)= [74]	-0.005114249471376731
(x)= [23]	0.0475252511262115	(x)= [49]	-0.02103178601593447	(x)= [75]	0.005437151557883184
(x)= [24]	0.02631953002098781	(x)= [50]	0.04303939327371469	(x)= [76]	0.009800535789610711
(x)= [25]	-0.02661781185635281	(x)= [51]	0.007406391749957123	(x)= [77]	0.002154134578117116

Figure 2.6: FIR filter coefficients obtained with matLab script

It can be seen and verified from this table that the generated coefficients are symmetric. Indeed, the coefficients in the array *b* obey to the FIR filter relationship $b(n) = b(N-1-n)$, hence giving a linear phase frequency response. Moreover, the sum of all the coefficients equals to zero and this has for result to block any DC component. The file *fir_coeff.txt* is in appendix 2 and has been changed to be read by a C code.

3. FIR filter implementation

As this implementation is based on the laboratory 3 codes, the interrupts happen when a sample is read by the board, hence calling the *ISR_AIC()* function. The text file *fir_coeff.txt* generated by the Matlab script is included in the project by adding the line `#include "fir_coef.txt"` at the start of the *intio.c* file. The workspace hierarchy is as shown in the figure 3.0.1 below.

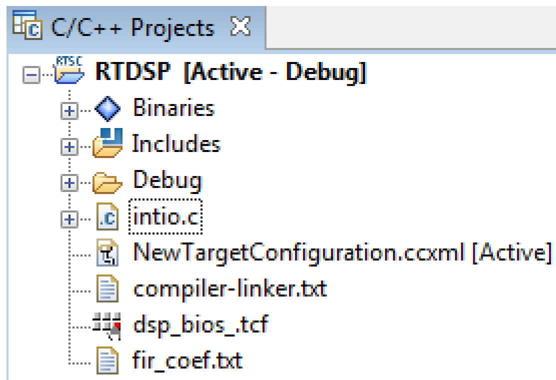

Figure 3.0.1: Project workspace hierarchy

3.1. First FIR filter implementation

The first implementation of the FIR filter is done by using a delay buffer combined with a MAC loop. The full code *intio.c* is attached in appendix 3.

A constant *N* is defined by `#define N (sizeof(b)/sizeof(b[0]))` and corresponds to the order of the filter (78 here). It is used to declare the delay input buffer `int x[N]` and limit the loops in all of our implementations.

The function *init_var()* shown in the figure below sets all the values of the array *x* to zero in order to avoid any miscalculations. It is called at the start of the *main().* This is again done in all the next implementations of the FIR filter.

```
/************************************** init_var() ***************************************/

void init_var(void) /*MODIFICATION_01 - Initializes variables (array x to zero)*/
{
    memset(x, 0, sizeof(x));
}
```

Figure 3.1.1: Function init_var()

In this first implementation, the interrupt service routine *ISR_AIC*() is defined as in figure 3.1.3 and the function *non_circ_FIR()* as in figure 3.1.4.

```
/************************** INTERRUPT SERVICE ROUTINE *******************************/

void ISR_AIC(void)
{
    int n;
    mono_write_16Bit(non_circ_FIR()); /*MODIFICATION_01 - Output filtered sample.*/
    /*MODIFICATION_01 - Move data along buffer from lower element to next higher*/
    for(n = N - 1; n > 0; n--)
    {
        x[n] = x[n - 1];
    }
    x[0] = mono_read_16Bit(); /*MODIFICATION_01 - Stores new sample in delay buffer*/
}
```

Figure 3.1.2: Function ISR_AIC() called by interrupt

The *mono_write_16Bit()* function writes the filtered sample to the output ports (as a MONO signal). The delay in the input delay buffer is achieved with the for loop in N – 1 iterations. Please refer to the comments in the code for further explanations. Note that “MODIFICATION_01” corresponds to the modifications done in the first implementation. Hence, “MODIFICATION_02” will correspond to the second implementation for example.

```
/****************************** non_circ_FIR() ***********************************/

double non_circ_FIR(void)
{
/*MODIFICATION_01 - Performs the FIR filtering, so the convolution between
the filter coefficients of b and the samples of the delay buffer x.
It uses indexes to move to the next element in x and b (01). */
    double sum = 0;
    int n;
    for(n = 0; n < N; n++) //note: for loops are 2 cycles faster than while loops.
    {
        sum += x[n] * b[n];
    }
    return sum;
}
```

Figure 3.1.3: Function non_circ_FIR() called by ISR_AIC()

This function performs the convolution between the input delay buffer *x* and the coefficients *b*. This MAC operations are done in the for loop through N iterations.

To test the correct operation of the implemented FIR filter, the following configuration is used.

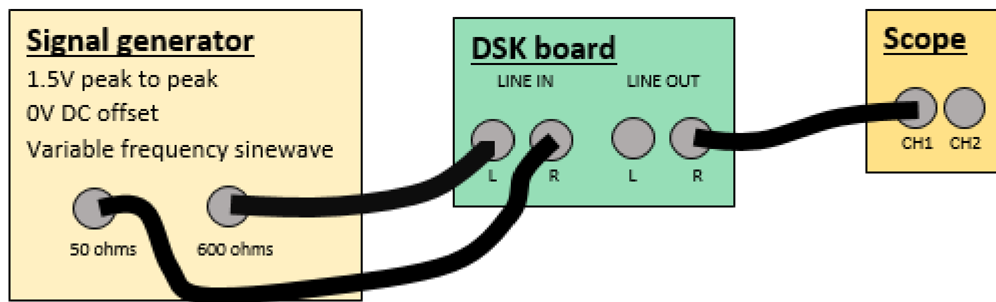

Figure 3.1.4: Connection configuration for input and output ports of the DSK board

The following results are obtained at various frequencies.

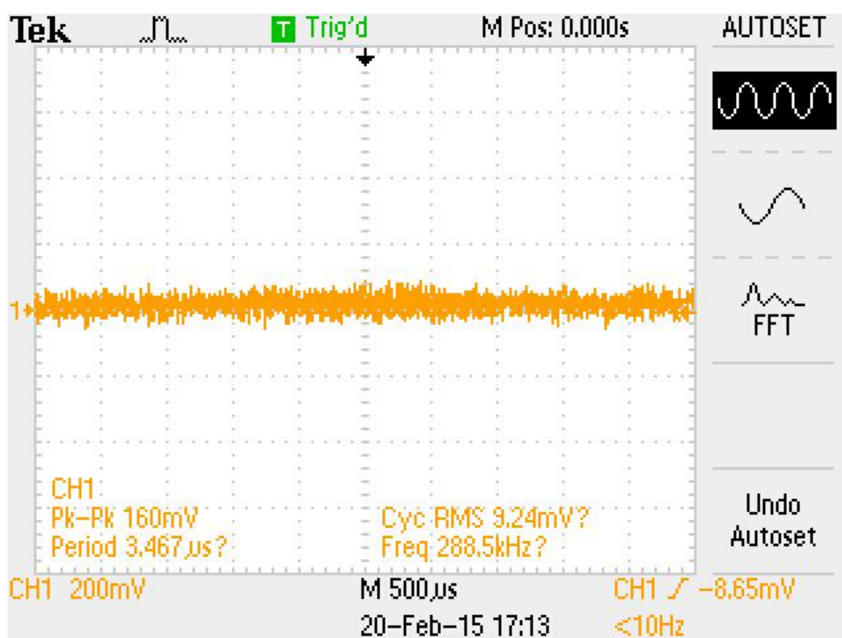

Figure 3.1.5: Scope measurement of output of DSK board, at stop band frequency range (<240 Hz; > 2000 Hz)

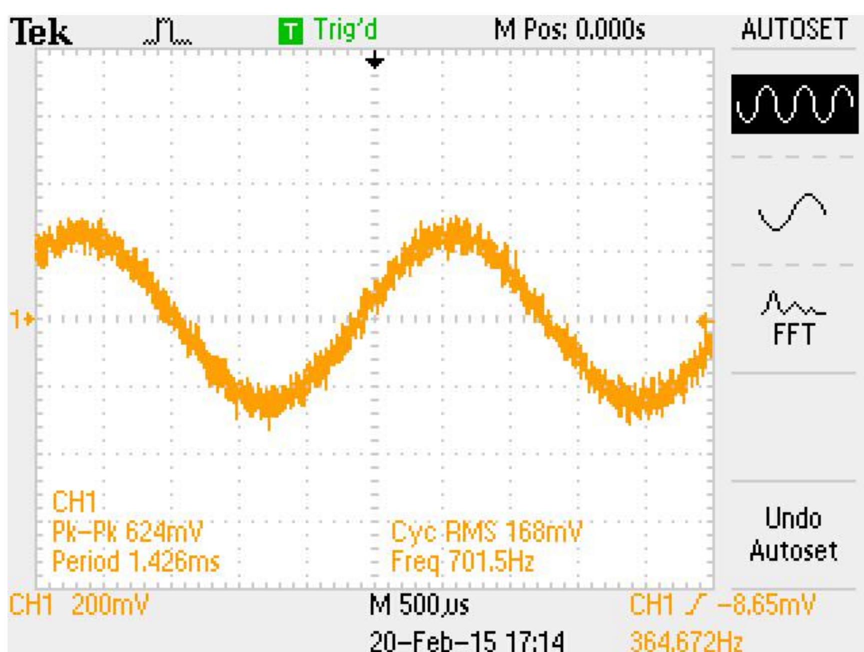

Figure 3.1.6: Scope measurement of output of DSK board, at pass band frequency range (364 Hz)

