IMPORTANT NOTE

Maintanence and updates to the document were frozen in year 2004 (revision 2.2.7 of AnadigmDesigner2).

This document is superceeded and replaced by the extensive "help" and "Documentation" included with and inside the software tool intself (AnadigmDesigner2).

Some users may find this document useful as a single printable file,
BEWARE !
IT IS NOW OBSOLETE, MANY IMPROVEMENTS AND ENHANCEMENTS TO AnadigmDesigner2 ARE NOT DOCUMENTED HERE

AnadigmDesigner®2

User Manual

Anadigm®'s second-generation AnadigmDesigner®2 EDA tool lets you design and implement dynamically reconfigurable analog circuits within a matter of minutes. Build your circuit by dragging and dropping Configurable Analog Modules (CAMs), each of which can be used to implement a range of analog functions for which you set the parameters. AnadigmDesigner®2 includes a time domain functional simulator which provides a convenient way to assess your circuit's behavior without the need for a lab set-up. The simulator's user interface is intuitive and easily learned. Most of the steps are the same that you would take while bench testing. Whether or not you're an analog expert, you can build a complete analog system rapidly, simulate it immediately, and then just point and click to download it to an FPAA chip for testing and validation.

AnadigmDesigner®2 is the world's first EDA product that lets you develop designs using FPAAs that can be reconfigured by the MCU in real-time to change the function they perform within a system or to adapt on-the-fly to maintain precision despite system degradation and aging. AnadigmDesigner®2 takes your design and automatically translates it into C-code that allows the design to be adjusted and controlled by a microprocessor within an embedded system. That means you can now control and adjust analog functions using system software in real time - a breakthrough capability for the analog world!

- AnadigmDesigner® tools allow complex circuits to be designed with a simple drag-and-drop graphical interface
- Proven, easy-to-use design tools for analog circuit designs
- No need for analog expertise to build complete analog systems
- A growing library of reusable CAMs pre-package common analog functions
- Expert system synthesis tools AnadigmFilter™ and AnadigmPID™ automate complex circuit design
- Circuit building blocks are abstracted to a functional level that can be manipulated in AnadigmDesigner®2
- Build complete analog systems rapidly, simulate immediately, and then download to the chip for instant verification
- Built-in multi-chip time domain simulator, four-channel oscilloscope interface, and arbitrary waveform simulation
- Automatically generates C-code to allow analog functions to be adjusted and controlled directly by a microprocessor within an embedded system

Anadigm® reserves the right to make any changes without further notice to any products herein. Anadigm® makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Anadigm® assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including with out limitation consequential or incidental damages. "Typical" parameters can and do vary in different applications. All operating parameters, including "Typicals" must be validated for each customer application by customer's technical experts. Anadigm® does not in this manual convey any license under its patent rights nor the rights of others. Anadigm® software and associated products cannot be used except strictly in accordance with an Anadigm® software license. The terms of the appropriate Anadigm® software license shall prevail over the above terms to the extent of any inconsistency.

The AnadigmDesigner®2 software version 2.2.7 (and later versions) will allow designs created for any of the Anadigmvortex (ANx20E04 or ANx21E04) devices to be evaluated on the AN221D04 platform. To check which version you are using, from inside AnadigmDesigner®2 go to the menu bar and click on Help, then click on About AnadigmDesigner2. To download the latest version of AnadigmDesigner®2 please go to www.anadigm.com.

AnadigmDesigner®2 software version 2.2.7 (and later versions), uses the AN221E04 device as the default device. To change this use the menu item *Edit* → *Insert new chip*, or change the default chip type by choosing *Settings* → *Preferences*. If this version of the software is being used to develop designs targeted to the AN220D04 evaluation board, be sure that the AN220E04 is the device selected within the design software.

Quick Start

AnadigmDesigner®2 User Interface

Detailed Reference Guide

Designing for Low Power Operation

Configuration File Formats

Functional Simulator

Hosted Configuration

AnadigmFilter

AnadigmPID

1 Quick Start

1.1 What does this software do?

AnadigmDesigner®2 software allows you to quickly and easily construct complex analog circuits by selecting, placing and wiring together building block sub-circuits referred to as CAMs (Configurable Analog Modules). The analog circuits that you build can be downloaded to Anadigm®'s Field Programmable Analog Array (FPAA). The FPAA will then function as the circuit you constructed. The results of your analog design can be viewed immediately using a signal generator and an oscilloscope.

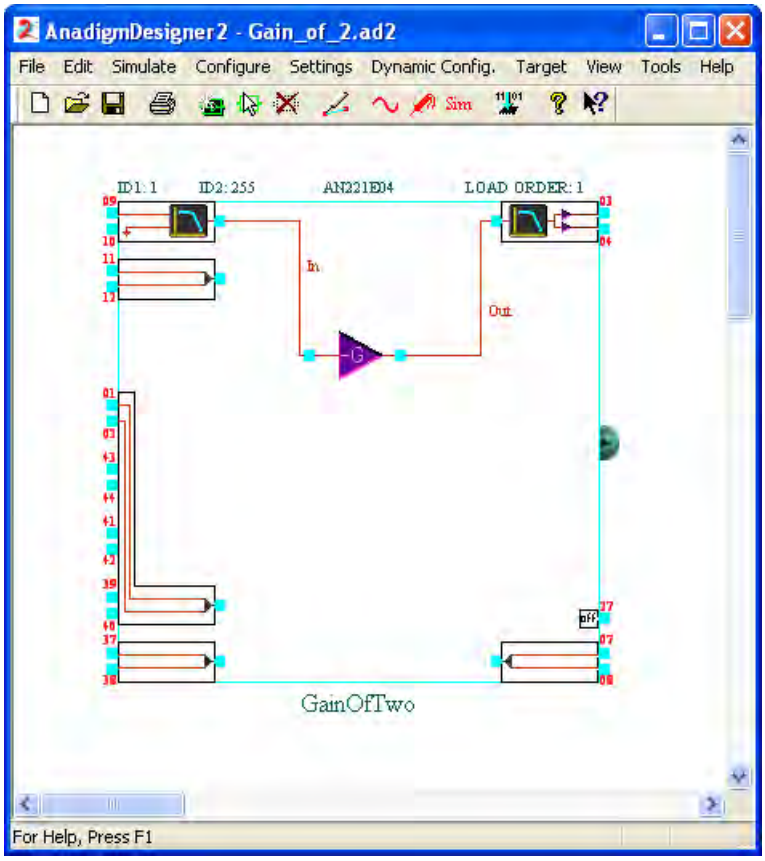
Since Anadigm®'s FPAA technology is SRAM based, the chip can be reprogrammed as many times as desired so you can try out as many circuits as you like.

Also note that multiple, independent circuits can be constructed and run simultaneously within a single device. For example, two completely independent filter networks, each with its own inputs and outputs can be constructed; the parameters and operation of one completely independent of the other. AnadigmDesigner®2 also accommodates large designs which may span across multiple devices.

AnadigmDesigner®2 also generates 'intelligent' configuration data for you as ready-made C Code which can run on a companion microprocessor. This enables on-the-fly reconfiguration of the AN220E04, AN221E04, or AN221E02 devices by a host processor.

A functional simulator is included in to facilitate circuit design and experimentation without the need for any lab equipment. The simulator features an intuitive user interface and displays time domain results graphically.

1.2 Quick Test Drive - Simple Example Circuit



Load in a Simple Example Circuit

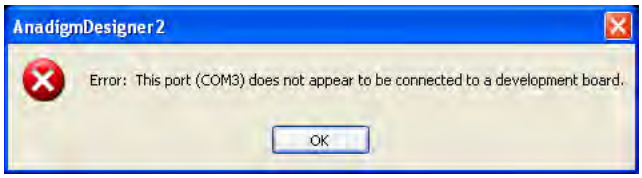
Within AnadigmDesigner®2, use the left mouse button and click on the *File* menu item. This brings up the *File* menu pull-down. Click on the *Open*... menu item. This brings up a file selection dialog box.

Browse to the "Circuits" folder (in the AnadigmDesigner®2 installation folder) then down to the “ANx20 Examples” folder. Locate the "Gain_of_2.ad2" entry, place the cursor over it, and double-click the left mouse button. The gain stage example circuit will be loaded into AnadigmDesigner®2 and displayed in the design window as shown here. (The on screen image has a black background.) This example circuit consists of just a single ended input feeding an inverting gain stage. The output is a differential signal prefaced by a reconstruction filter.

Once a design has been completed it may be saved for future edits, simulated using the built in functional simulator or down-loaded to an Anadigm® FPAA.

Serial Port Download

Assuming you have an evaluation board connected to your PC's serial port, and further assuming that a signal generator and oscilloscope are attached to the proper terminals, we are now ready to download the configuration data to the chip. The gain setting for the GainInv CAM is "2" and the output signal will be inverted.



Downloading to the FPAA on an evaluation board is accomplished by selecting the menu item *Configure → Write configuration data to serial port*. The download takes approximately 2 seconds. If any communications problems are encountered during the download process, AnadigmDesigner®2 will create a pop-up info window describing the problem.

Using a signal generator, apply a signal to pin 9 of the device. The voltage of the input signal should be centered about Voltage Main Reference (VMR). The gained & inverted signal (shown to the right as Vout) can be probed on pin 3, with the complimentary signal on pin 4.

The gain of the GainInv CAM for this circuit is set to two. A double-left click over the GainInv CAM will pop-up the Set CAM Parameters dialog box. Adjust the gain to some new value then again download the data to the serial port to immediately see the effect of the new gain value on the amplitude of the output waveform.

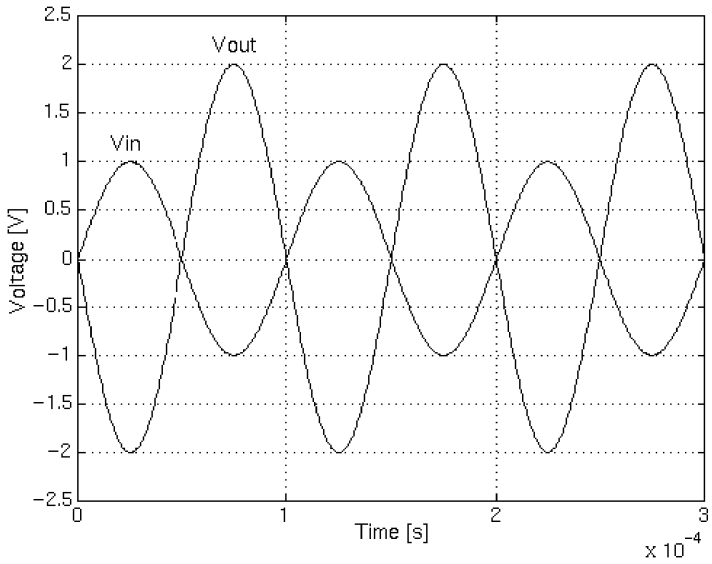


Figure 1 – The Simple Example Circuit in Action.

1.3 Brief Tutorial - Creating a New Circuit

At this point, you already know enough to start driving the design system. Still it will be instructive to go through at least one "from scratch" design just to make sure we cover all the basics. The following simple example is based on the popular AN220E04 array. The completed circuit will be similar to the Gain_of_2 circuit of the previous section.

The design steps are straight forward and have a natural flow. For any design, you follow the same basic steps:

1. Select, Adjust and Place CAMs
2. Connect the CAMs and IO Cells with wires
3. Download the configuration data to the FPAA

Start with a Blank Design Window

At any point, you can choose *File* → *New* and all contents of the main window will be cleared.

Select a CAM

Click on the  symbol (*Get New CAM* tool button) in the tool bar just above the main window (or type the “m” keystroke shortcut) to bring up the CAM Selection dialog box. Use the scroll bar on the right to scroll down to the (GainInv) Inverting Gain Stage CAM. Place the mouse pointer over it and left-click to select. Another left-click on the Create CAM button (or Enter keystroke) and a ghosted image of the CAM will be attached to the mouse cursor, ready to be placed into the FPAA represented in the design window. Once the CAM is dropped into place (another left-click), the Set CAM Parameters dialog box associated with that CAM appears (depending on the *Settings* → *Preferences...* choices made under the CAM tab).

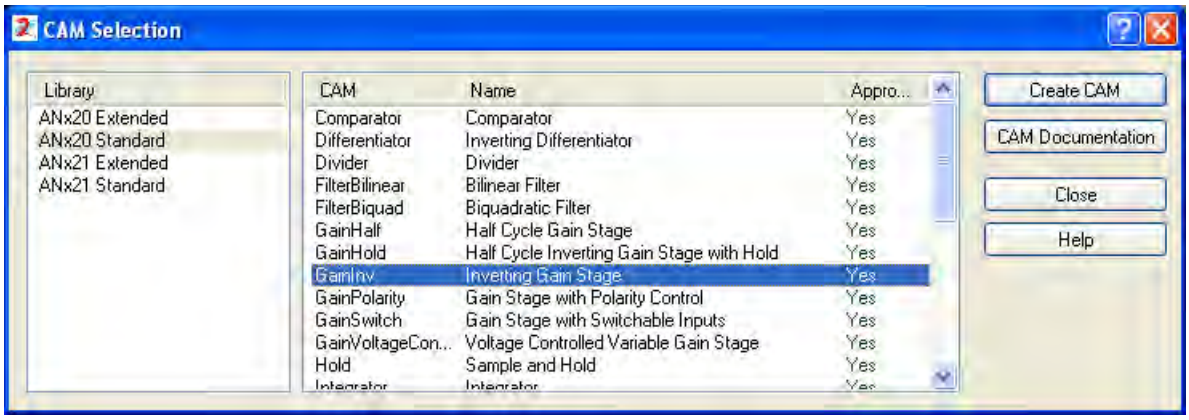


Figure 2 – The CAM Select Dialog Box

Adjust the CAM

The contents of the Set CAM Parameters dialog box vary with the CAM selected, but in general it will always contain all the user adjustable parameters available for that particular CAM. In this particular instance, the only parameter to set for the gain stage is its gain. The "realized" column reflects what the FPAA will be able to actually achieve.

There is a control which allows you to select which of the 4 internal clocks will drive the CAM. There is also the "Documentation" button which pops up a quick help window which fully describes the features of the selected CAM. Once you hit the “OK” button, the dialog box will dismiss itself.

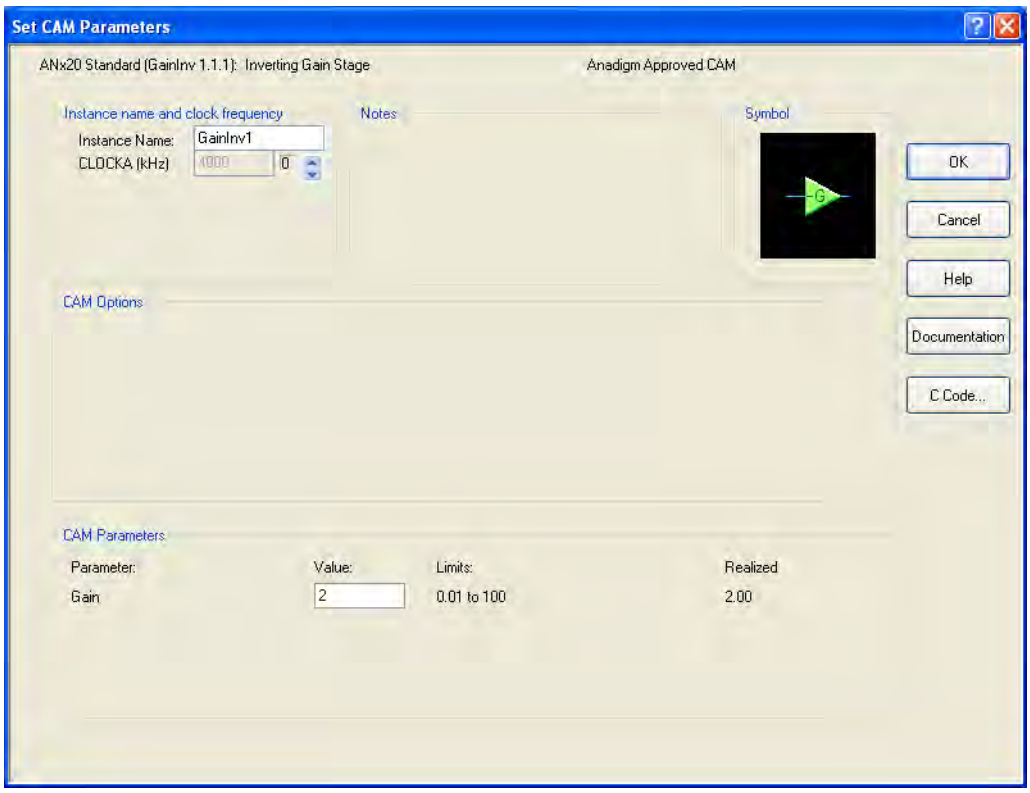


Figure 3 – Set CAM Parameters

Wire the Circuit

Enter the wire mode using the “w” keystroke shortcut. The mouse cursor should now look like a drawing pen. Using a left-click/drag/release mouse action, wire the CAM similarly to what is shown above in Section 1.2, Quick Test Drive - Simple Example Circuit.

Circuit connections in the FPAA array are accomplished by setting switches within the chip. Some nodes cannot be directly connected in the design window because no switch exists within the array that could directly connect them. Other connections are just plain illegal. For example, AnadigmDesigner®2 will not allow you to short two outputs together. In either case, the “drawing pen” cursor will shift to a “forbidden” cursor as a visual cue that the current placement is disallowed.

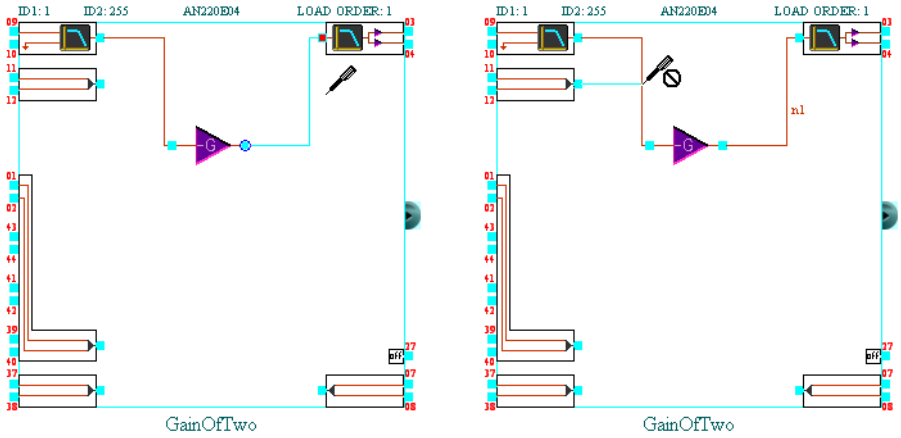


Figure 4 – Left, Legal Connection. Right, The Illegal Connection Attempt is Disallowed.

Figure 4 contains two screen shots taken during wiring. On the left, the drawing pen cursor indicates that the termination point (the input port of an Output Cell) is a valid location to drop the wire’s end. Releasing the left mouse button will cause the wire to snap to this adjacent port. On the right, the “forbidden” cursor is a visual cue that the shorting of two outputs will not be allowed.

Use the “e” keystroke short cut to get back into edit mode. Right click over the Input Cell being used, select the “CAM Settings” item from the resulting pop-up, and convert the Input Cell to “Single-ended” “Amplifier” with “Anti Alias Filter”.

Download the Configuration Data

Using the “Ctrl-w” keystroke shortcut, download the configuration data into the array and look at the oscilloscope screen. If you don’t happen to have a Sinewave oscillator to use as a test input to the FPAA you can you can simply include the one from the CAM library. Just drop it in and wire it up internally. You are now free to change whatever you wish in the design. For example, double-left mouse click on the center of the gain stage symbol to bring up the Set CAM Parameters dialog box to alter the value of the gain. Click OK to store the new gain value then again download the configuration data (Ctrl-w), and look at the oscilloscope to see the effect of the new gain value.

2 AnadigmDesigner®2 User Interface

AnadigmDesigner®2 presents the user with an intuitive interface. The entire program window is sizable to any practical size you feel most comfortable working in. Focus on ease of use continues with a complete, but not overburdened set of familiar and well organized pull down menus; each containing just the options you would expect. A few thoughtfully selected iconic push-button shortcuts are also provided in a tear away palette that can be floated anywhere on screen.

The design window contains a view of the FPAA device(s). This depiction shows the external pins (numbered for easy reference), their associated Input and Output cells, and a large central region which can be populated with CAMs and wiring. Notice in this particular screen shot, that the shortcut palette has been undocked from its default position just under the pull down menus and floated to the right of the main window. It only takes a simple click and drag to tear it away and float it anywhere on the display. The shortcut palette can also be docked to any of the other three sides of the main window just as easily.

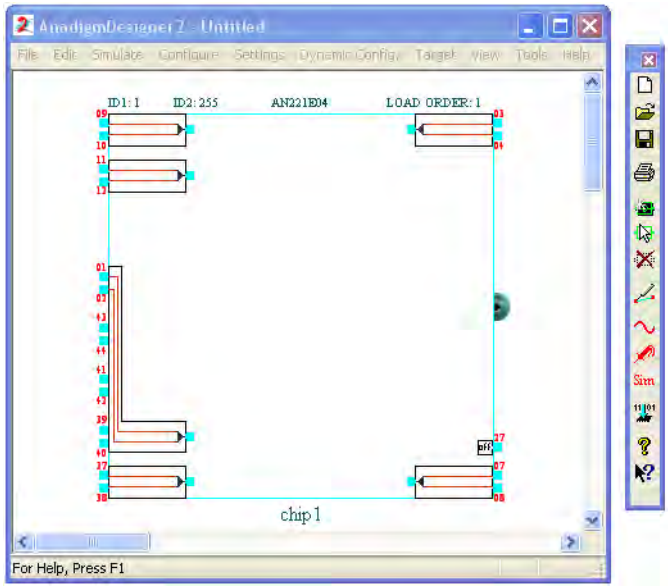


Figure 5 – New Design – Undocked Tool Palette

2.1 The Short Cut Buttons and Associated Keystroke Shortcuts

When first invoked, the program displays a docked pallet of tool buttons associated with the most commonly used functions of the software just below the pull-down menu bar. Because these features are so often used, most of these functions also have brief keystroke shortcuts associated with them as well. More experienced users will soon ignore the short cut buttons all together in favor of the speedier one and two keystroke shortcuts. This toolbar can then be hidden by deselecting it in the *View* pull-down menu.

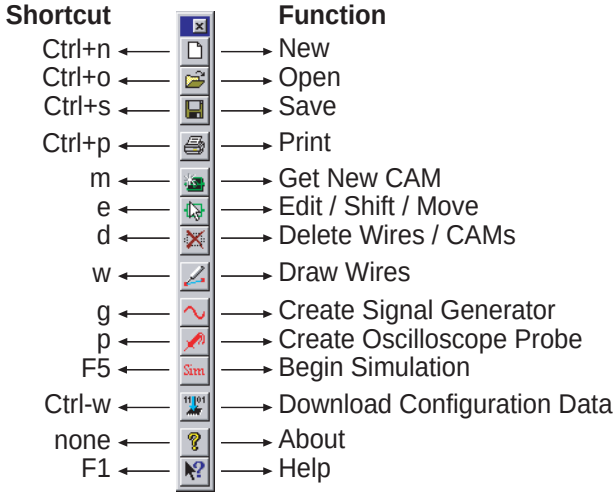


Figure 6 – Short Cut Button and Keystrokes

This particular view of the short cut palette shows it undocked from the design window and resized to a vertical aspect.

Considerable emphasis has been placed on the tool's ease of use and intuitive feel. Conventional keystroke combinations are used for quick access to generic features such as Save, Print and Help.

Single keystroke access is available for the most common editing functions such as Wire Mode, Delete Mode, Edit Mode and Get New CAM.

2.2 An Overview of the Pull-Down Menus

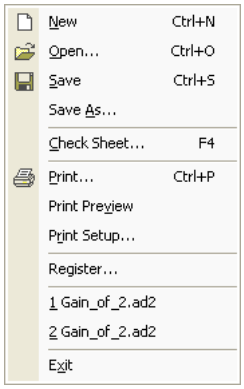
All of the menu items associated with each of the pull-down menus are addressed in detail in Section 3, Detailed Reference Guide. The contents of the *File* menu should be familiar to most experienced Windows users. The balance of the pull down menus are designed to contain an intuitive organization of all the features available within AnadigmDesigner®2.

Figure 7 – An Overview of All Available Menu Items

3 Detailed Reference Guide

This section is devoted to providing a detailed description of the basic menu items of AnadigmDesigner®2. Descriptions of the more comprehensive features are reserved for individual sections: Section 6, Functional Simulator, Section 7, Hosted Configuration, Section 8, AnadigmFilter, and Section 9, AnadigmPID.

3.1 "File" Pull-Down Menu



New

The *New* command creates a from scratch design window. If there is a design currently open that has been modified since its last save, invoking *New* will result in the program prompting for a save of the current work, prior to opening a fresh design window.

Open...

The *Open* command will display a standard file selection dialog box. In the context of AnadigmDesigner®2, you will be browsing for a .ad2 file containing a previously saved design. If the contents of the current design window have been modified since the last save, the program will give you the opportunity to save prior to overwriting with the contents of the file to be opened.

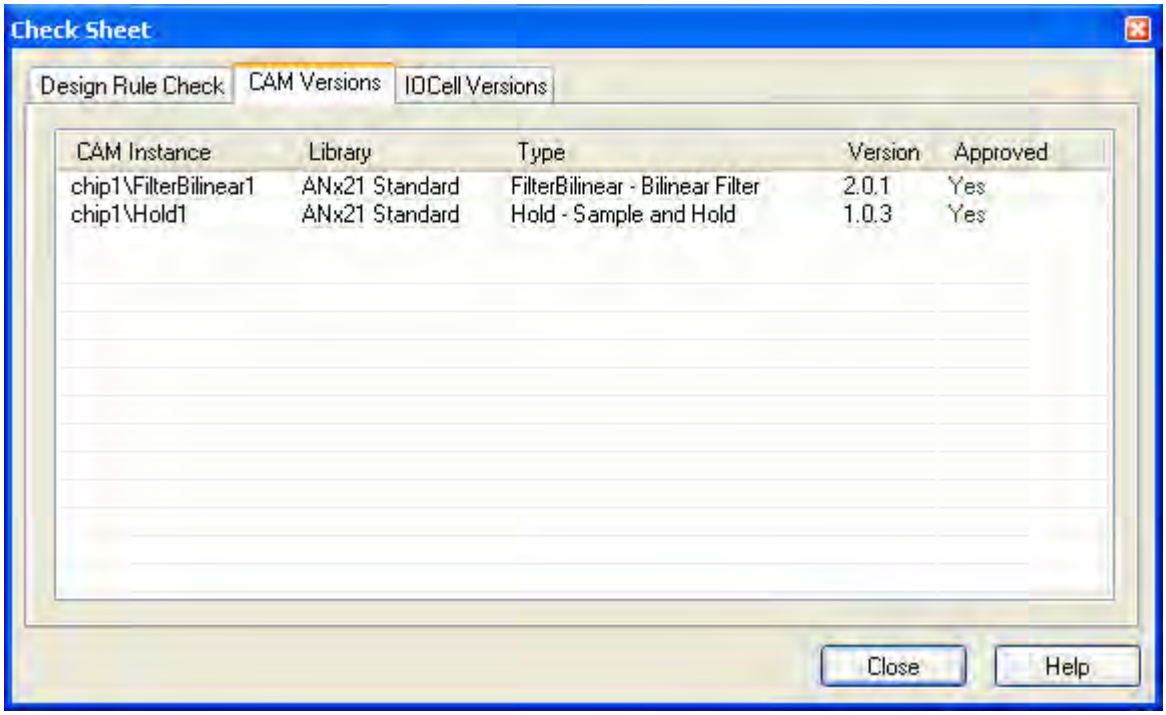
Save

The *Save* command will save the current design to its associated .ad2 file. If the design window happens to be currently “Untitled”, then *Save* will behave the same as the *Save As* menu item.

