Apex Filter Demo Application Note

Rev: 1.1.0

Date: 8th Nov 2011, Appended 25th Sept 2013

Frequency 1.00 kHz
Gain 1.00
Q 0.707
Balance 0.00

This Demonstration document contains a total 4 files, if you have only this pdf text document, go here
http://www.anadigm.com//sup_AppNoteLib.asp
to find the zipped Application executable, utility and AnadigmDesigner®2 files.

The author of this document assumes that the reader has:-

A copy of AnadigmDesigner®2 and some knowledge of using his tool.
Anadigm Development Kit AN231K04-DVLP3.

And if the reader wants to modify, duplicate or use this description as a starting point for new demonstration or application to control Anadigm's FPAA then also;_

A copy of Visual Studio or Visual C++ installed onto the same PC.
A basic knowledge of C++ software language.

TABLE OF CONTENTS

1 Purpose

This document describes how to create a PC driven Demonstration of Anadigm's FPAA technology. A PC application in the form of a GUI (Graphical user interface) is created and then used to control the parameters of a pair of filter CAMs implemented onto the AN231K04 development board.

The purpose of the demonstration is to illustrate the ease with which CAM parameters can be dynamically changed in real time.

This document describes how to create, step by step, a demonstration of a dynamically controlled stereo audio filter using Microsoft Visual C++. The demonstration allows the user to control the gain, centre frequency, Q-factor and balance of a stereo band-pass filter using a PC application (a GUI) with sliders and text boxes.

The application can be used to control the circuit within the AN231E04 FPAA device on an Anadigm AN231K04 development board, for ease of use the board has added audio sockets.

If a source of music is plugged into the board and speakers connected to the output, then the user can hear the effect of varying the parameters in real time.

This application can then be used as a stepping stone for the user to go on and create more complex dynamically controlled applications.

This document describes how to:

- set up the hardware
- create the AnadigmDesigner®2 circuit
- create a GUI application with Visual C++
- run the demonstration

2 Hardware Setup

2.1 Development Board Modifications

Figure 1 shows how to connect up the input stereo audio socket on the AN231K04 development board. The input signals will go through Rauch low pass filters. The development board has a set of footprints provided to make Rauch filters to inputs 3 and 4 of the AN231E04 device. Since the 2 audio channels are single-ended signals, 2 jumpers are added to connect the negative Rauch inputs to ground (figures 1 and 3).

Please refer to the AN231K04-DVLP3 Board User Guide for a detailed description and guide for using the Board. This document can found here Http://www.anadigm.com/_doc/UM231000-K001.pdf

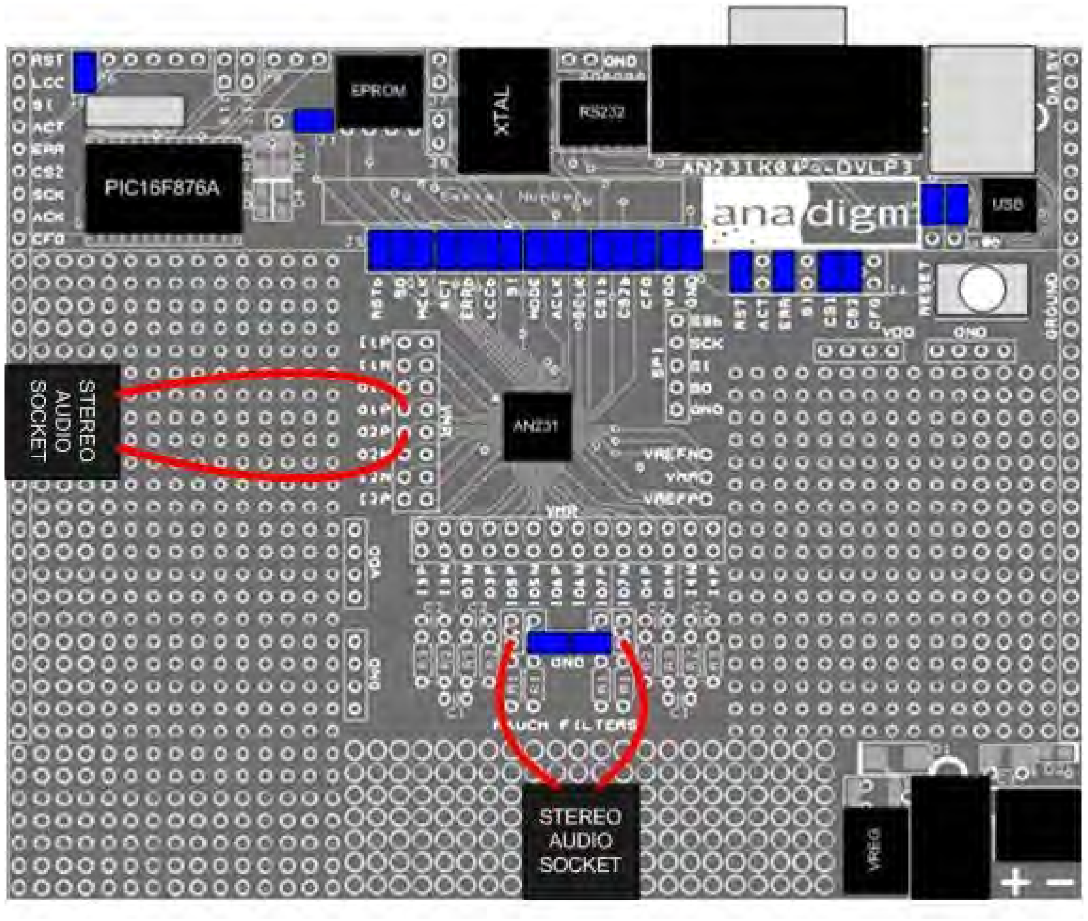


Figure 1: Stereo sockets added to AN231K04 development board

NOTE: the J15 jumper settings shown in figure 1 are for RS232 communication. This application will work with either RS232 or USB. To use USB communication the jumpers on J15 should be placed in the lower position.