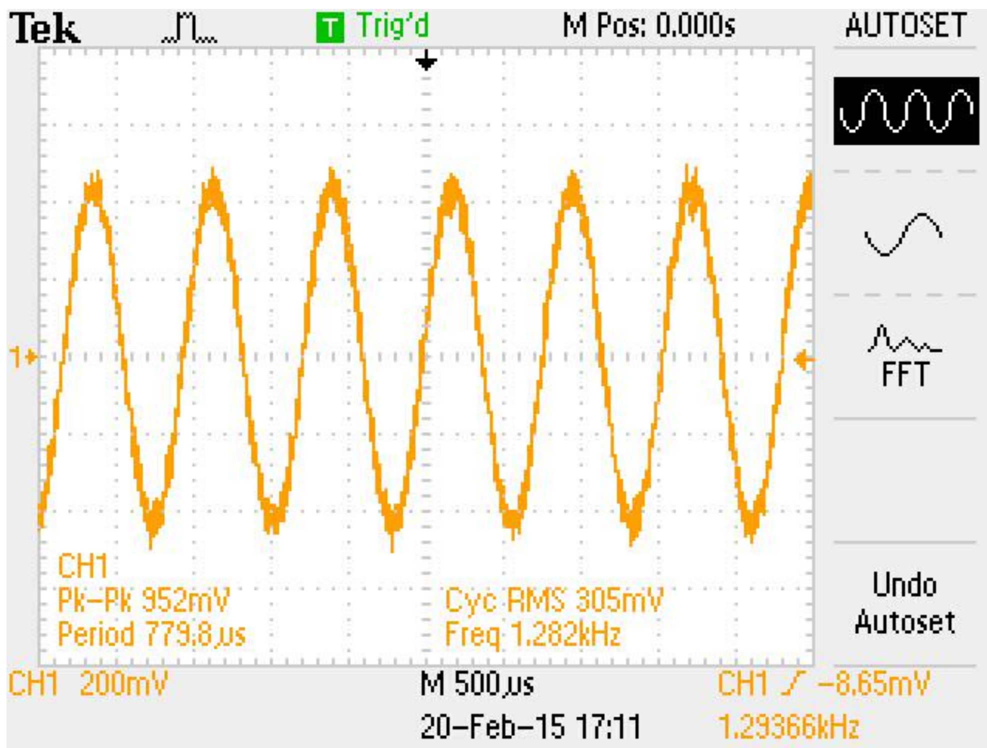

Figure 3.1.7: Scope measurement of output of DSK board, at pass band frequency range (1293 Hz)

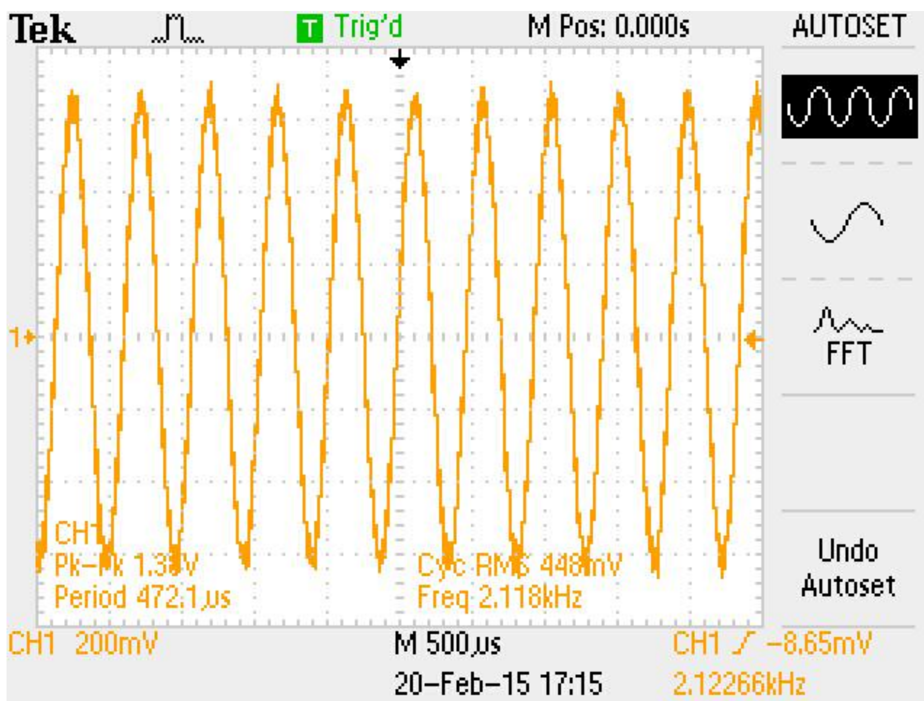

Figure 3.1.8: Scope measurement of output of DSK board, at second transition band frequency range (2122 Hz)

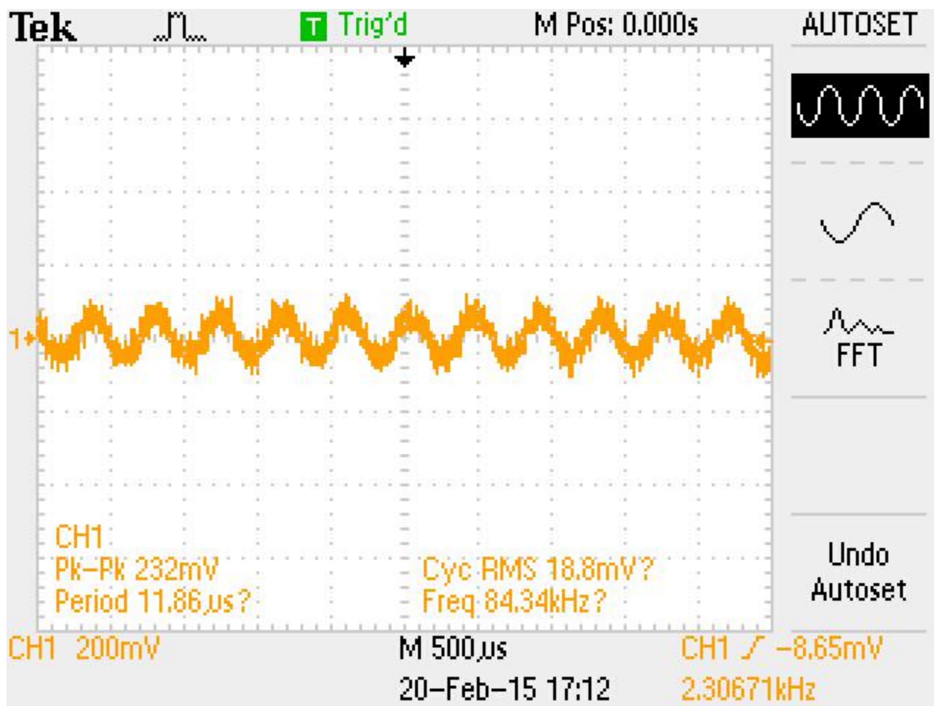

Figure 3.1.9: Scope measurement of output of DSK board, at stop band frequency range (2306 Hz)

The FIR filter implemented hence acts as it was expected to. The signal is absent in the stop bands and is the same signal in the pass band, between 240 Hz and 2 KHz. However, the amplitude of the output signal is 4 times reduced in comparison with the input amplitude. This is due to the two potential divider at each input port of the DSK board. These are shown in the figure 3.6 below.

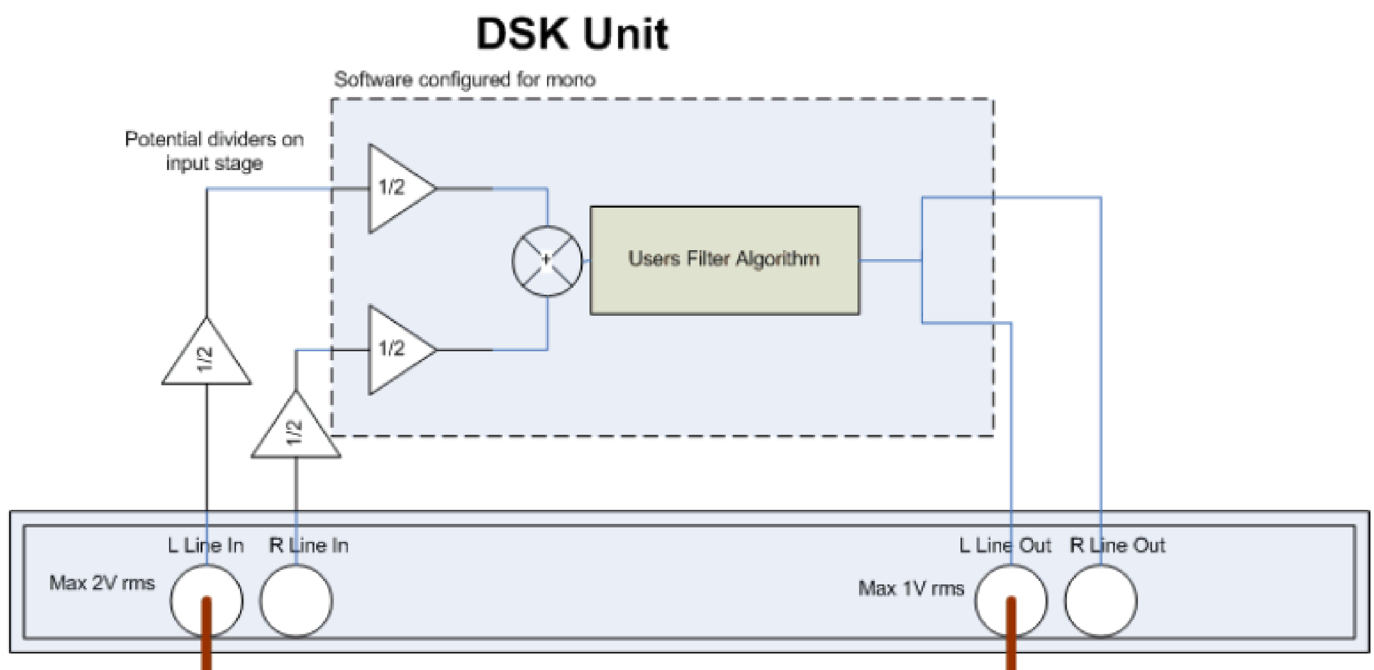

Figure 3.1.10: Potential dividers at the input ports of the DSK board

To evaluate the performance of the code, the profiling clock method will be used. First, the clock has to be enabled as shown below:

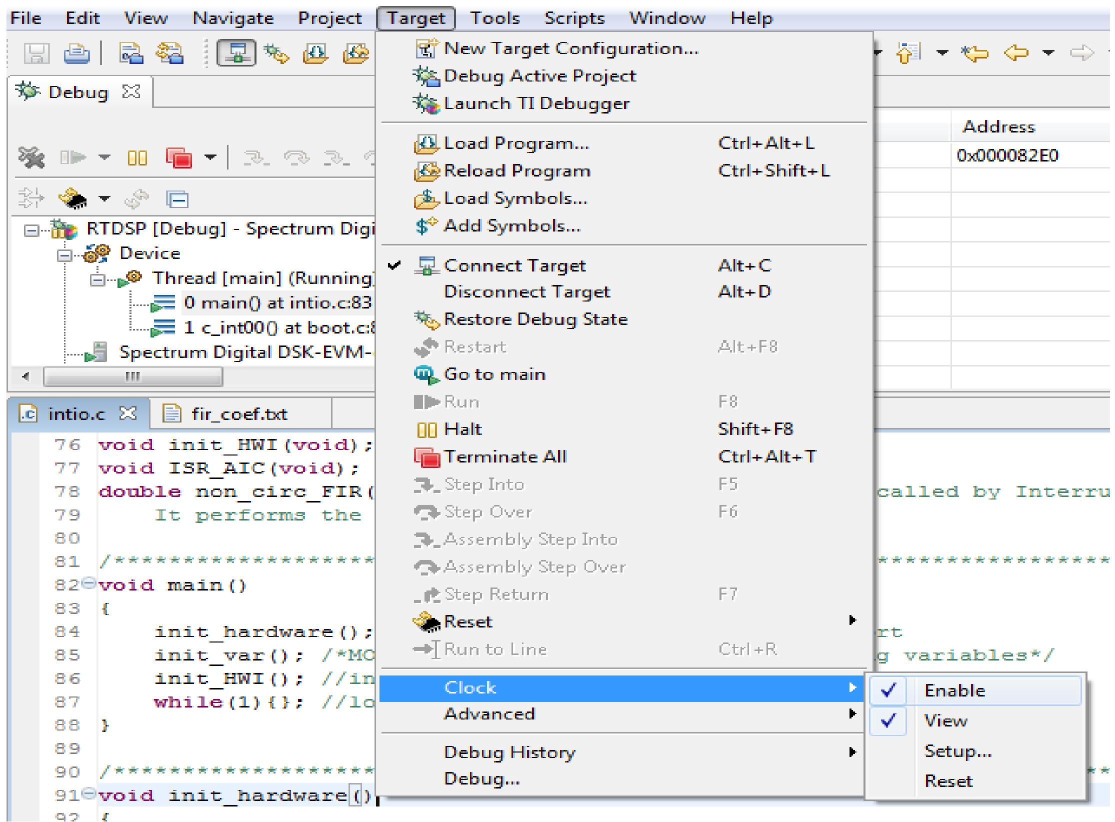

Figure 3.1.11: Menu to enable the clock profiling method

Two breakpoints are inserted so that they surround the interrupt routine (see figure 3.1.7). This will allow to reset the clock to zero at the beginning of the interrupt routine and to measure how many cycles are consumed by the entire interrupt routine. This method is used for all the future implementations of the FIR filter.

```
127 void ISR_AIC(void)
128 {
129     int n;
130     mono_write_16Bit
131     /*MODIFICATION_(
132     for(n = N - 1; r
133     {
134         x[n] = x[n -
135     }
136     x[0] = mono_reac
137 }
```

Figure 3.1.12: Breakpoints position to measure the number of cycles

There are 4 different compiler optimisation levels which can be accessed from **Project->Properties->C/C++ build->C6000 compiler->Basic options** but only the –o0 and –o2 optimisation options will be used. The –o0 optimisation level makes the optimisation occur only at a single statement level, while the –o2 option optimizes at the function level. The number of cycles are obtained without and with these compiler optimisations. These cycles measurements will be with these optimisations and breakpoints for all the future implementations of the FIR filter.

The following results are obtained for our first implementation.

Optimization	None	-o0	-o2
Measurement 1	5211	4977	1109
Measurement 2	5211	4977	1109
Measurement 3	5211	4977	1108
Measurement 4	5210	4977	1108
Best case	5210	4977	1108

Figure 3.1.13: Cycles measured for different optimisation levels (implementation 1)

This shows that the optimisation –o2 is the best with a best case of 1108 cycles used by the interrupt routine.

3.2. Second FIR filter implementation: only one for loop

The second implementation of the FIR filter is based on the first implementation described above. The main difference is that the MAC operations needed for the convolution and the delay operations are now done in only one for loop instead of two. The full code for this implementation is in appendix 4.

The functions ISR_AIC() and non_circ_FIR() are shown in the figure 3.2.1.

```
/************************* INTERRUPT SERVICE ROUTINE ********************************/

void ISR_AIC(void)
{
    mono_write_16Bit(non_circ_FIR()); /*MODIFICATION_01 - Output filtered sample.*/
    x[0] = mono_read_16Bit(); /*MODIFICATION_01 - Stores new sample in delay buffer*/
}

/******************************** non_circ_FIR() ************************************/
double non_circ_FIR(void)
{
/*MODIFICATION_01 - Performs the FIR filtering so the convolution between
the filter coefficients b and the samples in the delay buffer x.
It uses indexes to move to the next element in x and b (01). */
/*MODIFICATION_02 - The shifting is done in the MAC protocol.
    Only one for loop is needed now. */
    int n;
    double sum = x[0] * b[0];
    for(n = N-1; n > 0; n--)
    {
        sum += x[n] * b[n];
        x[n] = x[n - 1]; /*MODIFICATION_02 - Shifting in MAC protocol.*/
    }
    return sum;
}
```

Figure 3.2.1: Functions ISR_AIC() and non_circ_FIR()

The delay and MAC operations are now executed in one for loop in the non_circ_FIR() function. This should improve the speed of the interrupt routine.

The new implementation has been tested using the signal generator at various frequencies and satisfies the requirements (same waveforms as for the first implementation). The following results were obtained concerning the cycles used by the interrupt routine.

Optimization	None	-o0	-o2
Measurement 1	4403	4327	580
Measurement 2	4403	4326	581
Measurement 3	4404	4326	581
Measurement 4	4403	4326	580
Best case	4403	4326	580

Figure 3.2.2: Cycles measured for different optimisation levels (implementation 2)

This table clearly shows the gain in speed between the two implementations. 807 cycles were gained without optimisation, 651 with –o0 and 528 with the –o2 optimisation level. Also, the reduction in the differences of cycles between the two first implementations demonstrates the ability of the –o0 and –o2 compiler optimizations.

3.3. Third FIR filter implementation: using pointers

A third implementation was made using pointers as we thought using them would increase the speed of the interrupt routine. Indeed, the second implementation used index accessing in a for loop. Each of these loops contains 2 load operations: one for the address of the start of the array and one for the index value *n*. On the other hand, by using pointers, only loading the start address of the array is needed.

The only modifications (signalled by *MODIFICATION_03*) are made in the *non_circ_FIR()* function from the second implementation. This function is shown in figure 3.3.1 and the full code for this third implementation is shown in appendix 5.

```
/******************************* non_circ_FIR() ************************************/

double non_circ_FIR(void)
{
/*MODIFICATION_03 - Performs the FIR filtering so the convolution between
the filter coefficients b and the samples in the delay buffer x.
It uses pointers to move to the next element in x and b (02). */
    int n;
    double sum;
    int *xp = x; /*MODIFICATION_03 - Pointer to first element of x*/
    double *bp = b; /*MODIFICATION_03 - Pointer to first element of b*/
    sum = (*xp++) * (*bp++); /*MODIFICATION_03 - MAC using pointers*/
    for(n = N - 1; n > 0; n--)
    {
        sum += (*xp++) * (*bp++); /*MODIFICATION_03 - MAC using pointers*/
        x[n] = x[n - 1]; /*MODIFICATION_01 - Move data along buffer from lower element
to next higher*/
    }
    return sum;
}
```

Figure 3.3.1: Functions non_circ_FIR() using pointers instead of index accessing

The new implementation has been tested using the signal generator at various frequencies and satisfies the specifications. The following results were obtained concerning the cycles used by the interrupt routine.

Optimization	None	-o0	-o2
Measurement 1	5108	4722	745
Measurement 2	5107	4722	745
Measurement 3	5107	4722	744
Measurement 4	5107	4722	744
Best case	5107	4722	744

Figure 3.3.2: Cycles measured for different optimisation levels (implementation 3, using pointers)

The performance of the pointer implementation is disappointing as it takes way more cycles than the second implementation. Even without optimization, the number of clock cycles consumed by the interrupt routine is higher than the index accessing implementation. Moreover, a higher number of clock cycles for the index accessing loop at full optimization is expected since the compiler will optimize best the form of convolution it recognizes best, such as the index accessing loops.

For now, the second implementation is the best implementation so far.

3.4. Fourth FIR filter implementation: Circular buffer

In order to decrease furthermore the number of cycles needed by the interrupt service routine, a circular buffer is implemented to avoid the copy of a large set of data for each new sample. Indeed, in the previous implementations, data was moved around to delay the input buffer each time a new sample was received. This new algorithm updates a pointer to the newest element in the input buffer array. The full code is in appendix 6.