Save As...

The *Save As* command saves the current design using a new .ad2 appended filename. A typical application would be to open some reference design, make some modifications and *Save As* some new name. This will not affect the contents of the reference design's .ad2 file which was first opened.

Check Sheet...

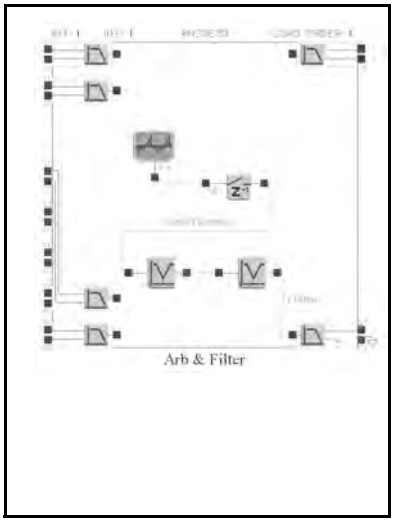


Every .ad2 design file carries within it all the information required by any installation of AnadigmDesigner®2 to re-open that design exactly as it was last saved. Placing a CAM or an IO Cell in a design results in the creation of a complete copy of that CAM or IO Cell from the current library or software installation into the .ad2 file. Libraries then, are only referenced when creating new instances within a design. There is no interaction with the currently installed libraries when simply opening a .ad2 file or adjusting CAM or IO Cell parameters in an existing design. The Check Sheet command however forces a comparison between the loaded design and the currently installed libraries and software. Any CAM or IO Cell version differences between the instantiated components of the design and the current library and software installation are discovered and highlighted.

Invoking the Check Sheet command results in a pop-up information window, arranged in columns. The CAM Versions tab presents: the fully qualified CAM instance names of the current design in the first column, the library names from which the CAMs were originally selected in the second, the CAM types in the third, and in the fourth, the most recent version of that CAM available in the current library of the same name (if installed) from which the CAM was originally selected.

The IOCell Versions tab brings up similar information comparing the design's instantiated IO Cells with the IO Cells available in the current installation directory structure of the software.

Status of the version checking for both the CAMs and IO Cells is color coded. Dark Green text indicates that the version of the component instantiated in the design matches what is available in the software installation. Dark Red indicates that the CAM or IO Cell was not found in the installed library. Dark Blue indicates that the instance version is old compared to the installed library; replacing the CAM or IO Cell with the newer version may be appropriate. Dark Yellow indicates that the instance version in the design is newer then the library or software; a software update may be appropriate.



Chip Name: Arb & Filter
Chip Type: AN220E04

Master Clock: 16000.000
System Clock: 4000.000
Clock 0: 2000.000
Clock 1: 4000.000
Clock 2: 1000.000
Clock 3: 250.000

CAMs assigned to this chip:

AN20 Input Cell (Version 1.0)
Parameter Corner Frequency = 496
Input = Differential
Low Offset Chopper = Off
Anti-Alias Filter = Active

AN20 Input Cell with Pad Select (Version 1.0)
Parameter Corner Frequency = 496
Input = Differential
Low Offset Chopper = Off
Anti-Alias Filter = Active
Input Line Select = A

Print...

The *Print* command brings up a standard print dialog box. Printing an open design is a great way to document the design in a concise way. Both a graphic depiction of the design and textual description of its contents are delivered to your printer.

The first page of the printed report is a graphic depiction of the current design screen. All of the design's CAMs and connections are shown along with Primary and Alternate ID values, Chip Name and Load Order assignment.

This view is handy for the PCB designer. It clearly shows the input and output pin numbers. The following pages of the printed report documents all CAMs used and their parameter settings.

If in the course of design, you elect to change one or more clock frequencies, then these sheets are a handy reference to guide you back to those affected CAMs.

These report pages also serve as a convenient design review reference.

Print Preview

The *Print Preview* command allows you to view the report to be printed out one or two pages at a time. There are tool bar buttons available for zooming in and out, flipping through the pages, printing and dismissing the window.

Print Setup...

The *Print Setup* command brings up a standard print dialog box. From this pop-up, you can select paper size and orientation as well as a printer. You may also adjust print options specific to the selected printer.

Register...

The *Register* command brings up a dialog box in which allows you to type in your License Identifier and License Key. Trial license keys can be obtained free of charge at the Anadigm® web site, http://www.anadigm.com.

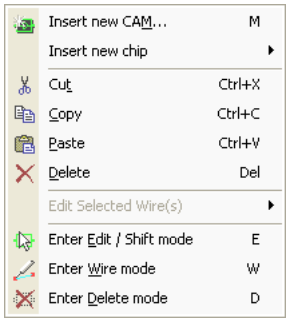
Recent Files

The second to last section of the *File* pull-down menu is a short list of the most recently used design files (sometimes referred to as an MRU list). From this convenient location, you can quickly left click over the desired design and AnadigmDesigner®2 will re-load that design.

Exit

A left click over the *Exit* item will immediately close down AnadigmDesigner®2. If your design was modified since the most recent save, then AnadigmDesigner®2 will give you the opportunity to save your changes before exiting.

3.2 "Edit" Pull-Down Menu



The items under this menu constitute those features through which the majority of the work gets done. As you become familiar with the software, you will probably soon abandon GUI access to these features in favor of the m, e, w, and d keystroke short cuts.

Insert new CAM

Selecting and placing a CAM begins with a pull down menu selection of *Edit* → *Insert New CAM* or its “md.” keystroke shortcut. The CAM Selection window will pop up next, with available CAMs in a scrollable window in the right half of the pop-up.

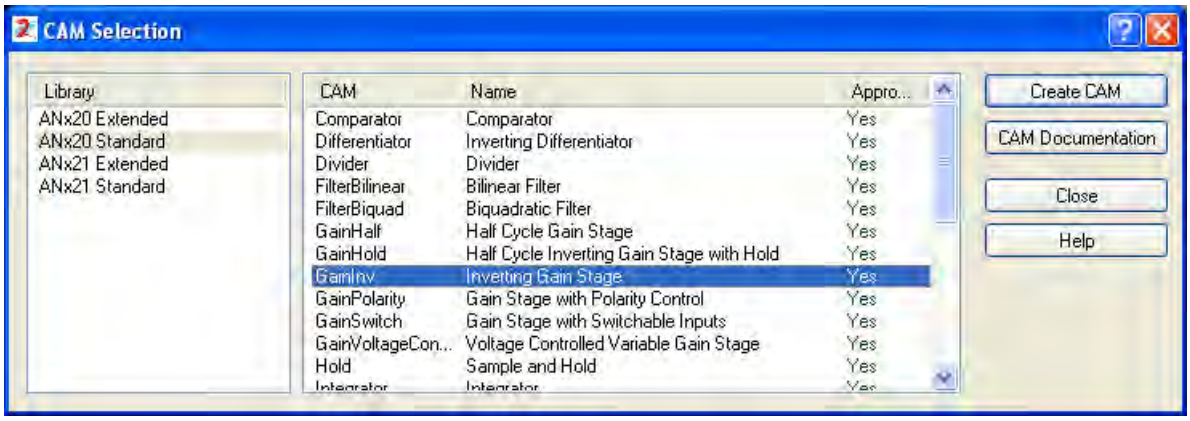


Figure 8 – The CAM Selection Dialog Box

A left-click over the desired CAM selects it. Another left-click on the Create CAM button (or Enter keystroke) and a ghosted image of the CAM attaches to the mouse cursor, ready to be placed into the FPAA represented in the design window. Once the CAM is dropped into place (another left-click), the

Set CAM Parameters dialog box associated with that CAM appears (depending on the *Settings* → *Preferences...* choices made under the CAM tab). The contents of the Set CAM Parameters dialog box vary with the CAM selected, but in general it will always contain all the user adjustable parameters available for that particular CAM. If a particular design requires many CAMs, then a second chip instance should be placed using the *Edit* → *Insert new chip* pull-down menu selection.

The Set CAM Parameters dialog can be reopened at any time from within the design window. A double left-click over any placed CAM will bring up its Set CAM Parameters pop-up It is in this dialog that all the pertinent parameters of this particular placement the CAM are established. AnadigmDesigner®2 defaults to a reasonable set of default parameters but you are free to change any text entry box or radio button with a white background to any in-range value that suits your needs.

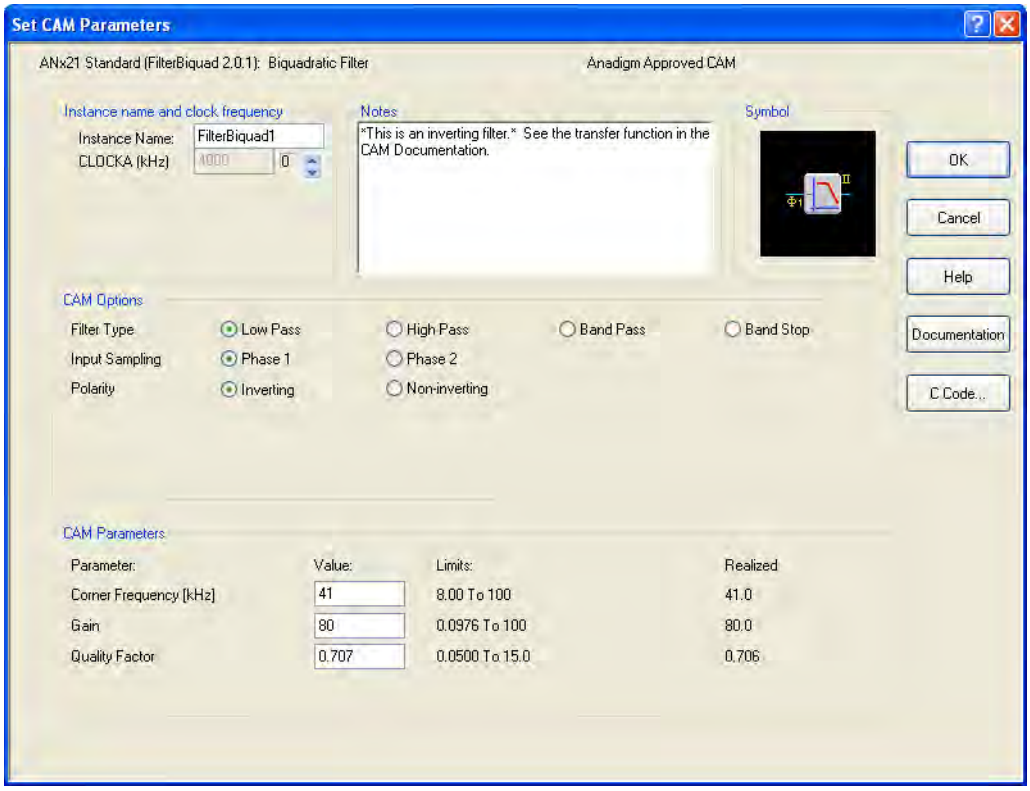


Figure 9 – A Typical Set CAM Parameters Dialog Box

The feedback elements of a CAB are restricted to a limited set of discrete values so the realized numbers may be slightly off from your input numbers. This quantization error is demonstrated in the sample above. The achieved filter Quality Factor is 0.706, just slightly off from the specified 0.707. AnadigmDesigner®2 automatically optimizes CAB programming to adjust for optimal performance while realizing parameters as close as possible to the desired values.

Changing the clock frequencies within the array after a CAM has been placed and parameterized may affect its response. If it is necessary for you to change a clock setting after your CAMs are placed and parameterized, then you will need to go back and right click on all the CAMs assigned to that clock and make the appropriate corrections.

Additional details on each of the parameters, limits and performance values can be found in on-line documentation available for each CAM.

Moving a CAM is accomplished in the same fashion as moving a wire or a wire label. As the edit mode cursor is moved to within close proximity of a placed CAM it will switch from an arrow to an arrow with a small 4-way arrow icon adjacent. Figure 10 is a screen shot of a typical CAM move in process.

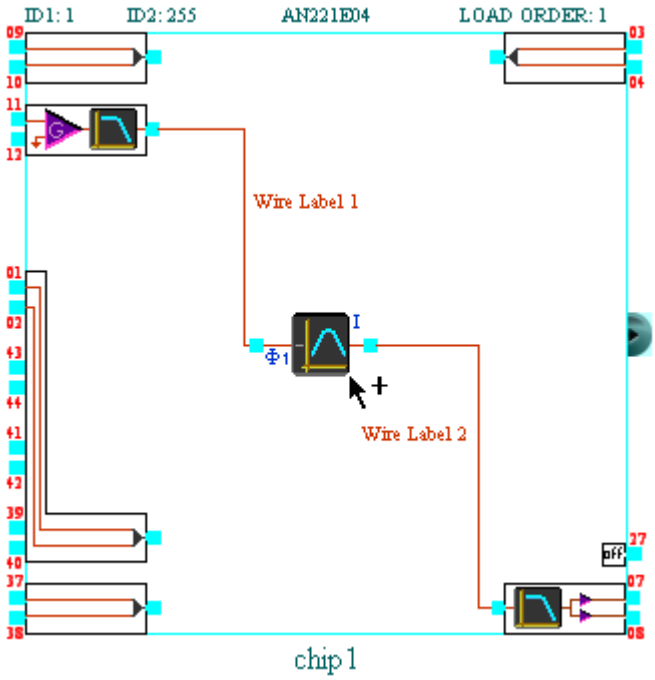
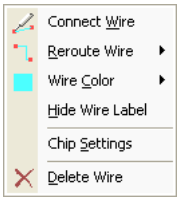


Figure 10 – CAM Dragging

Insert new chip

The *Insert new chip* selection places a second chip instance within the design window. If during the course of design, adding another CAM would exceed the capacity of the array, AnadigmDesigner®2 will not allow its placement. In this instance, it becomes necessary to add a second device to the design, using this command.

Edit Selected Wire(s)



Selecting a wire by right clicking over it will result in its being highlighted and the adjacent selection box will pop up. There are several commands available to fine tune the way the wire is presented in the design window. The color and routing method are both adjustable. There is a selection item for hiding (or unhiding) the wire label. There are also controls for deleting the wire and adding a new connection to the wire. Finally, as with most other areas of the design window, the right click also brings up access to a control allowing adjusting of chip level parameters.

The standard colors available to select from are Red, Green and Blue. If an alternate color is preferred, selecting “Other” will bring up another dialog window which will allow selection of any possible displayable color. In more complex designs, color coding the wires can be an effective and convenient way to keep track of what is going on where in the schematic. Neither the wire’s color nor the wire’s label text influences its actual connectivity in any way.

In addition to being able to change the color of the wire, it is also possible to change the method used to display the wire graphic. The “Straight Line” option converts the route to a simple straight line between its two endpoints. “Horizontal Start” begins the route horizontally and ends vertically at the terminus. “Vertical Start” does just the opposite.

Enter Edit / Shift mode

The Edit / Shift mode is the default mode of a design session. In this mode a left click and drag over a placed CAM will stick the CAM to the mouse cursor and allow it to be dragged to another location within the design window. The keystroke shortcut for entering this mode is “e”.

A handy extension to the Edit / Shift mode allows you to "clone" an instantiated CAM. While in Edit / Shift mode, doing a right-click and drag over a CAM, will result in a ghosted clone of that CAM being attached to your cursor. Releasing the right mouse button drops the clone into place and pops up a menu selection asking whether the CAM is to be moved or copied. No connection information is carried over to the new location for a copied CAM, but all of the original CAM’s parameters are retained; it is cloned..

Enter Wire mode

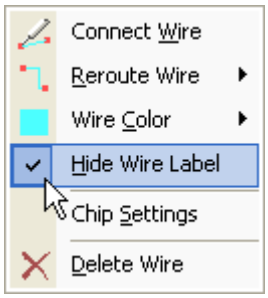
The *Enter Wire mode* command or its “w” keystroke shortcut moves the tool into wiring mode. When in wiring mode, the mouse cursor will either be a drawing pen or a drawing pen with an adjacent forbidden symbol. (Please refer to Figure 4 for examples of each.) The drawing pen indicates that a valid connection point is sufficiently close to snap to, allowing the start or termination of a wire with a single left click. A forbidden symbol indicates that any start or termination actions will not be allowed.

When in the Edit / Shift mode, the cursor automatically changes to a drawing pen when moved close to an allowable connection point. Conversely, when in Wire Mode a right-click brings up a pop-up menu allowing a quick change into Edit / Shift or Delete modes.

By default, all new wires placed into the design have a descriptive test wire label associated with them. This is a graphic only label, it has no bearing on the connectivity of the wires drawn. i.e. Having two or more wires labeled with the same name will not result in a short.

A single left click over a net name will convert the cursor into a familiar blinking vertical text insertion cursor. New wire label text can then be entered in the normal way. A wire label may be up to 20 characters long. If the net does not happen to have a name visible, there are two possible reasons.

The first (and least likely) is that the wire label is nothing but space characters. If this is the case, the only way to find the label in order to change it is to move the (edit mode) cursor along the length of the wire until the arrow cursor converts to a arrow with an adjacent “I-beam” symbol. A single left click at that point will cause the text insertion cursor to activate within the wire label.



The more likely scenario is that the label for the wire is selected to be hidden. A single right click over the wire itself, will cause the wire to highlight (indicating its selection) and the associated selection box pop up will appear. Deselecting the *Hide Wire Label* checkbox will result in the wire label re-appearing.

Both wires and wire labels can be moved to create a more visually pleasing schematic representation of your design.

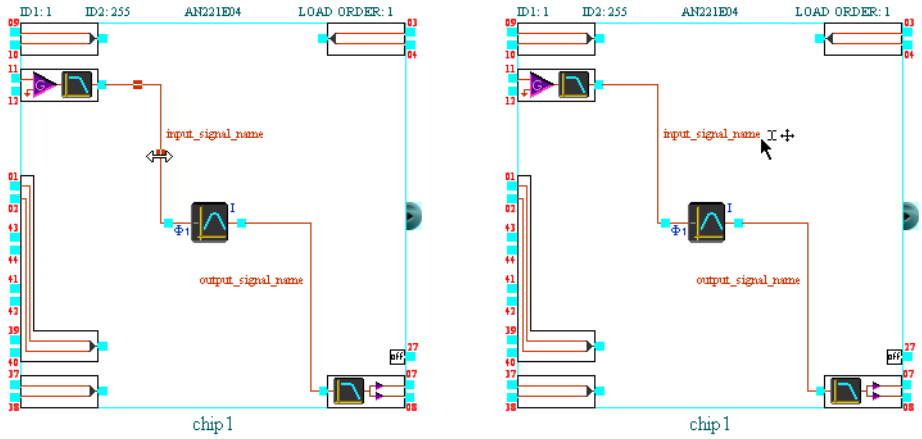


Figure 11 – Left, Wire Drag — Right, Label Drag.

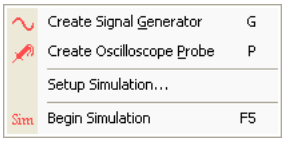
The left side of Figure 11 is a screen shot showing a typical wire drag in action. When in edit (or wire) modes, if the cursor gets sufficiently close to a wire, small square grab handles will activate on each segment of the wire and the cursor will switch from an arrow (or drawing pen) to double headed arrow. Left clicking and dragging one of these grab handles with this new cursor allows the associated line segment to be dragged orthogonally. The attached segments stretch to accommodate the move.

The right side of Figure 11 is a screen shot showing a typical wire label drag. If the mouse cursor gets sufficiently close to a wire label, the cursor will switch from a pointer or pen to a pointer with small “I-beam” and “4-way arrow” icons adjacent. Left clicking and dragging at this point allows the label to be moved to another valid location. Valid locations for wire labels are all within close proximity to the wire itself.

Enter Delete mode

The *Enter Delete mode* command or its “d” keystroke shortcut is used to delete placed CAMs or connections in a design. Once in the Delete mode, the cursor gets an "X" symbol added to it. Just left click over the unwanted CAM or connection element and it disappears from your design.

3.3 "Simulate" Pull-Down Menu



AnadigmDesigner®2 comes with a built in functional simulator. The simulator provides a convenient way to analyze your circuit designs whenever you are away from your test bench. Section 6, Functional Simulator, covers this portion of the design system in much greater detail, but for the sake of easy reference, the pull-down menu items are briefly described below.

Create Signal Generator

The *Create Signal Generator* command or its “g” keystroke shortcut will attach a signal generator icon to the mouse cursor. Drag the icon over to the desired Input Cell’s internal connection port, and use a left-click to drop it in place. A right-click over the generator icon will bring up a programmable parameter dialogue window. Up to 4 signal generators are allowed in the design to provide simulation stimuli.

Create Oscilloscope Probe

The behavior of this menu item is very much the same as described for *Create Signal Generator*. This menu selection or its “p” keystroke shortcut instead leaves an oscilloscope probe icon attached to the mouse cursor. A left-click over any valid wire connection point, drops the color keyed probe in place. Up to 4 of these probes may be placed in the design in order to monitor simulation results.

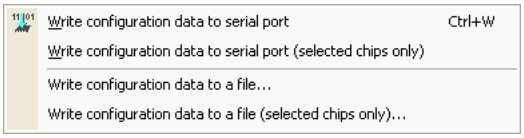
Setup Simulation...

The *Setup Simulation* command pops up a simulation setup dialog window. The details for various entries in this window are discussed in detail in Section 6, Functional Simulator.

Begin Simulation

The *Begin Simulation* command or its “F5” shortcut key, will begin a simulation sequence. Upon conclusion of the simulation, the Oscilloscope window pops up displaying all the simulation results available. The ESC key will abort the simulation run (so long as the AnadigmDesigner®2 is the active window).

3.4 "Configure" Pull-Down Menu



The *Configure* pull-down menu has a series of selections that all do the same basic thing. They each move FPAA configuration data out of the design system to either an RS-232 port or a configuration data file. Detailed descriptions of the serial port data stream and configuration file formats are addressed later in this manual (Section 7, Hosted Configuration).

Write configuration data to serial port
Write configuration data to serial port (selected chips only)

The *Write configuration data to serial port* command or its “Ctrl-w” keystroke shortcut writes the configuration data for the FPAA devices being displayed to the PC's serial port. Many of the Anadigm® evaluation products have the appropriate RS-232 circuitry on board for support of this function, obviating the need for programming Serial Boot PROMs or writing microcontroller hosting programs.

The *Write configuration data to serial port (selected chips only)* command writes the configuration data only for the currently selected FPAA instance(s) to the PC’s serial port. Multiple FPAA instances can be selected by holding down the ctrl key while left-clicking the boundaries of the desired devices.

Write configuration data to a file...
Write configuration data to a file (selected chips only)...

These commands write the configuration data, for the current or selected FPAA instances to a data file. By default, the file will be written into the same directory that the active circuit file (.ad2 file) resides in and will inherent the same root file name. The configuration data file may be used to program a serial PROM or used as a data file for microcontroller hosted FPAA designs.

Some of the older versions of EPROM programmer software packages are unable to handle file names greater than eight characters so you might wish to choose your circuit name with this in mind.

There are several different file formats available for storing configuration data. The available formats are presented in the “Save as type:” selection control within the *Write Configuration Dialog* pop-up.

AHF file (*.ahf)

AHF or ASCII Hex File format is a common basic data interchange format. Configuration data bytes are represented as hexidecimal ASCII character pairs.

S1 File (*.ms1)

Like .AHF, the ".ms1" file contains the ASCII hexidecimal representation of the FPAA’s configuration data. The data is presented in an industry standard S-Record format file.

S2 File (*.ms2)

S2 records differ slightly from S1 records in that they have 24 bit address fields whereas S1 records allocate only 16 bits of addressing per record.

Binary File (*.bin)

The configuration data is written to a file in space efficient binary format. The file may be used to program a serial PROM chip or used as a data file for microcontroller hosted FPAA designs.

Reversed Option

An alternate format available for each of the file format selections mentioned above reverses the bit ordering of the configuration data bytes. It is not uncommon to encounter serial PROM programmers which require programming data to be presented this way.

The options described in the following section support some of the more detailed facets of the FPAA's configuration logic. In order to best assimilate the information presented below, you should first read the configuration logic section of the FPAA User Manual.

Configuration File Options

WIthin the Write Configuration Dialog there is a Configuration File Options button which brings up the dialog shown in Figure 12.

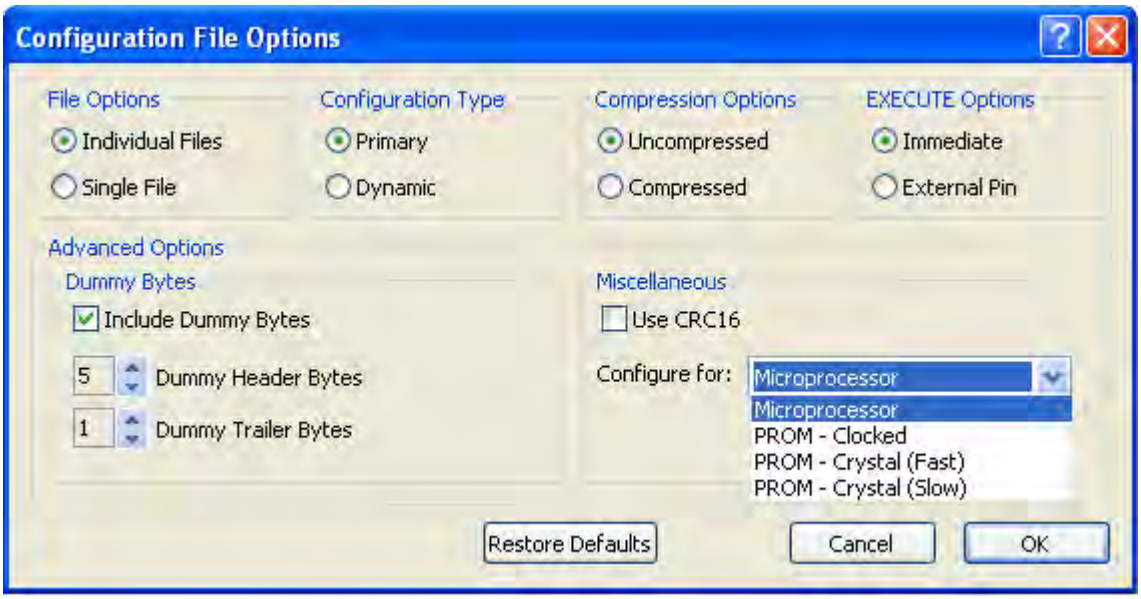


Figure 12 – Configuration File Options from Write Configuration Data

File Options

It is possible to have multiple FPAA instances open concurrently. In such cases, these options select between storing configuration data for all the devices in a single data file or into individual files.

