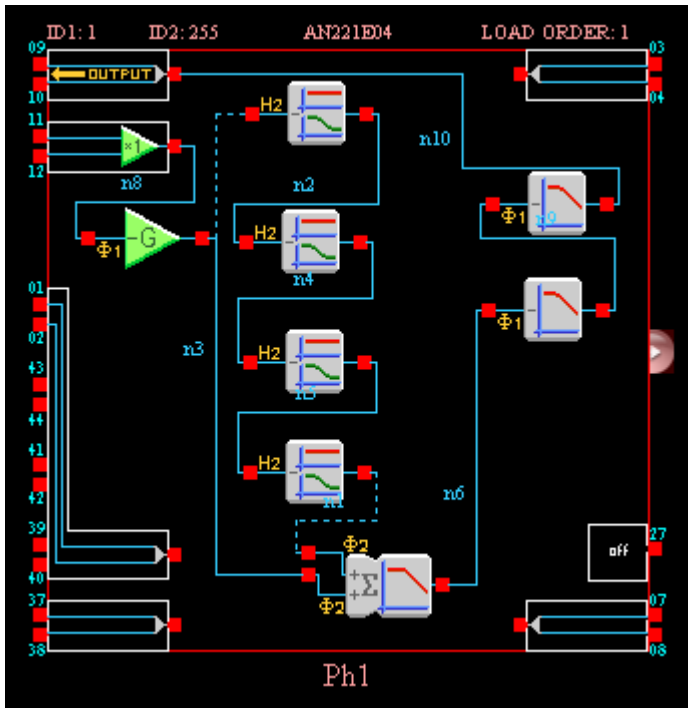
Audio Effects - Phase Shifter

Rev: 1.0.3

Date: 7th April 2004



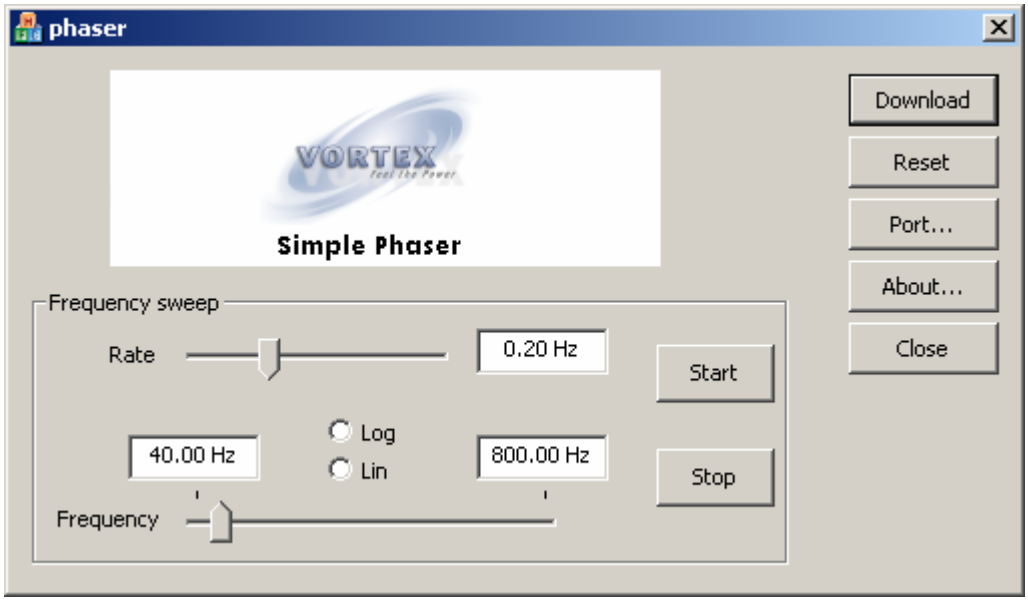


TABLE OF CONTENTS

1 Purpose

This document describes a reference design for implementing a simple audio phase shifter using Anadigm®'s AN221E04 FPAA device.

The circuit is accompanied by a software controller which performs the frequency sweep of the phaser. The implementation included in this kit uses algorithmic programming of the phaser elements, and runs on a Windows platform. Alternative methods can be used, targeting other processor platforms, which are less memory and processor intensive.

The 4-stage audio effect uses 2.5 CABs of an AN221E04 FPAA.

This document also discusses design variations which offer different design and software options to the user, and provides some bench characterizations of the design. These FPAA designs are included as AnadigmDesigner®2 design files as part of this Starter Kit.

2 Setup

2.1 Boards and interface

2.1.1 Inputs and outputs

The AN221K04 development board should be connected to the PC via the serial interface & cable provided. Depending on the type of signalling involved, the user may wish to make some custom modifications to the board. This section recommends various options.

It is recommended that all signalling into and out of the evaluation board be done differentially.