The ISR_AIC() function is slightly modified as described below.

```
/************************* INTERRUPT SERVICE ROUTINE *******************************/

void ISR_AIC(void)
{
    mono_write_16Bit(circ_FIR(mono_read_16Bit()));
    /*MODIFICATION_01 - Output filtered sample.*/
}
```

Figure 3.4.1: Functions ISR_AIC()

The circ_FIR() function is shown in the figure 3.4.2 below.

```
/********************************* circ_FIR() ***************************************/

double circ_FIR(double sample_in)
{
/*MODIFICATION_04 - Performs the FIR filtering.
This function uses a circular buffer in order to execute the convolution. (04)*/
    double sum = 0;
    double *bp = b; /*MODIFICATION_04 - Coefficient variable pointer.
                                This pointer is first set to the first element of b.*/
    int *x_start = x; /*MODIFICATION_04 - Input buffer constant pointer.
                                This pointer points to the first element of x.*/
    double *bp_end = b + N; /*MODIFICATION_04 - Coefficient constant pointer.
                                This pointer points to the last element of b.*/
    int *xp = x_start + offset;
/*MODIFICATION_04 - Input buffer variable pointer. This pointer is first set to the
'offset'th element of x.*/
    *xp = sample_in; /*MODIFICATION_04 - Stores the new sample at the address of xp.*/
    while(xp >= x_start)
    {
        sum += (*xp--) * (*bp++);
        /*MODIFICATION_04 - Shifts the address of the element of the input buffer
        from the 'offset' address to the first element's address (x_start).
        It shifts at the same time the address of the element of the coefficients
        from the first element's address to the 'offset' address.*/
    }
    xp = x_start + N - 1;
/*MODIFICATION_04 -Reinitialises xp to the end of the array.*/
    while(bp < bp_end)
    {
        sum += (*xp--) * (*bp++);
        /*MODIFICATION_04 - Shifts the address of the element of the coefficients
        from the 'offset' address to the last element's address (bp_end).
        It shifts at the same time the address of the element of the input buffer
        from the last (bp_end) element's address to the 'bp_end - offset' address.*/
    }
    offset--; /*MODIFICATION_04 - Decrements the offset*/
    if(offset < 0)
    {
        offset = N - 1;
/*MODIFICATION_04 - Restores the offset to N-1 when out of bound.*/
    }
    return sum;
}
```

Figure 3.4.2: Functions circ_FIR() using pointers

The function contains 4 pointers and each of their functions are described in the comments (*MODIFICATION_04*). All the working of this circular buffer algorithm is also described in the comments of the code above.

The variable *offset* is defined as a global variable by `int offset = N - 1;` and is decremented each time the interrupt service routine is called. It is also reinitialised to N – 1 whenever it is out of bound (negative). Because of this *offset* decrement, each time the ISR is called, the number of loops in the first while loop will decrease as the number of loops in the second while loop will increase.

The new implementation has been tested using the signal generator at various frequencies and satisfies the specifications. The following results were obtained concerning the cycles used by the interrupt routine.

Optimization	None	-o0	-o2
Measurement 1	3944	3945	806
Measurement 2	3942	3942	805
Measurement 3	3940	3940	805
Measurement 4	3939	3936	806
Best case	3939	3936	805

Figure 3.4.3: Cycles measured for different optimisation levels (implementation 4, using circular buffers)

Without optimization, this implementation is slightly quicker than the second implementation (non circular buffer with index accessing). However, with the optimisations applied, this code consumes more cycles in the ISR in comparison with the second implementation.

One last implementation was done by replacing pointers by index accessing in the circular buffer algorithm. The modifications are only done in the circ_FIR() function as shown below:

```
/********************************** circ_FIR() ***************************************/

double circ_FIR(double sample_in)
{
/*MODIFICATION_04 - Performs the FIR filtering.
This function uses a circular buffer in order to execute the convolution. (04)*/
    double sum = 0;
    int n = 0;
    x[offset] = sample_in;
/*MODIFICATION_05 - Stores the new sample at the address of xp.*/
    for(n = 0; n <= offset; n++)
    {
        sum += x[offset - n] * b[n];
    }
    for(n = 0; n < N - offset; n++)
    {
        sum += x[N - 1 - n] * b[offset + n];
    }
    offset--; /*MODIFICATION_04 - Decrements the offset*/
    if(offset < 0)
    {
        offset = N - 1; /*MODIFICATION_04 - Restores the offset to N-1 when out of
bound.*/
    }
    return sum;
}
```

Figure 3.4.4: Functions circ_FIR() using index accessing

This last implementation of the circular buffer gave us the right signals on the oscilloscope and better results concerning the speed of the ISR.

Optimization	None	-o0	-o2
Measurement 1	3944	3945	787
Measurement 2	3942	3942	788
Measurement 3	3940	3940	787
Measurement 4	3939	3936	787
Best case	3939	3936	787

Figure 3.4.5: Cycles measured for different optimisation levels (implementation 5, using circular buffers and indexes)

4. Spectrum analyser and measurement of frequency response

The frequency response of our designed FIR filter (using implementation 2) is measured with the APX 515 audio analyser and the apx500 software. The results obtained for the magnitude and the phase match to a certain extent the expected results from the MatLab simulation.

The magnitude response of the FIR filter is shown in the figure 4.1 below.

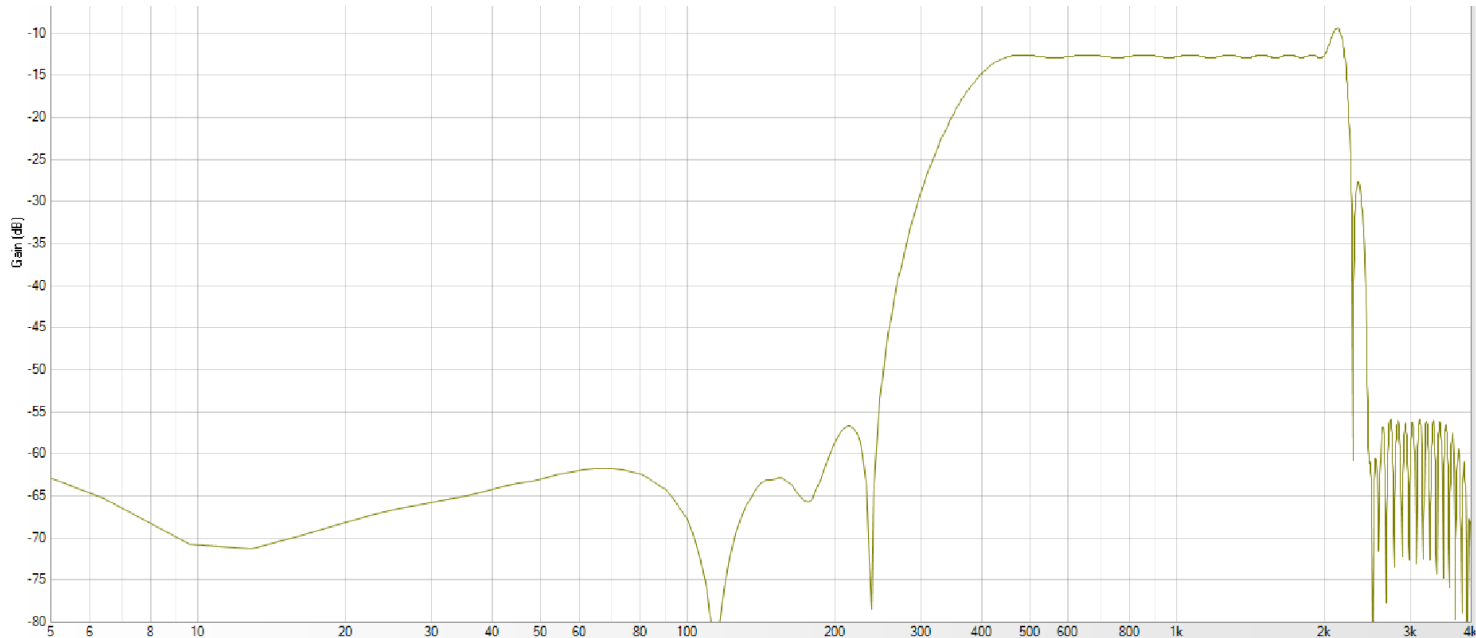

Figure 4.1: Magnitude response measured with APX 500

The magnitude response in the pass band frequency range is not 0 dB as expected but around -12.5 dB. This value corresponds to a linear gain of approximately 0.25. This corresponds to the 0.25 gain factor due to the potential dividers at the inputs of the DSK board as described earlier in the report.

Using the Matlab command *zplane*(*b*), where *b* is the coefficients of the FIR filter, a zero plane of the filter has been plotted as shown in the figure 4.2.