Configuration Type

The first time an FPAA gets loaded out of reset, it requires a Primary configuration. A Primary configuration data set includes the JTAG ID for the FPAA device type and establishes the ID1 for the particular device. Once an FPAA has completed a Primary configuration, only a Dynamic (a.k.a. Update) configuration data set can be loaded. A Dynamic configuration data set does not include a JTAG ID field.

Compression Options

After a power-on reset (or after holding ERRb low for 16 clock cycles) the FPAA will zero out Configuration SRAM and stand ready to accept a Primary Configuration. Since the reset put the Configuration SRAM into a known (all 0's) state, the only data that really needs to be transferred is non-zero bytes. Recall that the data blocks of a Primary Configuration are prefaced with address and byte count data. A compressed configuration file isn't really compressed in the conventional sense, instead it is a configu-

ration data file which typically contains many data blocks of non-zero data. An uncompressed configuration file will by contrast contain just a few large data blocks, but most of the data will be 0's.

EXECUTE Options

Normally, when an configuration data set finishes loading into an FPAA's Shadow SRAM, the data will at once shift into the Configuration SRAM. This happens when the Control Byte of the data blocks has its ENDEXECUTE bit set to 1. Setting this option to *Immediate* will set the ENDEXECUTE bit to 1. Setting this option to *External Pin*, clears the ENDEXECUTE bit to 0. A configuration data set with this condition can be loaded into the FPAA, but the transfer from Shadow SRAM to Configuration SRAM will only happen with the assertion of the device's EXECUTE pin. Please refer to the Configuration Logic section of the device User Manual for further details.

Dummy Bytes

The controls associated with these options add prefix and suffix all zero dummy bytes to the configuration data files. The configuration state machine needs extra clocks before and after the configuration data to cycle through its state machines. Appending dummy bytes ahead and behind of the configuration data file is a simple way to facilitate this. Please refer to the Configuration Logic section of the device User Manual for further details.

Use CRC16

A configuration data block may conclude with either a hard coded 0x2A or a two byte CRC-16 checksum. Enabling this option creates configuration data sets with CRC-16 checksums. Please refer to the Configuration Logic section of the device User Manual for further details.

Configure for:

There are four options available under the drop down selection box associated with *Configure for:* Microprocessor, PROM - Clocked, PROM - Crystal (Fast), PROM - Crystal (Slow). Each of these options set configuration bits that control of some of the logic associated with the generation and routing of clocks.

Configure for: Microprocessor

With this selection, the internal analog clock is derived from the ACLK pin. In order for the analog circuitry to function as expected, the ACLK pin must be driven continuously with a clock of fixed frequency.

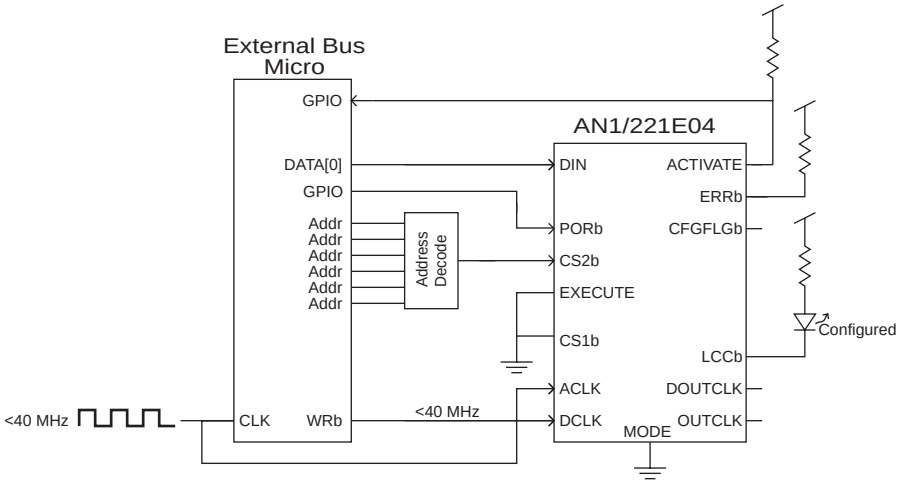


Figure 13 – A Typical Connection Scheme for a Host Microprocessor

Configure for: PROM - Clocked

With this selection, the internal analog clock is instead derived form the DCLK pin. Accordingly, it is now the DCLK pin that must be driven continuously with a clock of fixed frequency in order for the analog circuitry to function as expected.

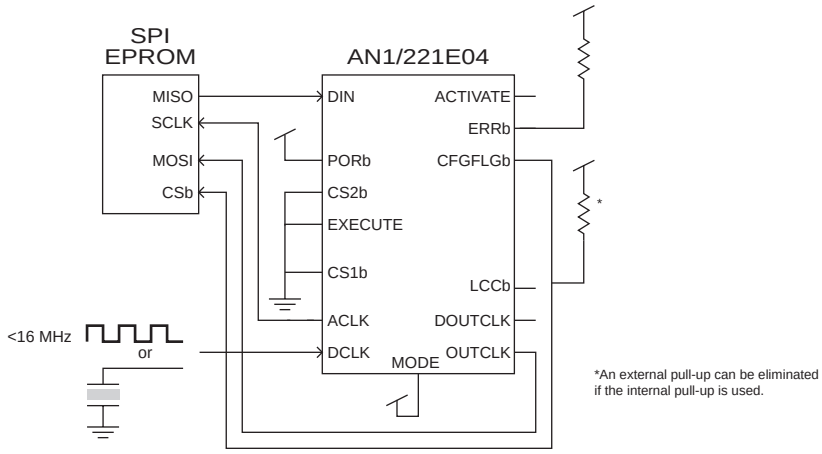


Figure 14 – A Typical Connection for Clocked PROM

Configure for: PROM - Crystal (Fast)
Configure for: PROM - Crystal (Slow)

These two configuration options are best discussed together. The first selection results in the device's DCLK signal being routed to its DOUTCLK as soon as the configuration completes. The second selection results in this same DCLK to DOUTCLK behavior *and* the internal analog clock is derived form the ACLK pin (as described just above for Configure for: Microprocessor).

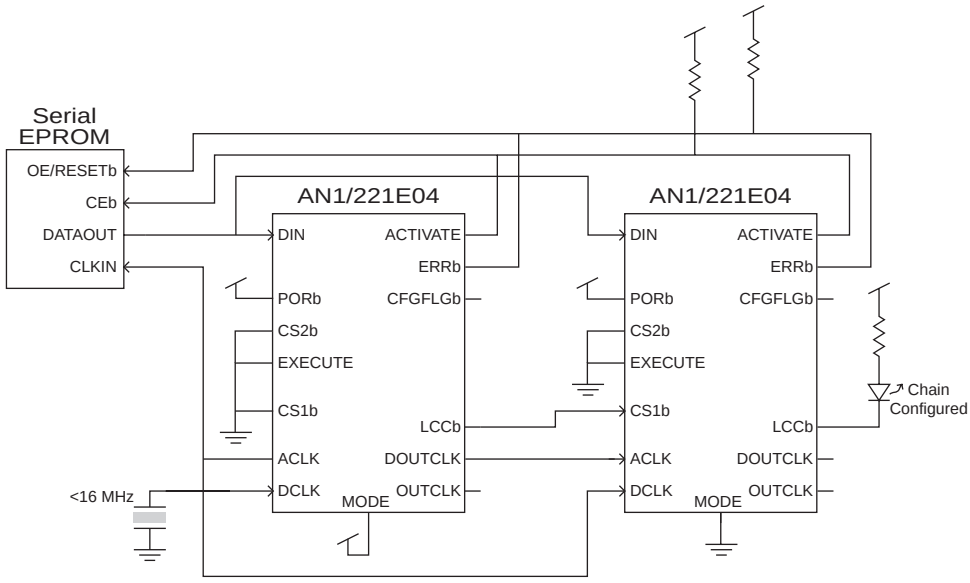
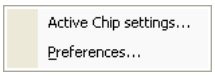


Figure 15 – A Typical Connection for Multiple Devices from a Single PROM

In the example above, the first device in the configuration chain should have a configuration file configured for *PROM - Crystal (Fast)*; DCLK will be routed to DOUTCLK as soon as the configuration completes. The second (and all subsequent) device(s) in the configuration chain should have a configuration file configured for *PROM - Crystal (Slow)*; DCLK will be routed to DOUTCLK and the internal analog clock will be derived from ACLK pin.

3.5 "Settings" Pull-Down Menu

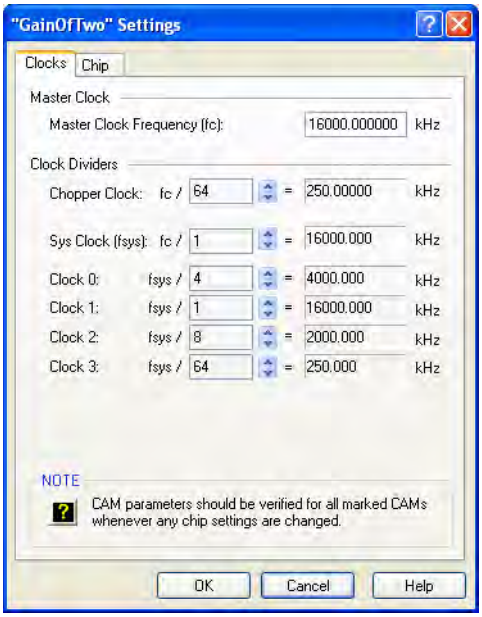


The *Settings* pull-down menu is something of a catch all area for various global settings applicable to device and the AnadigmDesigner®2 software.

3.5.1 Active Chip Settings...

This multi-tabbed window provides access to parameter settings for the currently active chip. If there are multiple chips in the design window, the active chip is highlighted with a red border. If there is a single chip in the design window, then it is the active chip.

Clocks (Tab)



The AnadigmDesigner®2 design window allows for more than one chip instance to be open at a time. It is most likely the case (and is the recommended situation) that each of the devices are driven off a common clock, but this is not a requirement. It is therefore necessary to provide a dialog box for chip level clock settings for each of the devices in the design. The Clocks tab of the “Active Chip settings...” selection from the *Settings* pull-down menu provides such a dialog box.

Parameters set in this tab of the dialog box only apply to the chip instance that currently has focus in the design window, the active chip. This in reinforced in the title bar of the dialog box. The chip’s name becomes part of the title bar. This is the same name displayed in the bottom center of the chip graphic in the design window. The chip name is used in the generation of C code, explained more fully in Section 7, Hosted Configuration.

Master Clock

All the device’s switched capacitor circuits and chopper amplifiers need a regular clock signal in order to operate predictably. It is critical for AnadigmDesigner®2 to know the exact frequency of the Master Clock, as nearly all programmable parameter calculations are a function of a switched capacitor clock derived directly from this frequency. The top text entry field of the Chip Setting dialog box is the location to provide this information to AnadigmDesigner®2.

Chopper Clock

One portion of the analog array influenced by clock frequency are the ultra low input offset chopper amps associated with analog input cells. Setting a lower clock frequency for these amplifiers improves settling time, while setting a higher value allows for better clock noise attenuation by the subsequent continuous time filter.

Sys Clock and Clock[3:0]

There is a System Clock divider that takes the analog clock input frequency (Master Clock, fc) and pre-scales it down to the analog System Clock frequency (fsys). The system clock is further divided down to form the four clocks used by all internal switch cap circuits, Clock[3:0]. The final control of this dialog box establishes which of the internal clocks will be routed to the device’s OUTCLK pin.

Chip (Tab)

Additional chip level parameters include controls for power consumption, pull-up enables and support for PROM based configuration. As with the parameters in the Clock tab, parameters set in the Chip tab of the dialog box only apply to the chip instance that currently has focus in the design window, the active chip.

Power Settings

AnadigmDesigner®2 can configure the FPAA to operate in either a High Bandwidth mode or a Low Power mode. In the Low Power mode all of the CAB circuits are biased for low power operation. For a more detailed look at the low power features, please refer to Section 4, Designing for Low Power Operation.

Enable Pull-ups

The on-chip pull-ups associated with the device's DIN, CFGFLGb and ACTIVATE pins are programmable. Selecting the associated check box modifies the data set written to configuration data files to enable these pull-up devices. These settings have no affect on serial port download data; this ensures compatibility with Anadigm® evaluation platform products.

3.5.2 Preferences...

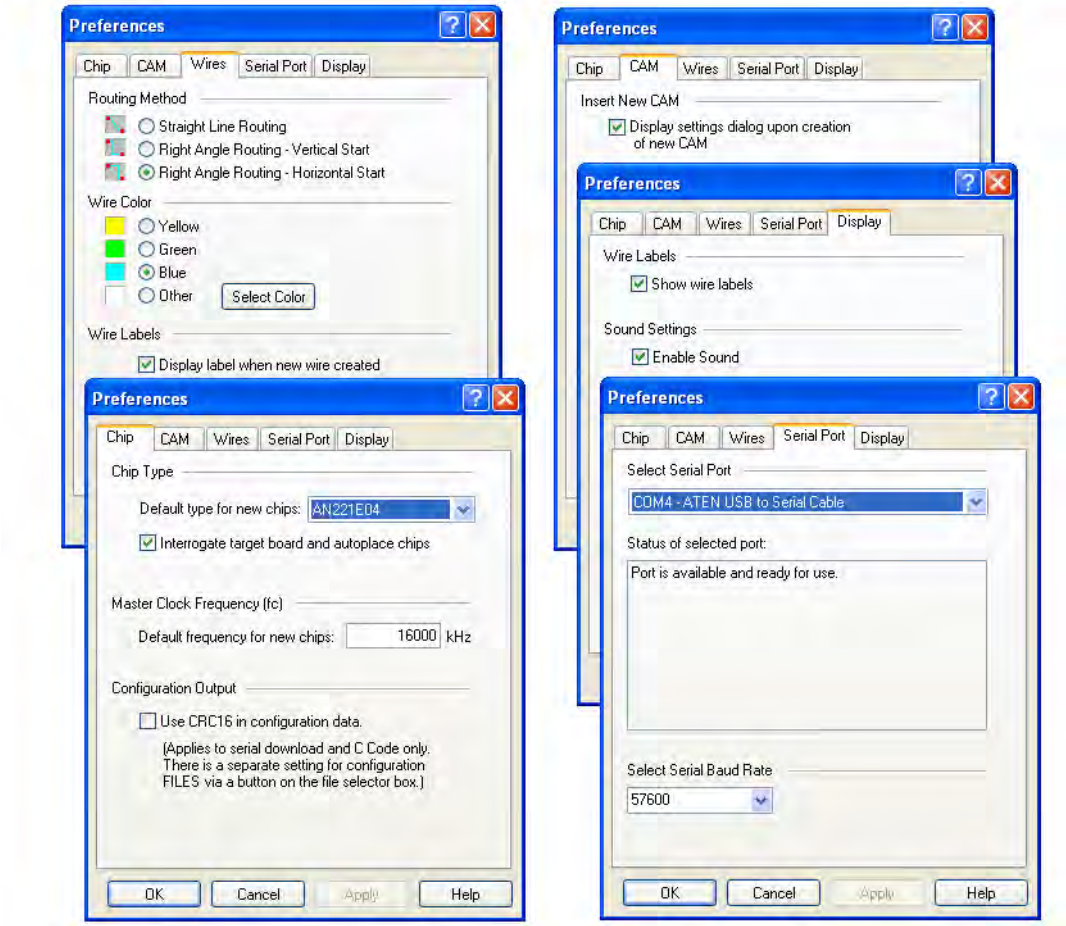


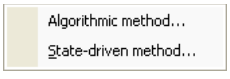
Figure 16 – Chip, CAM, Wires, Serial Port, and Display Preferences

The Preferences dialog box contains five sub dialogs, each accessible through their own tabs. All of the features are self explanatory, but there are a few features worthy of individual mention. The *Display* tab contains the “Show wire labels” check box which controls the display of *all* of the wire labels. Without this global control, labels would have to be turned off and on individually (by right clicking over a wire and selecting the “Hide Wire Label” command). This tab also contains the “Enable Sound” check box. Some users appreciate the audible feedback, others do not. Either way, this control is of interest to all.

The *Wires* tab contains the controls which determine the default label font, wire color and graphical wire routing style.

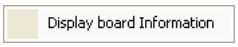
The *Serial Port* tab is used by AnadigmDesigner®2 to determine which serial port is connected to the target system and what Baud rate to use.

3.6 “Dynamic Config.” Pull-Down Menu



AnadigmDesigner®2 goes beyond just presenting an analog design environment and generating the configuration data file for the FPAA. This new generation of design system also generates C Code which enables a companion microprocessor to dynamically update the device operation by making the requisite C function calls. The full details are discussed separately in Section 7, Hosted Configuration.

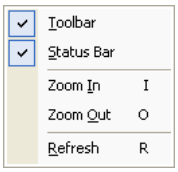
3.7 “Target” Pull-Down Menu



The Target pull-down menu provides access to functions associated with a target FPAA evaluation board. The assumption here is that the target board contains a microcontroller running a compatible version of the Anadigm® Boot Kernel (ABK). Usually this would be an Anadigm® supplied evaluation board but may also be a user designed target running the ABK.

This command returns a text message which contains information about the version of the ABK firmware that is currently running.

3.8 "View" Pull-Down Menu



Toolbar

The Toolbar check box controls whether or not the toolbar containing the program’s shortcut buttons is displayed. As with most other programs, the tool bar can be anchored to any of the other three sides of the design window, or floated undocked anywhere else on the screen. A left-click and drag over the short grey bar at the left most edge of the toolbar is all that is required to undock it from its default position at the top of the screen.

Status Bar

This check box controls whether or not the status bar appears at the bottom of the design window. It is highly recommended that you leave this feature enabled as the status bar provides useful user feedback information.

Zoom In and Zoom Out

These controls and their “i” and “o” keystroke shortcuts control the zoom setting of the design window. This is most useful when designs span more than a single FPAA device.

Refresh

No operating system’s graphics sub-system is without flaw. On rare occasions, distracting graphic artifacts may interfere with the presentation of more useful information within the design window. A quick swipe of the “r” keystroke shortcut for this command will force a screen redraw and this always immediately remedies the annoyance.

3.9 "Tools" Pull-Down Menu

This is where extensions to the basic AnadigmDesigner®2 program get plugged in. Among the first of these extensions available are AnadigmFilter™. and AnadigmPID™.

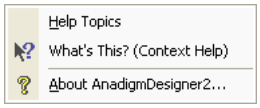
AnadigmFilter

AnadigmFilter™ is an intuitive but extremely powerful filter design tool mated to AnadigmDesigner®2. AnadigmFilter™ accepts basic filter parameters as inputs and immediately returns the required CAMs along with CAM parameter and connection settings to AnadigmDesigner®2. Filter synthesis of nearly any practical order and type is immediate and sure. Please refer to Section 8, AnadigmFilter for full details.

AnadigmPID

AnadigmPID™ is a powerful design aid for the construction of Proportional-Integral-Derivative control circuits. Please refer to Section 9, AnadigmPID for full details.

3.9.1 "Help" Pull-Down Menu



Topics

This control brings up the familiar help window which provides access to a well organized an expansive collection of topics. The content is extensively linked and thoroughly indexed.

What's This? (Context Help)

Selecting this control converts the normal mouse arrow cursor into an arrow with a question mark. Pointing and left clicking over any significant element in the design window will cause an informational pop-up window to appear.

About AnadigmDesigner2...

The *About AnadigmDesigner2* menu item returns the current version of the software in a pop-up window.

3.10 Resource Panel

AnadigmDesigner®2 recalculates the consumed device resources and re-estimates power consumption every time a CAM is placed, anytime parameters are changed in a CAM or IO, or anytime the chip's settings are changed. The resource panel associated with each chip in the design window presents this information graphically. To activate the resource panel, left click over the small triangle centered along the right edge of the chip graphic (see Figure18).

AnadigmDesigner®2 considers all the chip, IO and CAM settings and presents a power estimate in mW in the upper section of the resource panel. This power calculation is updated anytime the design is changed.

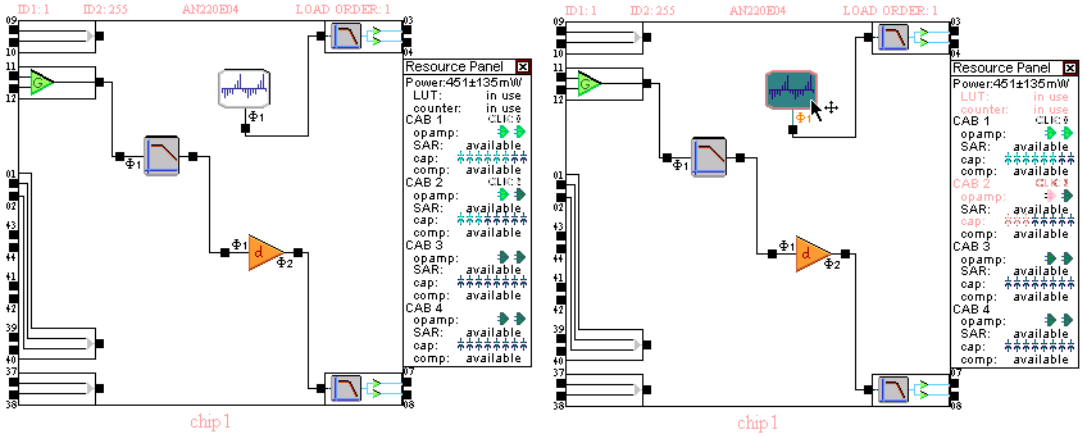


Figure 17 – Left-Clicking on the Arbitrary Waveform CAM Highlights its Associated Resources

The resource panel also displays the FPAA's physical resource availability. As CAMs are added to the design, AnadigmDesigner®2 automatically places them into a CAB and updates the resource panel. Left clicking over a placed CAM in the chip graphic will high light that CAM and the resources assigned to it. Conversely, left clicking over an assigned resource in the resource panel, will cause the resource to high light, along with its associated CAM. In right half of Figure 17, the Arbitrary Waveform generator has been selected with a left click. This complex CAM consumed the LUT and counter resources as well as one op-amp in CAB 2 along with 3 of 8 of that CAB's capacitor banks.

4 Designing for Low Power Operation

Low power is often a requirement in system designs. AnadigmDesigner®2 has several features specifically targeted at conserving power in such applications.

4.1 High Bandwidth vs. Low Power

The menu item *Settings → Active Chip settings...* opens up the Chip Settings dialog box. From this dialog the device can be configured to operate in either a High Bandwidth mode or a Low Power mode. In the Low Power mode all of the CAB circuits are biased for low power operation.

The power consumed by a design in the Low Power mode, ranges from one-third to one-half of the power consumed by the same design in the High Bandwidth mode. When the Low Power mode is selected, a '?' symbol appears on top of the CAMs placed in the design. This '?' symbol warns the users to open the CAM dialog box and make sure the settings are correct for the Low Power mode.

The Low Power mode is currently being characterized to determine the effect of lower biasing on the bandwidth/performance of the different CAMs. Until then, customers using the Low Power mode are advised to verify their designs on the bench.

4.2 Power Estimator

There is a Resource Panel that can be accessed by left clicking on the small triangle on the right side of the chip schematic. The Resource Panel includes a power estimate which is completely interactive and reflects a live power estimate as the design changes. Estimated power consumed is expressed in mW and the software provides a first-order approximation of the variability inherent in the estimate.

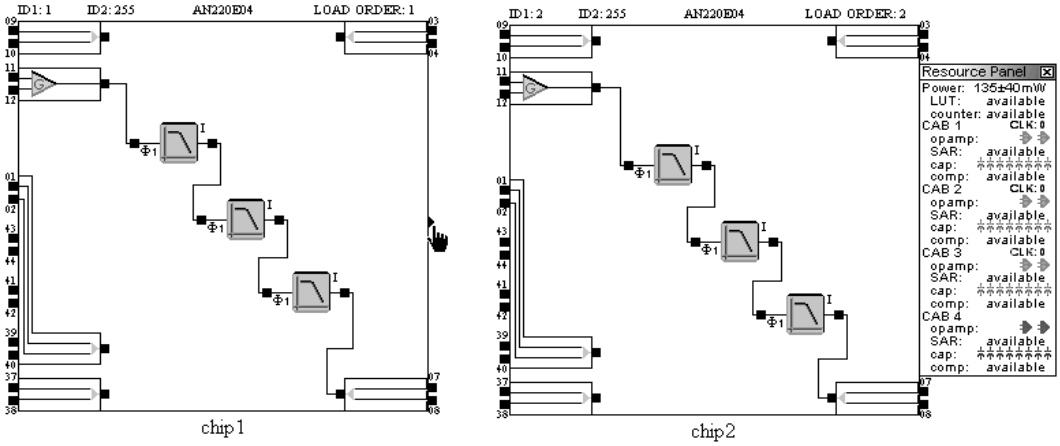


Figure 18 – Left Click on the Triangle to bring up the Power Estimate Resource Panel

4.3 Very Low Power – Dynamic Sleep Mode

Section 7 covers the entire C Code generation topic in detail, but this particular feature warrants mention here. AnadigmDesigner®2 can automatically create C Code representing the entire design. The C Code includes a function to generate a configuration data set that will put the device into sleep mode. Typically, a host processor calls the GetSleepData function then reconfigures the device with the resulting configuration data, placing the AN220E40 into a very low power Dynamic Sleep mode.

In Dynamic Sleep mode all analog functions are turned off except the crystal oscillator. In Dynamic Sleep, the device will consume power in the range of only 1 mW to 10 mW.

This function returns a pointer to a volatile memory block. If you wish to retain the data you must copy it into a separate buffer.

More detail on this low power sleep enabling function can be found in Section 7.4.9, GetSleepData.

5 Configuration File Formats

5.1 The Serial Port Data Stream

Selecting the *Configure* → *Write configuration data to Serial Port* pull down menu item (or using the “Ctrl-w” shortcut), causes AnadigmDesigner®2 to download configuration data to an attached target via a serial port. The serial port used is established under the *Settings* → *Preferences* (*Serial Port* tab) pull down menu.