The reader is referred to the AN221K04 Evaluation Board User Manual, (Anadigm Document Number UM30900-U010) for full details.

The above user manual recommends various input and output interface circuits for ease of connection to audio sources and active speakers.

For formal performance tests all measurement instrumentation should connect directly to the FPAA header pins without any additional circuitry in the signal path.

To interface to the device, it is assumed that a 0V-to-2V level shift is required. The simplest way to do this for audio is to build a simple subsonic filter.

If a differential input signal is available, the designer should apply the input signal through a pair of identical capacitors connected in series with the signal to input terminal I2 as shown in Figure 1. These capacitors work with the unity-gain input stage to build a high-pass filter with a corner frequency (using 100nF capacitors) of around 10Hz.

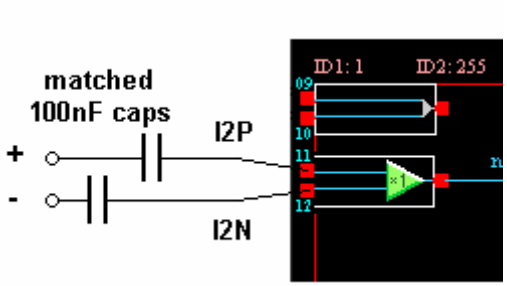


Figure 1

If the input signal is single-ended, then an alternative input arrangement is shown in Figure 2. If using this arrangement, the gain of the summing filter stage should be doubled (see section 3).

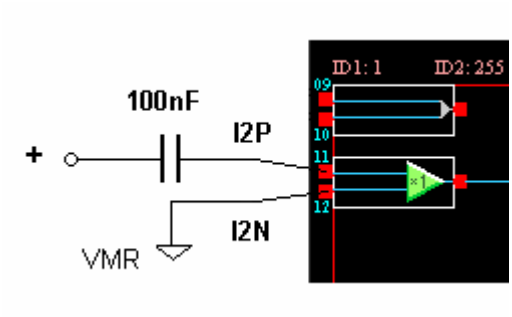


Figure 2

Figure 3 shows a differential-to-single-ended converter for a single-channel audio output, with 2V->0V common mode level shift.

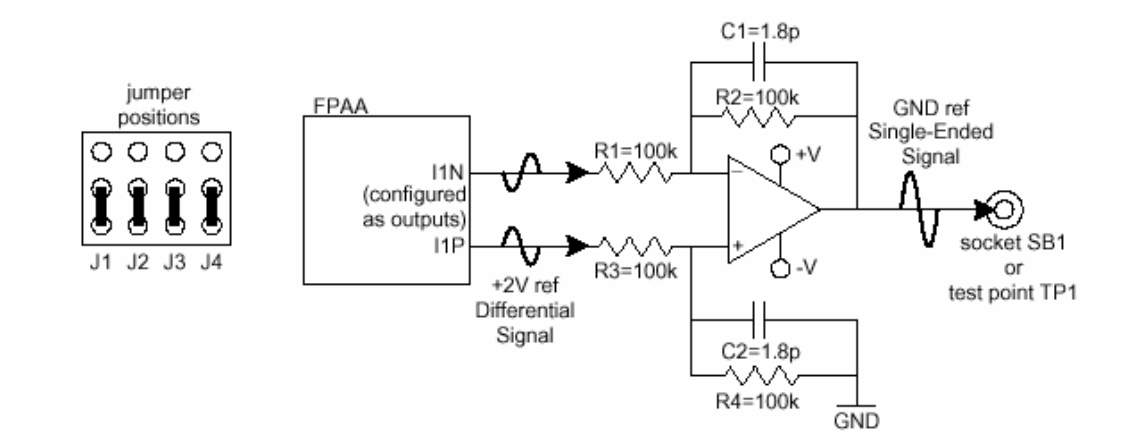


Figure 3

Alternatively, a single-ended output can be taken directly from the I1P pin, though the user is reminded of the 2V DC reference used for this signal. This can be adjusted using

- DC level shift or
- Series decoupling capacitor (e.g. 1µF)

☞ *N.B. If using the input arrangement described above, remember to **remove all jumpers J5-8** to isolate input I2x from the board components.*

☞ *For more complete information on the AN221K04 evaluation board, the user is referred to the user manual:*

AN221K04 Evaluation Board User Manual (click to download).
www.anadigm.com/support/literature library/Doc.No UM30900-U010

2.2 Software installation

The Starter Kit includes a simple Windows application called ***phaser.exe***.

This software works in concert with the FPAA circuit design, delivering real-time dynamic update data to the phaser to effect the frequency sweep. In this case, the software is targeted at a Windows® platform.

See Section 5 for more details.

Having installed the Starter Kit, no further installation is necessary. Simply double-click on this file or invoke it from the Windows Start menu.