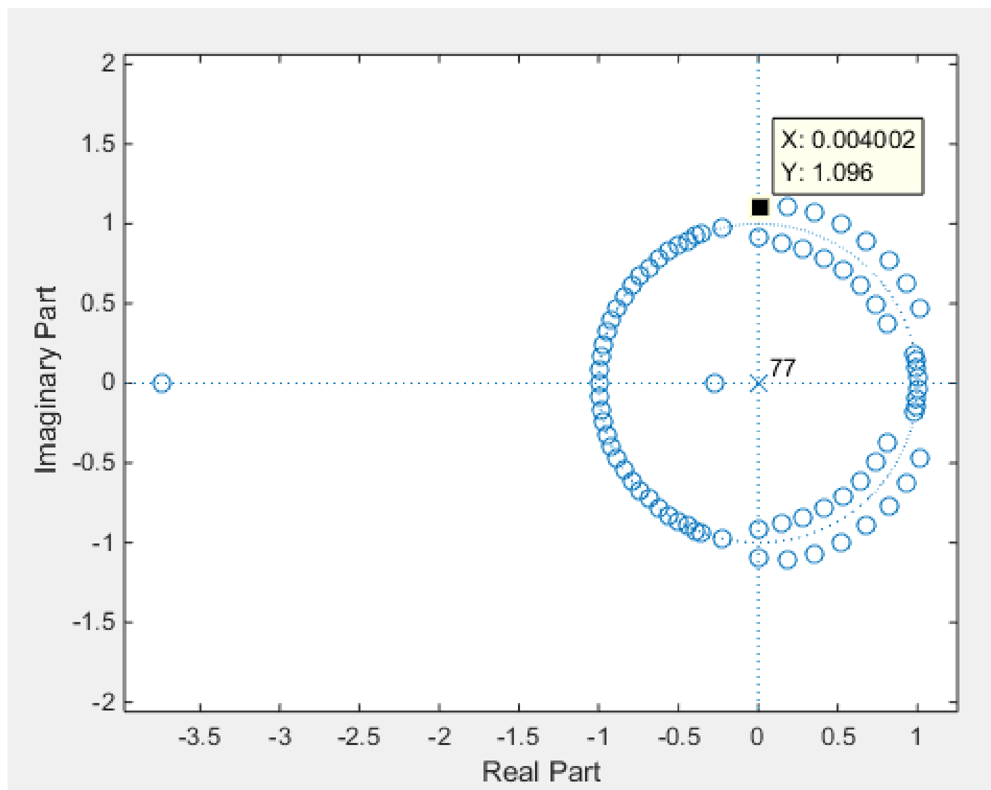

Figure 4.2: MatLab zero map of the FIR filter generated

The PI/2 axis represents half of the Nyquist rate, 2 KHz. Most of the zeroes of the filter are located under 2 KHz which is the highest corner frequency of the FIR filter. These zeroes on the unit circle will bring our gain to 0 dB for frequencies lower than 2 KHz. The zeroes placed to the right side of the PI/2 axis correspond to the ripple observable in the stop band. The ripple in the pass band is due to the position of the zeroes which are not exactly placed on the unit circle.

In order to show the phase is linear as expected, the group delay is observed using the APX 515. The following figure is obtained.

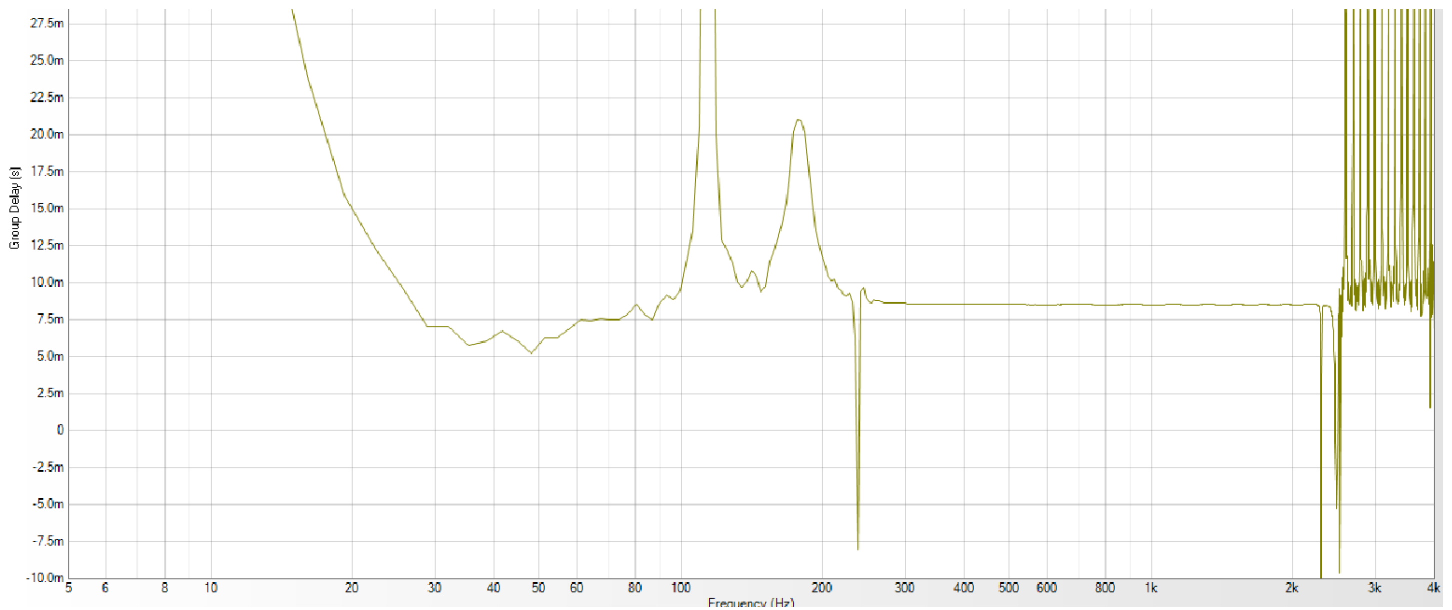

Figure 4.3: Group delay response measured with APX 500

As the group delay is constant between 230 Hz and 2200 Hz, the phase of the designed FIR filter is linear in the pass band as it was described in MatLab.

Appendices

Appendix 1: MatLab script to generate FIR filter coefficients

FIRdesign.m

```
%% ************* MATLAB FIR design script ************* %%

%% Authors: Alexandra Rouhana and Quentin McGaw %%
%% Date: February 2015 %%

% ~~~~~~~~~~~~~~~~ F U N C T I O N S ~~~~~~~~~~~~~~~~
% ~~~ db_2_gain function
db_2_gain = @(db_input) 10^(db_input/20);
    %Note: This is the definition of an anonymous function.
    %Note: This functions converts decibels into linear form.


% ~~~~~~~~~~~~~~~ P A R A M E T E R S ~~~~~~~~~~~~~~~
% ~~~ Debug parameters ~~~
display_firpmod_dev = false;
display_firpmod = false;
% ~~~ Frequency parameters ~~~
Fsampling = 8000; %sampling frequency (C6713 sampling)
Fcutoff = [240 440 2000 2450]; %Cut off frequencies
    %Note: size of array must be 2*sizeof(amp_cutoff) - 2
    %Note: asymmetric transition frequencies gives the "bump"
% ~~~ Gain parameters ~~~
PB_gain = db_2_gain(0); %Pass band gain
SB_gain = 0; %Stop band gain
Acutoff = [SB_gain PB_gain SB_gain]; %Cut off amplitudes
% ~~~ Ripple parameters ~~~
PB_ripple=db_2_gain(0.4); %pass band ripple
SB_ripple=db_2_gain(-48); %stop band ripple
PB_dev=(PB_ripple - 1)/(PB_ripple + 1));
dev = [SB_ripple PB_dev SB_ripple]; %deviation
    %Note: size of array must be sizeof(Acutoff)
if (display_firpmod_dev == true)
    dev
end


% ~~~~~~~~~~~~~ C A L C U L A T I O N S ~~~~~~~~~~~~~
% ~~~ Calculation of firpmord
[N,normFBEdges,FBAmplitudes,weights]=firpmord(Fcutoff,Acutoff,dev,Fsampling);
    %Note: N is the order of the filter
if (display_firpmod == true)
    N, normFBEdges, FBAmplitudes, weights
end
% ~~~ Calculation of firpm
b = firpm(N, normFBEdges, FBAmplitudes, weights);
    %Note: FIRcoefficients is a N+1 row vector


% ~~~~~~~~~~~~~~~~~ R E S U L T S ~~~~~~~~~~~~~~~~~~
freqz(b);
title('FIR filter designed to specifications')
save fir_coef.txt b -ASCII -DOUBLE -TABS;
%Note: In order to make the file fir_coef.txt readable
%      by the C code, the tabulations separating each
%      coefficient of the array of doubles b are replaced
%      with ", " using Notepad++. Some other modifications
%      are done as shown in the fir_coef.txt (appendices).
```

Appendix 2: fir_coeff.txt file containing coefficients generated by FIRdesign.m script

fir_coef.txt

```
double b[] = {2.1541345781171156e-03, 9.8005357896107110e-03, 5.4371515578831835e-03, -
5.1142494713767306e-03, -1.5135069206497014e-03, 8.1626176755367728e-03, -
2.2532582955269659e-03, -1.3880259340097808e-02, 2.6476296509414955e-04,
7.0485156237517404e-03, -1.5478437646509018e-02, -1.8217657715374400e-02,
8.9170579328378512e-03, 3.4387876650046100e-03, -2.4949016057598996e-02, -
5.8739698182686557e-03, 2.5269528500471299e-02, -7.9215383201721758e-04, -
2.0480143786109503e-02, 2.2652567579213391e-02, 3.7662528152726152e-02, -
8.4580293721033713e-03, -4.6883367511284715e-03, 4.7525251126211504e-02,
2.6319530020987809e-02, -2.6617811856352806e-02, 7.4063917499571225e-03,
4.3039393273714685e-02, -2.1031786015934469e-02, -5.6468863518531656e-02,
6.3712774691533393e-03, -1.5339983444065050e-04, -9.7621445849171984e-02, -
7.9149808187196066e-02, 1.1569930906362279e-02, -7.9355477758959903e-02, -
2.0852326113273578e-01, 1.1031540970205075e-02, 4.0837539878781343e-01,
4.0837539878781343e-01, 1.1031540970205075e-02, -2.0852326113273578e-01, -
7.9355477758959903e-02, 1.1569930906362279e-02, -7.9149808187196066e-02, -
9.7621445849171984e-02, -1.5339983444065050e-04, 6.3712774691533393e-03, -
5.6468863518531656e-02, -2.1031786015934469e-02, 4.3039393273714685e-02,
7.4063917499571225e-03, -2.6617811856352806e-02, 2.6319530020987809e-02,
4.7525251126211504e-02, -4.6883367511284715e-03, -8.4580293721033713e-03,
3.7662528152726152e-02, 2.2652567579213391e-02, -2.0480143786109503e-02, -
7.9215383201721758e-04, 2.5269528500471299e-02, -5.8739698182686557e-03, -
2.4949016057598996e-02, 3.4387876650046100e-03, 8.9170579328378512e-03, -
1.8217657715374400e-02, -1.5478437646509018e-02, 7.0485156237517404e-03,
2.6476296509414955e-04, -1.3880259340097808e-02, -2.2532582955269659e-03,
8.1626176755367728e-03, -1.5135069206497014e-03, -5.1142494713767306e-03,
5.4371515578831835e-03, 9.8005357896107110e-03, 2.1541345781171156e-03};
```