5.2 S-Record Format

A second industry standard format for transferring programming information between tools is the S-Record format. The file is easy to understand once you have been introduced to it. The first and last couple of lines of a S-Record format file for an AN10E40 are shown below:

```
S224000000D5B72200100105C000400000000000000000000000000000000000000000000000017
S22400002000000000000000000FF000000000000000000000000000000000000000000000000BC
...
S2170002400000000000000000000000000000000000002A7C
S9030000FC
```

A field by field dissection of the first S-Record within a configuration file is given below:

```
S224000000D5B72200100105C000400000000000000000000000000000000000000000000000017
S2 – tells downstream tools that this record's address field is 24 bits in length
S1 – in this location would instead indicate 16 bit of address
 24 – tells downstream tools that 36 (hex 24) bytes follow.
  000000 – tells downstream tools that the starting address for this data
       is 0x000000.
     D5 – the first byte of configuration data, the Sync Header is always D5
       B7220010 – for Anadigm® FPAAs, these 4 bytes represent a unique device
            type identifier.
                 This is the end of S-Record check byte – 17
```

A field by field dissection of the last S-Record within a configuration file is given below:

```
S9030000FC
S9 – tells downstream tools that this is the termination record for the file.
 03 – byte count for remainder of record
  0000 – dummy data
    FC – checkbyte
```

5.3 ASCII Hex File Format

ASCII Hex File Format is nothing more than raw configuration data bytes represented as ASCII Hex pairs, with a CR LF sequence suffixing each byte pair.

```
00(CR)(LF)
00(CR)(LF)
00(CR)(LF)
00(CR)(LF)
00(CR)(LF)
D5(CR)(LF)
B7(CR)(LF)
22(CR)(LF)
00(CR)(LF)
. . .
```

The first 5 bytes are dummy clocking data. The next byte is the Sync byte. The following 4 bytes comprise the 32 bit JTAG ID code for first revision AN220E04.

5.4 Binary File Format

The binary file format contains the configuration data with no ASCII Hex conversion.

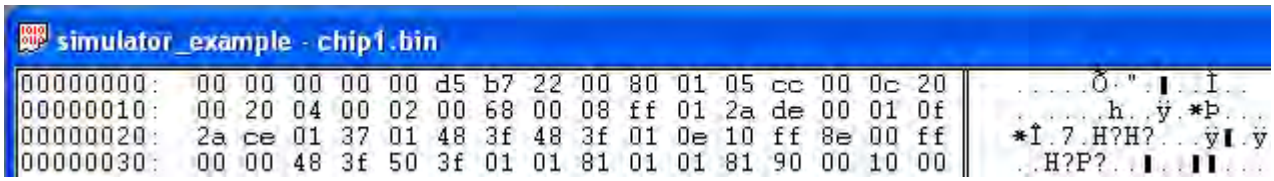


Figure 19 – BInary File Contents

In the figure above, a binary file viewer was used to examine the contents of a .bin file. Comparing this view with the .ahf file above in Section 5.3, ASCII Hex File Format, an easy correlation between the 5 all-zero dummy prefix bytes and the 4 bytes of the JTAG ID can be established between the two files.

5.5 Why Reversed?

Below are the first lines of two S-Record configuration files for the same design. The first line is from a "normal" S-Record file, the second from a reversed. The first byte of device configuration data for each record is bolded.

```
S224000000D5B72200100105C0004000000000000000000000000000000000000000000000017
S224000000ABED44000880A00300020000000000000000000000000000000000000000000000D2
```

The binary representations of the first data bytes "D5" and "AB" are:

```
D5 = 11010101
AB = 10101011
```

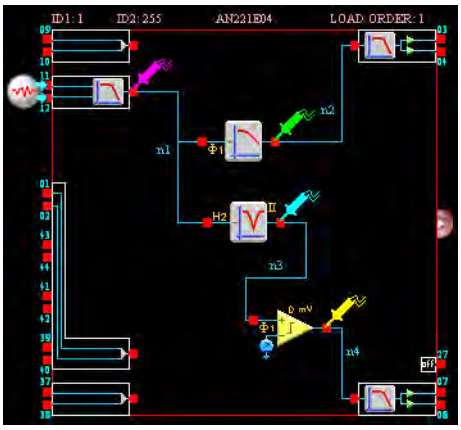
You can see that the bits within each data byte of the second (reversed) file are ordered opposite the first. The reason this is done is that data comes out of a serial EPROM in the opposite order in which it was programmed in. This quirky behavior of serial EPROMs has been around since their market introduction and has simply stuck. Data must be shifted into the FPAA most significant bit first.

If you intend to use serial EPROMs for booting an Anadigm FPAA, and your PROM programmer does not take this "first in last out" behavior into account (most do not), then you will likely want to use the "reversed" version of either of .ahf or .ms2 configuration file formats.

6 Functional Simulator

AnadigmDesigner®2 includes a time domain functional simulator which provides a convenient way to assess your circuit's behavior without the need for a lab set-up. A lab rig provides for more precise circuit evaluation under all conditions, but you may not always have a test bench at your disposal. For the casual investigator then, this functional simulator is a convenient alternative.

6.1 Simulator Overview



The simulator's user interface is intuitive and easily learned. Most of the steps are the same that you would take while bench testing any circuit:

- Wire up your circuit
- Attach and set up your signal generator(s)
- Attach Oscilloscope probes
- Set up the simulation parameters
- Launch the simulator

A screen shot of a partially designed circuit ready for simulation is shown here. A single generator drives both High Pass and Band Stop filters. The output of the Band Stop filter is routed to a two input comparator. The second input to the comparator is driven by a user defined voltage. The input waveform (an arbitrary type - swept sine of 1V peak amplitude) is applied to an Input Cell programmed for single ended operation. Four probes were placed to monitor the generator's output, the outputs of each of the filters, and the output of the comparator. F5 is the keyboard shortcut to launch the simulation. As the simulation completes, the Oscilloscope window pops up presenting your simulation results for review.

Figure 20 – Oscilloscope Display. High Pass, Swept Input, Band Stop & Comparator signals.

6.2 Simulator Performance

A CAM's simulation equations are contained as text strings within each CAM's .cam library file. For CAM's included with a particular software release, the simulation equations also reside in a compiled form within the executable files of the system. The compiled equations are used whenever available and greatly increase the speed of the simulator. If a compiled simulation model is not available (e.g. a user defined CAM, or a CAM created after a software release) then the simulator will evaluate the simulation equations within the .cam file using a run time interpreter; simulation speed will be noticeably slower.

The current simulation algorithms are not designed to handle zero-delay loops in your circuit design. Zero-delay loops are loops containing CAMs that have “zero-delay” characteristics i.e., the output immediately reflects the input with no intervening clock step.

The AnadigmDesigner®2 functional simulator is designed for use only with circuits in which all CAMs along an analog signal path use the same clock. While it is possible for the user to mix clocks along a signal path, the simulation results may not be correct for such a circuit. Anadigm® does not recommend using the functional simulator for mixed clock signal paths.

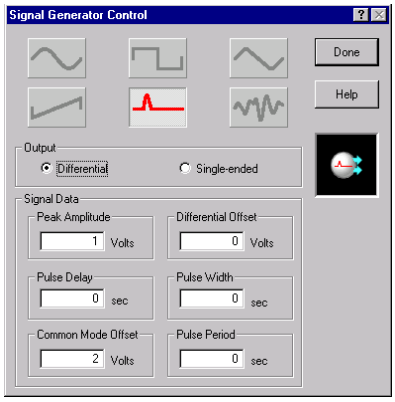
The simulator provides an expedient and convenient environment in which you can acquaint yourself with the unique Anadigm® analog design environment.

6.3 Signal Generators

Up to eight different signal generators may be placed on your design. Each generator used, must be attached to an Input Cell’s input pin (or pin pair). When the Input Cell is configured as a differential input, use a signal generator with its output also configured as differential.

Either the “g” keystroke shortcut, the "sine wave" toolbar shortcut button or the pull down menu selection *Simulate* → *Create Signal Generator* attaches a generator icon to your mouse cursor. Drag the icon to the desired Input Cell’s input pin (or pin pair) and left-click to drop it in place. Right-click over the generator icon to bring up its parameter dialogue window.

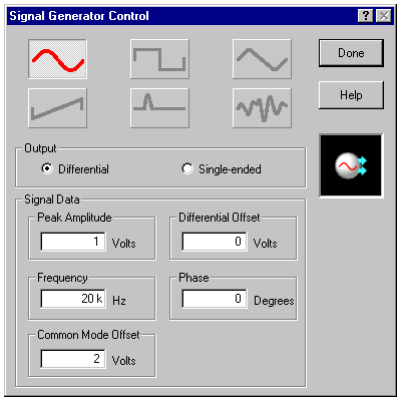
6.3.1 Pulse Function



The Pulse function allows construction of a single pulse (set Pulse Period to 0) or a pulse train. A control unique to the Pulse function is Pulse Delay. This control allows application of the pulse to be held off from the simulation for a programmable number of seconds.

All of the generators may be set up to drive either differential or single ended. When set as differential drivers, additional controls are available to set common mode and differential offset voltages.

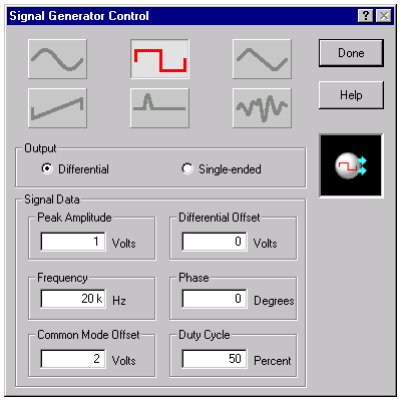
6.3.2 Sine, Sawtooth and Triangle Wave Generators



Three of the signal generators: Sine, Ramp and Triangle, share a common set of user adjustable parameters. Amplitude and Frequency are self explanatory.

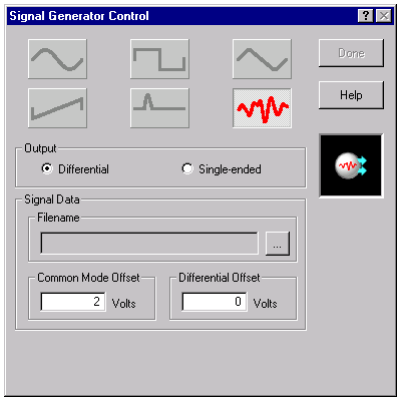
Each of these signals is symmetric about VMR. (Please recall that all internal analog processing is done with respect to Voltage Main Reference, VMR.) Setting a positive Voltage Offset shifts the entire applied signal up with respect to VMR. The default offset of 0 Volts means that these signals are symmetric about VMR (2.0 V w.r.t. AVSS).

6.3.3 Square Wave Generator



The square wave signal generator has all the same parameters as just discussed above, with the addition of Duty Cycle. The duty cycle may be adjusted to provide a waveform to meet whatever unique signal requirements you might have

6.3.4 Signal Data File Generator



The data file waveform generator allows you to insert a waveform of your own design. With this feature, there is not a waveform in the world you could dream up that can't be handled by the simulator.

The file browser dialog allows you to specify a file of type: .wav, .csv or .txt.

The .csv and .txt File Format

Each record of the ASCII .csv or .txt file contains a time and amplitude data pair. The simulator "connects the dots" to assemble a piecewise linear wave form. The first field of each record is a time value (in seconds). The second field is an amplitude value (in volts). A space or tab character may separate the two fields of each record in a .txt file. A comma is the separating delimiter for .csv files.

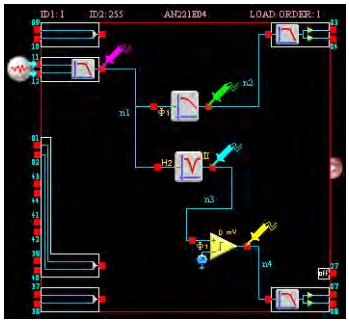
A spread sheet program is a handy utility to use when generating piecewise linear signal data files.

Values may be expressed in either decimal or scientific notation format. When expressing time values in scientific notation format, be sure to use as many decimal places as necessary to establish each time value as unique. The simulator requires that a piecewise linear waveform be presented with monotonically increasing time values. Too few digits in the time representation and the simulator will recognize two adjacent time stamps to have the same value. An error will be reported.

The .wav File Format

Experienced PC users are familiar with .wav format sound files. The waveform sample rate is encoded in the .wav file, so there is no need for a user entry. Only PCM encoding is recognized. Only the first channel of information (typically Left) is processed.

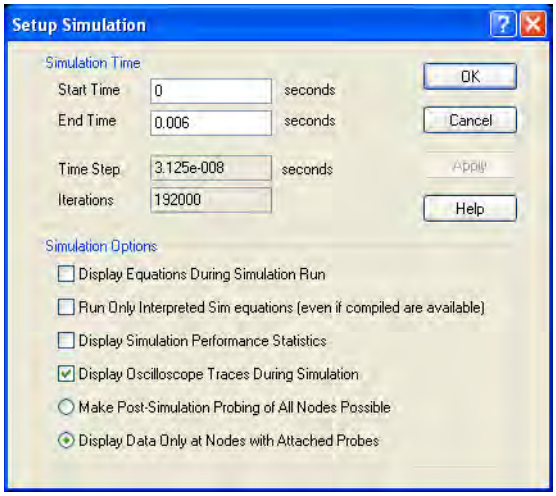
6.4 Oscilloscope Probes



Up to four different Oscilloscope Probes may be placed on your design. They may be placed on any active wire, CAM input or output, or any Input or Output Cell.

Either the “p” keystroke shortcut, the "probe" toolbar shortcut button or the pull down menu selection *Simulate* → *Create Oscilloscope Probe* attaches a probe icon to your mouse cursor. Drag it to the desired location and left-click to drop it in place. The color coded probes match the waveform colors in the Oscilloscope Window (see below).

6.5 Simulation Set Up and Run



The pull down menu *Simulate* → *Setup Simulation* allows you to establish the start and stop time of your simulation run. The units are in seconds.

The time step parameter is automatically calculated and inserted for you as 1/2 the period of the fastest System Clock frequency (fsys). Iterations refers to the number of simulation steps. Time step and number of iterations is only adjusted after selecting OK or Apply.

Launching the simulation is accomplished using the pull down menu selection *Simulate* → *Begin Simulation* or simply pressing the F5 shortcut key. "Display Equations During Simulation Run" is something similar to single stepping through the simulation. This feature is typically only invoked by advanced developers. The simulation performance statistics can be used to show how long your simulation is taking, how often each particular CAM is running, and whether any of your CAMs are running interpreted simulation code. All node data can be saved for post simulation exploration.

6.6 Oscilloscope Window - Viewing Simulation Results

All the simulation results are now available for graphical analysis. Each of the waveforms is color keyed to each of the scope probes dropped in the circuit. The display of each waveform can be toggled on and off by pressing its associated "Channel" button. The signal amplitude display scale can be adjusting using the associated "Volts Per Division" control. A "Position" control is also available that allows you to separate the waveforms vertically for easier viewing.

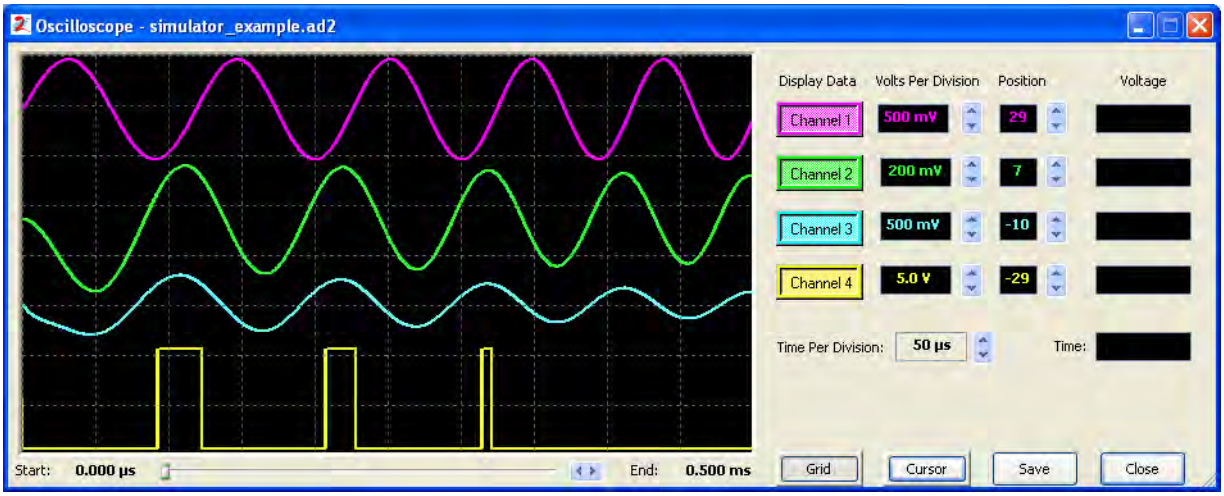


Figure 21 – Oscilloscope Display, Zoomed In to Show Simulation Details

A single vertical cursor is available to drag left and right within the waveform window. There are four "Voltage" displays, one for each waveform and a single "Time" indicator. As the cursor is moved horizontally, the Voltage and Time indicators all update to reflect that moment's simulation results. Moving the waveforms up and down using the "Position" control does not affect the Voltage readings.

The "Time Per Division" control allows the data in the waveform window to be scaled horizontally. Below the waveform window, there is a horizontal slider that allows you to pan through time if the horizontal scale is such that less than all of the waveform is presented. The Start and End times of the displayed waveforms are also presented. The additional controls toggle the cursor display and waveform grid display on and off, or Close the window.

This simulation uses arbitrary waveform signal generator. A swept sine data file was constructed and applied to the generator (Channel 1).

Referring to both Figures 20 and 21: Channel 3 shows the output of the band stop filter doing its job nicely, centered about 14 kHz. The amplitude of the output of the Low Pass filter assigned to Channel 2 can be seen to decrease in even the extremely narrow band displayed in Figure 21.

Channel 4 shows the output of the comparator. The comparator's positive input is driven by the output of the Band Stop filter (Channel 3) and its negative input driven by an internally generated reference voltage. The comparator output pulses narrow as its input waveform amplitude decreases. Eventually, the amplitude is so attenuated by the Band Stop filter that the comparator can find no valid trip point.

In Figure 22 you can see that the user defined input waveform definition ended at about 4.8 mS, but the simulation was run for a bit longer. This instructive screen shot demonstrates the vertical cursor. Positioned well after the input waveform data ends, you can see the current cursor time is 5.183 mS and that moment's instantaneous voltages for each of the channels. Its also instructive to review the DC response of the circuit at this point in time. The input waveform data (Channel 1) happened to end with an amplitude of -144.64 mV. Being a piece wise linear arbitrary waveform, the simulator holds the final value in place.

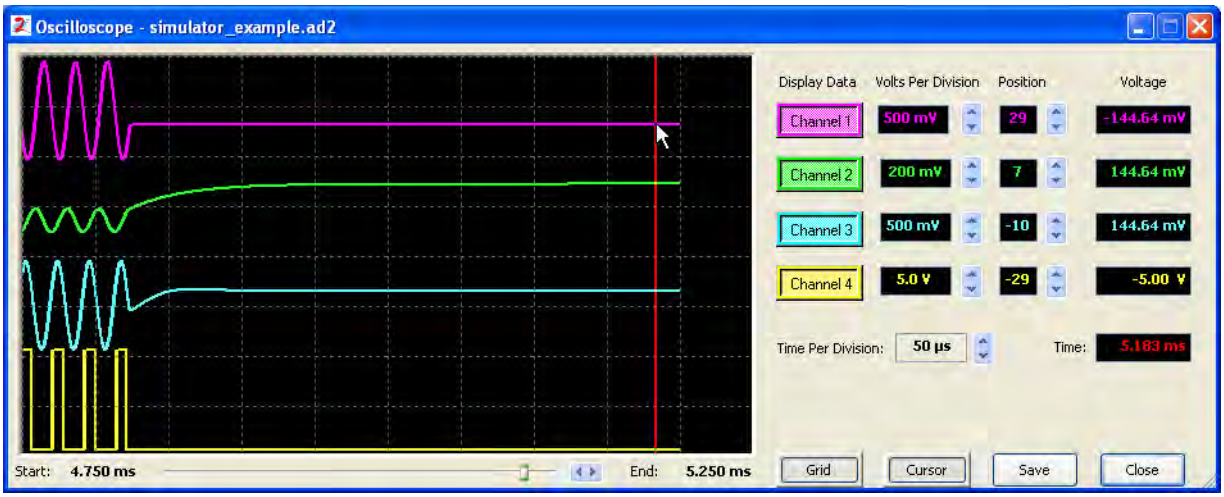


Figure 22 – End of Simulation - Detailing Circuit Behavior as Stimulus goes to DC

The Band Stop filter (Channel 3) was centered at 14 kHz. Obviously then DC is in one of the pass bands. With a gain of 1, you can see that the input voltage scaled by exactly 1 to yield a -144.64 mV output. Likewise for the Low Pass. The comparator output on Channel 4 is railed low.

7 Hosted Configuration

The preceding chapters of this manual dealt primarily with the use of AnadigmDesigner®2 to generate static configurations for one or more devices. The simplest use model involves designing the analog circuit, saving the associated configuration data file, and programing a serial EPROM with that data file. The FPAA can then configure itself on power-up from that EPROM. While statically configured programmable analog provides system designers with a powerful new resource, the true advantages of programmable analog can not be fully appreciated until the FPAA is hosted by a companion processor and its configuration is conducted as a response to changing system requirements.

This chapter discusses the following hosted operations: Algorithmic Dynamic Configuration, State Driven Dynamic Configuration, and static configuration. It is assumed that the reader is familiar with the use of AnadigmDesigner®2 for static configuration design, embedded system software design using C language, and the FPAA device features including the details of the configuration interface.

7.1 Hosted Operation

All of the Anadigm® FPAAs include a flexible configuration interface that is easily connected to companion microprocessor. The AN120E04 and AN121E04 devices are best suited to systems where the FPAA will normally be configured at power-up or after system reset. The configuration interfaces of the AN220E04, AN221E04 and AN221E02 devices include special functionality which allows for reconfiguration data to be loaded into the device on-the-fly without the need to reset the device, Dynamic Configuration..

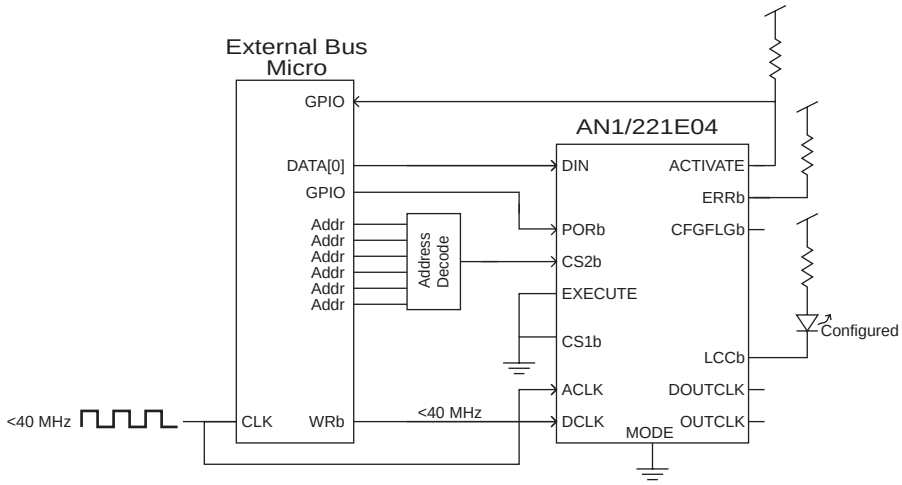


Figure 23 – A Typical Host Connection to an AN120E04 or an AN220E04

7.1.1 Understanding the Difference Between Static and Dynamic Configuration

AnadigmDesigner®2 creates several different types of data and C code source files in support of static and dynamic configuration.

Static Configuration

Using the basic features of AnadigmDesigner®2, it is easy to create circuits and their associated configuration data sets for both AN12xE04 and AN22xE0x devices. Using a host processor, a static configuration data set can be simply transferred to the FPAA. It is of course possible for a host processor to store several complete configuration data sets and completely reconfigure the attached FPAA(s) whenever desired. (For an AN120E04 and AN121E04 devices, a reset sequence is required prior to reconfiguration. The AN220E04, AN221E04 and AN221E02 devices on the other hand, allow complete or partial new configuration data to be loaded on-the-fly without the need to reset the device.) The

hosted transfer of static configuration data sets is fine for applications where the required analog circuit behavior is known beforehand.

For applications in which analog behavior must be adjusted on-the-fly, C Code generation features of AnadigmDesigner®2 offer convenient methods for a host processor to actually *create* and access the data that is required to configure and reconfigure the FPAA.

Dynamic Configuration

The power of C Code generation is enabled by the AN220E04's, AN221E04's and AN221E02's ability to be *dynamically* reconfigured (not available on the AN120E04 or AN121E04). Dynamic reconfiguration means that parts of the analog circuit or an entirely new analog circuit can be downloaded to the FPAA on-the-fly without the need to reset the FPAA. The new configuration is activated in a single clock cycle. C Code generation ensures that only the minimum amount of configuration data necessary to execute the change is generated, making the reconfigurations as small and as fast as possible. Using these features, baseline analog functions can be downloaded into the FPAA and then updated on-the-fly whenever changing application requirements warrant (Algorithmic Dynamic Configuration) or pre-compiled circuits topologies can be downloaded at will (State Driven Dynamic Configuration).

7.2 Algorithmic Dynamic Configuration
(AN220E04,AN221E04 & AN221E02 only)

Algorithmic Dynamic Configuration refers to the use of C Code by a companion host processor to actually create and download reconfiguration data for the attached FPAA on-the-fly in response to changing analog signal processing requirements. Algorithmic Dynamic Configuration features support the initial Primary Configuration of the FPAA and subsequently allow only the programmable parameters of the analog circuit to be adjusted; the circuit topology is static. One example application is an adjustable filter where the number and type of filter stages is fixed, but the corner frequency, Q and gain are to be adjusted on-the-fly.

Each CAM has associated C Code functions that are designed to manipulate its programmable parameters. Using AnadigmDesigner®2 Algorithmic Dynamic Configuration features, C Code files containing these functions can be generated. The C Code that is generated contains important information about the low-level components of the circuit. The C Code functions use this information to dynamically generate reconfiguration data for the chip as they are called. As each function is called, the data it produces to reconfigure the chip is appended to a data buffer. The functions do not directly reconfigure the chip, but rather build the data that is required to reconfigure the chip. When it is time to send the data to the chip, C Code API functions, such as GetReconfigData are called to retrieve the reconfiguration data buffer. It is then the responsibility of the host processor to transfer this data into the FPAA.

The following subsections walks through a prototypical design illustrating the use C Code generation for Algorithmic Dynamic Configuration.

7.2.1 Design the Circuit

The example is a simple audio filter. The circuit puts the low frequencies on the left channel and the high frequencies on the right channel. The design is not as compact or efficient as it could be, but uses a variety of CAMs suitable for this tutorial.

