

EE3-19

# Real Time Digital Signal Processing

Course homepage: <http://learn.imperial.ac.uk>

## *Lab 4 – Real-time Implementation of FIR Filters*

Paul D. Mitcheson  
[paul.mitcheson@imperial.ac.uk](mailto:paul.mitcheson@imperial.ac.uk)  
Room 1112, EEE

Imperial College  
London

## Objectives

- Learn to design FIR filters using Matlab.
- Implement the FIR filter using the C6713 DSK system in real-time in C and assembly
- Make your FIR filter operate as fast as you can
- Measure the filter characteristics using a network or spectrum analyzer.

## The Problem

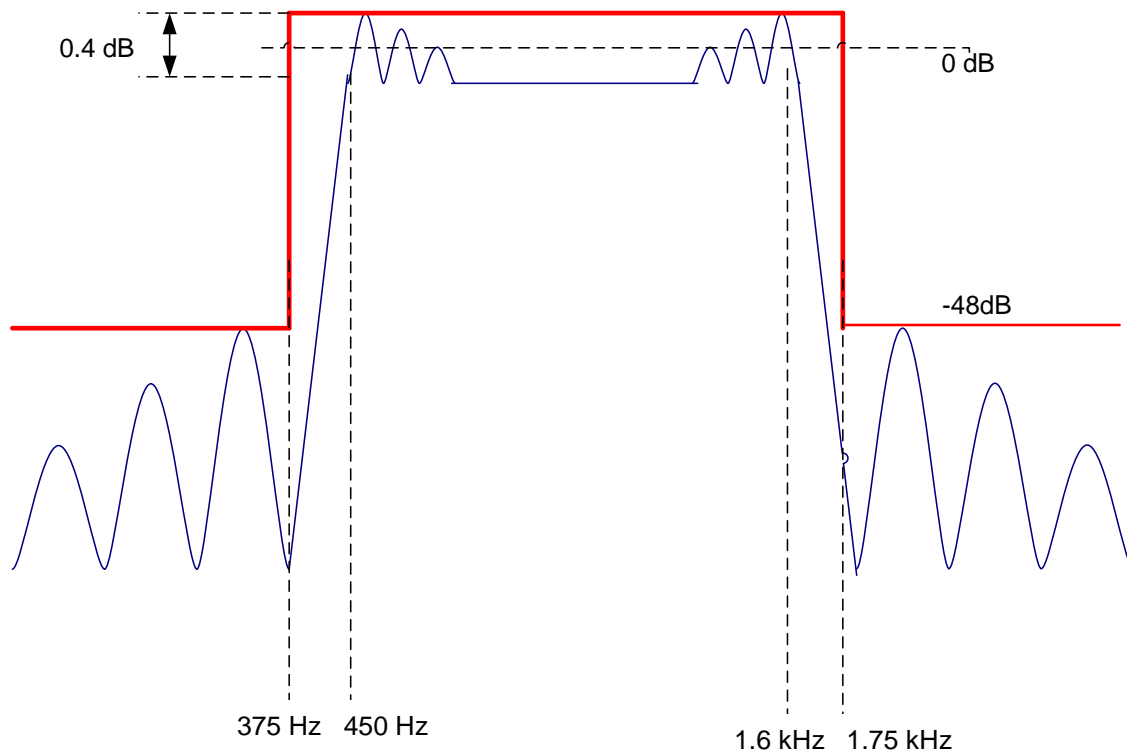
The transfer function of an  $M^{\text{th}}$  order FIR filter is given by:

$$H(z) = b_0 + b_1z^{-1} + \dots + b_Mz^{-M}$$

The corresponding time-domain difference equation is:

$$y(n) = b_0x(n) + b_1x(n-1) + \dots + b_Mx(n-M)$$

Our objective in this lab session is to design a FIR filter that satisfies the following specification (this drawing is conceptual and so is not to scale):



## ***Filter Design Using Matlab***

Your first step is to design this filter in Matlab using the Parks-McClelland algorithm. Read the description of the Matlab functions `firpm` and `firpmord` using Matlab help. You may also find the function `freqz` useful in calculating the frequency response of the filter. Also remember the sampling frequency of the C6713 system will be set to 8 kHz.