NOTE2: figure 1 shows direct connections from the AN231E04 to the output socket. This is applicable to situations where the output device has a floating ground such as headphones. If the output device is ground referenced then O1P and O2P should be connected to the output socket through 0.1nF (or larger) capacitors.

The components to make up the Rauch filters are not provided on the board, so the user must calculate the component values and fit the components. The equations for calculating Rauch filter components are fairly complicated, so an Excel spread sheet is available on the Anadigm website to help the user calculate the values. It is suggested that the following component values be used:

R1 = 10kΩ

R2 = 10kΩ

R3 = 4.7kΩ

C1 = 1nF

C2 = 470pF

These component values will give the filter response shown in figure 2 below:

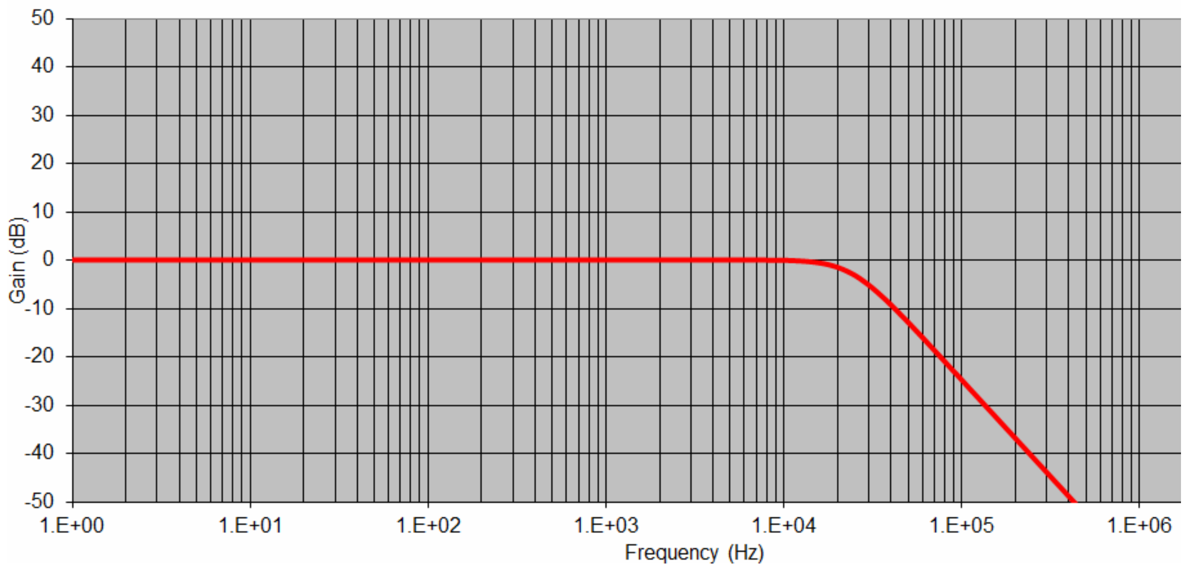


Figure 2: Rauch low pass filter response

Different parameters can be realised using different component values, the values above are a good solution for audio applications. The cut-off frequency is set to ~20kHz, just above the audio range and well below the 100kHz clock frequency of the filter CAMs.

The single-ended Rauch filter circuit is shown in figure 3. The components are labelled on the AN231K04 board and the component values suggested above should be added to the board by the user.

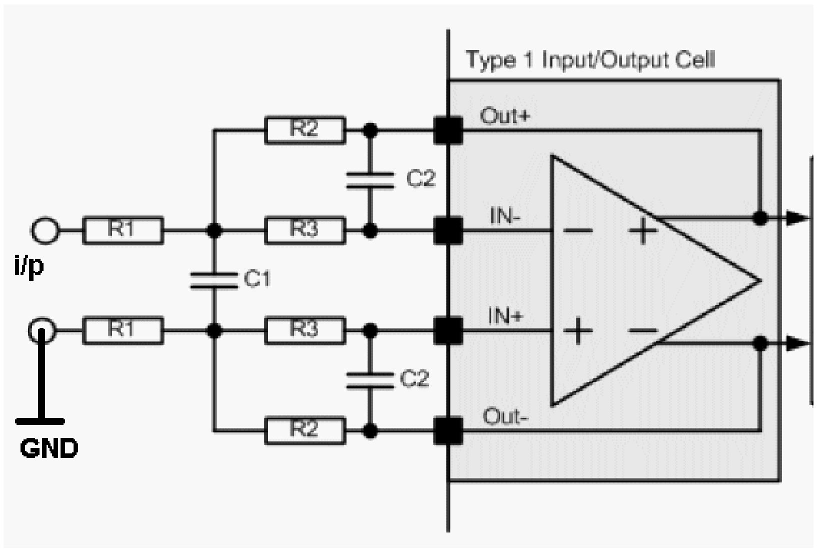


Figure 3: Rauch single-ended low pass filter

A second stereo audio socket should be added to the development board to provide the audio output (see figure 1). The 2 channels of this socket should be connected to O1P and O2P of the AN231E04. If using a receiver that is floating such as headphones, the connections can be made direct to the output socket. If using ground referenced active speakers then O1P and O2P should be connected to the audio socket via capacitors (0.1nF ceramic capacitors should be sufficient). This is to remove the +1.5V common mode offset of the output signals.

NOTE: the ground connections of both audio sockets should be connected to board ground. The audio input should be ground referenced.

Alternatively. it is also possible to connect a fully floating output (e.g.headphones) directly across the differential outputs of the FPAA, left channel audio from between O1N and O1P (no ground connection), similarly Right channel audio from O2N and O2P. This differential connection will provide 6dB higher power or volume. However it could be dangerous, if a ground referenced speaker is connected by accident the FPAA may be damaged.