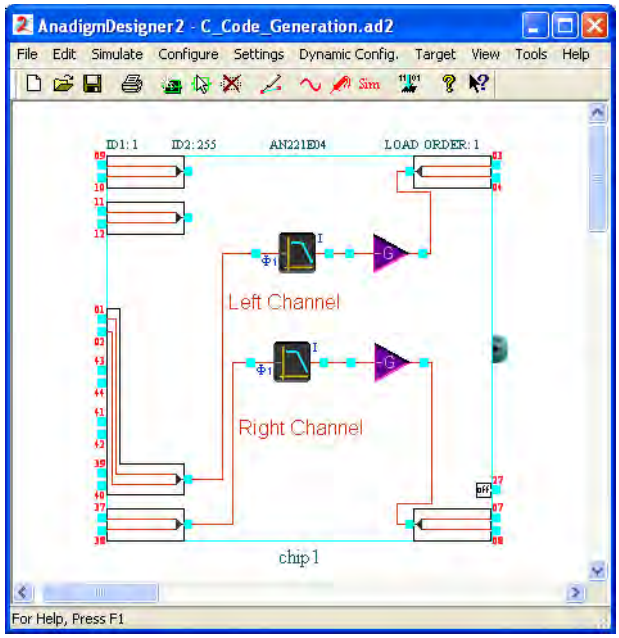


Figure 24 – Example Circuit for C Code Generation

7.2.2 Choosing Functions to Generate

After completing the design of a circuit, a decision must be made as to which parts of the circuit will need to be dynamically reconfigured. With this information in hand, C Code functions from each CAM in the circuit should be generated into the C Code.

7.2.3 Choose the CAM C Code Functions

For this design example, assume the need to dynamically reconfigure the high-pass filter on the right channel, and the gain stage on the left channel. By default all CAM C Code functions are on. In order to generate compact C Code, turn off the function generation for the CAMs that will not be dynamically reconfigured.

To bring up the C Code functions for a particular CAM:

1. Right-click on the CAM and choose “C Code Functions” from the pop-up menu. Or,
2. Right-click on the CAM and chose “CAM Settings” then click on the “C Code…” button.

Using one of these methods, open the “C Code Functions Window” for the gain stage on the right channel. The CAM instance name **GainInv_Right** appears in the title bar as a convenient reminder. There is no need to dynamically reconfigure this CAM, so deselect all of the functions it offers.

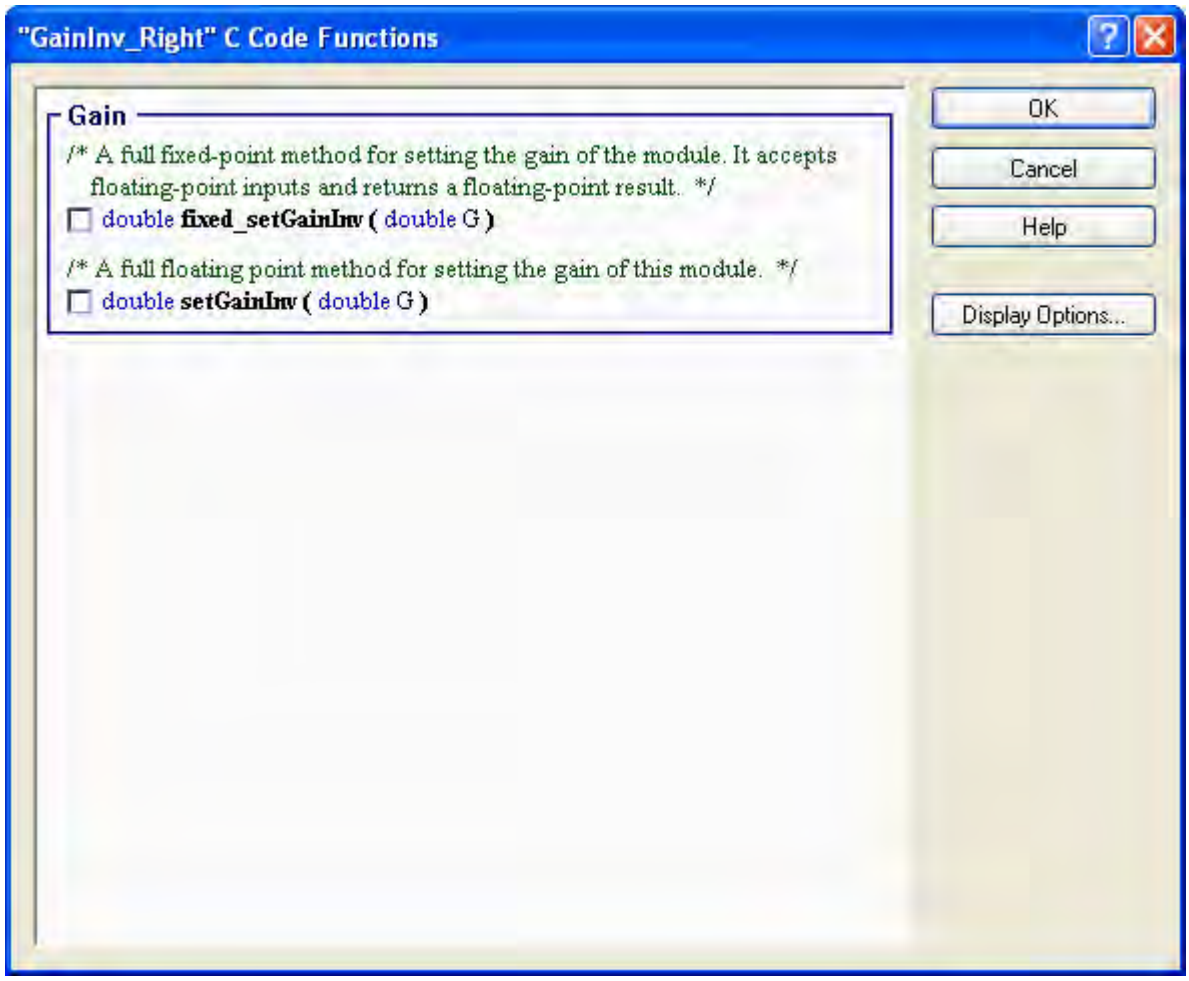


Figure 25 – Disabling C Code Generation for Particular Functions of a CAM Instance – GainInv_Right

Next, turn off all of the functions for low-pass filter, for the left channel. This time use the “CAM C Code Functions Window” to turn off the functions. To open this window, choose *Dynamic Config → Algorithmic Method…* and left-click the “CAM Functions...” button. This window will show us the C Code functions of every CAM in the circuit organized in several different ways.

As it first opens, the window presents a list of all the available CAM Types in the Selection Pane (right side). The window shows that all functions for the CAM Type **ANx21 Standard\GainInv** are turned off. This means that no C Code functions will be generated for any CAM in the circuit that is this type. There is only one instance (**GainInv_Right)** of this CAM Type (GainInv) and all of its functions are turned off.

Notice that the multiplexed input cell (**ADdata\ANx21_INMUX**) has C Code functions and they are turned on. There is no requirement to program the multiplexed input, so uncheck the box to turn off C Code generation for all of its functions.

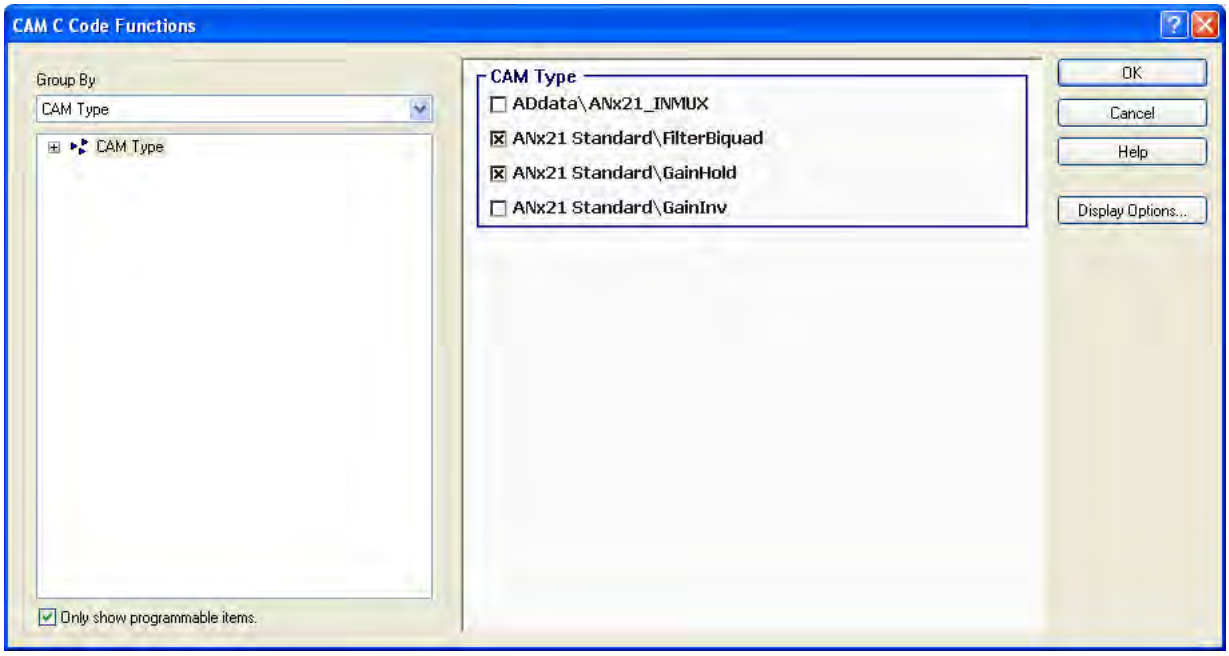


Figure 26 – Disabling C Code Generation by CAM Type – ADdata\ANx21_INMUX

Next, turn off the functions for the low-pass filter. The window shown in Figure 20 cannot be used to do this. If the box associated with CAM Type **ANx21 Standard\FilterBiquad** were unchecked, it would turn off the C Code functions for *all* CAMs of that type, meaning it would also turn off the functions of the high-pass filter. To achieve the desired goal of turning off C Code generation for a particular instance, click on the “Group By” selection box and choose “Instance Name” from the list. The selection pane on the right will show a list of all the CAM instances in the circuit. Notice that the **GainInv_Right** and **InputCell4** (the multiplexed input)

already have all of their C Code turned off. To turn off the functions for the low-pass filter on the left channel, click on (uncheck) the box next to **Filter_Left**.

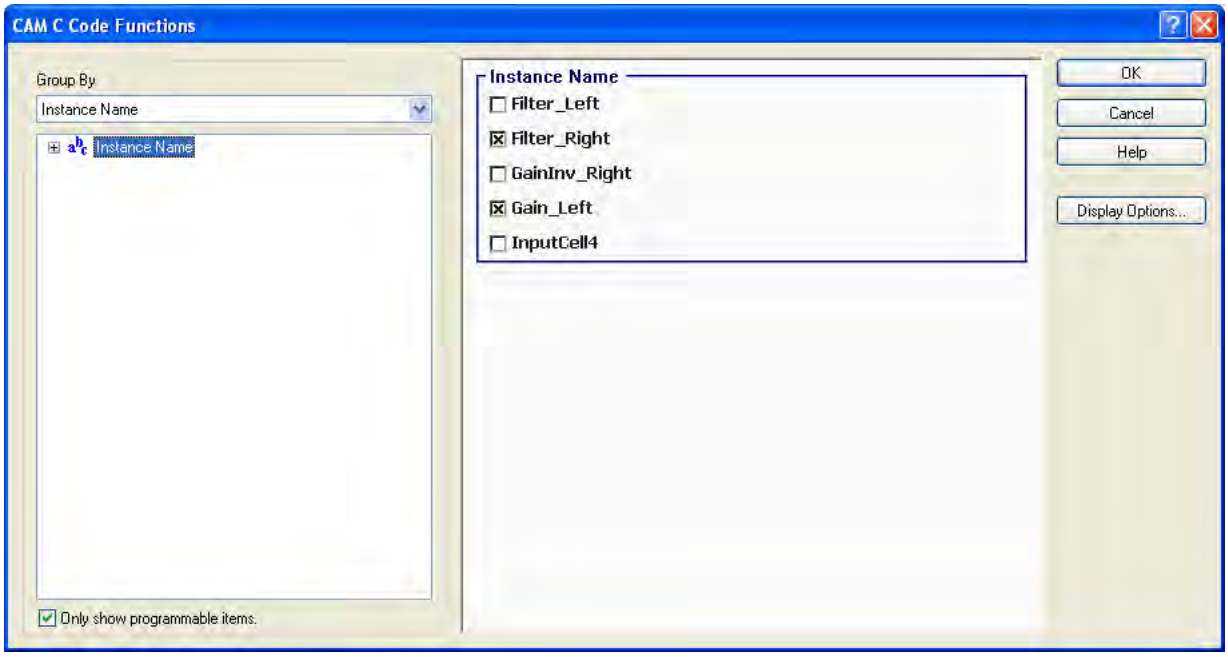


Figure 27 – Disabling C Code Generation by CAM Instance Name – Filter_Left

Right now, all of the C Code functions for the **Filter_Right** and the **Gain_Left** will be generated. We may not need every function they have to offer, so we will probe a little further to suppress the C Code generation of the unwanted functions.

By clicking on the '+' next to Instance Name in the CAM Explorer (on the left) we can expand the node to list all of the CAM instance names. First select **Filter_Right**. The Selection Pane (right side) then shows all of the available C Code functions for that CAM instance. For this example, there is no requirement to explicitly

change capacitor values using C Code, so disable the function `SetBQHighPassCaps`. Next select **Gain_Left** in the CAM Explorer. Turn off the `fixed_setGainHold` function by unchecking it in the Selection Pane.

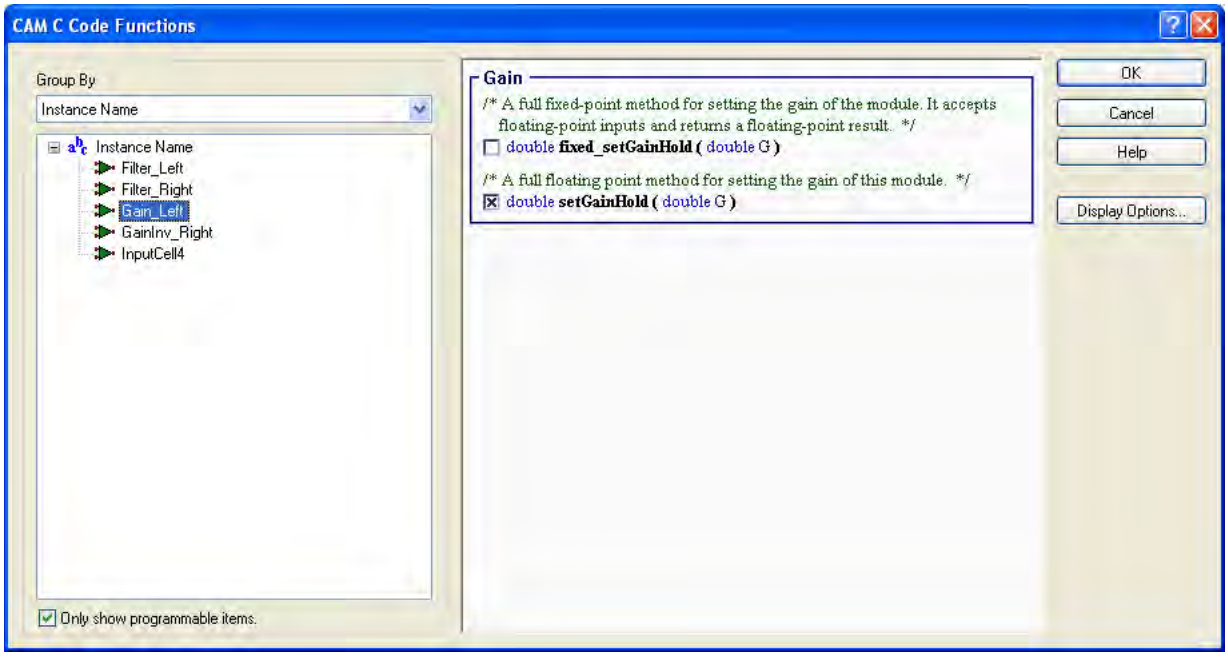


Figure 28 – Disabling C Code Generation of a Particular Function of a Particular Instance

Now both the **Gain_Left** and **Filter_Right** CAMs each have one C Code function turned on and one C Code function turned off. Now select the root “Instance Name” in the CAM Explorer, using the “Group By” drop down menu. In the Selection Pane, notice that the check in box next to these two CAMs is gray, indicating that some, but not all of the C Code functions for these CAMs have been disabled.

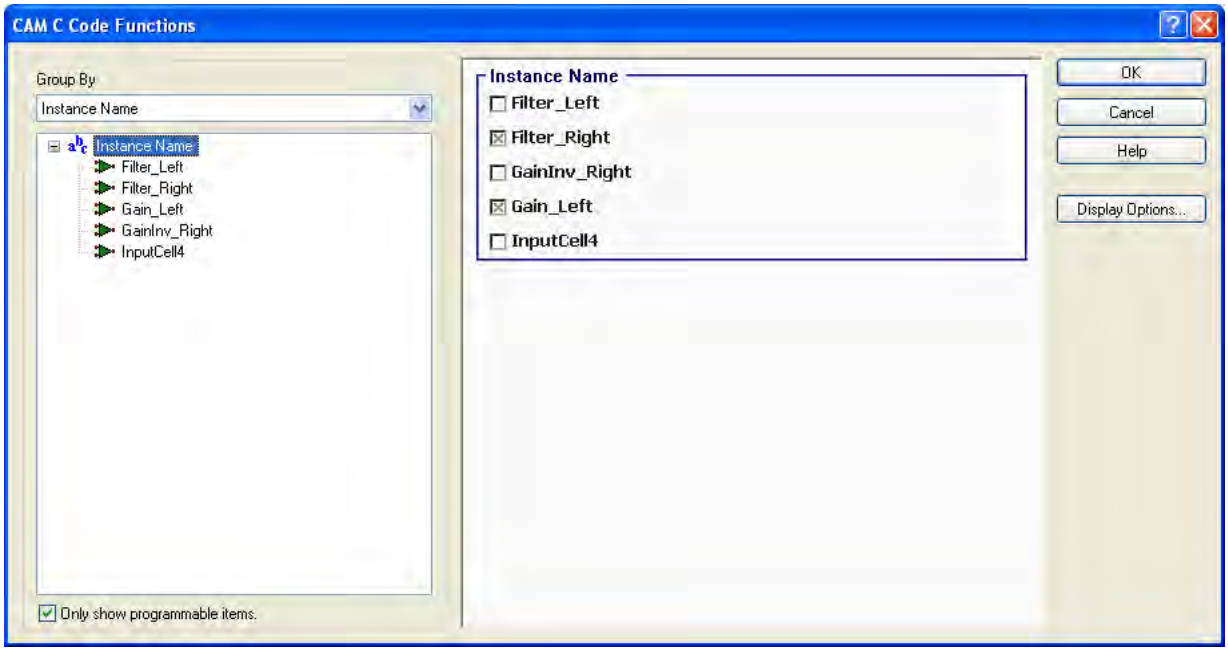


Figure 29 – Composite Status of C Code Generation for all CAM Instances

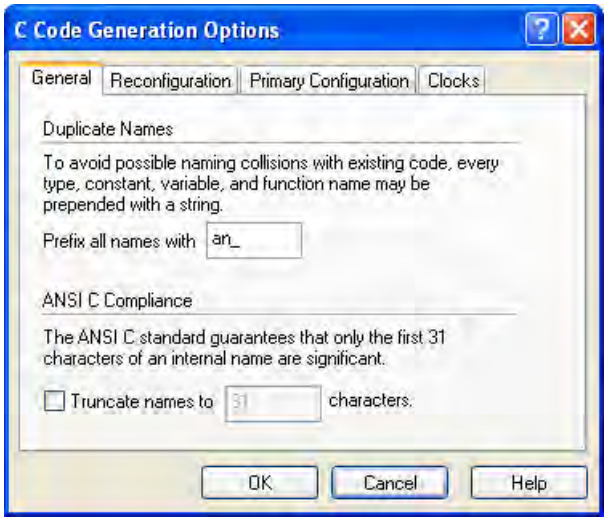
7.2.4 Generating the Code

This section will demonstrate the process for generating the C Code files. It is assumed that all of the steps in the previous section have been completed.

Setting the Generation Options

The “C Code Generation Options Window” can be opened by:

1. *Choose C Code → Generation Options…* from the AnadigmDesigner®2 main menu, or
2. *Choose C Code → Generate…* from the AnadigmDesigner®2 main menu, then click on the Generation Options… button.



General Tab

Stay with the default settings on the General Tab. To avoid name collisions with any existing code, all functions and variables generated can be prefixed with a string. By default, this string is "`an_`". So the `setGainHold` function will be generated as `an_setGainHold`. AnadigmDesigner®2 maintains ANSI C compliance by truncating all names to 31 characters. The generation engine will do the truncation intelligently, and will not allow two names to collide.

Reconfiguration Tab

The Reconfiguration Tab allows control of allocation of the memory used to store the reconfiguration data that is dynamically created in the C Code. As each CAM C Code function is called by the host program, configuration data is added to the reconfiguration data buffer. The size of this buffer is controlled in this tab. Check the Enable automatic growth, to avoid buffer overrun.

Primary Configuration Tab

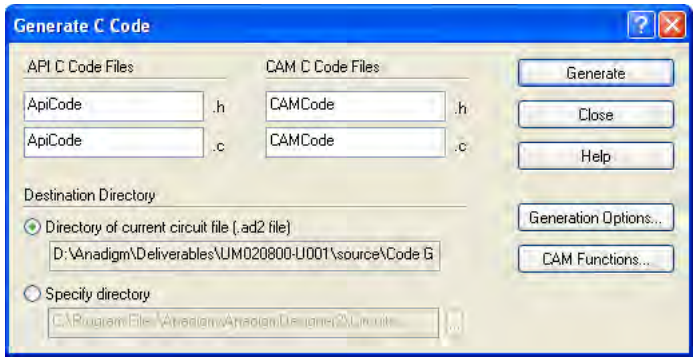
This tab selects the devices that will be configured initially using C Code. Leave the box next to the device name checked.

Clocks Tab

The clocks tab allows selection of the devices for which dynamic reconfiguration of clock divisors is required. Leave this unchecked for now

Creating the C Code Files

To create the C Code files, open the “Generate C Code Window”. Do this by choosing *C Code → Generate…* from the AnadigmDesigner®2 main menu. This window allows us to choose the names of the files that get generated, and specify the directory where they will be created. Keep the default file names and generate them to the same directory as the circuit file.



Everything is now ready to go. Press the “Generate” button to create the files. A prompt will appear that asks for confirmation to overwrite the files if they already exist . After the files are generated, AnadigmDesigner®2 indicates that C Code was generated with success.

The functions are now created and host programming using the C Code API can begin.

7.3 Controls Reference - Algorithmic Dynamic Configuration

7.3.1 CAM C Code Functions Window

Right-click over a CAM and select “C Code Functions”. This window displays the available C Code functions for that particular CAM.

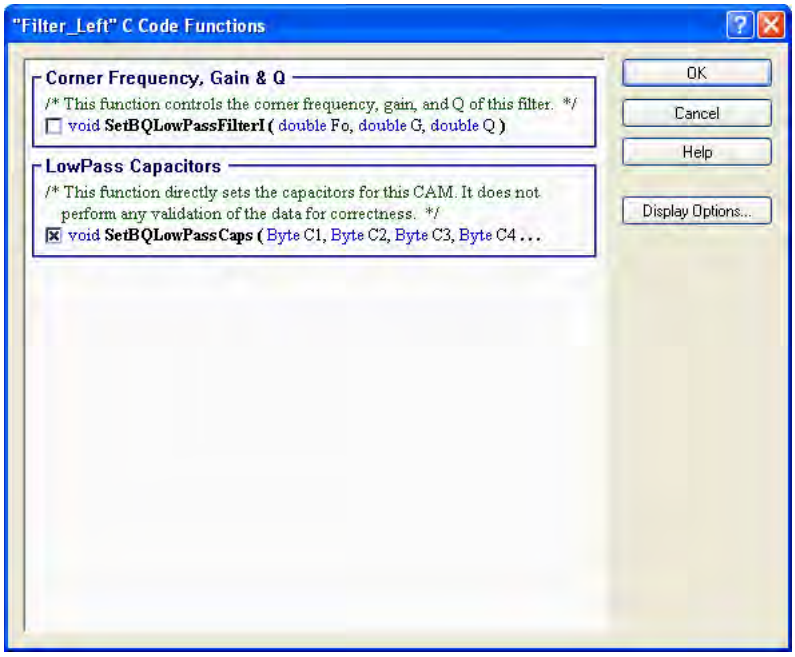


Figure 30 – C Code Functions Window for Instance Filter_Left

Function Display

Only functions relevant to the current CAM options, as set in the Set CAM Parameters window, will be displayed. In the example shown above, the CAM is set up to be a low pass filter. Had it been setup to be a high pass filter, a different set of functions would be available.

Functions are grouped in sets, as shown by the “Corner Frequency, Gain & Q” set and the “LowPass Capacitors” set in the window above. These sets are defined by the CAM and group related functions.

Figure 30 shows function descriptions, function names in bold, parameter arguments, and highlights the ANSI C data types. The display style is configurable from the “C Code Display Options” window.

Code Generation

If a function is checked it will be included in the generated C Code. An ID representing the CAM instance will be created. This ID will be passed as the first parameter to the generated function to indicate which CAM instance to act on. If a function is not checked in this window it is still possible the function will be generated if there is another CAM instance of the same CAM type which has checked the function. However, even though the function is generated, it is not possible to use the function within the C Code for CAMs that did not check the function, as the necessary ID's and supporting code would not be generated.

7.3.2 Global C Code Functions Window

Select *Dynamic Config* → *Algorithmic Method…*, left-click the “CAM Functions...” button, and expand each level of the hierarchy. This view allows full control over what functions get generated for all CAM instances.

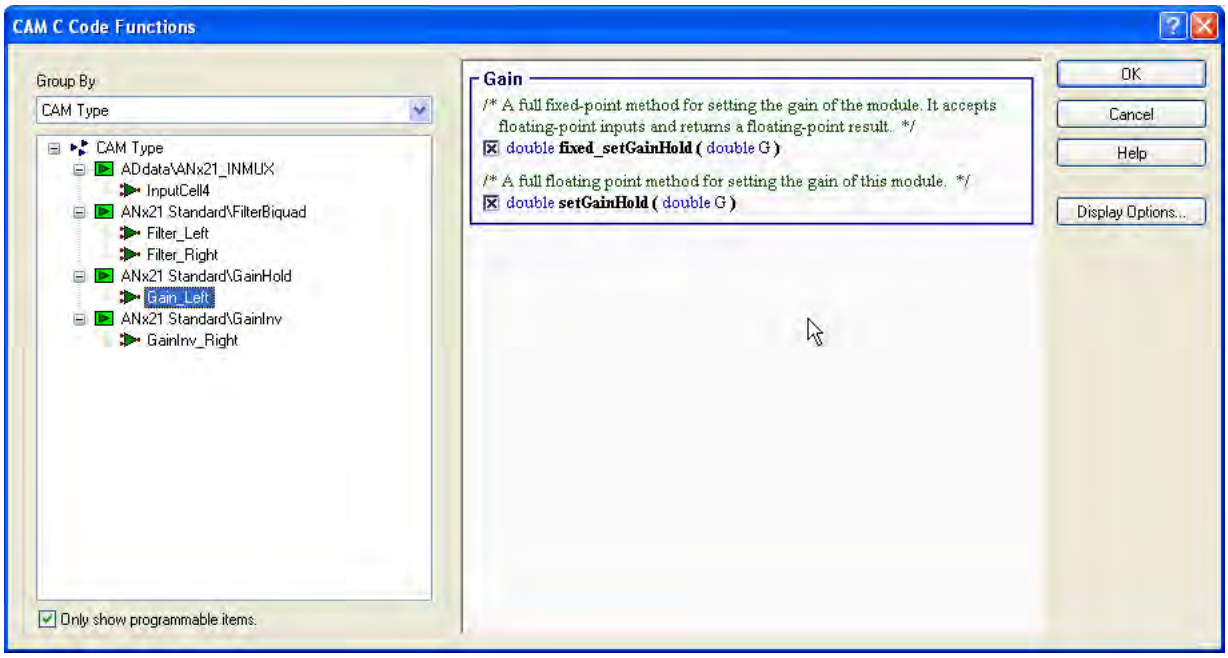


Figure 31 – Hierarchical View of all CAM Instances

Only Show Programmable Items

When checked (lower left of window), the window will only display CAMs which are programmable via C Code. If there are CAMs in the circuit, but none are C Code programmable, then no CAMs will be shown in this window. If the box is not checked, all CAMs in the circuit are listed in the window, regardless of their C Code programmability.