Using Matlab you should plot the frequency response of your filter showing clearly that the passband and stopband specifications are met. The resultant coefficients for your filter should be written to a text file called `fir_coef` in a format suitable for inclusion in a C program. For example, this file might contain:

```
// FIR filter coefficients
double b[] = {1.03344530e-0.2, 0.2356333e-0.3, -0.039871111-e0.1, ... };
```

The `save` command in Matlab is one method of saving workspace variables to a text file:

Save filename.txt coefs – ASCII –DOUBLE -TABS

This will save a variable called `coefs` to a file called `filename.txt` as double floating point variables delimited by tabs.

If you use any other method you may need to ensure that the coefficients are cast to the correct precision by issuing the command at the Matlab prompt:

```
format long e
```

To read the text file into your program:

```
# include "fir_coef.txt "
```

Because you will be measuring and comparing performance for different filter implementations you must use the same precision (double floating point) for all your coefficients and calculations in this lab.

## Filter implementation

You should build this on a copy of the Lab 3 session on interrupt I/O. Make a copy of the project folder for lab 3, rename it **lab4**. Make the necessary edits to *intio.c* as described below.

For the proper operation of the FIR filter it is required that the current sample (read using the special function `mono_read_16Bit()`) and the  $M-1$  previous samples be processed at the same time, where  $M$  is the order of the filter. Therefore, your interrupt service routine should perform the following 4 tasks:

### 1. Read the input sample from the codec.

### 2. Perform the delay operator

For an initial implementation of the delay operator an array shuffling method will suffice:

```
// define a 12 element delay buffer

#define N 12
double x[N];

//..somewhere in the ISR...

for (i = N-1;i>0;i--)
{
    x[i]=x[i-1];    /* move data along buffer from lower */
}                  /* element to next higher */
x[0] = sample_in;  /* put new sample into buffer */
```

### 3. Perform the convolution function.

### 4. Output $y(n)$ to the codec.

Implement the actual filtering code within a separate function that is called by the ISR (for example I called my filtering function `non_circ_FIR()`) This will make the measurement of instruction cycles easier.<sup>1</sup>

Test your implementation by attaching (at a suitable frequency) a sine wave to the input of the board (check the amplitude is not so large as to damage the input to the DSK<sup>2</sup> before making the connection!) Observe the output from the DSP (You can do this with any scope). A sinewave signal identical to the input (with no harmonics or distortions) should start to appear on the output and increase in amplitude around the first transition band and the opposite should occur at the next transition band. If you are having problems getting the correct response read appendix A: (debugging MAC algorithms.)

---

<sup>1</sup> A final executable would be compiled using the O3 compiler optimisation which removes the overhead of function calls, so the cost of doing this is not really an issue.

<sup>2</sup> Remember that the audio input is rated at  $2V_{RMS}$  which corresponds to 5.66V peak to peak. Do not exceed this voltage!

## ***Improving Algorithm performance with the compiler***

Benchmark the ISR of your algorithm in terms of instruction cycles (using the profiling clock method described in lab1). Do this for compiled versions of your code with the following compiler optimization options: <sup>3</sup> (1) no optimization (2) Option set to -o0 (3) option set to -o2. Take measurements a few times until the cycle count minimises and use the best case that you find for each filter implementation.

## ***Measurement of filter response***

Only use the spectrum analysers once you have checked that the filter is working using a scope and signal generator. Use one of the SR780 spectrum analyzers or the Audio Precision APX520 in the laboratory to measure the frequency response of your working system. Copies of the relevant pages from the SR780 manual and an explanation of the APX520/515 are included in the appendix.

**Compare the results of the SR780/APX520/APX515 with those obtained in Matlab.**

## ***A Faster Implementation***

Now convert your FIR filter to utilise circular buffers. Try to make it run as fast as you can. Then compare the run times of your best circular buffer implementation with your shuffle buffer implementation at each optimisation level.

---

<sup>3</sup> -o3 optimisation does not work with the profiling clock measurement method as the breakpoints can no longer be used.