2.2 The AnadigmDesigner®2 Circuit

Figure 4 shows the AnadigmDesigner®2 circuit for this demonstration. It is a very simple circuit that uses a single band pass biquad filter per channel. The purpose of this circuit is only to demonstrate control of CAMs, it is not recommended as an audio circuit.

NOTE: the circuit has the input OpAmps enabled for the inputs 3 and 4. This is important for making the Rauch filters.

NOTE2: the development board has an analog master clock of 16MHz. To achieve a CAM clock of 100kHz the FPAA internal clock divider has been set to 160.

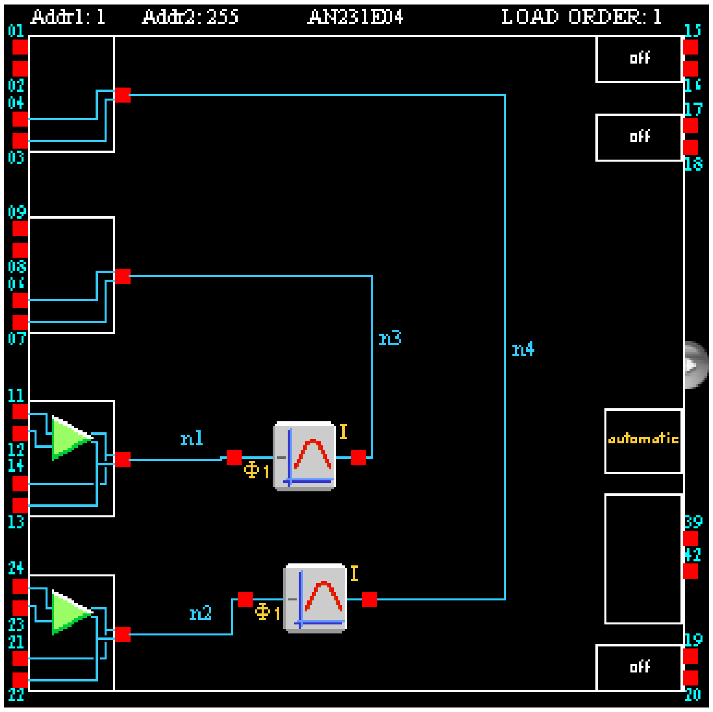


Figure 4: AnadigmDesigner®2 circuit

The default filter parameters are as follows:

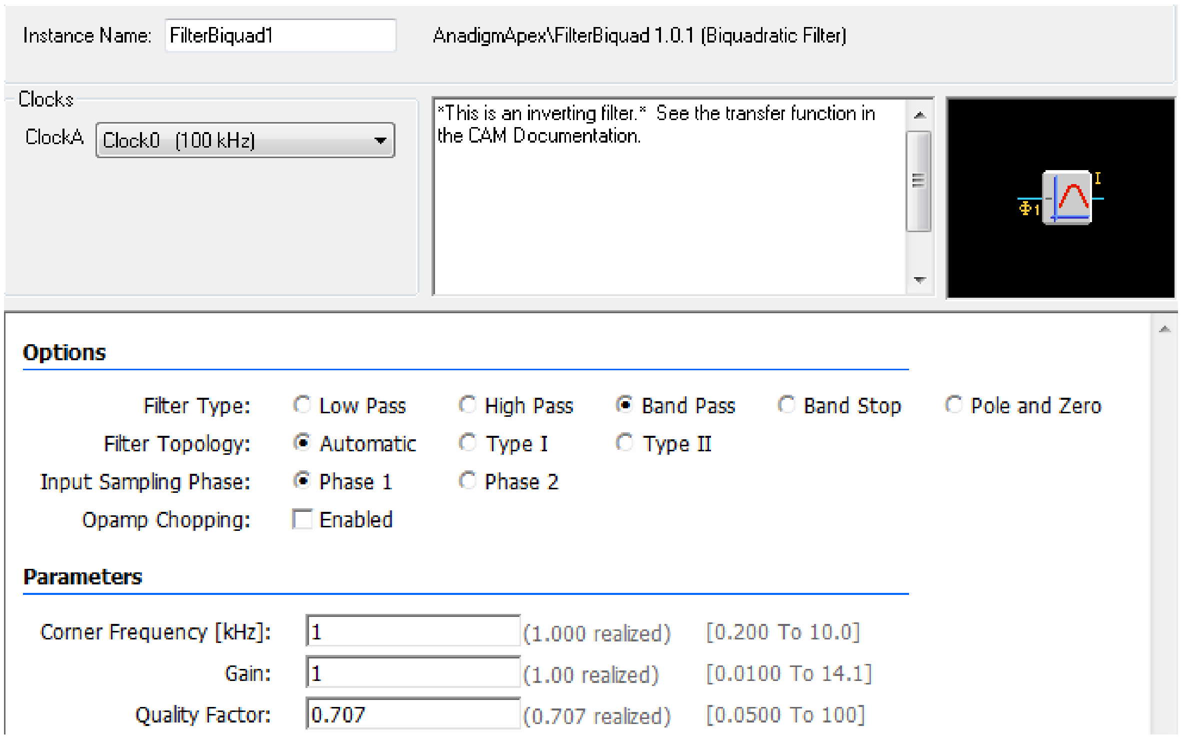
Fc = 100kHz (CAM clock, fixed for the demonstration)

Fo = 1kHz

G = 1

Q = 0.707

The parameter settings and clock settings are shown in figures 5 and 6:



Instance Name: FilterBiquad1

AnadigmApex\FilterBiquad 1.0.1 (Biquadratic Filter)

Clocks

ClockA: Clock0 (100 kHz)

This is an inverting filter. See the transfer function in the CAM Documentation.