The executable **phaser.exe** *is installed automatically as part of the Starter Kit. Its default location is*

<Effects Phaser install dir>\phaser Application

or it can be invoked from the Start menu:

Start->Anadigm->Starter Kits->Audio Effects – phaser starter kit>Phaser Application

This software will only operate with an AN221K04 evaluation kit, connected to the PC's *serial port* using standard serial connector.

Disclaimer:

Anadigm does not make any warranty or representation as to the functionality or otherwise of the "phaser" application and all warranties implied or express are excluded to the extent permitted by applicable law. Anadigm does not provide product support for this software.

3 Circuit description

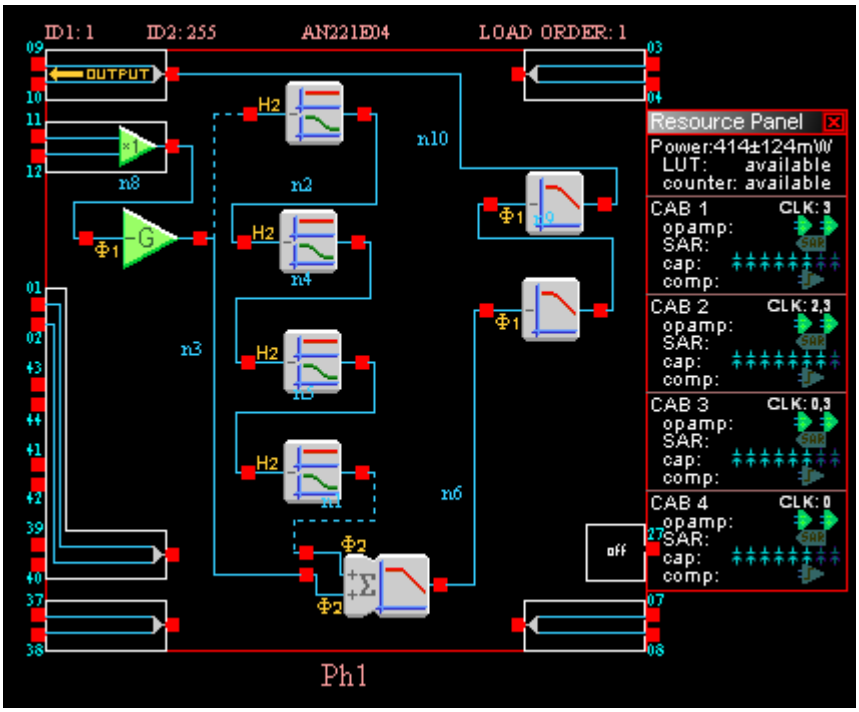


Figure 4 – Circuit of *phaser 4stage.ad2*

The circuit of Figure 4 contains the components of a classic phase shifter using four identical all-pass filter stages. This can be increased or reduced as desired. The output added to the input signal, generating constructive and destructive interference in the summed signal depending on phase differences between the signals.

A four-stage design delivers two notches in the logarithmic frequency response, symmetrically spaced each side of the pole frequency of the all-pass filters (see Section 6.1).

The objective of the design is to sweep the pole frequency of the all-pass filters thus sweeping these notches within the band of choice. This is done using dynamic reconfiguration of the all-pass filters (see Section 5).

The incoming signal has a gain of 15 applied to optimise for the dynamic range of the phaser circuit. The output has a Linkwitz-Riley 2nd order low-pass filter applied following the SumFilter, with corner frequency of 4kHz. The second stage attenuates the outgoing signal giving an overall gain of 3.0.

For **single-ended inputs**, the summing stage should have both input gains = **1.0**; for **differential inputs** the gains should be = **0.5**.

4 Circuit Variations

4.1 6-Stage Phaser

A variant of the Phaser circuit uses 6 stages instead of 4.

This design is included in the Starter Kit as ***phaser 6stage.ad2*** (Figure 5). See note below.