***The assembly code aspect of this lab is optional for those in need of a challenge! This part is for feedback only, and not assessed.***

## **Assembly version of FIR filter using circular buffering hardware**

The 6713 can use circular buffering hardware that eliminates the overhead of making the pointer “wrap” at the end of the buffer. In this section, working with an ASM code template you are going to set up the register that switches on circular buffering and write a 64 Bit IEEE floating point MAC algorithm that uses the circular buffer you configured. Finally you will streamline the code by making use of the parallelism and multiple function units of the hardware.

Read the TI Instruction Set Reference Guide (SPRU 733A) pages 2-10 to 2-12. This explains the register that allows the circular buffering to be made available to several registers. Two important restrictions for the buffer are it has to be an integer power of 2 in length in bytes (up to power 32) and will need to be aligned to a byte boundary equal to the buffer size in bytes. Also note since the buffer will store 64 bit floating point data the number of data elements it can hold will be:

$$(\text{buffer size in bytes})/(\text{Data element size in bytes})$$

The buffer size (in data elements) should be made larger than the coefficient array.

### **Preliminary setup**

Declare an array called `x_buffer` (as a global) in C, to the constraints given above, that will hold your delay buffer. Data can be made to align to the block size byte boundary by following the declaration with;

```
#pragma DATA_ALIGN(x_buffer, BUFFER_BYTE_SIZE)
```

Declare a global pointer called `X_PTR` that is initialised at declaration to point to the beginning of the buffer. This pointer will be used by the assembly code to locate where the next new sample should be stored each time the function is called.

Write a routine to initialise the delay buffer to all zeros.

Add the ASM template file into your project.

Add the following prototype into your C code:

```
extern void circ_FIR_DP(double **ptr, double *coef, double *input_samp, double  
*filtered_samp ,unsigned int numCoefs);
```

Data is passed and returned from the function using a number of arguments that require pointers. To pass the filtered sample for example just precede the variable name by the `&` symbol to obtain the address where the variable is stored. The ASM code can then modify the value held at this address and return the result of the calculation to C.

One parameter required is pointer to a pointer (\*\*ptr). This should be passed the address of the pointer you declared above. In this way the ASM code can modify the address that the pointer holds and store it back into the pointer.

In the appropriate place in your C code write a call to the assembly FIR function.

### ***Writing a MAC that operates on double precision floating point data***

Read through the comments in the ASM code to understand how the code works. Notice that delays have been inserted and no attempt has been made to use the parallelism of the hardware. This is an exercise for you!

Modify the AMR register set up routine so that the buffer size is correctly set.

Write the MAC routine with the required delays into the loop. You will need to browse SPRU 733A to find the necessary instructions.

Check that the code works using a signal generator and scope. When the code is working benchmark the code using the profiling tools. Modify the ASM code to make the code more efficient (make sure the code still works!).

Repeat until you can get a MAC that consumes the lowest number of cycles possible then compare your results with the C versions of the filter you wrote.

## ***Deliverables***

As before, you do not have to write a formal report including abstract, conclusions etc for this lab but the report should be tidy and comprehensible. Ensure you cover the points made below.

Marks are awarded as follows:

- Correct design of the filter in Matlab with proof of the expected frequency response [4]
- Implementation of a basic non-circular FIR filter in C with explanation of your code. Include benchmark data at different compiler optimisation levels. [6]
- Implementation of a circular buffer in C with explanation of how it works. Include benchmark data at different compiler optimisation levels. More marks are available for faster implementations of your buffer. If you write several versions of a circular buffer which operate at different speeds then you should explain them all. [14]
- Show the frequency response for your fastest implementation in C from network analyser. Is your filter linear phase? Is the gain what you expect? [5]

### ***Optional work for those in need of a challenge! For feedback only, and not assessed.***

- Assembly language implementation of working FIR filter in assembly. Explain what you have done.
- Additional marks for optimising you assembly code – with explanations of your modifications.
- Discussion of the relative performance of your different implementations of filter at different compiler optimisation levels. Look in the TI manuals to see a description of what the compiler does at different optimisation levels.

Please submit your assignment formatted as a pdf/word/other turnitin compatible document via the blackboard upload page for lab4.