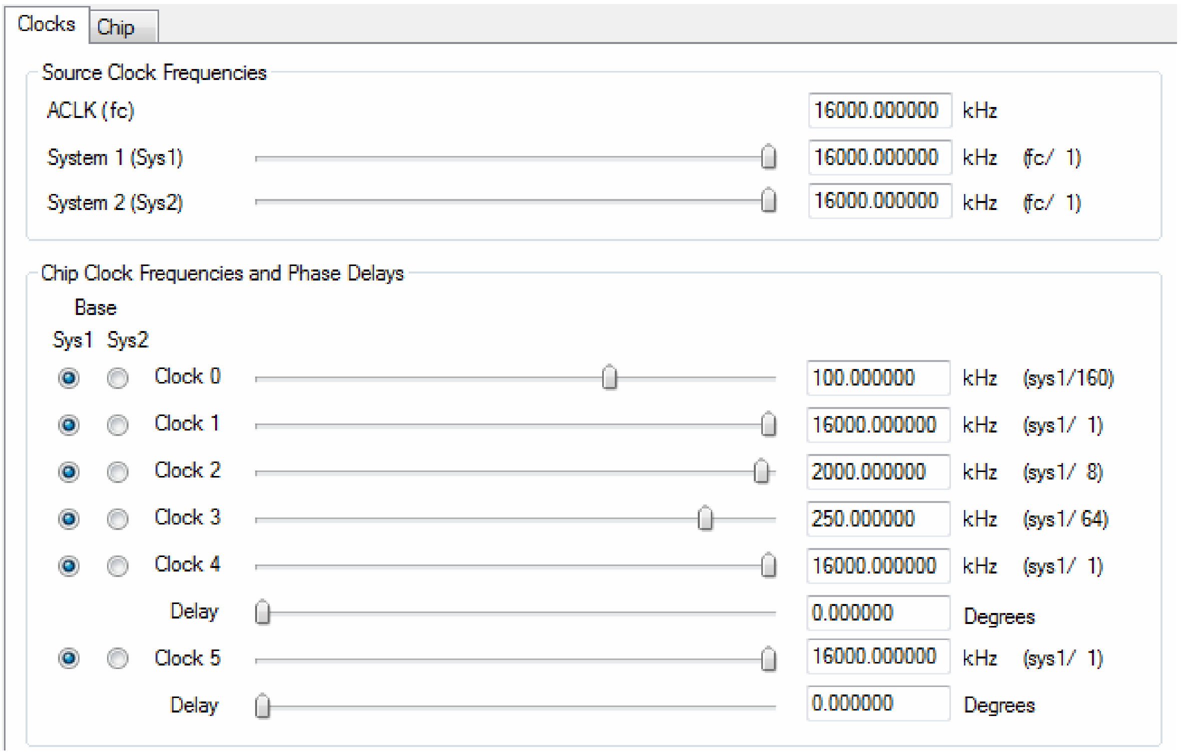
Options

Filter Type:	Low Pass	High Pass	[✓ Band Pass]	Band Stop	Pole and Zero
Filter Topology:	[✓ Automatic]	Type I	Type II		
Input Sampling Phase:	[✓ Phase 1]	Phase 2			
Opamp Chopping:	Enabled				

Parameters

Corner Frequency [kHz]:	1	(1.000 realized)	[0.200 To 10.0]
Gain:	1	(1.00 realized)	[0.0100 To 14.1]
Quality Factor:	0.707	(0.707 realized)	[0.0500 To 100]

Figure 5: CAM parameter settings



Clocks | Chip

Source Clock Frequencies

ACLK (fc)	16000.000000	kHz	
System 1 (Sys1)	16000.000000	kHz	(fc/ 1)
System 2 (Sys2)	16000.000000	kHz	(fc/ 1)

Chip Clock Frequencies and Phase Delays

Base Sys1	Base Sys2				
●	○	Clock 0	100.000000	kHz	(sys1/160)
●	○	Clock 1	16000.000000	kHz	(sys1/ 1)
●	○	Clock 2	2000.000000	kHz	(sys1/ 8)
●	○	Clock 3	250.000000	kHz	(sys1/ 64)
●	○	Clock 4	16000.000000	kHz	(sys1/ 1)
		Delay	0.000000	Degrees	
●	○	Clock 5	16000.000000	kHz	(sys1/ 1)
		Delay	0.000000	Degrees	

Figure 6: Clock settings

3 Creating The Software

3.1 Single FPAA Application

This section describes the steps to create the VC++ application. It is necessary to have Visual C++ v6.0 or a version of Visual Studio.

1. Create the circuit within AnadigmDesigner®2 (reference Fig 3).

2. Go to Dynamic Config menu and select Visual C++ Prototype.

3. Select VC++ version e.g. Visual C++ .NET 2005. Select directory and type in Project name, for the purpose of this document we will call it FilterDemo. Click on Generate. The project will be created and will open in Visual Studio.

4. In the solution explorer window in Visual Studio expand the FilterDemo directory and then expand the Source Files directory, double click on FilterDemoDlg.cpp and scroll down to the OnInitDialog function. Change the values of m_sampleMin, Max and Value to 0.2, 10.0 and 1.0 respectively. These are the slider min, max and default values for filter centre frequency.

5. Go to the UpdateSampleControls function and insert the code below at the bottom of the function. The functions come from the FPAA1.CAM.cpp files in the CCode dir.

```
m_FPAA1.FilterBiquad1.SetBQBandPassFilterI(m_sampleValue, 1.0, 0.707);
m_FPAA1.ExecuteReconfig(AUTO_EXEC);
m_FPAA1.FilterBiquad2.SetBQBandPassFilterI(m_sampleValue, 1.0, 0.707);
m_FPAA1.ExecuteReconfig(AUTO_EXEC);
```

This code makes the slider change the corner frequency of the 2 biquads. The two CAMs are named FilterBiquad1 and FilterBiquad2 in this case.

6. Also change the word Units to kHz in the following line of this function:

```
value.Format("%.2f kHz", m_sampleValue);
```

7. Change from Solution Explorer to Resource View in left window. If the resource view tab is not visible then go to the View menu at the top and select Resource View. There will be 3 directories - CCode, PortIO and "the project directory", in this case "FilterDemo". Expand the project directory, then the project.rc directory, and then the Dialog directory and then double click the project dialog called IDD_FILTERDEMO_DIALOG. This will show the GUI (Graphical User Interface).