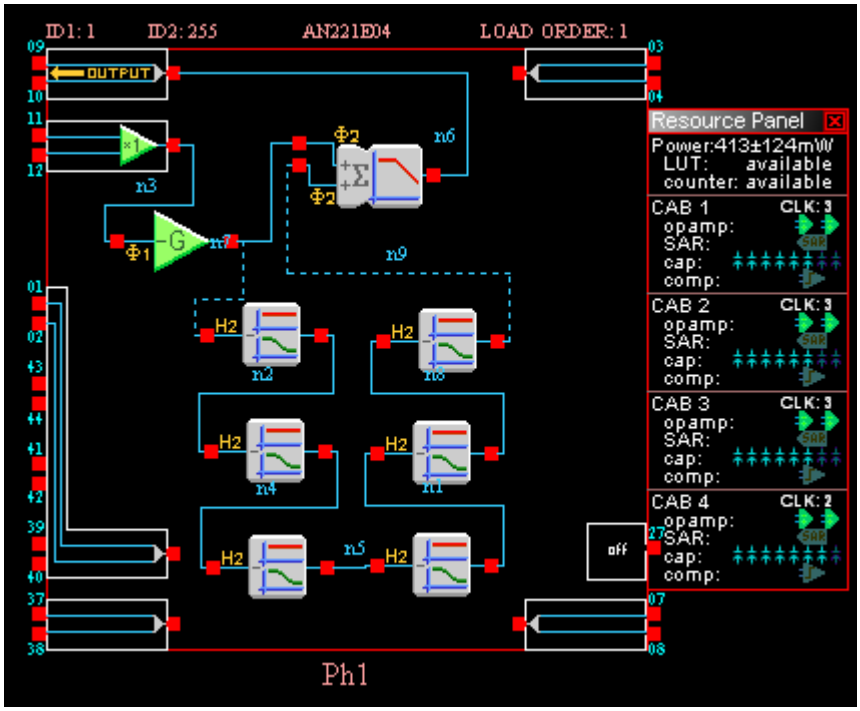


Figure 5 - Circuit of *phaser 6stage.ad2*

The 6-stage design introduces a third notch in the frequency response at the pole frequency of the filters (see Section 6.2). In this circuit, output signals are attenuated by the SumFilter.

4.2 Stereo 4-Stage Phaser

The Starter Kit also includes an alternative design for the Phaser circuit called ***stereo phaser 4stage.ad2*** (Figure 6).

This is basically the circuit described in Section 3, with the simple variation that for input frequencies near and above the pole frequency setting of the filters, there is up to 180 degrees of phase difference between the two output channels, giving a frequency-dependent stereo ‘spatial’ effect.

In this circuit, output signals are attenuated by the SumFilter.

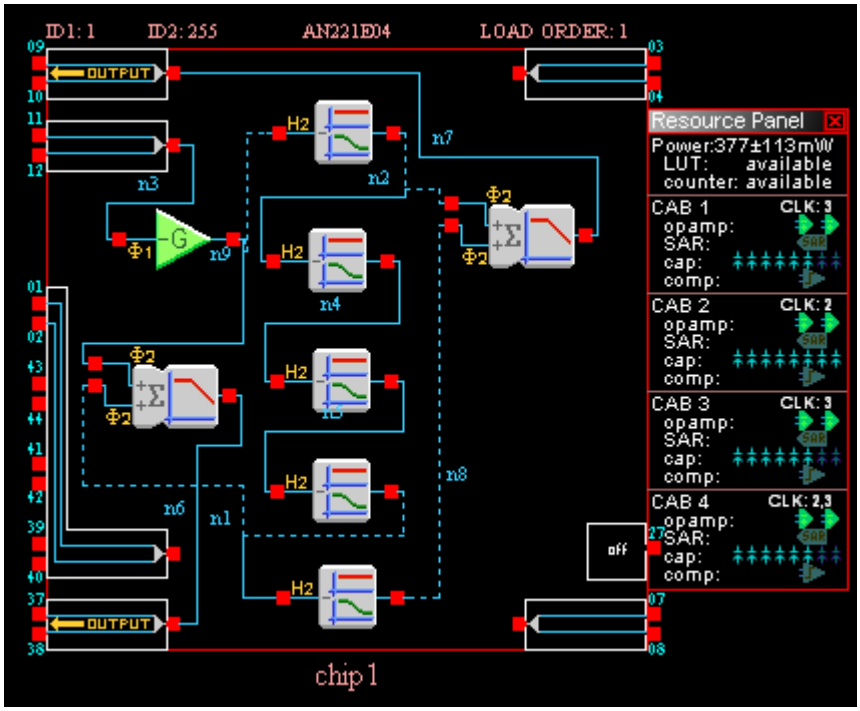


Figure 6 – Circuit of *stereo phaser 4stage.ad2*

4.3 Other Techniques

The above audio effects are based on the principle that a microprocessor of some description is available to perform the real-time dynamic update of the circuit. In an environment where the FPAA is being exploited to the fullest, this is likely to be true.

However, there may be a desire to build circuits which use internal analog voltage-based low frequency oscillators (LFOs) and voltage-controlled functions.

The **Customer CAM** library is a set of modules which are available in addition to the Standard release libraries. These contain elements such as the ***xVCFilt*** voltage-controlled bilinear filter which can substitute the all-pass ***FilterBilinear*** CAMs used in this Starter Kit. Anadigm is considering the implementation of an LFO CAM.

With these components a phase shifter that is entirely self-contained within the FPAA is feasible.

☞ *Details on the availability of **Customer CAMs** can be seen on the Customer Support pages of Anadigm's website*

www.anadigm.com

5 phaser.exe Software

5.1 Executable

This tool is the 'other half' of the phaser construction. It replaces the function of a Low Frequency Oscillator (LFO) that might have been used in traditional all-analog designs (see Section 4.3).