## ***Revision History***

20 <sup>th</sup> Jan 2010	Added APX500 notes and ASM MAC question
11 <sup>th</sup> Feb 2010	Added mark allocation information
18 <sup>th</sup> Nov 2010	Updated for CCS4 and Windows 7
2 <sup>nd</sup> Feb 2011	Modified mark allocations
18 <sup>th</sup> Jan 2014	Made latter parts unassessed to reduce workload for new EE3 structure

## ***Appendix A: Debugging MAC Algorithms***

This technique may be useful if your code has bugs that are difficult to find. In a nutshell it involves clocking some easy calculable data through the algorithm and ensuring that the correct results are obtained at the output.

The example describes the testing of a 2<sup>nd</sup> order FIR filter i.e. three elements:

$$y(0) = b_0.x_0 + b_1.x_1 + b_2.x_2$$

1. Set coefficients to easily calculable values. For example set the coefficients to:

$$\{b_0 \ b_1 \ b_2\} \text{ to } \{1, -2, 3\}$$

2. Don't have too many coefficients, keep the arithmetic simple.
3. Ensure that any delay buffers (input and output) have been set to zero. i.e.

$$\{X_0 \ X_1 \ X_2 \} \text{ to } \{0, 0, 0\}$$

Note that this zeroing functionality should have already been implemented as part of the initialization of your algorithm.

4. "Hard wire" a constant 1 to the input of your delay line. The filter will now receive a step that will move through the delay buffer on every cycle of your ISR code (see table below). You may want to modify this idea, e.g. define a global variable, and then implement a counter that counts from -10 to 10 then resets to -10 and repeats the pattern. Use this as your input to the filter. (This modification could be useful if you are having problems with implementing your input delay buffer as you will be able to see the consecutive values propagate through the buffer.)
5. Set breakpoints in your ISR at appropriate points, for example in the last line of your ISR (so you can check the final sum of the MAC).
6. Place watches as necessary on the variables of interest e.g. the sum, loop counters, partial sums.

7. Using your chosen test data, hand calculate the values and put the data in a table. Each row should represent an execution of your ISR code. For the IIR filter you will need extra columns for the output delay buffer!

b[0]=1	b[1]=-2	b[2]=3
--------	---------	--------

Each row represents the results of a complete execution of the ISR

x[0]	x[1]	x[2]	y sum
0	0	0	<b>0</b>
1	0	0	<b>1</b>
1	1	0	<b>-1</b>
1	1	1	<b>2</b>

8. Run the code to the breakpoints and compare the data given in your table to that given in the watch table. Any differences in results should help indicate where the bugs are.

## Appendix B: The SR780 User Manual (pages 1-11 to 1-14)

### Measuring a Transfer Function 1-11

## Measuring a Transfer Function

This example investigates the transfer function of the test filter (enclosed with this manual) using FFT measurements. You will use the SR780 source to provide a broad band chirp and both input channels to measure the input to and output from the device under test.

1. Press [System]  Press <Preset>  Press [Enter] to confirm Preset.	Display the System menu.  Preset returns the unit to its default settings.  Preset requires confirmation to prevent accidental reset. Wait until the self tests are completed.
2. Use a BNC Tee to connect the Source Output to the filter input and the Ch1 A Input.  Connect the filter output to the Ch2 A Input.	In this instrument, transfer function is defined as Ch2 response over Ch1 reference. Thus, Ch1 monitors the filter input (source output) and Ch2 measures the response of the device under test.
3. Press [Source]  Press <Chirp>  Press [Window]  Press <Window>  Select (Uniform) with the knob and press [Enter].	Select the Source menu.  Choose Chirp output. The output is an equal amplitude sine wave at each frequency bin of the FFT spectrum.  Select the Window menu.  Adjust the FFT Window function.  The Chirp source requires the use of the Uniform window since not all chirp frequency components are present at all points in the time record. The chirp is exactly periodic with the FFT time record and does not 'leak' with the uniform window.
4. Press [Auto Range Ch1]  Press [Auto Range Ch2]	Let the analyzer automatically set the Input Ranges to agree with the signals. Note that the Input Range readouts at the top of the screen are displayed in inverse when Auto Range is on.
5. Press [Freq]  Press <Span>  Use the knob to adjust the Span to 6.4 kHz and press [Enter].	Select the Frequency menu.  Adjust the FFT Span.  Set the Span to display the filter notch at 1 kHz.