8. Copy and paste the slider to make a new one (click on it and press CTRL C). Expand the group box (the faint box surrounding the slider and text label) and move the label down to make room for it. Right click the new slider and select Properties to change the name. Default name will be IDC_SAMPLE_SLIDER2. Leave as default for now.

9. Right click the slider and select Add Variable. Enter the variable name m_gainSlider and change the Access to private. Click Finish.

10. Go back to Solution Explorer and look at the dialog header file (FilterDemoDlg.h in Header Files directory). A new slider variable will be declared:

```
CSliderCtrl m_gainSlider;
```

11. Add variable declarations below for new slider:

```
double m_gainValue;
double m_gainMin;
double m_gainMax;
```

12. Look in dialog cpp file. Near top is DoDataExchange function. This will have the new slider in it:

```
DDX_Control(pDX, IDC_SAMPLE_SLIDER2, m_gainSLider);
```

13. Go to OnHScroll function near bottom and add new case statement below with new slider name and new slider limits. This should go below the close bracket of the previous case statement.

```
case IDC_SAMPLE_SLIDER2:
{
    // Get the current position of the slider
    int percent = ((CSliderCtrl*) pScrollBar)->GetPos();

    // Map the percent onto the actual sample value
    m_gainValue = m_gainMin + (percent / 100.0) * (m_gainMax - m_gainMin);

    // Synchronize the slider and the edit control
    UpdateGainControls();

   break;
}
```

14. Create a new function similar to UpdateSampleControls and call it UpdateGainControls and place it below the UpdateSampleControls function. Here is the code:

```
void FilterDemoDlg::UpdateGainControls()

{
    // Map the actual sample value to the range [0, 100] for the slider
    int percent = (int) (100.0 * (m_gainValue - m_gainMin) / (m_gainMax - m_gainMin));

    // Update the slider position
    m_gainSlider.SetPos(percent);
    m_gainSlider.Invalidate();

    m_FPAA1.FilterBiquad1.SetBQBandPassFilterI(m_sampleValue, m_gainValue, 0.707);
    m_FPAA1.ExecuteReconfig(AUTO_EXEC);
    m_FPAA1.FilterBiquad2.SetBQBandPassFilterI(m_sampleValue, m_gainValue, 0.707);
    m_FPAA1.ExecuteReconfig(AUTO_EXEC);
}
```

15. This function needs to be declared in the dialog header file, just below the declaration for the UpdateSamplesControls. The code is below:

```
void UpdateGainControls();
```

16. Go to the UpdateSampleControls function and change the last 4 lines to:

```
m_FPAA1.FilterBiquad1.SetBQBandPassFilterI(m_sampleValue, m_gainValue, 0.707);
m_FPAA1.ExecuteReconfig(AUTO_EXEC);
m_FPAA1.FilterBiquad2.SetBQBandPassFilterI(m_sampleValue, m_gainValue, 0.707);
m_FPAA1.ExecuteReconfig(AUTO_EXEC);
```

This ensures we keep the current gain value when we change the frequency.

17. In the OnInitDialog function near the top, add the following lines:

```
m_gainMin = 0.01;
m_gainMax = 10.0;
m_gainValue = 1.0;
```

and...

```
m_gainSlider.SetRange(0,100);
```

and...

```
UpdateGainControls();
```

For neatness add these lines below the equivalent lines for the first slider.

18. Build it and run it.

19. Go to the GUI again in the Resource View and copy and paste the edit box. Its name will be set to IDC_SAMPLE_EDIT2. Leave this the same. Add a variable in the same way we did with the slider i.e. right click on edit box and select Add Variable. Enter m_gainEdit into the Variable name box and change the access to private. Click on Finish.

20. Go to the dialog header file in the Solution explorer window and at the bottom the new edit box variable should have been added automatically:

```
CEdit m_gainEdit;
```

21. Go to the dialog cpp file in the Solution explorer window. In the DoDataExchange function we will see this code already added:

```
DDX_Control(pDX, IDC_SAMPLE_EDIT2, m_gainEdit);
```

22. Add this function at the bottom of the dialog cpp file:

```
void FilterDemoDlg::OnGainEditChanged()
{
   // Get the value in the edit box
   CString text;
   m_gainEdit.GetWindowText(text);

   // Convert to a double
   m_gainValue = atof(text);

   // Ensure the value is within the allowable range
   m_gainValue = max(min(m_gainValue, m_gainMax), m_gainMin);

   // Synchronize the slider and the edit control
   UpdateGainControls();
}
```

23. Go to PreTranslateMessage function in the same file and add the "else if" code as shown below in red:

```
BOOL FilterDemoDlg::PreTranslateMessage(MSG* pMsg)
{
    bool eatMessage = true;

    if (pMsg->message == WM_KEYDOWN && pMsg->wParam == VK_RETURN)
    {
        if (GetFocus() == &m_sampleEdit)
        {
            OnSampleEditChanged();
        }
        else if (GetFocus() == &m_gainEdit)
        {
            OnGainEditChanged();
        }
        else
        {
            eatMessage = false;
        }
     }
etc….
```

24. In the dialog header file, add the line below after the equivalent line for the frequency edit box (last line under "protected:")

```
afx_msg void OnGainEditChanged();
```

25. In the dialog cpp file, go to the UpdateGainControls function and add the code below after the gainSlider.Invalidate line:

```
// Format the text for the edit control
CString value;
value.Format("%.2f", m_gainValue);

// Update the edit control text
m_gainEdit.SetWindowText(value);
```

26. Build it and run it.

27. Now follow the same steps that were used to add the gain controls for adding Q controls.

28. Now do the same again for balance. This is slightly different in the way that the CAMs are configured. The balance slider is varied from –0.99 to +0.99 with default value of 0, and then the gain of one CAM is set to:

m_gainValue * (1 + m_balanceValue)

and the gain of the other CAM is set to:

m_gainValue * (1 - m_balanceValue)

NOTE: if the balance slider has extreme values of +/-1 then the code will try to set the gain of one CAM to zero when the slider is moved to its extreme position. This value is not allowed and an audible click is heard when this happens.