Selection Panel

The window on the right displays check boxes next to items that can be turned on and off. The items available in this window depend on what is selected in the tree on the left. A check means that C Code will be generated for that item.

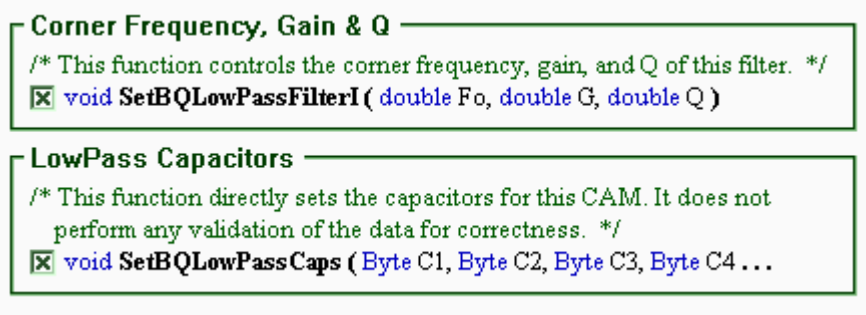
CAM Explorer

The window on the left is the CAM Explorer. Using it, it is possible to group CAMs in three different ways, as selected by the three choices in the Group By selection drop down menu.

Grouping by CAM Type

CAMs are sorted by their type, which is their library and unique CAM name within that library. Selecting the CAM Type root in the CAM Explorer gives a listing of all CAM Types, with check boxes, in the Selection Panel. If a box is checked, all functions for CAMs of that type are on. If it is not checked, all functions for CAMs of that type are off.

When selecting an actual CAM type in the CAM Explorer, such as **ANx21 Standard\Biquad**, the Selection Panel displays all functions available from all configurations of CAMs of that type. If a function is checked here, the function will be on for all CAMs of that type.



If a single CAM instance is selected in the CAM Explorer, such as **Filter_Left**, the Selection Panel is then identical to the CAM C Code Functions Window.

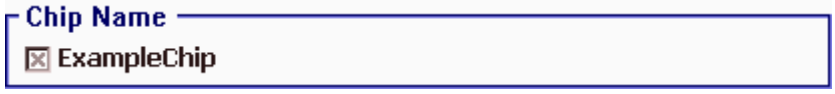
Grouping by Instance Name

CAMs are sorted alphabetically by instance name. Selecting the Instance Name root in the CAM Explorer causes the Selection Pane to present a list of CAM instances. If an instance is checked all of its C Code functions are on. If it is not checked, none of its C Code functions are on. It is possible for the check to be in an intermediate state if some of the functions are on, and others are off. This is signaled by a greyed out X.

If a single CAM instance is selected in the CAM Explorer, such as **GainInv_Left**, the Selection Panel is then identical to the CAM C Code Functions Window.

Grouping by Chip Name

CAMs are grouped according to the chip they are in. Selecting the Chip Name root in the CAM Explorer presents a list of chips with check boxes in the Selection Panel. If a box is checked, all functions for CAMs in that chip will be on. If a box is not checked, all functions for CAMs in that chip will be off. It is also possible for a check box to be in an intermediate state. This is the case if some of the functions in the CAMs in the chip are on, while others are off.



When a chip name is selected in the CAM Explorer then the Selection Pane will present a check box that is used to turn CAM functions for this chip on and off.

Below the chip name in the CAM Explorer are the further groupings of Instance Name and CAM Type. Selecting these items produces the same results as selecting the corresponding items when the CAM Explorer is being grouped by Instance Name or CAM Type. The difference is that the check boxes in the Selection Panel then only represent those things that are associated with the selected item in the CAM Explorer. For instance, if **ANx21 Standard\GainInv** CAM Type is selected under ExampleChip, removing the checks in the boxes for **ANx21 Standard\GainInv** will turn off all functions for only the **ANx21 Standard\GainInv** Type CAMs that are in ExampleChip.

In the following example, the C Code functions for the **ANx21 Standard\GainInv** CAMs in **ExampleChip** are turned off, but this will not affect the C Code functions for the **ANx21 Standard\GainInv** CAMs in **TestChip**.

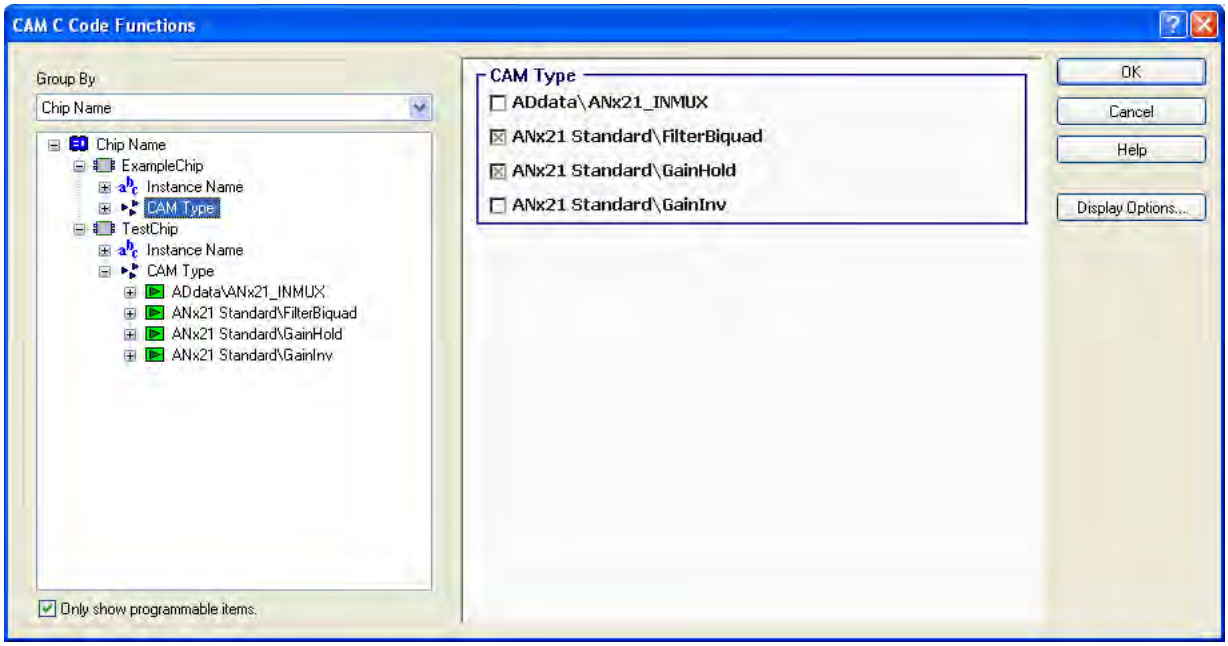
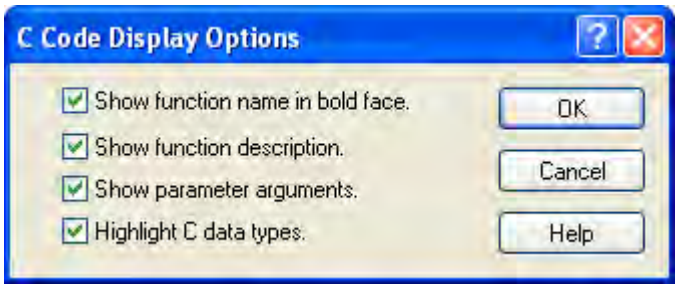


Figure 32 – Generating Code for Multiple Devices

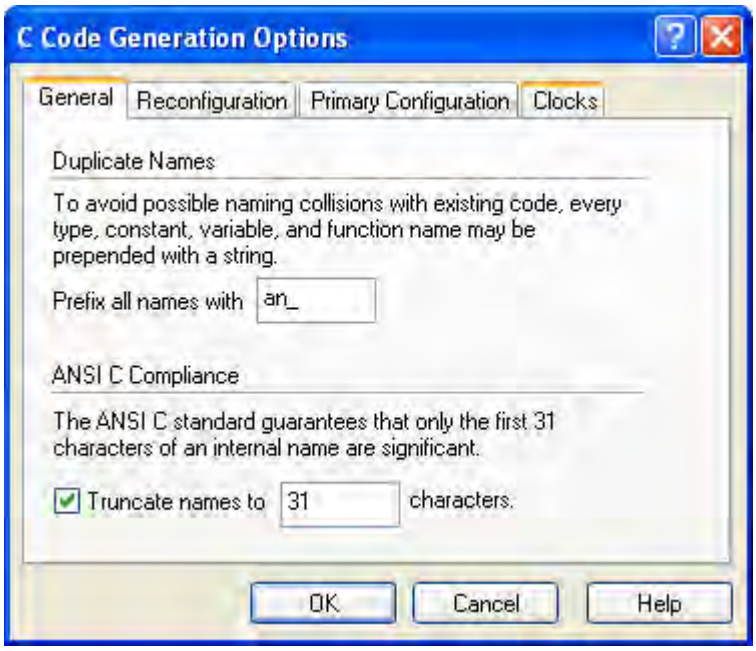
7.3.3 C Code Function Display Options

This window presents options that affect the display of the CAM C Code Functions Window and the Global C Code Functions Window. The options are self explanatory.



7.3.4 C Code Generation Options - General Tab

This tab deals with general output formatting, and does not affect the actual functionality of the code that is generated. The settings in this tab apply to all C Code files that are generated.



Duplicate Names

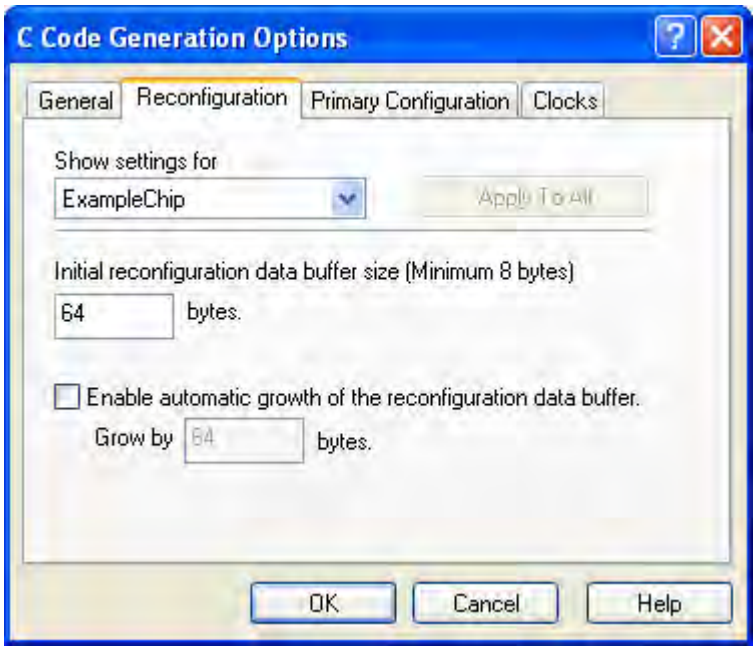
To avoid possible naming collisions with existing code, or simply to highlight variables and function names in the generated code when used with existing code, a prefix can be added to all generated types, constants, variables, and function names.

ANSI C Compliance

The ANSI C standard only guarantees that the first 31 characters of a name are unique. If there were two functions that were 33 characters and unique up to the last character, such as function MyCall…Data1() and MyCalll…Data2() then the compiler would actually create two functions with the internal names MyCall…Dat() and MyCall…Dat(). The linker would not be able to resolve this ambiguity, and the code would not compile. If the C Code is generated with name truncation enabled, the truncation is applied intelligently and no name collisions or linkage ambiguities will occur. In the previous example, the function names that would be generated are MyCall…D_1() and MyCall…D_2().

7.3.5 C Code Generation Options - Reconfiguration Tab

This tab is used to set reconfiguration options specific to each chip.



Using the Window

Every chip in the circuit that is reconfigurable will be listed in the “Show Settings for” drop down box. The settings in the lower half of the window apply to the chip selected in this box. The “Apply To All” button may be pressed to apply the settings for the currently selected chip to all reconfigurable chips in the circuit.

Data Buffer Size

When using the C Code API to build reconfiguration data, a separate memory block of reconfiguration data is maintained for each chip. Each chip has its own settings for the size of this block and what action the C Code should take if that size is not enough.

The Initial reconfiguration data buffer size is the number of bytes that will be initially allocated for the chip to store its reconfiguration data.

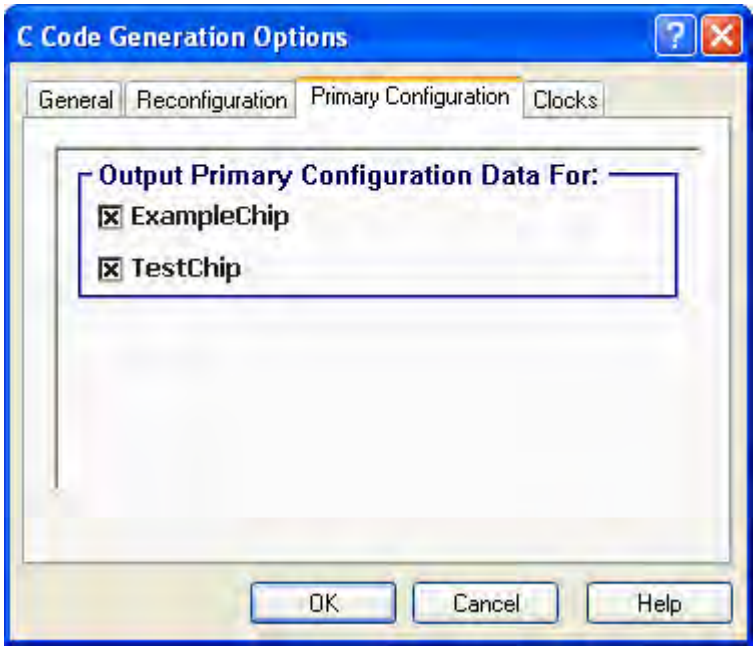
Automatic Buffer Growth

If it is uncertain that the initial buffer size will be enough to hold the entire reconfiguration data, then the buffer can be set to automatically grow when necessary. If the Enable automatic growth of the reconfiguration data buffer check box is checked, then the Grow by text box will be enabled. This text box holds the number of bytes to grow the memory block by if growth is necessary. If this is set to 32, and an API call requires one more byte in the buffer than is available, then 32 more bytes will be allocated, leaving 31 bytes for future data. If this is set to 4, and an API call requires 9 more bytes, then the minimum number of bytes that is a multiple of the grow by number will be allocated. In this case, 12 more bytes would be allocated.

The downside to enabling this feature is the time overhead that may be required by the microprocessor to dynamically allocate memory.

7.3.6 C Code Generation Options - Primary Configuration Tab

This tab is used to select which chips will have primary configuration data output in the C Code.



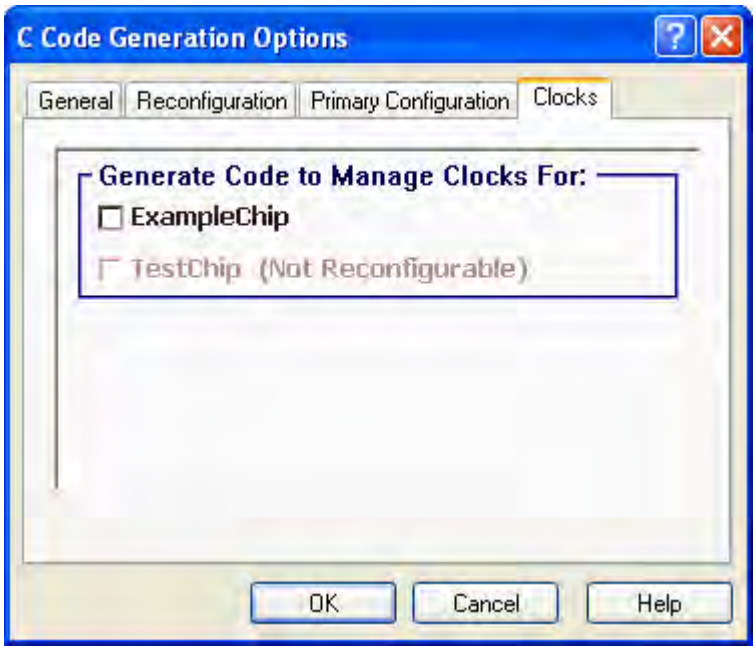
The primary configuration data can be generated for a chip regardless of what CAM C Code functions have been selected. If a chip is checked, then the primary configuration data is generated in the C Code files, and API functions are generated to access the data.

It may be desirable to disable the output of the primary configuration data if:

1. There are multiple chips in the circuit, but you will only be using the C Code for a sub-set of them.
2. The chip will get its initial configuration from something other than the microprocessor, but you need the C Code for reconfigurations.

7.3.7 C Code Generation Options - Clocks Tab

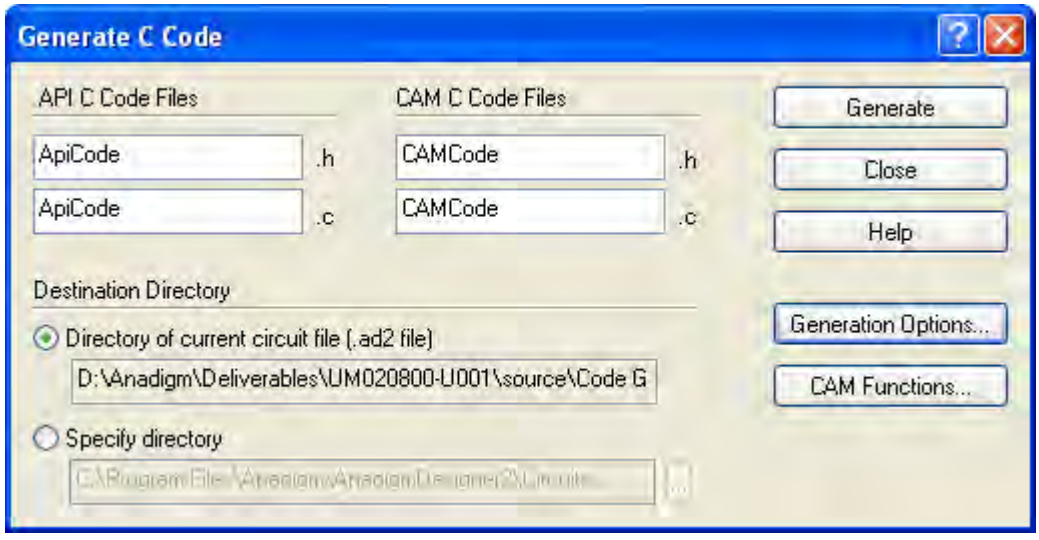
This tab is used to select which chips will be able to have their clock divisors manipulated with C Code.



The clock reconfiguration data can be generated for a chip regardless of what CAM C Code functions have been selected. If a chip is checked, then the necessary data and functions are generated to manipulate clock divisors for that chip.

7.3.8 Generate C Code Window

This window is used to select the names of the generated C Code files, as well as the directory where the files will be written.



API Code & CAM Code Files

There are always four files generated, no matter how many chips or CAMs are present, or what combination of generation options have been chosen. The CAM C Code Files will contain functions that have been checked in the CAM C Code Functions Window (or the Global CAM C Code Functions Window). All checked functions across all CAMs in the circuit are output to these two files. All of the API functions

and configuration data are output to the API C Code Files. These files may be renamed to any file name allowed by the operating system.

Destination Directory

Every file generated is output to the same directory. This output directory may be the directory of the current circuit file, or a directory that is specified. If the directory is that of the current circuit file, then anytime the location of the circuit file changes, this directory will change with it. This ensures that the C Code files will always be generated in the same directory as the corresponding .ad2 file. If Specify Directory is selected, then the output directory may be any directory accessible by the operating system. This directory will remain the same until it is explicitly changed.

Generate

The Generate button is pressed to create the files. A notification will be displayed when the generation is complete. Pressing the “Generation Options…” button opens the C Code Generation Options Window, and pressing the “CAM Functions…” button opens the Global C Code Functions Window.

7.4 C Code API Reference - Algorithmic Dynamic Configuration

This reference shows all C Code functions and types as they would be generated without the default prefix of "`an_`". For more information on the prefix, see Generation Options - General Tab.

Primary Configuration Functions

`GetPrimaryConfigData`	Gets the configuration data required to perform a primary configuration.
`GetResetData`	Gets the configuration data required to perform a soft reset.

Reconfiguration Functions

`InitializeReconfigData`	Prepares the data buffer that will hold the reconfiguration data.
`ShutdownReconfigData`	Destroys the reconfiguration data buffer created by InitializeReconfigData.
`ClearReconfigData`	Clears the reconfiguration data buffer without deallocating memory.
`GetReconfigData`	Gets the current reconfiguration data.
`GetReconfigControlFlags`	Gets the control byte of the reconfiguration data.
`SetReconfigControlFlags`	Sets the control byte of the reconfiguration data.

Power Management Functions

`GetSleepData`	Gets the configuration data required to put a chip into sleep mode.

Clock Functions

`GetMasterClock`	Gets the frequency in Hz of a master clock.
`GetClockDivisor`	Gets the current value of a clock divisor.
`SetClockDivisor`	Sets the value of a clock divisor.
`IncrementClockDivisor`	Increments the current value of a clock divisor to the next valid divisor value.
`DecrementClockDivisor`	Decrements the current value of a clock divisor to the next valid divisor value.
`ClockUpdatesFinished`	Clears Update_CLKS bit.

CAM Functions

Nearly all CAMs offered in the library have some programmable parameters controlled by associated functions. There are too many to list explicitly here. Generation of these functions is covered in more detail in Section 7.3.1, CAM C Code Functions Window.

7.4.1 GetPrimaryConfigData

```
const Byte* GetPrimaryConfigData(Chip chip, [out] int* pCount);
```

Description

Gets the configuration data required to perform a primary configuration. The data begins with the synch byte and ends with the last error byte. Data must be shifted into the FPAA most significant bit first.

Parameters

`chip`

The ID the C Code generated for the chip. This will be something like `chipName` or `an_chipName`, depending on the name of the chip and the generation options.

`pCount`

A pointer to a valid integer that will receive the number of bytes in the array returned by the function.

Return Value

A pointer to the configuration data that can be sent to the chip.

Example

```
/* Get a pointer to the primary configuration data */
int dataSize = 0;
const Byte* pData = GetPrimaryConfigData(chipName, &dataSize);
/* Send the data to the chip*/
. . .
```

7.4.2 GetResetData

```
const Byte* GetResetData(Chip chip, [out] int* pCount);
```

Description

Gets the configuration data required perform a soft reset on the chip. The data begins with the synch byte and ends with the control byte.

Parameters

`chip`

The ID the C Code generated for the chip. This will be something like `chipName` or `an_chipName`, depending on the name of the chip and the generation options.

`pCount`

A pointer to a valid integer that will receive the number of bytes in the array returned by the function.

Return Value

A pointer to the configuration data that can be sent to the chip.

Example

```
/* Get a pointer to the reset data */
int dataSize = 0;
const Byte* pData = GetResetData(chipName, &dataSize);
/* Send the data to the chip*/
. . .
```

7.4.3 InitializeReconfigData

```
void InitializeReconfigData(Chip chip);
```

Description

Prepares the data buffer that will hold the reconfiguration data for the chip.

Parameters

`chip`

The ID the C Code generated for the chip. This will be something like `chipName` or `an_chipName`, depending on the name of the chip and the generation options.

Return Value

None.

Remarks

This function must be called before calling any other reconfiguration functions for the chip. This function allocates memory. To deallocate the memory call `ShutdownReconfigData`. However, to simply clear the data in the buffer without memory deallocation call `ClearReconfigData`.

Example

```
/* Get a pointer to the primary configuration data */
int dataSize = 0;
const Byte* pData = GetPrimaryConfigData(an_chipName, &dataSize);

/* Send the data to the chip*/
. . .

/* Initialize reconfiguration data before using any CAM function */
InitializeReconfigData(an_chipName);

/* Calculate changes to CAM parameters then use CAM C Code */
. . .

/* Get a pointer to the reconfiguration data */
int dataSize = 0;
const Byte* pData = GetReconfigData(an_chipName, &dataSize);

/* Send the data to the chip */
. . .

/* OK. Data sent, now clear the buffer to get ready to call some more CAM func-
tions. */
ClearReconfigData(an_chipName);

/* Calculate new changes to CAM parameters then use CAM C Code */
. . .

/* Get a pointer to the new reconfiguration data */
const Byte* pData = GetReconfigData(an_chipName, &dataSize);
/* Send the data to the chip */
. . .

/* End of program clean up reconfiguration data */
ShutdownReconfigData(an_chipName);
```

7.4.4 ShutdownReconfigData

```
void ShutdownReconfigData(Chip chip);
```

Description

Destroys the reconfiguration data buffer created by `InitializeReconfigData`.

Parameters

`chip`

The ID the C Code generated for the chip. This will be something like `chipName` or `an_chipName`, depending on the name of the chip and the generation options.

Return Value

None.

Remarks

This function deallocates memory, and should be called after all reconfiguration processing for the chip is complete. To simply clear the data in the buffer in preparation for a new configuration, call `ClearReconfigData` instead.

Example

See `InitializeReconfigData`.

7.4.5 ClearReconfigData

```
void ClearReconfigData(Chip chip);
```

Description

Clears the data buffer that holds the reconfiguration data for the chip without deallocating memory.

Parameters

`chip`

The ID the C Code generated for the chip. This will be something like `chipName` or `an_chipName`, depending on the name of the chip and the generation options.

Return Value

None.

Remarks

It is preferable to call this function instead of repeated calls to `InitializeReconfigData` and `ShutdownReconfigData`. Repeated calls to these functions take time and may cause memory fragmentation.