SR780 Network Signal Analyzer

## 1-12 Measuring a Transfer Function

	<p>The top display (A) is measuring the filter input and should show a fairly flat spectrum. The bottom display (B) is measuring the filter output and should show a deep notch.</p> <p>Both displays are measuring absolute signal levels.</p>
<p>6. Press [Display Setup]</p> <p>Press &lt;Measurement&gt;</p> <p>Select (&lt;F2/F1&gt;) with the knob and press [Enter].</p> <p>Press &lt;Units&gt;</p> <p>Select (dB) with the knob and press [Enter].</p> <p>Press [Auto Scale A]</p>	<p>Select the Display Setup menu.</p> <p>Adjust the Measurement of the active display (A).</p> <p>Choose Transfer Function for the Measurement in DisplayA (top).</p> <p>Transfer Function is the ratio of the response (Ch2) to the input (Ch1) and is a unitless quantity.</p> <p>Change the Units.</p> <p>Choose dB units for the Transfer Function.</p> <p>Adjust the scale and reference for DisplayA to show the entire range of the data.</p>
<p>7. Press [Marker]</p> <p>Press &lt;Width&gt;</p> <p>Select (Normal) with the knob and press [Enter].</p> <p>Press &lt;Seeks&gt;</p> <p>Select (Min) with the knob and press [Enter].</p> <p>Move the Marker Region with the knob to find the notch frequency and depth. Or press [Marker Min].</p>	<p>Select the Marker menu.</p> <p>Adjust the Marker Width for DisplayA.</p> <p>Change to Normal Width (1/2 division).</p> <p>Adjust what the Marker Seeks within the Marker Region.</p> <p>Seek the Minimum of the data within the Marker Region.</p> <p>The Marker Region makes it easy to find narrow peaks and valleys in the graph. The notch should be around 1 kHz and about -60 dB deep.</p>
<p>8. Press [Display Options]</p> <p>Press &lt;X-Axis&gt;</p> <p>Select (Log) with the knob and press [Enter].</p>	<p>Select the Display Options menu.</p> <p>The graph might look better on a log x axis.</p> <p>Log scale is a common way to display filter response functions.</p>
<p>9. Let's show phase response on DisplayB (bottom).</p>	<p>The two displays have separate Measurements.</p>

You may wish to keep the x scale linear for easier comparisons to the Matlab plot.

SR780 Network Signal Analyzer

Press [Active Display]	Make DisplayB the active display. The active display has its Marker Position Bar (above the graph) highlighted.
Press [Display Setup]	Select the Display Setup menu. The setup of DisplayB (the active display) is now shown in the menu.
Press <Measurement>	Adjust the Measurement of DisplayB.
Select (<F2/F1>) with the knob and press [Enter].	Choose Transfer Function also.
Press <View>	The measured data is a set of complex values which can be viewed in a number of different ways.
Select (Phase) with the knob and press [Enter].	Choose Phase View to show the phase of the transfer function.
Press [Auto Scale B]	Scale DisplayB to show the entire phase transfer function.
10. Press [Display Options]	Select the Display Options menu.
Press <X-Axis>	The graph looks better on a log x axis.
Select (Log) with the knob and press [Enter].	Now both displays have a log x axis.
11. Let's link the Markers together.	
Press [Active Display]	Make DisplayA (top) the active display.
Press [Marker]	Select the Marker menu.
Press <Width>	Adjust the Marker Width of DisplayA.
Select (Spot) with the knob and press [Enter].	Change the Marker Width to Spot.
12. Press [Link] and use the knob to move the marker.	The [Link] key links the two display markers together. This allows simultaneous readout of Transfer Function Magnitude (top) and Phase (bottom).
Press [Enter]	Pressing any key removes the link between the markers.

When you submit your coursework you may wish to work out how to display the phase un-wrapped here.