NOTE2: the equations above which modify the gains of each CAM must be used in all of the UpdateParameterControls functions. This will ensure that the balance is preserved at its most recent setting when other parameters are varied.

29. Labels are simple to add. Just go to the GUI in the resource view, copy and paste the existing label and type in new contents.

30. Finally it is suggested to change the Visual Studio project from Debug to Release configuration. This will create a much smaller .exe application. To do this, go to the Build menu and select Configuration Manager. Under “Active solution configuration:” in the pull down menu, select Release. Then build the project again. The .exe application will now be contained in a sub directory called Release.

NOTE: the full source code for this application can be requested by emailing Anadigm support.

3.2 Creating a 2 FPAA Application

Figure 7 shows how to chain 2 development boards together.

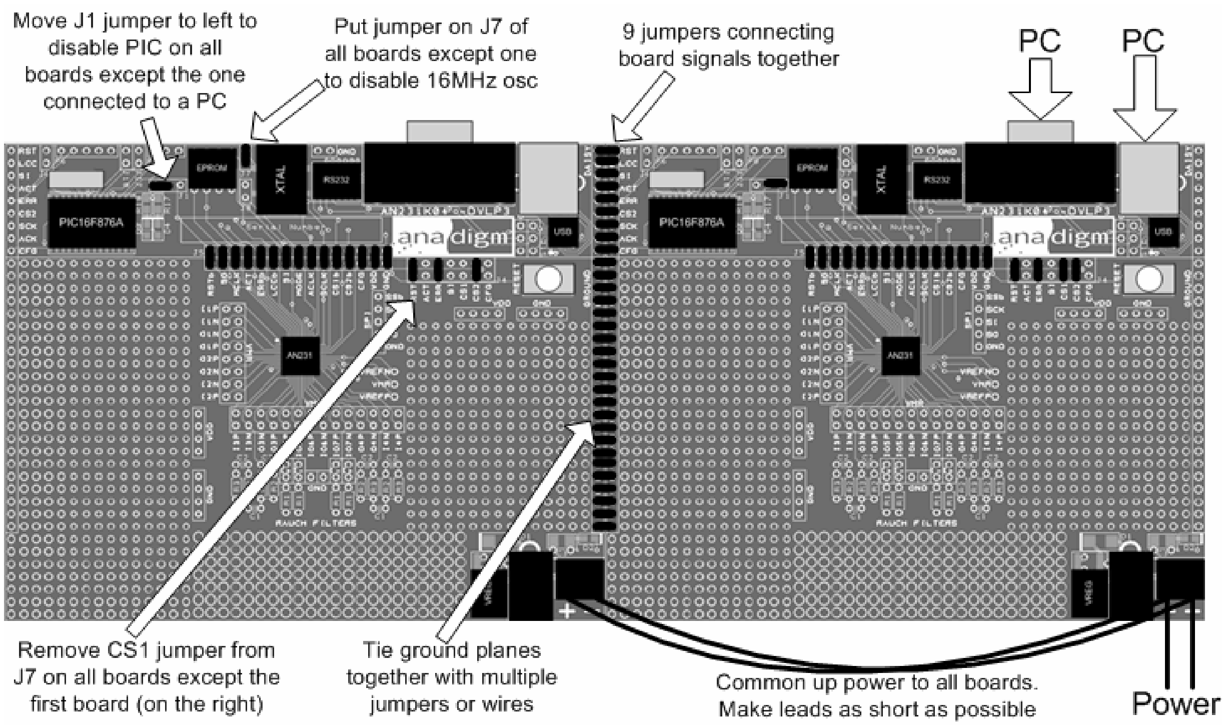

Move J1 jumper to left to disable PIC on all boards except the one connected to a PC
Put jumper on J7 of all boards except one to disable 16MHz osc
9 jumpers connecting board signals together
PC
PC
Remove CS1 jumper from J7 on all boards except the first board (on the right)
Tie ground planes together with multiple jumpers or wires
Common up power to all boards. Make leads as short as possible
Power

Figure 7: Chaining 2 Development Boards

In order to make a 2 FPAA prototype application, follow these steps:

1. Create 2 FPAA circuit in AD2. AD2 will automatically give the 2 devices the IDs FPAA1 and FPAA2. This is shown in figure 8.