Appendix 3: First implementation of FIR filter (intio.c)

Implementation1_noncircular_index.c

```
/*************************************************************************************
        ,      DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING
                ,   ,     IMPERIAL COLLEGE LONDON

            ,     EE 3.19: Real Time Digital Signal Processing
                ,       Dr Paul Mitcheson and Daniel Harvey

            ,           , LAB 4: Real-time Implementation of FIR Filters

            ,           ********* I N T I O. C **********

  Implementation of FIR filter using indexes to execute the convolution function.
  MODIFICATIONS made by Quentin McGaw and Alexandra Rouhana, during February 2015
  MODIFICATION_01 correspond to this implementation.

 ************************************************************************************
                Updated for use on 6713 DSK by Danny Harvey: May-Aug 2006
                Updated for CCS V4 Sept 10
 ************************************************************************************/
/*
 *  You should modify the code so that interrupts are used to service the
 *  audio port.
 */
/**************************** Pre-processor statements ******************************/

#include <stdlib.h> //Included so program can make use of DSP/BIOS configuration tool.
#include "dsp_bios_cfg.h"
#include "dsk6713.h"
#include "dsk6713_aic23.h"
/* The file dsk6713.h must be included in every program that uses the BSL.  This
   example also includes dsk6713_aic23.h because it uses the
   AIC23 codec module (audio interface). */
#include <math.h> //math library (trigonometric functions)
//Some functions to help with writing/reading the audio ports when using interrupts.
#include <helper_functions_ISR.h>
typedef int bool; //Defines boolean type for debugging
#define true 1
#define false 0
#include "fir_coef.txt" /*MODIFICATION_01 - Includes the array of doubles b[].
                                    It represents the filter coefficients.*/
#include <string.h> /*MODIFICATION_01 - Library to use the memset function.*/
#define N (sizeof(b)/sizeof(b[0]))
/*MODIFICATION_01 - N is the order of the filter (number of coefficients).
    This is found by calculating the number of elements in the array of coefficients.
    In our case, N is set to 78. */

/****************************** Global declarations *********************************/

/* Audio port configuration settings: these values set registers in the AIC23 audio
   interface to configure it. See TI doc SLWS106D 3-3 to 3-10 for more info. */
DSK6713_AIC23_Config Config = { \
             /**********************************************************************/
             /*   REGISTER,           FUNCTION       ,     SETTINGS          */
             /**********************************************************************/\
    0x0017,  /* 0 LEFTINVOL  Left line input channel volume  0dB                  */\
    0x0017,  /* 1 RIGHTINVOL Right line input channel volume 0dB                  */\
    0x01f9,  /* 2 LEFTHPVOL  Left channel headphone volume   0dB                  */\
    0x01f9,  /* 3 RIGHTHPVOL Right channel headphone volume  0dB                  */\
    0x0011,  /* 4 ANAPATH    Analog audio path control       DAC on, Mic boost 20dB*/\
```

```c
    0x0000,  /* 5 DIGPATH    Digital audio path control      All Filters off       */\
    0x0000,  /* 6 DPOWERDOWN Power down control              All Hardware on       */\
    0x0043,  /* 7 DIGIF      Digital audio interface format  16 bit                */\
    0x008d,  /* 8 SAMPLERATE Sample rate control             8 KHZ                 */\
    0x0001   /* 9 DIGACT     Digital interface activation    On                    */\
           /********************************************************************/
};


// Codec handle:- a variable used to identify audio interface
DSK6713_AIC23_CodecHandle H_Codec;
int x[N]; /*MODIFICATION_01 - Defines a 78 elements input delay buffer*/

 /******************************* Function prototypes ********************************/
void init_hardware(void);
void init_var(void); /*MODIFICATION_01 - Initialises x[N] to zero using memset*/
void init_HWI(void);
void ISR_AIC(void); //Interrupt function
double non_circ_FIR(void); /*MODIFICATION_01 - Function called by Interrupt function.
    It performs the filtering of the input sample.*/

/********************************** Main routine ************************************/
void main()
{
    init_hardware(); //initialize board and the audio port
    init_var(); /*MODIFICATION_01 - Function initializing variables*/
    init_HWI(); //initialize hardware interrupts
    while(1){}; //loop indefinitely, waiting for interrupts
}

/********************************** init_hardware() *********************************/
void init_hardware()
{
    // Initialize the board support library, must be called first
    DSK6713_init();
    // Start the AIC23 codec using the settings defined above in config
    H_Codec = DSK6713_AIC23_openCodec(0, &Config);
    /* Function below sets the number of bits in word used by MSBSP (serial port) for
    receives from AIC23 (audio port). We are using a 32 bit packet containing two
    16 bit numbers hence 32BIT is set for  receive */
    MCBSP_FSETS(RCR1, RWDLEN1, 32BIT);
    /* Configures interrupt to activate on each consecutive available 32 bits
    from Audio port hence an interrupt is generated for each L & R sample pair */
    MCBSP_FSETS(SPCR1, RINTM, FRM);
    /* These commands do the same thing as above but applied to data transfers to
    the audio port */
    MCBSP_FSETS(XCR1, XWDLEN1, 32BIT);
    MCBSP_FSETS(SPCR1, XINTM, FRM);
}

/************************************* init_var() ***********************************/
void init_var(void) /*MODIFICATION_01 - Initializes variables (array x to zero)*/
{
    memset(x, 0, sizeof(x));
}

/********************************** init_HWI() **************************************/
void init_HWI(void)
{
    IRQ_globalDisable();            // Globally disables interrupt4
    IRQ_nmiEnable();                // Enables the NMI interrupt (used by the debugger)
    IRQ_map(IRQ_EVT_RINT1,4);       // Maps an event to a physical interrupt
    IRQ_enable(IRQ_EVT_RINT1);      // Enables the event
    IRQ_globalEnable();             // Globally enables interrupts
}

/************************* INTERRUPT SERVICE ROUTINE ******************************/
```

```
void ISR_AIC(void)
{
    int n;
    mono_write_16Bit(non_circ_FIR()); /*MODIFICATION_01 - Output filtered sample.*/
    /*MODIFICATION_01 - Move data along buffer from lower element to next higher*/
    for(n = N - 1; n > 0; n--)
    {
        x[n] = x[n - 1];
    }
    x[0] = mono_read_16Bit(); /*MODIFICATION_01 - Stores new sample in delay buffer*/
}

/******************************** non_circ_FIR() ************************************/
double non_circ_FIR(void)
{
/*MODIFICATION_01 - Performs the FIR filtering, so the convolution between
the filter coefficients of b and the samples of the delay buffer x.
It uses indexes to move to the next element in x and b (01). */
    double sum = 0;
    int n;
    for(n = 0; n < N; n++) //note: for loops are 2 cycles faster than while loops.
    {
        sum += x[n] * b[n];
    }
    return sum;
}
```

Appendix 4: Second implementation of FIR filter (intio.c) (only one for loop)