## 1-14 Measuring a Transfer Function

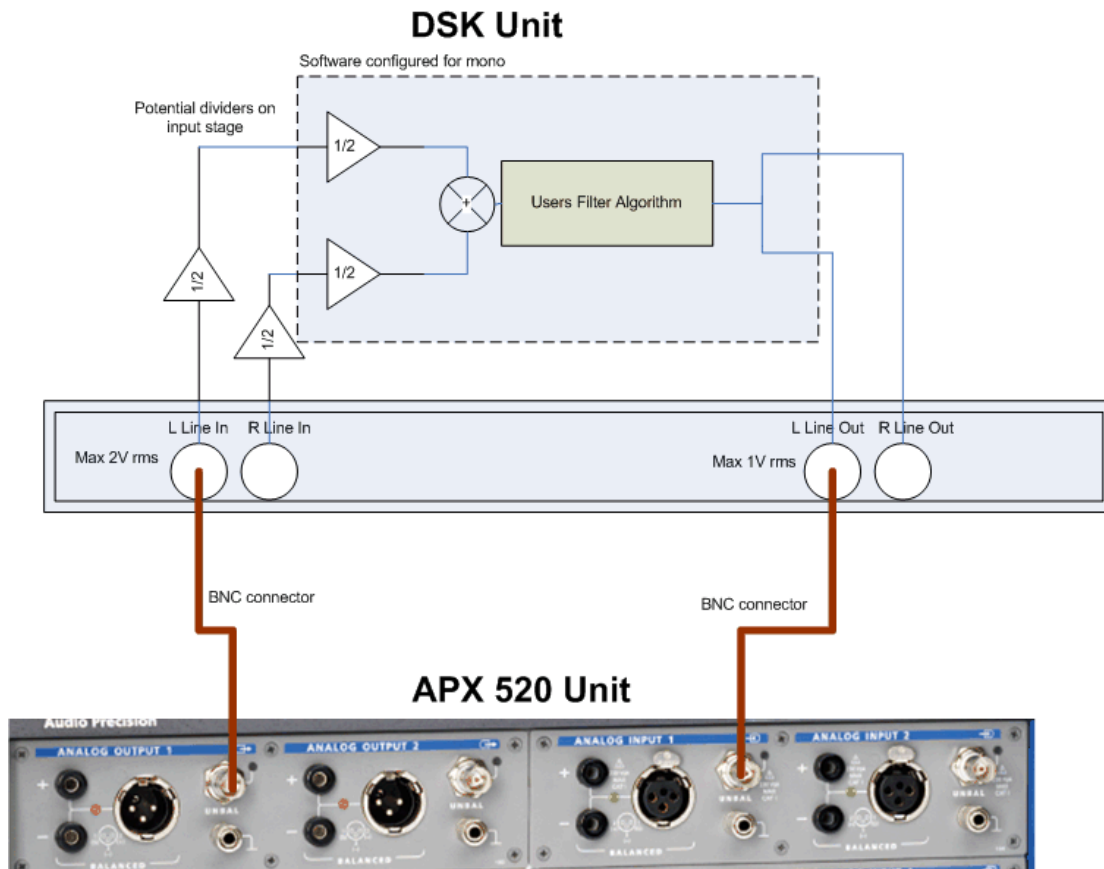
---

To permanently link the Markers, go to the Marker menu.	
Press [Marker]	Select the Marker menu.
Press <Marker>	Adjust the Marker Type.
Select (Link) with the knob and press [Enter].	Linked Markers move together. Since we changed the DisplayA Marker to Linked, moving the DisplayA Marker moves the DisplayB Marker.
Move the Marker with the knob.	If DisplayB is active, moving its Marker does not move the DisplayA Marker. To do this, change the DisplayB Marker Type to Linked also.
	This concludes this measurement example. You should have a feeling for the basic operation of two channel measurements and the use of [Active Display].

---

## Appendix C: Audio Precision APX500 Audio Analyser

Ensure the USB cable is connected between the computer and the APX unit.  
Connect BNC connectors to the DSP board as shown below. Ensure you understand the gain issues involved here and how this will affect your results.