The tool contains the configuration and parameter controls for the circuit contained in ***phaser 4stage.ad2.***

Install and run the phaser.exe program as described in Section 2.2.

When phaser.exe is invoked, the dialog shown in Figure 7 appears.

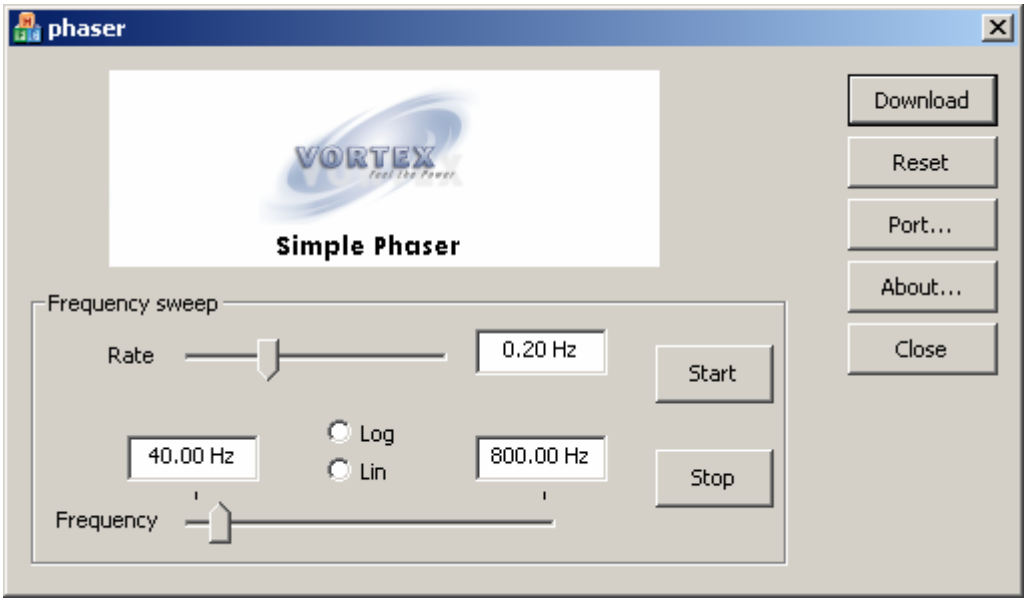


Figure 7

Press "***Port…***" to select the serial port to which the evaluation board is connected.

First, a configuration should be loaded to the device. This is done by pressing ***Download***.

The main control is the ***Frequency*** slider. The range settings for this control are user-selectable, and are entered in the Edit boxes provided. The slider can be manually adjusted or can be put into free-run mode by pressing ***Start***.

The rate of the sweep is set by the ***Rate*** slider or Edit box.

5.2 Source Code

This Starter Kit includes all the source code for Microsoft Visual C++® 6.0 to generate the ***phaser.exe*** executable. This code was initially generated

using the ***Visual C++ Prototyper*** tool within AnadigmDesigner®2 and then customized.

AnadigmDesigner®2 automatically generates **C-code application programming interfaces (APIs)** for all user-selected elements in a given design. These are based on a general-purpose ANSI-C standard for deployment to as wide a base of processor platforms as possible

The ***Visual C++ Prototyper*** is a tool for automatically generating a software project for Microsoft® Visual C++® , targeting a Windows platform. This is for rapid prototyping of software control. The ***Prototyper*** 'wraps' the API C-code in C++ class library functions and makes simple Windows-based controllers a breeze to create.

*See AnadigmDesigner®2 in-tool help information for more details on **C-code APIs** under the **Dynamic Configuration** feature.*

*Additionally, details on the use and features of AnadigmDesigner®2's **Visual C++ Prototyper** tool are fully described in the AnadigmDesigner®2 in-tool help information, and in a separate purpose-built Starter Kit. The latter is shortly to be released on the Customer Support pages on Anadigm's website.*

www.anadigm.com

A brief mention will be made here of the construction of the key elements in the ***phaser*** code as a guide to the reader.

The main class library in which all customizations were made is called ***phaserDlg***.

The three functions of particular interest in this class relate to the dynamic reconfiguration of the all-pass ***FilterBilinear*** CAMs and the sweeping of their pole frequencies.

```
//--------------------------------------------------------------------
//
// This function is called by MFC when the main dialog
// edit controls are changed
//
void phaserDlg::setFilters(double freq)
{
        // reconfigure the all-pass biquads
        m_Ph1.AP1.setFilterAllPass(freq);
        m_Ph1.AP2.setFilterAllPass(freq);
        m_Ph1.AP3.setFilterAllPass(freq);
        m_Ph1.AP4.setFilterAllPass(freq);
        m_Ph1.ExecuteReconfig();
}
```

Figure 8

Figure 8 shows the function ***setFilters()*** which reconfigures all the ***FilterBilinear*** CAMs' pole frequencies. The structure of the C-code wrappers is such that the respective CAMs' API C-code functions are accessed by simple naming conventions. In this example "***Ph1***" refers to the user-allocated name of the FPAA device in AnadigmDesigner®2. "***AP[n]***" refers to the user-allocated instance name of each respective all-pass ***FilterBilinear*** CAM.