Implementation2_noncircular_index_oneForLoop.c

```
/*************************************************************************************

        ,      DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING
                ,   ,    IMPERIAL COLLEGE LONDON

            ,      EE 3.19: Real Time Digital Signal Processing
                ,       Dr Paul Mitcheson and Daniel Harvey

            ,           , LAB 4: Real-time Implementation of FIR Filters

            ,           ********* I N T I O. C **********

  Implementation of FIR filter using indexes to execute the convolution function.
  MODIFICATIONS made by Quentin McGaw and Alexandra Rouhana, during February 2015
  MODIFICATION_02 correspond to this implementation.

 ************************************************************************************
               Updated for use on 6713 DSK by Danny Harvey: May-Aug 2006
               Updated for CCS V4 Sept 10
 ************************************************************************************/
/*
 *  You should modify the code so that interrupts are used to service the
 *  audio port.
 */
/***************************** Pre-processor statements ******************************/

#include <stdlib.h> //Included so program can make use of DSP/BIOS configuration tool.
#include "dsp_bios_cfg.h"
#include "dsk6713.h"
#include "dsk6713_aic23.h"
/* The file dsk6713.h must be included in every program that uses the BSL.  This
   example also includes dsk6713_aic23.h because it uses the
   AIC23 codec module (audio interface). */
#include <math.h> //math library (trigonometric functions)
//Some functions to help with writing/reading the audio ports when using interrupts.
#include <helper_functions_ISR.h>
typedef int bool; //Defines boolean type for debugging
#define true 1
#define false 0
#include "fir_coef.txt" /*MODIFICATION_01 - Includes the array of doubles b[].
                                    It represents the filter coefficients.*/
#include <string.h> /*MODIFICATION_01 - Library to use the memset function.*/
#define N (sizeof(b)/sizeof(b[0]))
/*MODIFICATION_01 - N is the order of the filter (number of coefficients).
    This is found by calculating the number of elements in the array of coefficients.
    In our case, N is set to 78. */

/******************************** Global declarations ********************************/

/* Audio port configuration settings: these values set registers in the AIC23 audio
   interface to configure it. See TI doc SLWS106D 3-3 to 3-10 for more info. */
DSK6713_AIC23_Config Config = { \
             /**********************************************************************/
             /*   REGISTER,            FUNCTION      ,     SETTINGS         */
             /**********************************************************************/\
    0x0017,  /* 0 LEFTINVOL  Left line input channel volume  0dB                   */\
    0x0017,  /* 1 RIGHTINVOL Right line input channel volume 0dB                   */\
    0x01f9,  /* 2 LEFTHPVOL  Left channel headphone volume   0dB                   */\
    0x01f9,  /* 3 RIGHTHPVOL Right channel headphone volume  0dB                   */\
    0x0011,  /* 4 ANAPATH    Analog audio path control       DAC on, Mic boost 20dB*/\
```

```
    0x0000,  /* 5 DIGPATH    Digital audio path control      All Filters off       */\
    0x0000,  /* 6 DPOWERDOWN Power down control              All Hardware on       */\
    0x0043,  /* 7 DIGIF      Digital audio interface format  16 bit                */\
    0x008d,  /* 8 SAMPLERATE Sample rate control             8 KHZ                 */\
    0x0001   /* 9 DIGACT     Digital interface activation    On                    */\
            /*********************************************************************/
};


// Codec handle:- a variable used to identify audio interface
DSK6713_AIC23_CodecHandle H_Codec;
int x[N]; /*MODIFICATION_01 - Defines a 78 elements input delay buffer*/

 /******************************* Function prototypes ********************************/
void init_hardware(void);
void init_var(void); /*MODIFICATION_01 - Initialises x[N] to zero using memset*/
void init_HWI(void);
void ISR_AIC(void); //Interrupt function
double non_circ_FIR(void); /*MODIFICATION_01 - Function called by Interrupt function.
    It performs the filtering of the input sample.*/

/********************************** Main routine ************************************/
void main()
{
    init_hardware(); //initialize board and the audio port
    init_var(); /*MODIFICATION_01 - Function initializing variables*/
    init_HWI(); //initialize hardware interrupts
    while(1){}; //loop indefinitely, waiting for interrupts
}

/********************************** init_hardware() *********************************/
void init_hardware()
{
    // Initialize the board support library, must be called first
    DSK6713_init();
    // Start the AIC23 codec using the settings defined above in config
    H_Codec = DSK6713_AIC23_openCodec(0, &Config);
    /* Function below sets the number of bits in word used by MSBSP (serial port) for
    receives from AIC23 (audio port). We are using a 32 bit packet containing two
    16 bit numbers hence 32BIT is set for  receive */
    MCBSP_FSETS(RCR1, RWDLEN1, 32BIT);
    /* Configures interrupt to activate on each consecutive available 32 bits
    from Audio port hence an interrupt is generated for each L & R sample pair */
    MCBSP_FSETS(SPCR1, RINTM, FRM);
    /* These commands do the same thing as above but applied to data transfers to
    the audio port */
    MCBSP_FSETS(XCR1, XWDLEN1, 32BIT);
    MCBSP_FSETS(SPCR1, XINTM, FRM);
}

/************************************* init_var() ***********************************/
void init_var(void) /*MODIFICATION_01 - Initializes variables (array x to zero)*/
{
    memset(x, 0, sizeof(x));
}

/********************************** init_HWI() **************************************/
void init_HWI(void)
{
    IRQ_globalDisable();            // Globally disables interrupt4
    IRQ_nmiEnable();                // Enables the NMI interrupt (used by the debugger)
    IRQ_map(IRQ_EVT_RINT1,4);       // Maps an event to a physical interrupt
    IRQ_enable(IRQ_EVT_RINT1);      // Enables the event
    IRQ_globalEnable();             // Globally enables interrupts
}

/************************** INTERRUPT SERVICE ROUTINE *******************************/
```

```c
void ISR_AIC(void)
{
    mono_write_16Bit(non_circ_FIR()); /*MODIFICATION_01 - Output filtered sample.*/
    x[0] = mono_read_16Bit(); /*MODIFICATION_01 - Stores new sample in delay buffer*/
}

/****************************** non_circ_FIR() ***********************************/
double non_circ_FIR(void)
{
/*MODIFICATION_01 - Performs the FIR filtering so the convolution between
the filter coefficients b and the samples in the delay buffer x.
It uses indexes to move to the next element in x and b (01). */
/*MODIFICATION_02 - The shifting is done in the MAC protocol.
    Only one for loop is needed now. */
    int n;
    double sum = x[0] * b[0];
    for(n = N-1; n > 0; n--)
    {
        sum += x[n] * b[n];
        x[n] = x[n - 1]; /*MODIFICATION_02 - Shifting in MAC protocol.*/
    }
    return sum;
}
```

Appendix 5: Third implementation of FIR filter (intio.c) (using pointers)

Implementation3_noncircular_pointers.c

```
/*************************************************************************************
            ,       DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING
                       ,   ,     IMPERIAL COLLEGE LONDON

                 ,       EE 3.19: Real Time Digital Signal Processing
                     ,       Dr Paul Mitcheson and Daniel Harvey

                 ,            , LAB 4: Real-time Implementation of FIR Filters

                 ,            ********* I N T I O. C **********

  Implementation of FIR filter using indexes to execute the convolution function.
  MODIFICATIONS made by Quentin McGaw and Alexandra Rouhana, during February 2015
  MODIFICATION_03 correspond to this implementation.

 ************************************************************************************
              Updated for use on 6713 DSK by Danny Harvey: May-Aug 2006
              Updated for CCS V4 Sept 10
 ************************************************************************************/
/*
 *  You should modify the code so that interrupts are used to service the
 *  audio port.
 */
/**************************** Pre-processor statements ******************************/

#include <stdlib.h> //Included so program can make use of DSP/BIOS configuration tool.
#include "dsp_bios_cfg.h"
#include "dsk6713.h"
#include "dsk6713_aic23.h"
/* The file dsk6713.h must be included in every program that uses the BSL.  This
   example also includes dsk6713_aic23.h because it uses the
   AIC23 codec module (audio interface). */
#include <math.h> //math library (trigonometric functions)
//Some functions to help with writing/reading the audio ports when using interrupts.
#include <helper_functions_ISR.h>
typedef int bool; //Defines boolean type for debugging
#define true 1
#define false 0
#include "fir_coef.txt" /*MODIFICATION_01 - Includes the array of doubles b[].
                                     It represents the filter coefficients.*/
#include <string.h> /*MODIFICATION_01 - Library to use the memset function.*/
#define N (sizeof(b)/sizeof(b[0]))
/*MODIFICATION_01 - N is the order of the filter (number of coefficients).
    This is found by calculating the number of elements in the array of coefficients.
    In our case, N is set to 78. */

/****************************** Global declarations *********************************/

/* Audio port configuration settings: these values set registers in the AIC23 audio
   interface to configure it. See TI doc SLWS106D 3-3 to 3-10 for more info. */
DSK6713_AIC23_Config Config = { \
             /*******************************************************************/
             /*   REGISTER,            FUNCTION      ,     SETTINGS         */
             /*******************************************************************/\
    0x0017,  /* 0 LEFTINVOL  Left line input channel volume  0dB                 */\
    0x0017,  /* 1 RIGHTINVOL Right line input channel volume 0dB                 */\
    0x01f9,  /* 2 LEFTHPVOL  Left channel headphone volume   0dB                 */\
    0x01f9,  /* 3 RIGHTHPVOL Right channel headphone volume  0dB                 */\
    0x0011,  /* 4 ANAPATH    Analog audio path control       DAC on, Mic boost 20dB*/\
    0x0000,  /* 5 DIGPATH    Digital audio path control      All Filters off     */\
```

```
    0x0000,  /* 6 DPOWERDOWN Power down control               All Hardware on       */\
    0x0043,  /* 7 DIGIF      Digital audio interface format  16 bit                */\
    0x008d,  /* 8 SAMPLERATE Sample rate control             8 KHZ                 */\
    0x0001   /* 9 DIGACT     Digital interface activation    On                    */\
            /**********************************************************************/
};


// Codec handle:- a variable used to identify audio interface
DSK6713_AIC23_CodecHandle H_Codec;
int x[N]; /*MODIFICATION_01 - Defines a 78 elements input delay buffer*/

 /******************************* Function prototypes ********************************/
void init_hardware(void);
void init_var(void); /*MODIFICATION_01 - Initialises x[N] to zero using memset*/
void init_HWI(void);
void ISR_AIC(void); //Interrupt function
double non_circ_FIR(void); /*MODIFICATION_01 - Function called by Interrupt function.
    It performs the filtering of the input sample.*/

/********************************** Main routine ************************************/
void main()
{
    init_hardware(); //initialize board and the audio port
    init_var(); /*MODIFICATION_01 - Function initializing variables*/
    init_HWI(); //initialize hardware interrupts
    while(1){}; //loop indefinitely, waiting for interrupts
}

/********************************** init_hardware() *********************************/
void init_hardware()
{
    // Initialize the board support library, must be called first
    DSK6713_init();
    // Start the AIC23 codec using the settings defined above in config
    H_Codec = DSK6713_AIC23_openCodec(0, &Config);
    /* Function below sets the number of bits in word used by MSBSP (serial port) for
    receives from AIC23 (audio port). We are using a 32 bit packet containing two
    16 bit numbers hence 32BIT is set for  receive */
    MCBSP_FSETS(RCR1, RWDLEN1, 32BIT);
    /* Configures interrupt to activate on each consecutive available 32 bits
    from Audio port hence an interrupt is generated for each L & R sample pair */
    MCBSP_FSETS(SPCR1, RINTM, FRM);
    /* These commands do the same thing as above but applied to data transfers to
    the audio port */
    MCBSP_FSETS(XCR1, XWDLEN1, 32BIT);
    MCBSP_FSETS(SPCR1, XINTM, FRM);
}

/************************************* init_var() ***********************************/
void init_var(void) /*MODIFICATION_01 - Initializes variables (array x to zero)*/
{
    memset(x, 0, sizeof(x));
}

/********************************** init_HWI() **************************************/
void init_HWI(void)
{
    IRQ_globalDisable();             // Globally disables interrupt4
    IRQ_nmiEnable();                 // Enables the NMI interrupt (used by the debugger)
    IRQ_map(IRQ_EVT_RINT1,4);        // Maps an event to a physical interrupt
    IRQ_enable(IRQ_EVT_RINT1);       // Enables the event
    IRQ_globalEnable();              // Globally enables interrupts
}

/************************* INTERRUPT SERVICE ROUTINE *******************************/
void ISR_AIC(void)
```

```
{
    mono_write_16Bit(non_circ_FIR()); /*MODIFICATION_01 - Output filtered sample.*/
    x[0] = mono_read_16Bit(); /*MODIFICATION_01 - Stores new sample in delay buffer*/
}

/******************************** non_circ_FIR() ************************************/
double non_circ_FIR(void)
{
/*MODIFICATION_03 - Performs the FIR filtering so the convolution between
the filter coefficients b and the samples in the delay buffer x.
It uses pointers to move to the next element in x and b (02). */
    int n;
    double sum;
    int *xp = x; /*MODIFICATION_03 - Pointer to first element of x*/
    double *bp = b; /*MODIFICATION_03 - Pointer to first element of b*/
    sum = (*xp++) * (*bp++); /*MODIFICATION_03 - MAC using pointers*/
    for(n = N - 1; n > 0; n--)
    {
        sum += (*xp++) * (*bp++); /*MODIFICATION_03 - MAC using pointers*/
        x[n] = x[n - 1]; /*MODIFICATION_01 - Move data along buffer from lower element
to next higher*/
    }
    return sum;
}
```