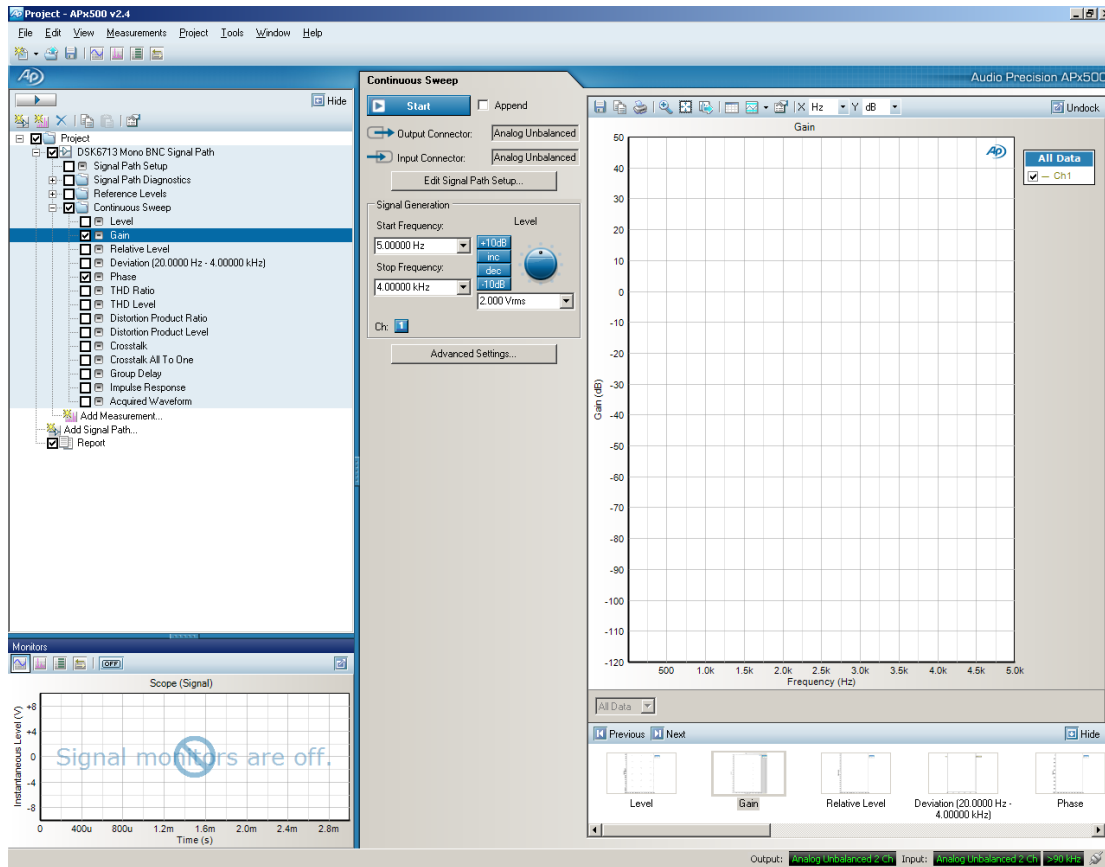




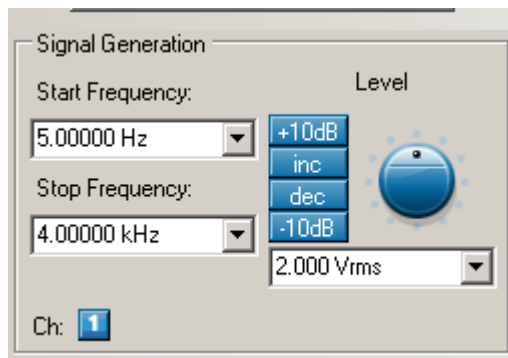
Ensure the switch on the front panel of the APX unit is on.

Start the software by clicking the apx500 icon on the desktop

You will be presented with the screen below.



The unit is configured to sweep frequencies between 5HZ and 4KHz (to suit the 8KHz sampling). The input voltage has been set to match the 2V input of the DSK6713 (do not exceed this!)