Example

See InitializeReconfigData

7.4.6 GetReconfigData

const Byte* GetReconfigData(Chip chip, [out] int* pCount);

Description

Gets the current reconfiguration data for the chip. The data begins with the synch byte and ends with the last error byte. Data must be shifted into the FPAA most significant bit first.

Parameters

`chip`

The ID the C Code generated for the chip. This will be something like `chipName` or `an_chipName`, depending on the name of the chip and the generation options.

`pCount`

A pointer to a valid integer that will receive the number of bytes in the array returned by the function.

Return Value

A pointer to the reconfiguration data that can be sent to the chip.

Example

```
/* Get a pointer to the reconfiguration data */
int dataSize = 0;
const Byte* pData = GetReconfigData(chipName, &dataSize);

/* Send the data to the chip */
. . .

/* OK. Data sent, now clear the buffer to get ready to call some more CAM func-
tions. */
ClearReconfigData(chipName);
```

7.4.7 GetReconfigControlFlags

```
ControlByte GetReconfigControlFlags(Chip chip);
```

Description

Gets the control byte in the reconfiguration data for the chip.

Parameters

`chip`

The ID the C Code generated for the chip. This will be something like `chipName` or `an_chipName`, depending on the name of the chip and the generation options.

Return Value

A combination of one or more `ControlByte` flags generated by the C Code. The possible flags are:

```
ControlByte_Pullups
ControlByte_EndExecute
ControlByte_SReset
ControlByte_Read
ControlByte_InhibitRdbck
ControlByte_ResetAll
```

Remarks

Using the ANSI C bitwise-AND operator it is possible to test for specific flags in the control byte.

Example

```
/* Get the bits in the control byte */
ControlByte controlByte = GetReconfigControlFlags(chipName);

/* Do something if the Reset bit is set */
if (controlByte & ControlByte_SReset)
{
/* … */
}
```

7.4.8 SetReconfigControlFlags

```
void SetReconfigControlFlags(Chip chip, ControlByte nFlags);
```

Description

Sets bits in the control byte of the reconfiguration data for the chip.

Parameters

`chip`

The ID the C Code generated for the chip. This will be something like `chipName` or `an_chipName`, depending on the name of the chip and the generation options.

`nFlags`

A combination of one or more `ControlByte` flags generated by the C Code. The possible flags are:

```
ControlByte_Pullups
ControlByte_EndExecute
ControlByte_SReset
ControlByte_Read
ControlByte_InhibitRdbck
ControlByte_ResetAll
```

Return Value

None.

Remarks

Using the ANSI C bitwise-OR operator it is possible to set multiple flags at once. For instance:

Example

```
SetReconfigControlFlags(chipName, ControlByte_Pullups | ControlByte_EndExecute);
```

If it is also possible to turn flags on and off using a combination of bitwise operators and the `GetReconfigControlFlags` function. This example turns off the EndExecute bit, turns on the InhibitRdbck bit, and keeps the other bits as they were:

```
SetReconfigControlFlags(chipName, ControlByte_InhibitRdbck |
(~ControlByte_EndExecute & GetReconfigControlFlags(chipName));
```

7.4.9 GetSleepData

```
const Byte* GetSleepData(Chip chip, [out] int* pCount, Bool powerDown);
```

Description

Gets the configuration data required to put the chip into sleep mode.

Parameters

`chip`

The ID the C Code generated for the chip. This will be something like `chipName` or `an_chipName`, depending on the name of the chip and the generation options.

`pCount`

A pointer to a valid integer that will receive the number of bytes in the array returned by the function.

`powerDown`

If non-zero, all analogue functions will be turned off except the crystal oscillator. If zero, all analogue functions will be turned on.

Return Value

A pointer to the configuration data that can be sent to the chip. The memory containing the data is volatile. The data begins with the synch byte and ends with the last error byte.

Example

```
/* Get a pointer to the sleep data */
int dataSize = 0;
const Byte* pData = GetSleepData(chipName, &dataSize);

/* Send the data to the chip*/
. . .
```

7.4.10 GetMasterClock

```
Frequency GetMasterClock (Chip chip);
```

Description

Gets the frequency in Hz of a chip's master clock.

Parameters

`chip`

The ID the C Code generated for the chip This will be something like `chipName` or `an_chipName`, depending on the name of the chip and the generation options.

Return Value

The frequency of the master clock in Hz.

7.4.11 GetClockDivisor

short GetClockDivisor (Chip chip,ClockDivisor nDivisor);

Description

Gets the current value of a clock divisor for a chip.

Parameters

`chip`

The ID the C Code generated for the chip. This will be something like `chipName` or `an_chipName`, depending on the name of the chip and the generation options.

`nDivisor`

A clock divisor ID generated by the C Code. The possible values are:

ClockDivisor_PreScale
ClockDivisor_Clock0
ClockDivisor_Clock1
ClockDivisor_Clock2
ClockDivisor_Clock3
ClockDivisor_Chop1
ClockDivisor_Chop2

Return Value

The value of the clock divisor.

7.4.12 SetClockDivisor

void SetClockDivisor (Chip chip, ClockDivisor nDivisor, short value);

Description

Sets the value of a clock divisor for a chip.

Parameters

`chip`

The ID the C Code generated for the chip. This will be something like `chipName` or `an_chipName`, depending on the name of the chip and the generation options.

`nDivisor`

A clock divisor ID generated by the C Code. The possible values are:

ClockDivisor_PreScale
ClockDivisor_Clock0
ClockDivisor_Clock1
ClockDivisor_Clock2
ClockDivisor_Clock3
ClockDivisor_Chop1
ClockDivisor_Chop2

Return Value

None

Remarks

This function appends bytes to the current reconfiguration data for the chip. It is assumed that nDivisor is a valid clock divisor value. The change takes place immediately internally, meaning subsequent calls to `GetClockDivisor` for this chip and divisor will return the value set here. If nDivisor is equal to the current value of the clock divisor, then this function has no effect and no new bytes are appended to the reconfiguration data.

7.4.13 IncrementClockDivisor

```
void IncrementClockDivisor (Chip chip, ClockDivisor nDivisor);
```

Description

Increments the current value of a clock divisor to the next valid divisor value.

Parameters

`chip`

The ID the C Code generated for the chip. This will be something like `chipName` or `an_chipName`, depending on the name of the chip and the generation options.

`nDivisor`

A clock divisor ID generated by the C Code. The possible values are:

```
ClockDivisor_PreScale
ClockDivisor_Clock0
ClockDivisor_Clock1
ClockDivisor_Clock2
ClockDivisor_Clock3
ClockDivisor_Chop1
ClockDivisor_Chop2
```

Return Value

None.

Remarks

This function appends bytes to the current reconfiguration data for the chip. It is assumed that nDivisor is a valid clock divisor value. The change takes place immediately internally, meaning subsequent calls to `GetClockDivisor` for this chip and divisor will return the value set here. If the divisor is already at its maximum value, this function has no effect, and no new bytes will be appended to the reconfiguration data.

7.4.14 DecrementClockDivisor

void DecrementClockDivisor (Chip chip, ClockDivisor nDivisor);

Description

Decrements the current value of a clock divisor to the next valid divisor value.

Parameters

`chip`

The ID the C Code generated for the chip. This will be something like `chipName` or `an_chipName`, depending on the name of the chip and the generation options.

`nDivisor`

A clock divisor ID generated by the C Code. The possible values are:

```
ClockDivisor_PreScale
ClockDivisor_Clock0
ClockDivisor_Clock1
ClockDivisor_Clock2
ClockDivisor_Clock3
ClockDivisor_Chop1
ClockDivisor_Chop2
```

Return Value

None.

Remarks

This function appends bytes to the current reconfiguration data for the chip. It is assumed that nDivisor is a valid clock divisor value. The change takes place immediately internally, meaning subsequent calls to `GetClockDivisor` for this chip and divisor will return the value set here. If the divisor is already at its lowest value, this function has no effect, and no new bytes will be appended to the reconfiguration data.

7.4.15 ClockUpdatesFinished

void ClockUpdatesFinsihed (Chip chip);

Description

Clears the Update_CLKS bit.

Parameters

`chip`

The ID the C Code generated for the chip. This will be something like `chipName` or `an_chipName`, depending on the name of the chip and the generation options.

Return Value

None.

Remarks

When a reconfiguration that changes clock divisor values is sent, there is a Update_CLKS bit that must also be set for the clocks to actually be updated. All of the clock-setting functions (SetClockDivisor, IncrementClockDivisor, DecrementClockDivisor) ensure that this bit is set. However, after the reconfiguration is sent the bit must be turned off manually using this function. If this function is not called, the clocks will be reset and synchronized every time a reconfiguration is sent.

If the next reconfiguration data that will be sent also changes clock divisors, then it is not necessary to call this function, as the bit will just be turned back on by the clock-setting functions.

7.4.16 CAM Functions – Example Application

Each CAM contains C Code functions that are specific to changing its parameters The functions that get generated in the C Code are chosen using the C Code Functions Window and Global C Code Functions Window.

See the CAM documentation for more details on the C Code functionality each CAM contains.

Example Usage

This example assumes we selected the `setGain` function of a **GainInv** CAM named **MyGain**.

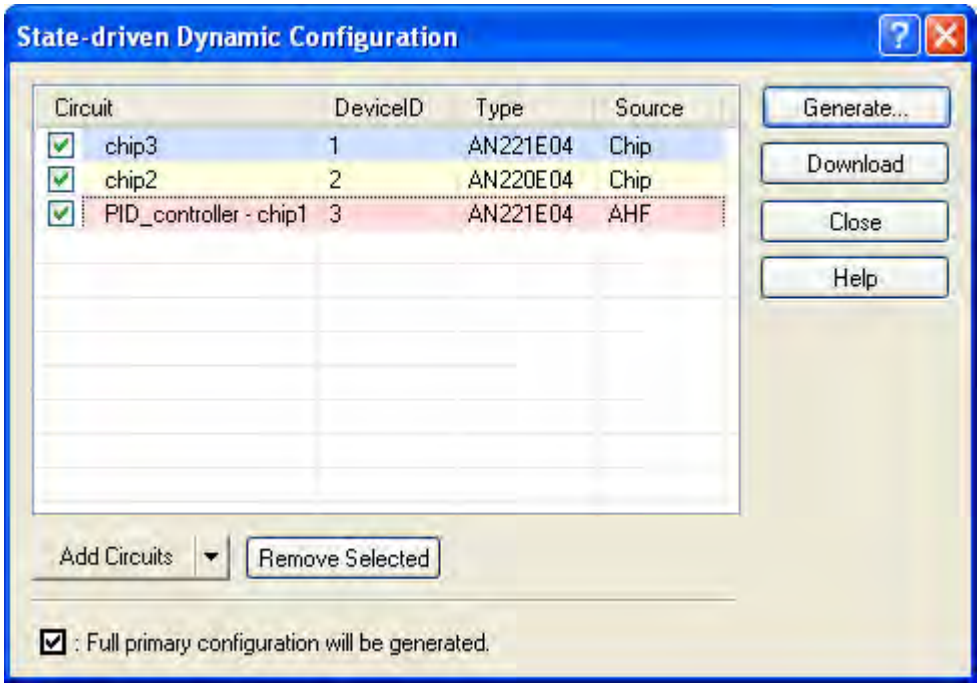
```
/* Get a pointer to the primary configuration data */
int dataSize = 0;
const Byte* pData = GetPrimaryConfigData(chipName, &dataSize);

/* Send the data to the chip*/
. . .

/* Get the reconfiguration buffer ready to go */
InitializeReconfigData(chipName);

/* Change the Gain just a bit */
setGain(chipName_MyGain, 6.0);

/* Now get a pointer to the reconfiguration data */
int dataSize = 0;
const Byte* pData = GetReconfigData(chipName, &dataSize);

/* Send the data to the chip */
. . .

/* OK. Data sent, now clear the buffer to get ready to call some more CAM func-
tions. */
ClearReconfigData(chipName);

/* Do some more changing of the gain */
. . .

/* We are all done doing C Code stuff. Destroy the reconfiguration buffer. */
ShutdownReconfigData(chipName);
```

7.5 State Driven Dynamic Configuration
(AN220E04,AN221E04 & AN221E02 only)

State Driven Dynamic Configuration is the use of C Code by a companion host processor to access and download pre-compiled reconfiguration data for the attached FPAA(s). Unlike Algorithmic Dynamic Configuration, State Driven Dynamic Configuration does not generate C Code for the host processor to create FPAA configuration data. State Driven Configuration instead creates pre-compiled data sets and just two C Code function calls for access to that data, for compilation into the host program.

When using Algorithmic Dynamic Configuration only CAM parameters can be adjusted. With State Driven Dynamic Configuration the entire circuit topology can be changed; the updated circuit can adjust just a single CAM parameter or replace the complete contents and connectivity.

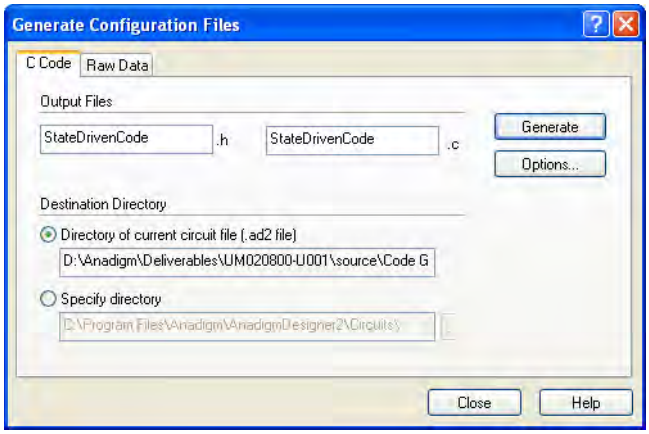
When the *Dynamic Config.* → *State-driven method..* menu item is selected, a chooser window pops up. Designs for which configuration data is to be compiled are added to this chooser window. Using the “Add Circuits” control, the designs may be imported from the active design session using the “From Chips...” option of this control. Designs may also be imported by using the “From Circuit Files...” in the form of existing .AHF configuration files.



Function naming for Algorithmic Dynamic Configuration is based on the chip name (e.g. “chip2”). In State Driven Dynamic Configuration the grouping of designs is by DeviceID (ID1). At least one version of each chip (of each DeviceID) must have a full Primary Configuration data set generated. All other chips of the same DeviceID will get difference data generated. The resultant compiled data file(s) covering several chips can thus be quite compact.

Left-clicking the Generate button will open up a next level dialog box which establishes values for various parameters associated with C Code and Raw Data file generation for State Driven Dynamic configuration.

7.5.1 C-Code



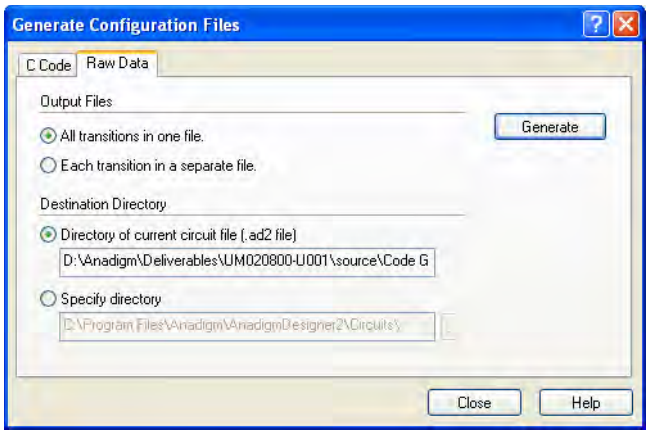
Very similar to the C Code generation dialog described in section 7.3.8, the C Code tab of this dialog establishes the root file names and locations for the generated files.

The .c file will contain constant arrays with Primary Configuration and reconfiguration data (difference data) required for all devices declared in the chooser and functions to access that data.

The .h file contains a prototype for the functions as well as a collection of typedefs and defines. This file is fully commented for clarity.

The host processor uses the function calls (described later in section 7.6) to gain access to the data in order to perform Primary Configurations and subsequent reconfigurations using the difference data.

7.5.2 Raw Data



In order to accommodate construction of host processor software in some other language than C (assembler for example), AnadigmDesigner®2 creates data files which contain the same Primary Configuration and reconfiguration (difference data) data sets in ASCII Hex Format (AHF). The data may be output into a single file (per DeviceID) or organized into separate files.

When output into one file, a .txt file containing the AHF data is created with root name of StateDrivenDevice*nnn*.txt; where *nnn* is the DeviceID number (ID1). The file contains only comment fields to delineate the data sets and the AHF formatted data itself.

When the “Each transition in a separate file.” radio button is selected, a subdirectory is created similarly named StateDrivenDevice*nnn*. Within that subdirectory, separate AHF formatted .txt files are generated without comment, each containing either Primary Configuration data or reconfiguration data (difference data). The files names alone provide guidance as to the contents. For example, “(Primary) chip1.txt” contains the Primary Configuration data for the device named “chip 1”.

7.6 C-Code API - State Driven Dynamic Configuration

The C-Code API for State Driven dynamic configuration is much reduced from Algorithmic Method. Only two functions are provided: one to assist in a Primary Configuration and another to assist in a device reconfiguration. This reference shows all C Code functions and types as they would be generated without the default prefix of "`an_`". For more information on the prefix, see Generation Options - General Tab.

Primary Configuration Functions

`GetCircuitPrimaryData` Gets the configuration data required to perform a primary configuration.

Reconfiguration Functions

`GetCircuitTransitionData` Gets the configuration data required to perform a reconfiguration.

7.6.1 GetCircuitPrimaryData

```
const Byte* GetCircuitPrimaryData (circuit nCircuit, [out] int* pCount);
```

Description

Gets the configuration data required to perform a primary configuration. The data begins with the synch byte and ends with the last error byte.

Parameters

`nCircuit`

The ID generated for the circuit. This will be something like chipName_001_Primary or an_chipName_001_Primary, depending on the name of the chip and the generation options.

`pCount`

A pointer to a valid integer that will receive the number of bytes in the array returned by the function.

Return Value

A pointer to the configuration data that can be sent to the chip.

Example

```
/* Get a pointer to the primary configuration data */
int dataSize = 0;
const Byte* pData = GetCircuitPrimaryData(chipName_001_Primary, &dataSize);
/* Send the data to the chip*/
. . .
```

7.6.2 GetCircuitTransitionData

```
const Byte* GetCircuitTransitionData (Circuit nCircuit, [out] int* pCount);
```

Description

Gets the configuration data required to transition from one circuit state to another. The data begins with the synch byte and ends with the control byte.

Parameters

`nCircuit`

The ID generated for the circuit. This will be something like chipName_001 or an_chipName_001, depending on the name of the chip and the generation options.

`pCount`

A pointer to a valid integer that will receive the number of bytes in the array returned by the function.

Return Value

A pointer to the reconfiguration data that can be sent to the chip.

Example

```
/* Get a pointer to the primary configuration data */
int dataSize = 0;
const Byte* pPrimaryData = GetCircuitPrimaryData(chipNameA_001_Primary, &dataSize);
/* Send the data to the chip*/
. . .

/* Get a pointer to reconfiguration to change to another circuit */
const Byte* pTransData = GetCircuitTransitionData(chipNameB_001, &dataSize);
/* Send the data to the chip*/
. . .
```

7.7 Static Configuration (all devices)

Static configuration by a host processor involves the transfer of a complete Primary Configuration data set after an FPAA reset sequence. Host driven Static Configuration is suitable for systems in which the FPAA is configured only at power-up or after a system reset.

AnadigmDesigner®2 supports various Static Configuration schemes by the generation of three unique forms of data. Two of the forms were discussed in sections 7.5.1 and 7.5.2, namely C-Code and Raw Data for State Driven dynamic reconfiguration. If the FPAA is first reset, it is practical to use only the Primary Configuration portions of the data generated by these features.

The third form of configuration data created by AnadigmDesigner®2 is the configuration data file. While intended for use with a PROM programmer, these various data file formats are perfectly suitable for access by a host processor for Static Configuration. In fact, the Raw Data format used in State Driven Dynamic Configuration is the same as the .ahf configuration file format described in section 5.3.

Since AN120E04 and AN121E04 devices do not accept reconfiguration data, only a reset sequence followed by a Primary Configuration sequence is possible for host driven Static Configuration.

8 AnadigmFilter

AnadigmFilter is a powerful design aid for the creation of higher order filters. The standard CAM libraries provide first and second order filters that only require user selection of corner frequency, gain and Q. These standard library filter elements may be cascaded to realize filters of higher order, but doing so effectively usually requires using supplementary filter design reference materials and significant manual calculations. As an alternative, AnadigmFilter completely automates the design and implementation of higher order filters. The tool is accessed via the *Tools* → *AnadigmFilter* menu selection.

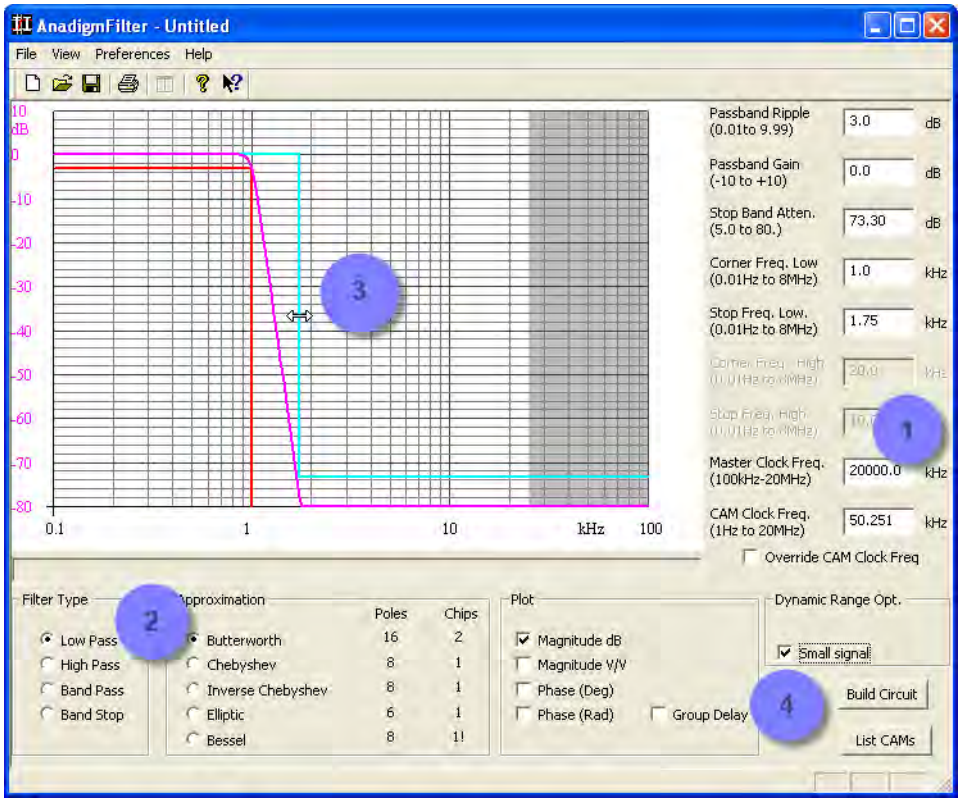


Figure 33 – The AnadigmFilter Main Window

The main window of AndigmFilter contains all of the controls that will be of interest to most uses. Designing and implementing a filter can be achieved in four steps. The figure above has numbered annotations to match these steps:

1. Tell AnadigmFilter the Master Clock frequency.
2. Select the desired type of filter (Low Pass, High Pass, Band Pass, Band Stop).
3. Drag the filter response limit boundaries to the desired frequency and gain settings.
4. Left-click the *Build Circuit* button.

AnadigmFilter will design the filter and export the design to AnadigmDesigner®2. AndigmFilter will create as many FPAA instances as required, and populate those instances with lower order filter CAMs complete with all the parameters set and connections made as necessary to realize the high order filter design. Filters of very high order can be designed and implemented in just moments. The main window of AnadigmFilter also has settings for the selection of the approximation type (Butterworth, Chebyshev, Inverse Chebyshev, Elliptical, and Bessel). There are controls available for governing what sort of plot is presented. Text entry boxes, allow for precise textual vs. graphical filter parameter input. The *Build Circuit* button transfers information out of AnadigmFilter and into AnadigmDesigner®2.

In the sample below, the 16 pole Butterworth filter approximation of Figure 33 was transferred into the design window of AnadigmDesigner®2. This particular filter required two FPAAs, however much more efficient approximations were available, with the others requiring only a portion of a single FPAA. The number of FPAAs required to achieve the filter is listed in the *Chips* column in the *Approximation* panel of AnadigmFilter. The exclamation mark next to the Bessel approximation in the example above informs the user that there was some problem. In this case, the response of the Bessel approximation fell outside of the gain and frequency boundaries.