Appendix 6: Fourth implementation of FIR filter (intio.c) (circular buffer, using pointers)

Implementation4_circular_pointers.c

```
/*************************************************************************************

       ,      DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING
               ,   ,    IMPERIAL COLLEGE LONDON

           ,     EE 3.19: Real Time Digital Signal Processing
               ,      Dr Paul Mitcheson and Daniel Harvey

           ,           , LAB 4: Real-time Implementation of FIR Filters

           ,           ********* I N T I O. C **********

  Implementation of FIR filter using indexes to execute the convolution function.
  MODIFICATIONS made by Quentin McGaw and Alexandra Rouhana, during February 2015
  MODIFICATION_04 correspond to this implementation.

 ************************************************************************************
               Updated for use on 6713 DSK by Danny Harvey: May-Aug 2006
               Updated for CCS V4 Sept 10
 ************************************************************************************/
/*
 *  You should modify the code so that interrupts are used to service the
 *  audio port.
 */
/**************************** Pre-processor statements ******************************/

#include <stdlib.h> //Included so program can make use of DSP/BIOS configuration tool.
#include "dsp_bios_cfg.h"
#include "dsk6713.h"
#include "dsk6713_aic23.h"
/* The file dsk6713.h must be included in every program that uses the BSL.  This
   example also includes dsk6713_aic23.h because it uses the
   AIC23 codec module (audio interface). */
#include <math.h> //math library (trigonometric functions)
//Some functions to help with writing/reading the audio ports when using interrupts.
#include <helper_functions_ISR.h>
typedef int bool; //Defines boolean type for debugging
#define true 1
#define false 0
#include "fir_coef.txt" /*MODIFICATION_01 - Includes the array of doubles b[].
                                    It represents the filter bp.*/
#include <string.h> /*MODIFICATION_01 - Library to use the memset function.*/
#define N (sizeof(b)/sizeof(b[0]))
/*MODIFICATION_01 - N is the order of the filter (number of coefficients).
    This is found by calculating the number of elements in the array of coefficients.
    In our case, N is set to 78. */

/****************************** Global declarations *********************************/

/* Audio port configuration settings: these values set registers in the AIC23 audio
   interface to configure it. See TI doc SLWS106D 3-3 to 3-10 for more info. */
DSK6713_AIC23_Config Config = { \
             /*******************************************************************/
             /*   REGISTER,            FUNCTION       ,     SETTINGS         */
             /*******************************************************************/\
    0x0017,  /* 0 LEFTINVOL  Left line input channel volume  0dB                 */\
    0x0017,  /* 1 RIGHTINVOL Right line input channel volume 0dB                 */\
    0x01f9,  /* 2 LEFTHPVOL  Left channel headphone volume   0dB                 */\
    0x01f9,  /* 3 RIGHTHPVOL Right channel headphone volume  0dB                 */\
    0x0011,  /* 4 ANAPATH    Analog audio path control       DAC on, Mic boost 20dB*/\
    0x0000,  /* 5 DIGPATH    Digital audio path control      All Filters off     */\
```

```
    0x0000,  /* 6 DPOWERDOWN Power down control              All Hardware on       */\
    0x0043,  /* 7 DIGIF      Digital audio interface format  16 bit                */\
    0x008d,  /* 8 SAMPLERATE Sample rate control             8 KHZ                 */\
    0x0001   /* 9 DIGACT     Digital interface activation    On                    */\
            /**********************************************************************/
};


// Codec handle:- a variable used to identify audio interface
DSK6713_AIC23_CodecHandle H_Codec;
int x[N]; /*MODIFICATION_01 - Defines a 78 elements input delay buffer*/
int offset = N - 1; /*MODIFICATION_04 - Used for the circular buffer. */

 /******************************* Function prototypes ********************************/
void init_hardware(void);
void init_var(void); /*MODIFICATION_01 - Initialises x[N] to zero using memset*/
void init_HWI(void);
void ISR_AIC(void); //Interrupt function
double circ_FIR(double sample_in);
/*MODIFICATION_04 - Function called by Interrupt function.
It performs the filtering of the input samples
This function uses a circular buffer using pointers. (04)*/

/********************************** Main routine ************************************/
void main()
{
    init_hardware(); //initialize board and the audio port
    init_var(); /*MODIFICATION_01 - Function initializing variables*/
    init_HWI(); //initialize hardware interrupts
    while(1){}; //loop indefinitely, waiting for interrupts
}

/********************************** init_hardware() *********************************/
void init_hardware()
{
    // Initialize the board support library, must be called first
    DSK6713_init();
    // Start the AIC23 codec using the settings defined above in config
    H_Codec = DSK6713_AIC23_openCodec(0, &Config);
    /* Function below sets the number of bits in word used by MSBSP (serial port) for
    receives from AIC23 (audio port). We are using a 32 bit packet containing two
    16 bit numbers hence 32BIT is set for  receive */
    MCBSP_FSETS(RCR1, RWDLEN1, 32BIT);
    /* Configures interrupt to activate on each consecutive available 32 bits
    from Audio port hence an interrupt is generated for each L & R sample pair */
    MCBSP_FSETS(SPCR1, RINTM, FRM);
    /* These commands do the same thing as above but applied to data transfers to
    the audio port */
    MCBSP_FSETS(XCR1, XWDLEN1, 32BIT);
    MCBSP_FSETS(SPCR1, XINTM, FRM);
}

/************************************* init_var() ***********************************/
void init_var(void) /*MODIFICATION_01 - Initializes variables (array x to zero)*/
{
    memset(x, 0, sizeof(x));
}

/********************************** init_HWI() **************************************/
void init_HWI(void)
{
    IRQ_globalDisable();            // Globally disables interrupt4
    IRQ_nmiEnable();                // Enables the NMI interrupt (used by the debugger)
    IRQ_map(IRQ_EVT_RINT1,4);       // Maps an event to a physical interrupt
    IRQ_enable(IRQ_EVT_RINT1);      // Enables the event
    IRQ_globalEnable();             // Globally enables interrupts
}
```

```c
/************************** INTERRUPT SERVICE ROUTINE ********************************/
void ISR_AIC(void)
{
    mono_write_16Bit(circ_FIR(mono_read_16Bit()));
    /*MODIFICATION_01 - Output filtered sample.*/
}

/********************************** circ_FIR() ***************************************/
double circ_FIR(double sample_in)
{
/*MODIFICATION_04 - Performs the FIR filtering.
This function uses a circular buffer in order to execute the convolution. (04)*/
    double sum = 0;
    double *bp = b; /*MODIFICATION_04 - Coefficient variable pointer.
                            This pointer is first set to the first element of b.*/
    int *x_start = x; /*MODIFICATION_04 - Input buffer constant pointer.
                            This pointer points to the first element of x.*/
    double *bp_end = b + N; /*MODIFICATION_04 - Coefficient constant pointer.
                            This pointer points to the last element of b.*/
    int *xp = x_start + offset; /*MODIFICATION_04 - Input buffer variable pointer.
                            This pointer is first set to the 'offset'th element of
x.*/
    *xp = sample_in; /*MODIFICATION_04 - Stores the new sample at the address of xp.*/
    while(xp >= x_start)
    {
        sum += (*xp--) * (*bp++);
        /*MODIFICATION_04 - Shifts the address of the element of the input buffer
        from the 'offset' address to the first element's address (x_start).
        It shifts at the same time the address of the element of the coefficients
        from the first element's address to the 'offset' address.*/
    }
    xp = x_start + N - 1; /*MODIFICATION_04 -Reinitialises xp to the end of the
array.*/
    while(bp < bp_end)
    {
        sum += (*xp--) * (*bp++);
        /*MODIFICATION_04 - Shifts the address of the element of the coefficients
        from the 'offset' address to the last element's address (bp_end).
        It shifts at the same time the address of the element of the input buffer
        from the last (bp_end) element's address to the 'bp_end - offset' address.*/
    }
    offset--; /*MODIFICATION_04 - Decrements the offset*/
    if(offset < 0)
    {
        offset = N - 1; /*MODIFICATION_04 - Restores the offset to N-1 when out of
bound.*/
    }
    return sum;
}
```