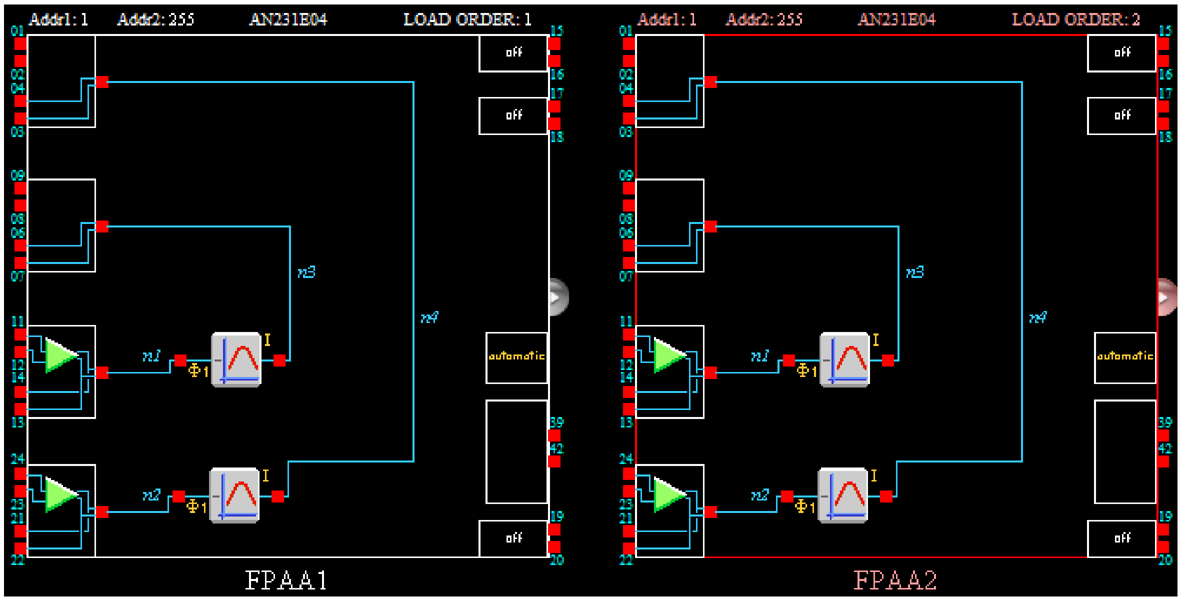


Figure 8: 2 FPAA Circuit

2. Go to Dynamic Config menu and select Visual C++ Prototype.

3. Select VC++ version e.g. Visual C++ .NET 2005. Select directory and type in Project name. Click on Generate.

4. In the ProjectDlg.cpp file, go to UpdateSampleControls function. In this function it is possible to add functions to control CAM parameters in both FPAAs. The functions come from the FPAA1_CAM.cpp files and the FPAA2_CAM.cpp files in the CCode directory (e.g. FPAA2_FilterBiquad1.cpp). Below is some example code:

```
m_FPAA1.FilterBiquad1.SetBQBandPassFilterI(Fo_1, Ampl_1);
m_FPAA1.ExecuteReconfig(AUTO_EXEC);
m_FPAA2.FilterBiquad1.SetBQBandPassFilterI(Fo_2, Ampl_2);
m_FPAA2.ExecuteReconfig(AUTO_EXEC);
```

This code is controlling the Biquad Filter CAM in FPAA1 and the Biquad Filter CAM in FPAA2. Note that the variables Fo_1, Ampl_1, Fo_2, Ampl_2 will have been declared in the header file ProjectDlg.h and initialised in the OnInitDialog() function in ProjectDlg.cpp.

5. In the OnDownload() function there should be code to do a primary configuration of both FPAAs. This code may look like this:

```
m_FPAA1.ExecutePrimaryConfig();
m_FPAA2.ExecutePrimaryConfig();
```

In this way it is very easy to create a VC++ prototype as described in section 3 of this document, but controlling 2 or more FPAAs.

4 Running the Demonstration

First connect either an RS232 or USB serial cable from the serial port of a PC to the AN231E04 development board.

Start AnadigmDesigner®2 , load the circuit filterdemo.ad2 and download this to the board.

A music source should be plugged into the input socket you mounted on the development board. The output socket you added should be connected to headphones or speakers. If the music source is active you should hear the music you chose to play running through the AN231E04 device and the filter circuit loaded

Try adjusting the Gain's, Center Frequencies and Q-factor of the two Biquad filter using circuit's parameters and AnadigmDesigner®2 , repeatedly reconfiguring the FPAA on the board by downloading. The objective here is to get an appreciation for the flexibility of the technology and the ease with which a pair of 2nd order filters parameters can be modified.

Having modified the circuit a few times (maybe 10 different settings), the user may start imaging a better more automated way to make such changes …….. back to the GUI we just built.

NOTE: This demo is not about the quality of the filter as an audio filter, the music and speaker (headphone) is only included so that the demo observer gets instant feedback associated with the circuit changes being made (feedback by hearing the effect of the filter).

Double click on the file you built using the descriptions above, file called FilterDemo.exe (you will find this in the Release or Debug sub directory in the FilterDemo directory), you may keep the music playing and the speaker attached. The GUI is shown in figure 7.

NOTE: If you are using the FilterDemo.exe file directly from a download of this documents zip file collection, you may see an error, WinIO file not found, if this is case you will need to add the three WINIO files to the same directory as the FilterDemo.exe file you are using.

First check the Port selection by clicking the Port button, this port selected should be the same as the port selection you used inside AnadigmDesigner®2 .

NOTE: This GUI and FPAA board hardware will function correctly if you leave AnadigmDesigner®2 software open. However AnadigmDesigner®2 will not be able to download to the FPAA board whilst the GUI is open, the GUI has constant access to the serial port and blocks other applications.

Next, click the Reset button to clear the circuit previously loaded from AnadigmDesigner®2 into your board's FPAA device, music will not be heard.

Then Click on "Download" and the music should again be heard in stereo through the headphones/speakers.

Now, use the sliders or edit boxes to adjust the filter settings.

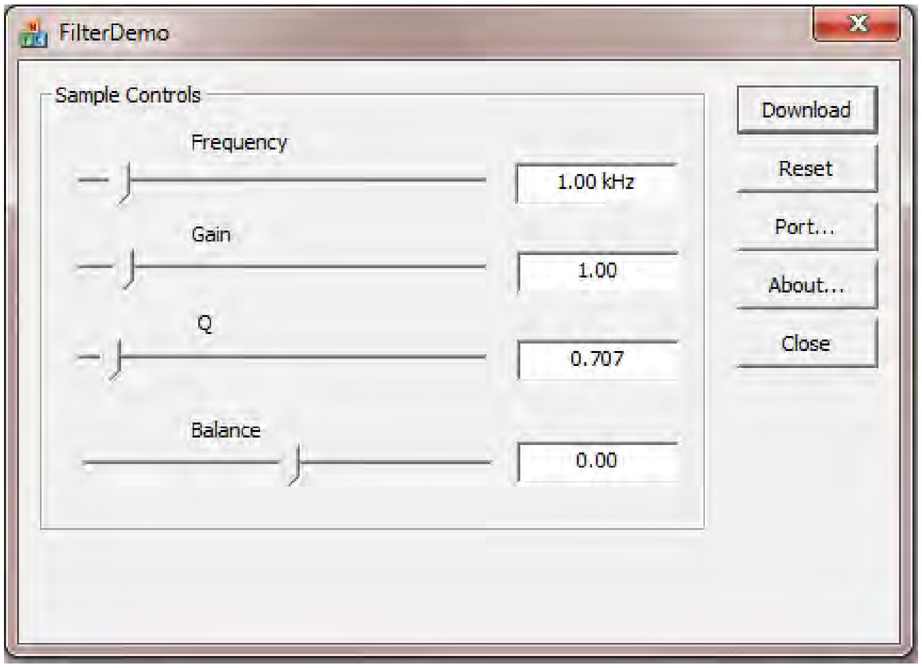
Anadigm suggests, try making the filter Q-factor higher, (eg Q=5) then holding the frequency slider, sweep slowly up and down the frequency spectrum and listen to the result. You may need to increase the volume a little to hear this clearly, increase Gain (e.g. G=8)