## Gain Plot

Press the start button (below the text “continuous sweep”) to obtain a plot



Explore the graphs in more detail using the buttons below:

zoom in button



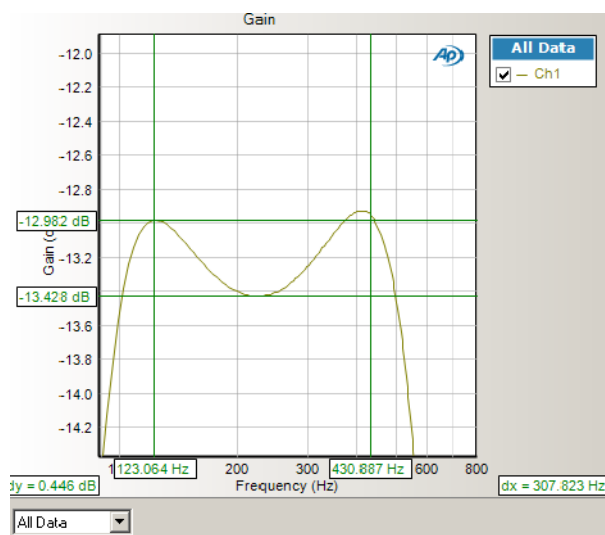
zoom out to original



Pan



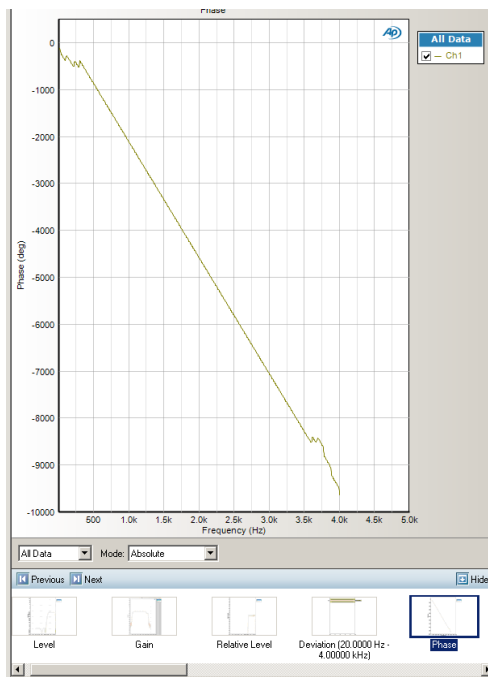
Cursors can be brought up by right clicking the graph and selecting cursors/x axis and cursors/y axis. Drag the cursors to where you wish to take a measurement (for example below).

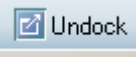


The combination of these buttons will allow you to make measurements more accurately.


## Phase plot

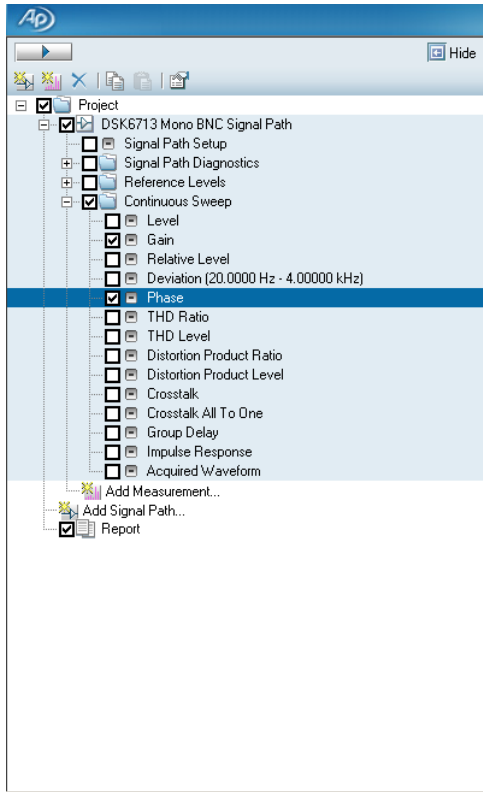
Select phase in the panel below the plot to show the phase plot. The graph can be zoomed/panned etc in a similar way to the gain plots.



Gain and phase plots can be undocked and viewed at the same time (side by side) using the undock button 

## Printing a report

Ensure gain and phase are ticked in the navigator as shown below then hit the start button in this section. 



A report will be shown on the screen which can be exported in a number of different formats or printed.