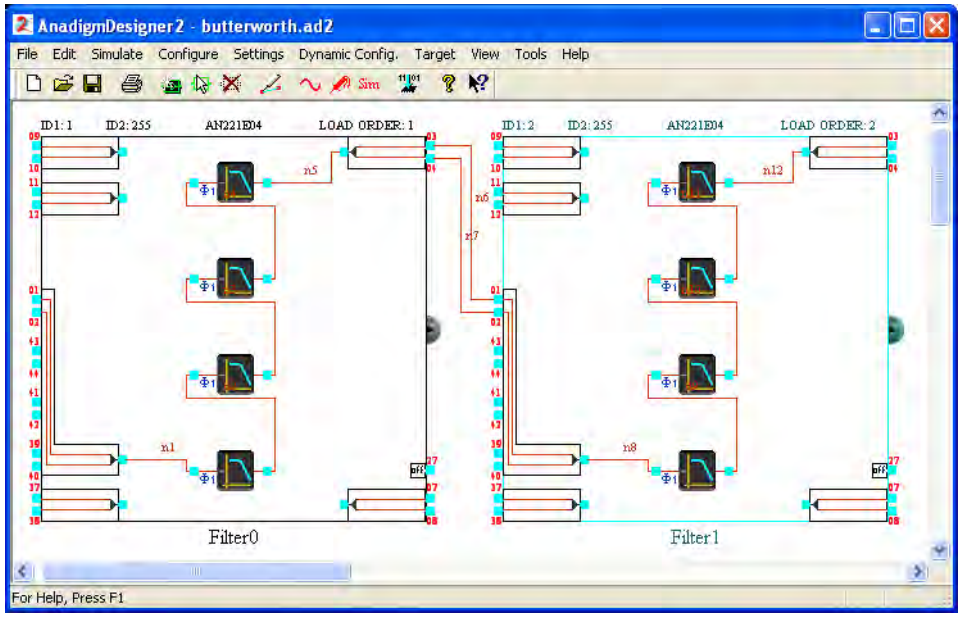


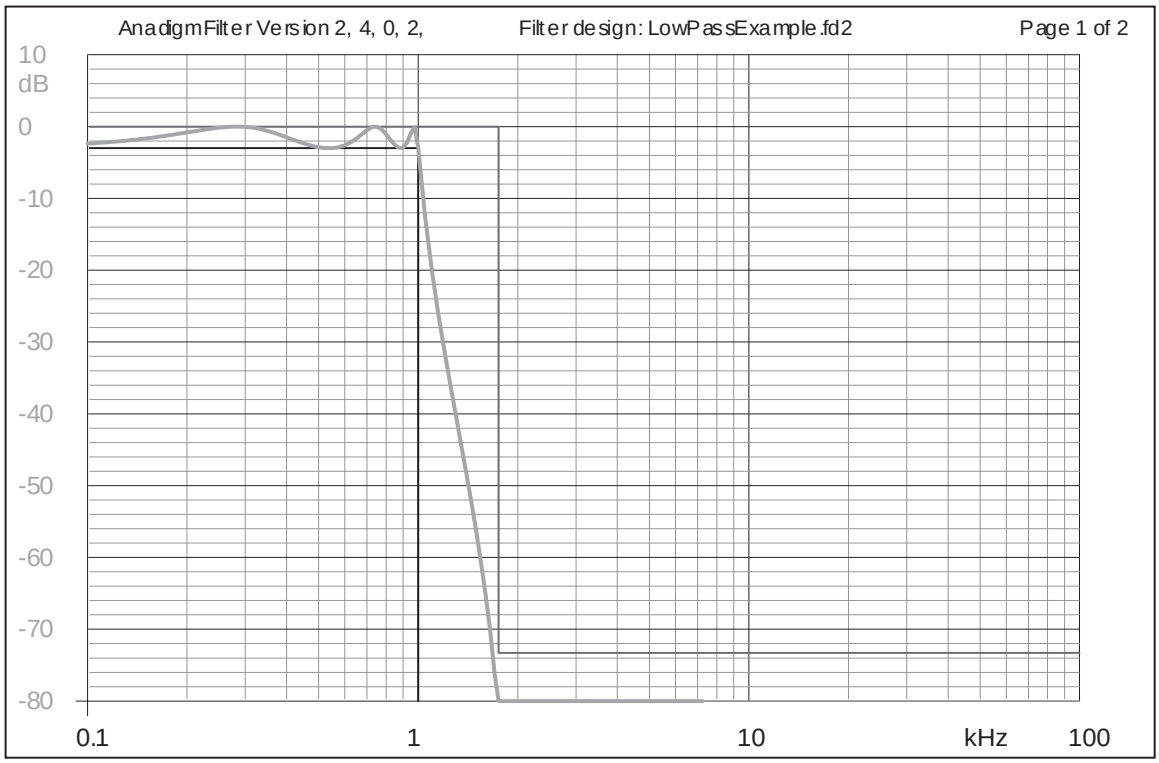
Figure 34 – Automatic Construction of High Order Filters

Off-Line Data Analysis

Filter response data can be exported for off-line analysis using the *File → Save Analysis File (CSV)* menu item. A .csv format file is created with columns for: Frequency, Magnitude [dB], Magnitude [V/V], Phase, and Group Delay.

Printing

The *File* → *Print* menu command creates a multi-page print out similar to the one shown in Figure 35.



AnadigmFilter Version 2, 4, 0, 2, Filter design: LowPassExample.fd2 Page 2 of 2

The parameters for this filter are saved in file: LowPassExample.fd2.fd2

Target Chip type:	AN221E04
Filter Type:	Low Pass
Approximation:	Elliptic

Filter is optimized for Small signal input

Filter Parameters:

Passband Ripple:	3 dB
Overall Gain:	0 dB
Stop Band Attenuation:	73.3 dB
Corner Fequency:	1 KHz
Stop Frequency:	1.75 KHz
Master Clock Frequency:	2e+007 KHz
CAM Clock Frequency:	49.2611 KHz

The following Anadigm CAMs are needed to realize this filter:

Chip: FILTER0

	Instance	Clock	Module	Type	Phase	Fo-Khz	DC Gn	HF Gn	Q	
CAM:	Stage0	ClockA	0	FILTERBIQUAD	BS	1	0.327	14.1	0.0392	1.06
CAM:	Stage1	ClockA	0	FILTERBIQUAD	BS	1	0.757	0.382	0.0392	3.84
CAM:	Stage2	ClockA	0	FILTERBIQUAD	BS	1	0.981	0.132	0.0392	15.5

Figure 35 – Sample Print Sheets from AnadigmFilter

Preferences

The only other controls that need any detailed explanation are associated with the *Preferences* menu item.

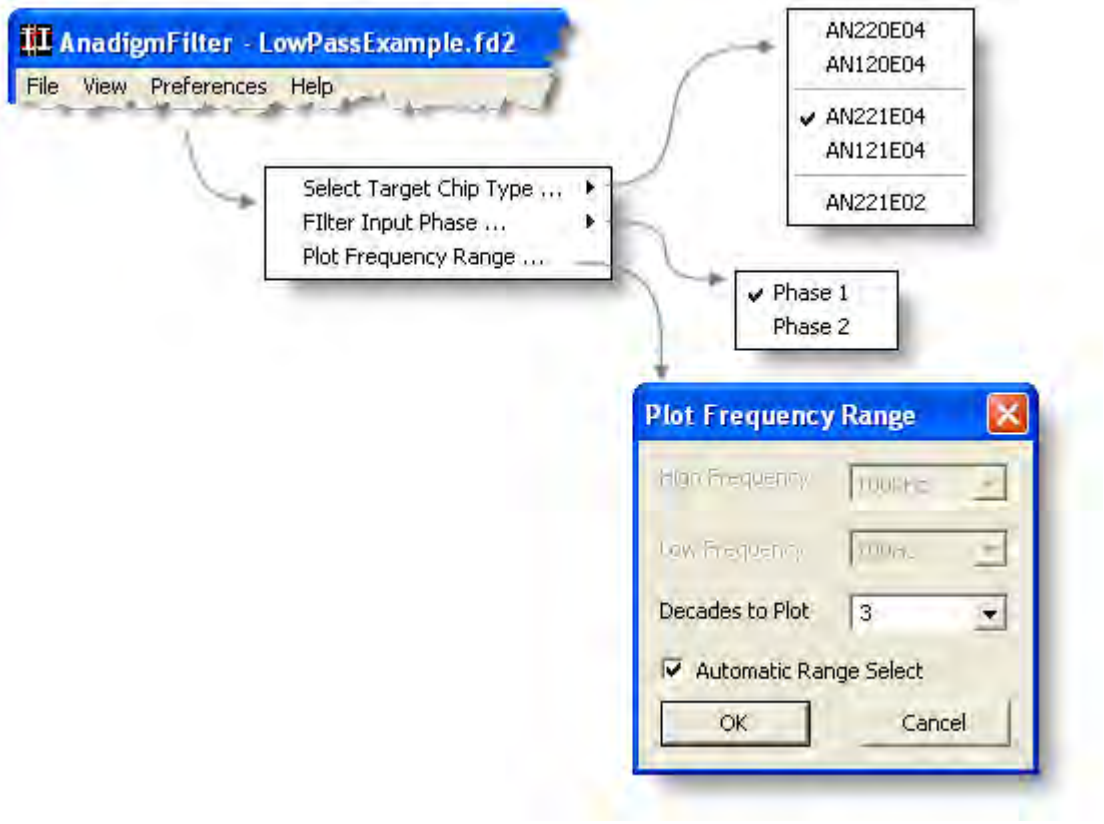


Figure 36 – Menu Sub-Items Under the Preferences Menu Heading

The controls contained within the *Preferences* menu tell AnadigmFilter what FPAA type it should map the filter design into and which phase of the clock the input signal should be sampled on. The input phase selection is usually of not much consequence as the signal is typically coming from outside the chip.

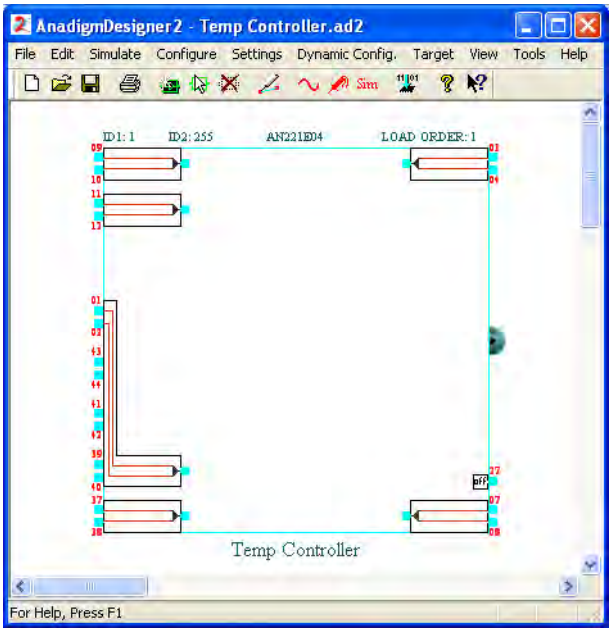
There are also a set of controls governing the appearance of the AnadigmFilter plot window.

Further Detailed Feature Descriptions

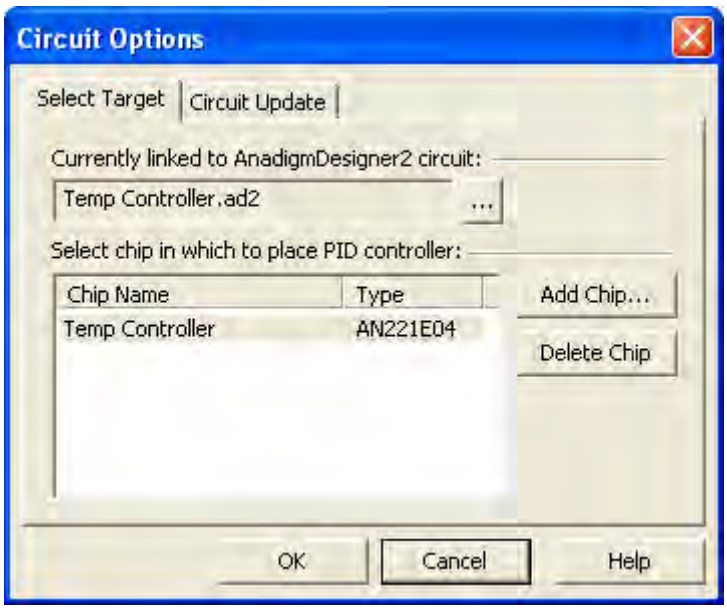
The Help menu system associated with AnadigmFilter is extensive. Detailed descriptions for every feature of AnadigmFilter are available for further examination.

9 AnadigmPID

AnadigmPID is a powerful design aid for the creation of closed loop control circuits. The tool uses Proportional (P), Integral (I) and Derivative (D) building blocks to construct common closed loop controller circuit topologies including, P, PI, PD and PID forms. The primary design inputs are simply the gains for each leg of the controller circuit. Controller circuit design data is automatically (and continuously) transferred from AnadigmPID into AnadigmDesigner®2 (and optionally onto a hardware evaluation platform).



Creation of a closed loop controller circuit begins with opening a new circuit within AnadigmDesigner®2. Although the FPAA instance of the new design window is empty, it will be convenient at this point to name the instance something meaningful and to also save the .AD2 file under a meaningful name.



.AnadigmPID is invoked from the *Tools* → *AnadigmPID* menu selection.

When the AnadigmPID is first invoked it needs to know what FPAA instance to communicate its design data to. Normally, only a single FPAA instance is open for edit, however it is possible to have multiple FPAAs open in AnadigmDesigner®2 and the *Circuit Options* selection window is used to establish contact with the desired one

Once this dialog is dismissed, the main window of AnadigmPID is presented.

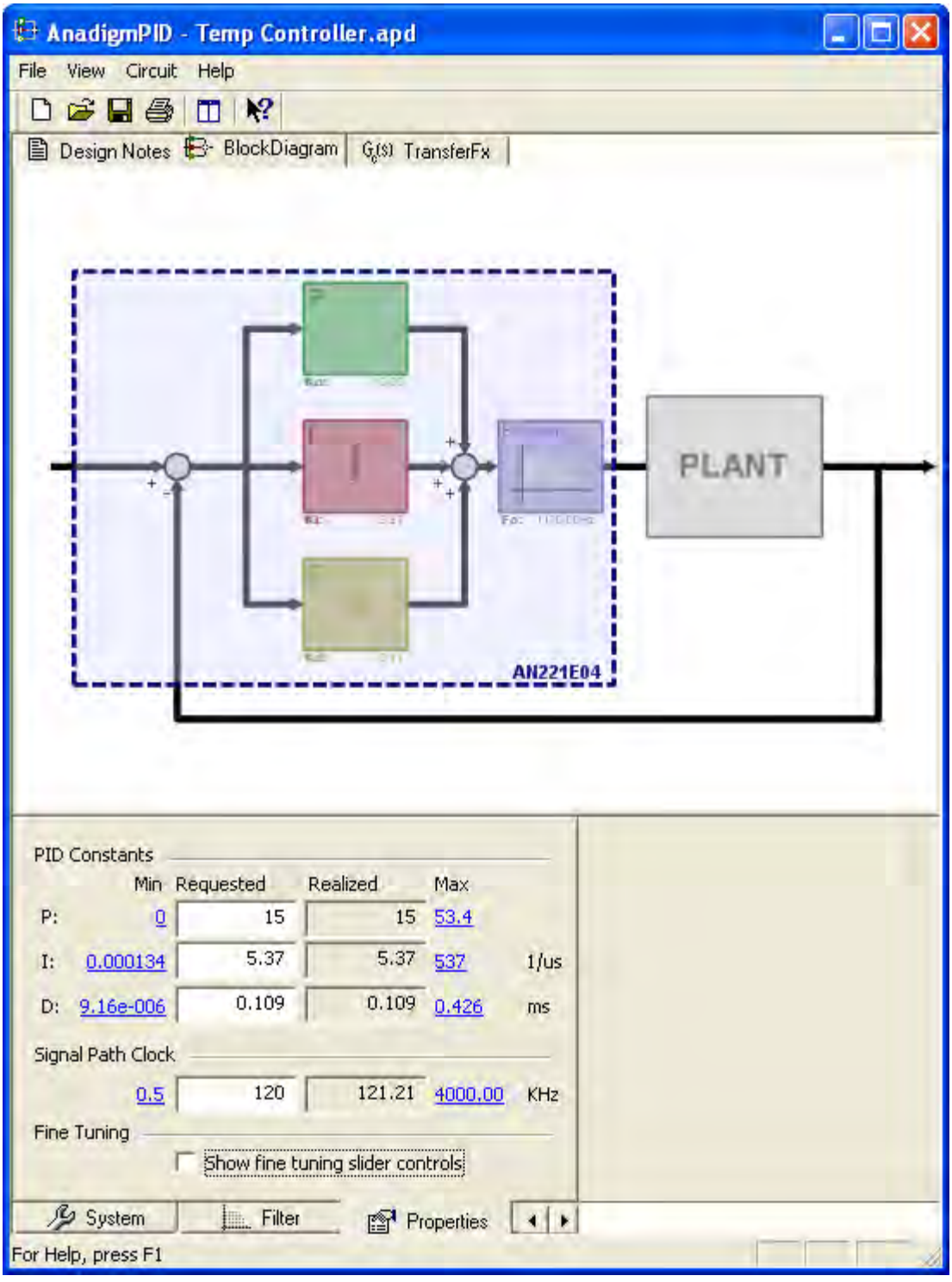
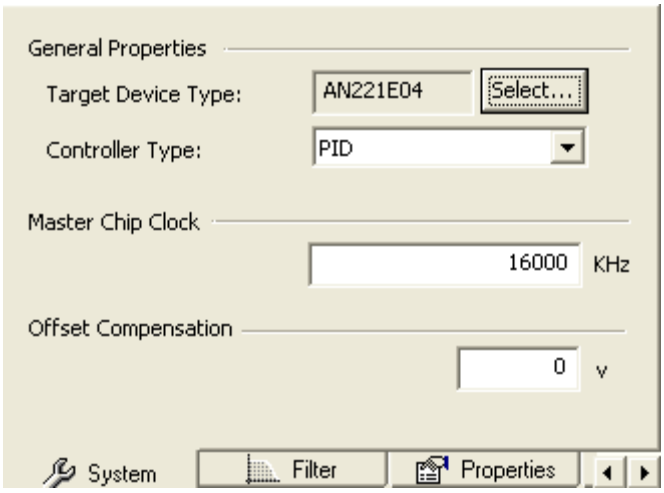


Figure 37 – AnadigmPID Main Window - Block Diagram View

The upper half of the AnadigmPID window is a tabbed window with tabs for: *Design Notes*, *Block Diagram* and *TransferFx* (Transfer Function). This brief introduction to the tool will address only the *Block Diagram* tab. The lower half of the AnadigmPID window is also a tabbed window. The tabs in this portion of the window include: *System*, *Filter*, *Properties* and *Input*.

System Tab



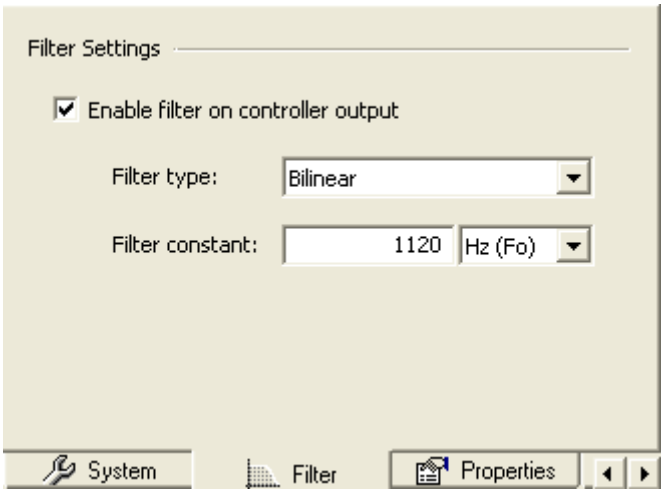
The *System* tab defines the target FPAA type, in this case and AN221E04. The *Select* button recalls the *Circuit Options* dialog described above.

Master Chip Clock sets the expected frequency of the analog clock for the device.

The *Controller Type:* drop down selection box includes choices for P, PI, PD and PID controller topologies (Proportional, Integral, Differential)

The *Offset Compensation* control adds DC offset to the output of the control circuit.

Filter Tab

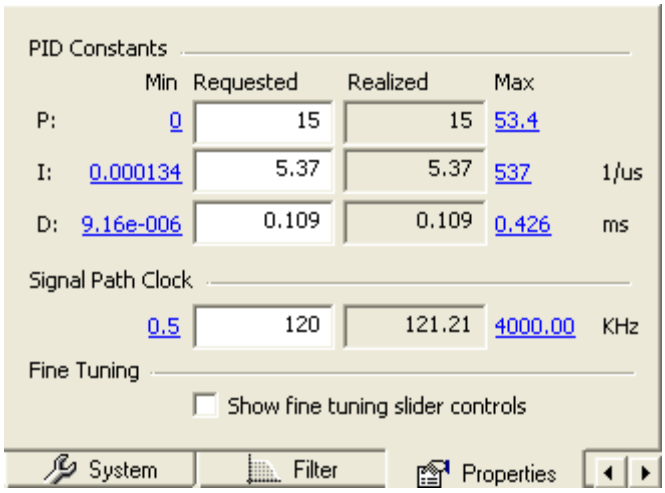


The *Filter* tab controls whether or not a low pass filter is included for the controller's output and sets parameters for that filter.

The filter type options available are bilinear and a higher order biquadratic.

The filter constant is usually specified as a corner frequency in Hz. Optionally, input units may be set to Radians/s or S.

Properties Tab



The *Properties* tab is where the constants associated with each leg of the controller circuit are set. Controls are available for K_P, K_I and K_D. The desired values for each of the constants are entered in the *Requested column*. The *Realized* column reflects what AnadigmPID was able to achieve.

The achievable ranges of each of the controls are not completely independent of one another. In particular the frequency set in the *Signal Path Clock* declares the frequency delivered to each of the CAMs in the signal path. Changing this control impacts all signal path CAMs and the range of achievable response for each. Clicking on any of the hyperlinked text within this tab will open a detailed help window.

Determining constant values appropriate to the plant or process being controlled is beyond the scope of this introductory manual.

A final control in the *Properties Tab* is the *Show fine tuning slider controls* check box. Selecting this control opens a window similar to the one shown below in Figure *38*. This control is especially useful when combined with the *Circuit* → *Continuous Download to Board* menu feature. Using these slider controls while configuration data is continuously downloaded to a target system enables the live tuning of controller circuits created with AnadigmPID.

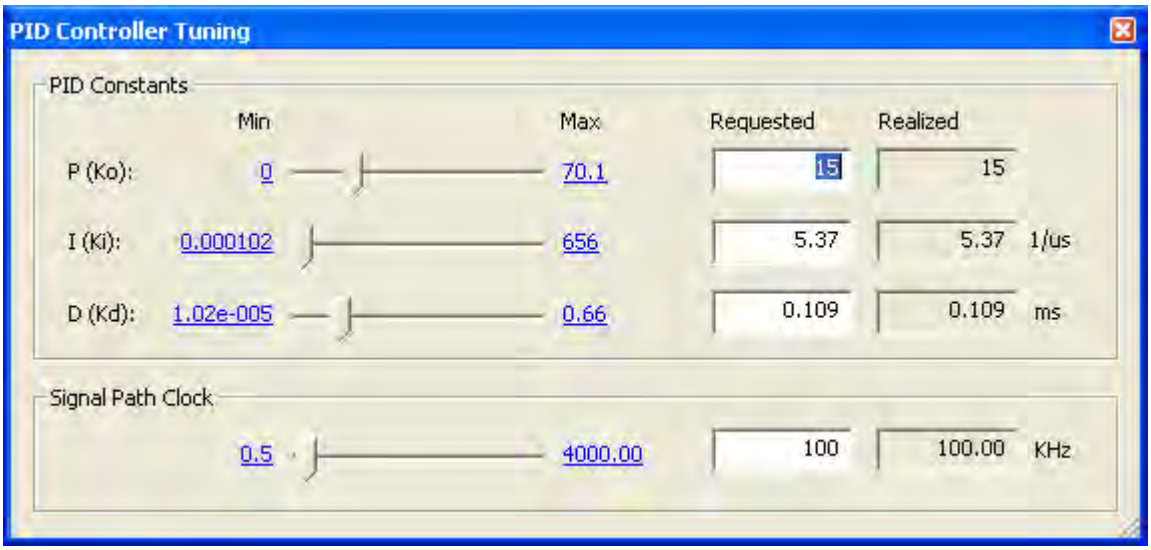
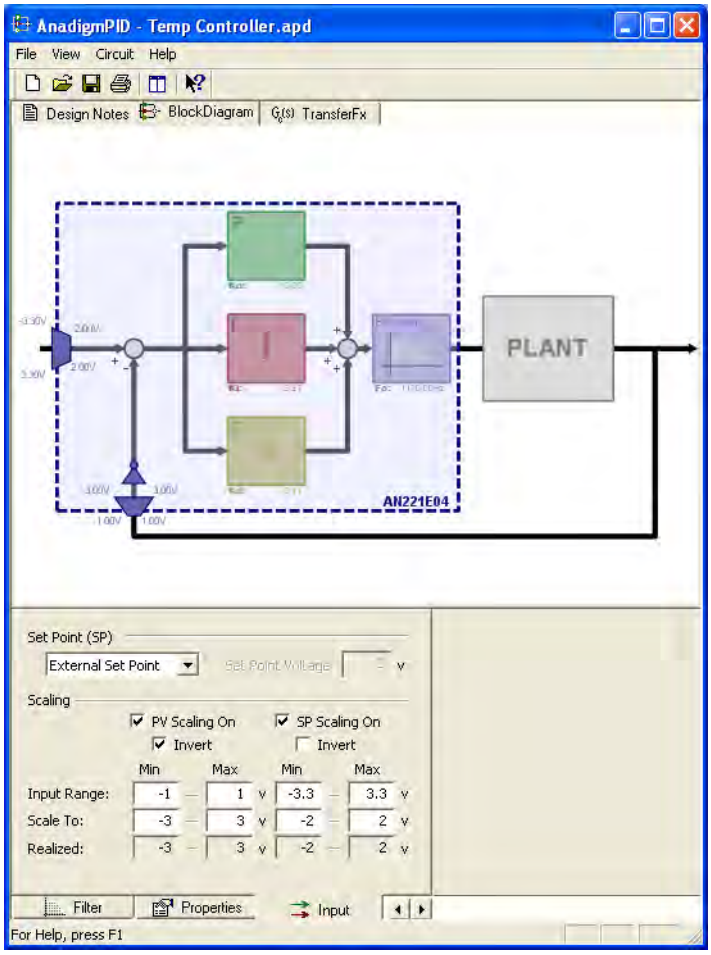


Figure 38 – Fine Tuning Slider Controls - Convenient for Tuning Live Systems

Input Tab



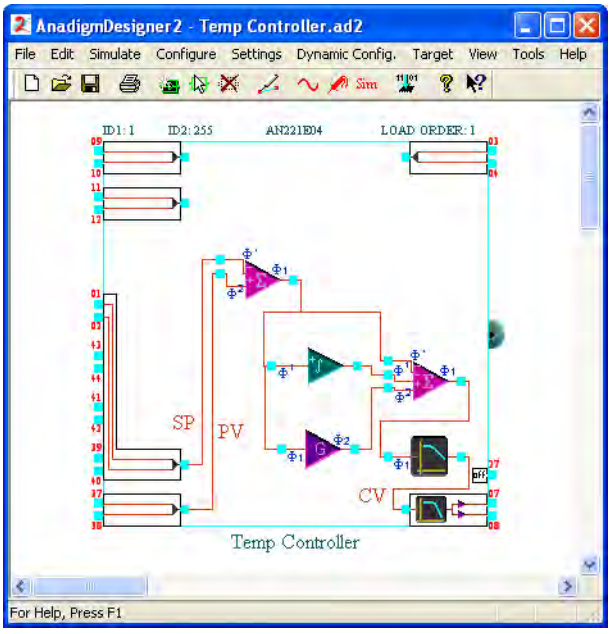
The *Input* tab allows for control over several aspects of the controller circuit inputs. Scaling and/or inversion can be applied to the PV signal (the feedback signal from the plant).

Similarly scaling and/or inversion is available for the Set Point signal (SP).

Selecting scaling adds gain ports at the FPAA boundary of the block diagram as shown to the left. Likewise, selecting inversion adds an inverter icon to the block diagram as a visual reminder of the setting.

Internal Set Point is one of the options available from the *Set Point (SP)* drop down control. Selecting this option places a Set Point voltage generator inside of the FPAA. This is useful in applications where the Set Point is invariant, or otherwise set under control of a companion host processor using dynamic reconfiguration.

Automatic Transfer of Design Data



Circuits designed within AnadigmPID are transferred live to the FPAA instance within AnadigmDesigner®2 that was first associated with the design. Wire labels SP, PV and CV make for easy identification of the Set Point, Plant Feedback and Controller Output respectively.

Once PID (PI, PD, or P) controller design is completed, manual modifications or additions to the FPAA are possible in the usual manner from within the AnadigmDesigner®2 main window.

Further Detailed Feature Descriptions

The Help menu system associated with AnadigmPID is extensive. Detailed descriptions for every feature of AnadigmFilter are available for further examination.

www.anadigm.com