Then perhaps choose a mid-audio range center frequency eg. 1kHz and try sweeping the Q-factor, you should be able to hear all the music with a low Q-factor and increasingly less of it as the Q-factor increases and removes all but the components close to 1kHz.

Try the same thing with a high and low center frequency (e.g.200Hz, 4kHz).

Remember as you move the slider that the GUI you built is calculating new circuit parameters, building an FPAA configuration file and sending it to the board where the FPAA is re-configured, this happens so fast you don't hear it happening nor is there any significant delay.

NOTE: USB port data streams are buffered and can be delayed by a PC operating system.



Frequency 1.00 kHz
Gain 1.00
Q 0.707
Balance 0.00

Figure 7: The GUI Application

NOTE: the limits of some CAM parameters are inter-related. This is true of the Biquad filter CAM, but the above application does not take this into account. See section 5.1 for a more detailed description of this problem.

5 Limitations and Improvements

5.1 CAM Parameter Limits

The limits of the parameters of some CAMs are inter-related. To understand this we suggest playing with the Biquad parameters in AnadigmDesigner®2 and the user will see that, for example, a low Q value reduces the maximum setting for corner frequency and gain. The CAM parameters dialog window inside AnadigmDesigner®2 has been programmed to understand this interaction and to limit the users CAM parameter range appropriately to ensure that only fully functional circuit parameters can be realized (look at the realized values and the range). The C-code used with the GUI does not contain any information about the interaction between parameters, nor does it have any limits other than those you entered when creating it, thus poor performing circuits can be realized. They can also be eliminated by changing parameter ranges and limits.

This effect can be heard in the above application, try reducing the Q to a low value and then increase the gain or move the balance to an extreme. The music will become distorted because the Biquad CAM cannot achieve the desired parameters.

An alternative and simple solution to this is to add a gain CAM into each channel of the AnadigmDesigner®2 circuit to provide an independent element for the control of volume and balance. Use the Biquad filters with a fixed gain of 1 and adjust the center frequency and Q-factor with the filters and the gain independently with the Gain CAM. The circuit is shown in figure 8.

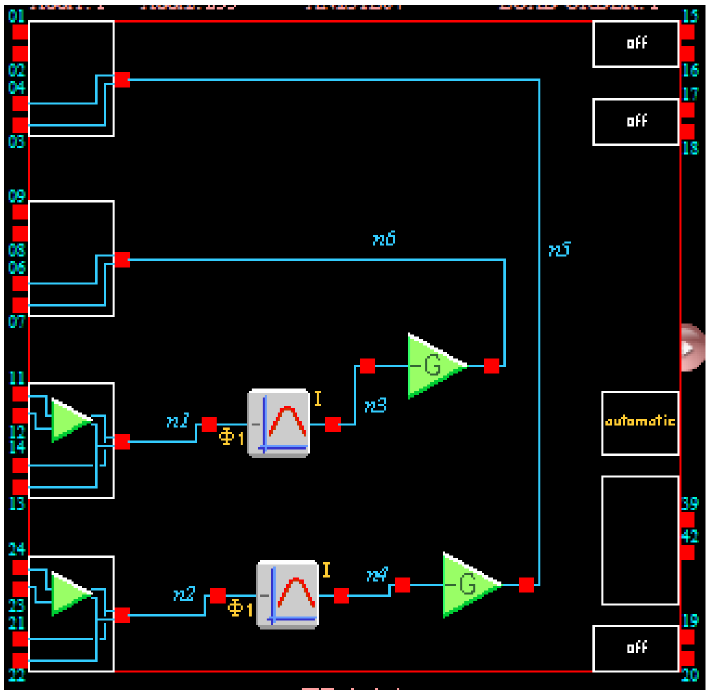


Figure 8: AnadigmDesigner®2 circuit with added gain control

5.2 Logarithmic Frequency Slider

The user will have noticed that the slider that controls frequency is linear. This is not ideal for controlling frequency in an audio application. It is far better to have a logarithmic slider control. We can achieve this by changing 2 lines in the code.

In the file FilterDemoDlg.cpp go to the function UpdateSampleControls(). Change the following line:

```
int percent = (int) (100.0 * (m_sampleValue - m_sampleMin) / (m_sampleMax - m_sampleMin));
```

to the following:

```
int percent = (int) (100.0 * log10(100.0 * m_sampleValue / m_sampleMax) / log10(100.0));
```

or more simply:

```
int percent = (int) (50.0 * log10(100.0 * m_sampleValue / m_sampleMax));
```

Then go to the function OnHScroll() in the same file. Here we find the same equation but rearranged to be in terms of m_sampleValue:

```
m_sampleValue = m_sampleMin + (percent / 100.0) * (m_sampleMax - m_sampleMin);
```

This line should be changed to:

```
m_sampleValue =  m_sampleMax * pow(10.0, (percent / 50.0)) / 100.0;
```

Now rebuild the project and try the new application.