The underlying C-code function is ***an_setFilterAllPass(an_CAM nCAM, double Fo)***.

The ***setFilters()*** function is invoked from another function, ***UpdateSampleControls()*** (Figure 9), which manages the synchronization of the user interface dialog and the settings of the physical filters.

```
//--------------------------------------------------------------------
//
// This function synchronises all dialog controls and issues an update
// to the filters in the phaser to the current setting.
//
// This is the only place where setfilters() is called.
//
void phaserDlg::UpdateSampleControls()
{
```

Figure 9

The ***UpdateSampleControls()*** function is, in turn, invoked from another function ***OnStart()*** (Figure 10) which contains all the timing and frequency co-ordination for the phaser.

```
//--------------------------------------------------------------------
//
// Starts the phaser sweep cycle & checks for messages as it goes
//
void phaserDlg::OnStart()
{
```

Figure 10

Using this method, a high degree of flexibility is easily added to the code. As may be evident, no optimization of the code has been performed for processor speed, memory utilization or elegance!

5.3 Software Efficiency

The software control techniques described in Section 5.2 make no attempt to optimize code. A practical situation may not have a Pentium-class processor available for rapid algorithmic calculations from filter parameters.

Clearly software efficiencies can and should be applied.

Figure 8 is a good example, where it can be seen that four successive calls are made to the same capacitor-setting algorithm, with the same parameter. This is clearly processor-inefficient, when the only practical difference between the four is the address of the capacitors being programmed.

Furthermore, this function is called repetitively for the same values as the frequency is swept up and down.

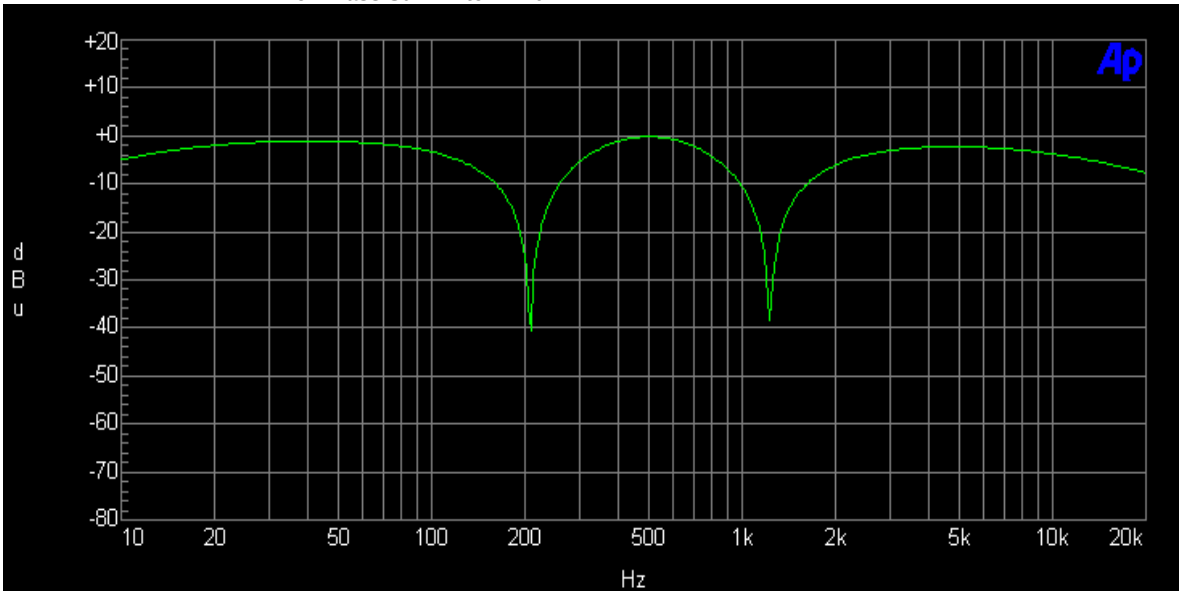
One example of a more sensible alternative would be to add a new function or modify the existing function ***an_setFilterAllPass()*** so that instead of directly programming the FPAA, it writes the capacitor values that it calculates into a vector or matrix in the code as a single-pass.

This way the algorithm itself need only be run once for each frequency range, and the ‘sweep’ algorithm simply plays out pre-calculated capacitor values. This is far more processor-efficient.

6 Performance

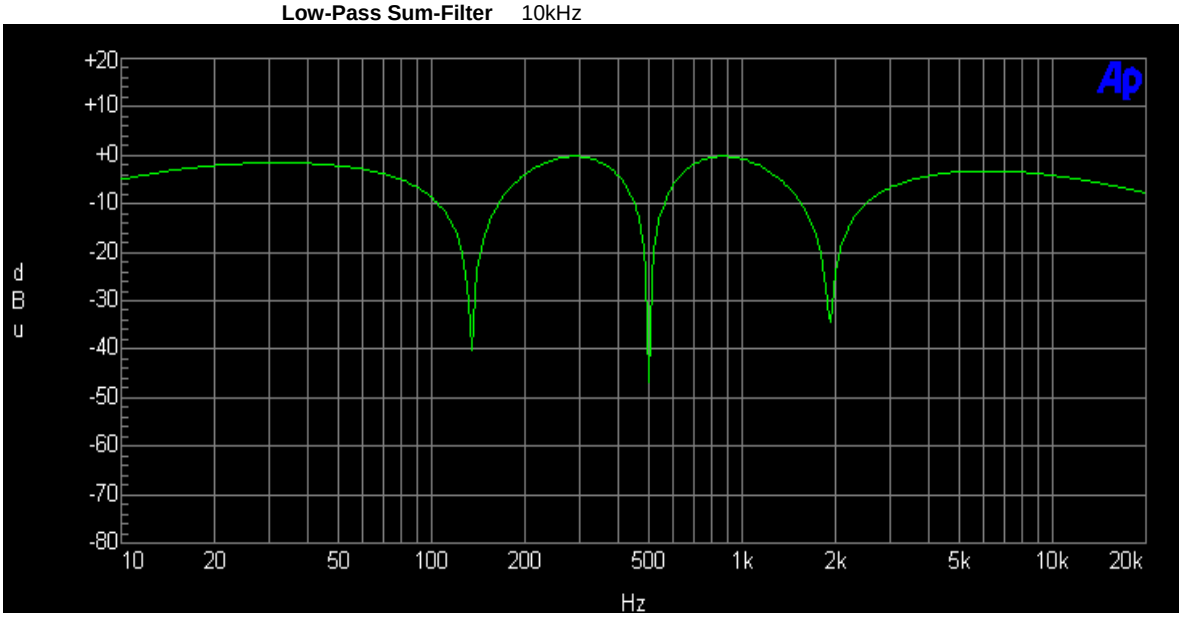
6.1 4-Stage Phaser response

Circuit: *phaser 4stage.ad2*
Input: 0dBu
All-Pass Corner Freq's: 500Hz
Low-Pass Sum-Filter 10kHz



6.2 6-Stage Phaser response

Circuit: *phaser 6stage.ad2*
Input: 0dBu
All-Pass Corner Freq's: 500Hz
Low-Pass Sum-Filter 10kHz



APPENDIX A – Alternative DC Input Interfacing Options

Figure 11 gives an example of a single-ended to differential converter for a single-channel audio input.

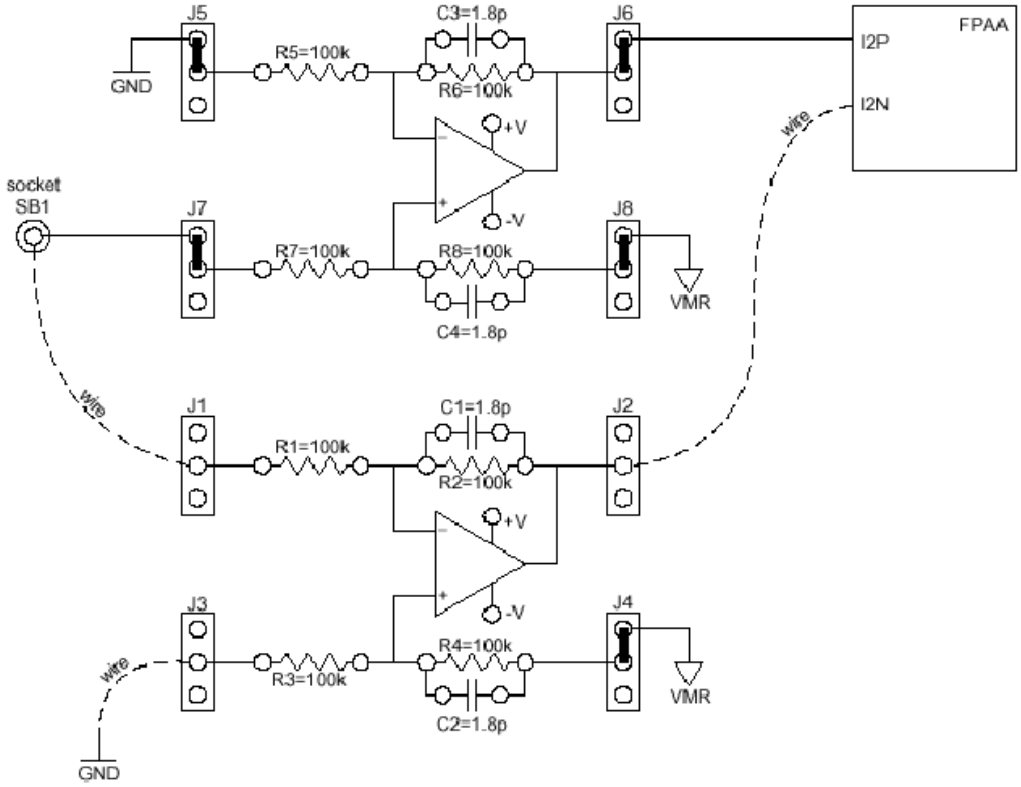


Figure 11

Depending on the nature of the audio signal source output drive (e.g. if it is AC-decoupled), it is possible that the common-mode of 2V will not be correctly established at the input using the circuit of Figure 11. An alternative is given in Figure 12.

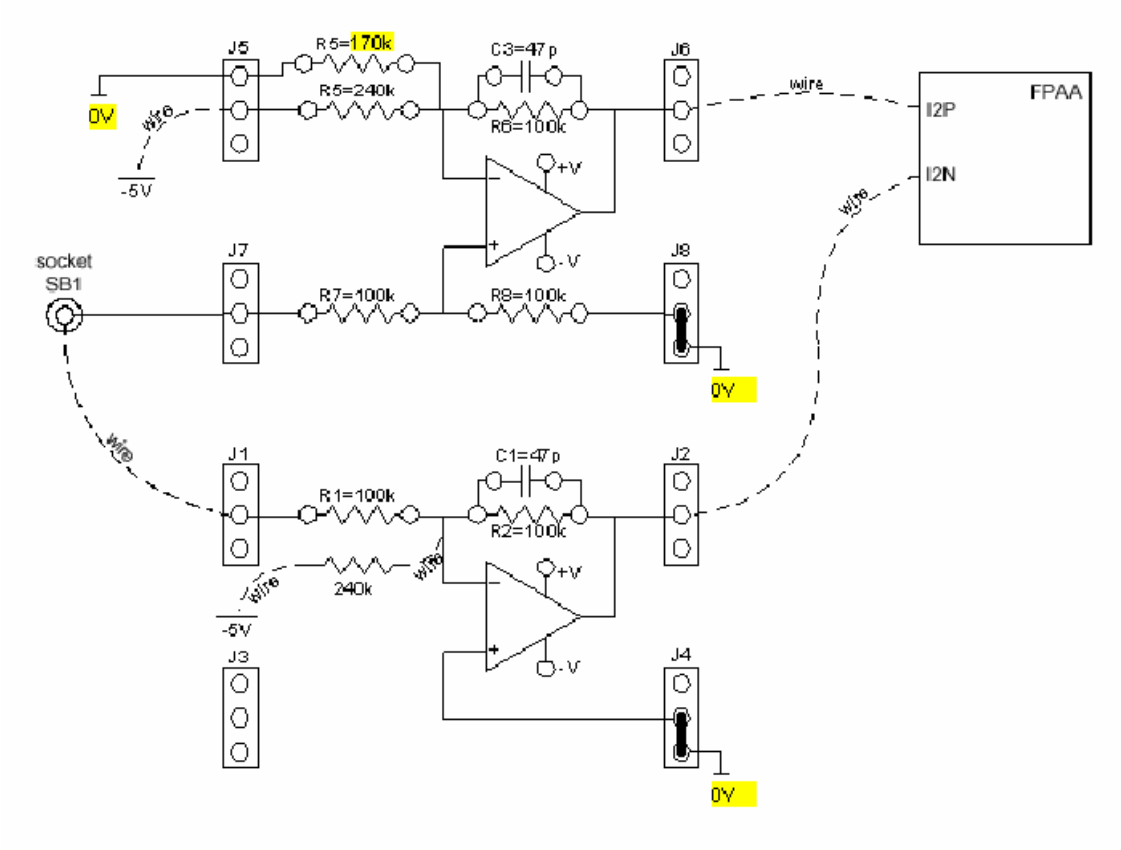

J5
R5=170k
R5=240k
C3=47p
J6
wire
FPAA
I2P
I2N
0V
wire
-5V
+V
-V
socket
SB1
J7
R7=100k
R8=100k
J8
0V
wire
J1
C1=47p
J2
R1=100k
R2=100k
wire
240k
wire
-5V
+V
-V
J3
J4
0V

Figure 12