Presentation Title: Introduction to VisualAudio®

Presenter Name: Paul Beckmann

Chapter 1: Introduction

Chapter 1a: Overview

Hello and welcome. My name is Paul Beckmann. I'm an engineer with Analog Devices and today I am going to be discussing an introduction to VisualAudio. This module provides an introduction to VisualAudio; a tool for rapid development of audio processing software. The examples and demonstrations today will be based on the Blackfin 533 EZ-KIT although VisualAudio also supports a large number of other Blackfin and SHARC processors. You will learn about the primary features of VisualAudio and how the tool can accelerate product development. You'll learn how to design audio processing layouts using the graphical editor and also a little bit about the underlying DSP software architecture. The target audience for this

presentation is embedded product developers. We assume some experience with audio and also some familiarity with the Blackfin processor and the VisualDSP++ development environment. This is an introductory module. A separate module, aimed specifically at audio algorithm developers, discusses VisualAudio's advanced features in more detail.

The outline for today's presentation is: First I give an overview of VisualAudio, describing the main features and benefits. Then I'll move on to a live demonstration and show you the tool in action. Then I'll talk about the DSP software architecture, discuss how VisualAudio is related to VisualDSP++, talk about the audio module library, real-time platforms, and then finally, conclude.

Chapter 2: VisualAudio Overview

Sub-chapter 2a: What Is VisualAudio?

Let's begin with an overview of VisualAudio. What is VisualAudio? It's a tool for streamlining audio product development. It consists of three components. First of all, there is VisualAudio designer; it's a graphical audio processing design application. There's a separate audio module library consisting of commonly used audio functions. And finally, there are example platforms – these real-time frameworks run on the EZ-KIT hardware and provide audio I/O. VisualAudio is designed for product development engineers; people who have to put together efficient products with audio capabilities. VisualAudio provides most of the standard software components found in audio products and finally, VisualAudio generates MIPS and memory optimized code. The code that VisualAudio generates is suitable for inclusion into a final product without further optimization.

Sub-chapter 2b: Blackfin vs SHARC

VisualAudio supports both the Blackfin and the SHARC processor families from Analog Devices. The Blackfin processor is a native 16-bit processor with SIMD capabilities. Within VisualAudio, we use a 32-bit fixed point representation for all audio. Blackfin also has a rich set of microcontroller features and has a full external memory interface. The SHARC processor, on the other hand, is a 32-bit floating point DSP, also with SIMD capabilities. There's an external memory interface that varies among the processor versions. Both architectures come in a variety of models with integrated audio peripherals. You'll find serial ports, S/PDIF transceivers, hardware sampling rate converters and so forth. Both processor families are supported by similar platforms and complementary sets of audio modules and decoders.

Let's compare and contrast the Blackfin and the SHARC processors. The SHARC is ideal for products whose primary function is audio, where there is a significant amount of audio processing. This includes audio/video receivers, professional audio systems, and high-end automotive systems. The Blackfin processor, on the other hand, is ideal for products that have functions in addition to audio. This includes portable media players, automotive head units and telematics systems, networked media nodes, mass market pro audio and mid-end and entry-level

automotive amplifiers. As a rule of thumb, the SHARC processor is three to four times as efficient as a Blackfin in processing audio per MIP. Still, given Blackfin's high clock rate, it's still a very significant and feature rich audio processor.

Sub-chapter 2c: Supported Hardware

This slide gives an overview of the EZ-KIT evaluation hardware supported by VisualAudio. On the left hand side is a list of the EZ-KITs which contain SHARC processors and on the right you'll see the Blackfin processors. It begins with the 262 EZ-KIT. It has 2 analog inputs and 8 analog outputs and also, a single S/PDIF input. Then there's the 364 EZ-KIT. It also has 2 in /8 out analog and contains S/PDIF inputs and outputs. And there's also the 369 EZ-KIT. That has a similar set of I/O as a 364 EZ-KIT. There's also an audio extender card, which is coming soon, which is going to provide 8 audio inputs and 16 audio outputs and this audio extender card works with the complete line here of SHARC EZ-KITs.

On the right hand side are the EZ-KITs for the Blackfin processors. There's one for the 533 EZ-KIT; that's the one I'm going to be using today. It has 4 analog inputs and 6 analog outputs. There's a separate Blackfin 537 EZ-KIT. By itself, it has two inputs and two analog outputs and in addition, there's an audio extender card that works with the Blackfin 537 EZ-KIT and that provides 8 analog inputs and 16 analog outputs and also S/PDIF input and output. So based on your application, which processor family you're using and your I/O needs, you would select a suitable EZ-KIT.

Sub-chapter 2d: Key Benefits

Now the key benefits for VisualAudio: I've broken up into two separate categories; one for audio product development and one for IP developers, that is, those people who are developing audio algorithms.

Let's start with the benefits for product developers. First of all, VisualAudio provides a starting point and methodology for audio product development. You don't have to start from scratch. Many of the pieces are provided. VisualAudio reduces development time, cost and risk. It allows engineers to focus on differentiating their products rather than implementing standard features. What's also nice about VisualAudio, is it provides access to audio IP, that is audio algorithms, in a consistent format.

Now for audio IP developers, VisualAudio features streamlined audio IP development. It allows you to tune and test and develop algorithms more quickly. VisualAudio also serves as a nice demonstration platform and also VisualAudio provides a consistent format to deliver audio IP in.

Chapter 3: On-line Demo

Sub-chapter 3a: Demo Overview

Now, let's move on to a live demo. First of all, I'm going to begin and talk about creating a new system in VisualAudio, show you how to design the layout using the drag-and-drop editor, go through code generation, and then we're going to build and run the executable on the EZ-KIT hardware using VisualDSP++ and finally, we're going to conclude with real-time tuning.

Now the demo set up I have is as follows: First of all, I have, in terms of hardware, I have a Blackfin 533 EZ-KIT here. I'm using a high performance USB emulator. That's the connection between the VisualDSP++ debugger and the Blackfin processor. Although I'm using a high performance emulator, you can also use VisualAudio with the built-in USB emulator that's on the EZ-KIT itself. I have a line level audio source coming in here, that's coming from my PC and then I have two audio outputs here that go to a set of powered speakers.

In terms of my software setup, I have VisualAudio installed and also VisualDSP++. And the steps I'm going to go through; I'm going to create an audio processing design using the graphical editor. Again, I'm going to generate code, build and run the executable on the EZ-KIT and finally, tune the system in real-time.

Sub-chapter 3b: Audio Layout

Now I'm going to switch over to VisualAudio. Here's the main VisualAudio window. This is the way VisualAudio looks when it's first started. I'm going to begin by going under the system menu and selecting "New System." First of all, I have to give it a name. I'm going to call it "DemoSystem" and the next thing I do is I select a platform file. Now as I mentioned, each of the EZ-KITs has an associated platform file. So I'm going to browse to the location of the platforms, under VisualAudio Platforms and right now I have three different platforms installed; two SHARC and one Blackfin.

I'm going to select the folder with the 533 platform and select this XML file. This XML file is a text file which describes the capabilities of the target hardware to VisualAudio. It contains, for example, the number of inputs and outputs, the sampling rates supported, what "TickSizes" or block sizes the layout supports and so forth. After I've selected this, I click "OK" and what VisualAudio does now, is it goes through a list of audio module directories and it goes through each of the audio modules within the directories and finds the ones that are compatible with the selected Blackfin 533 processor.

Now what you have on the left is what's called the audio module pallet. It's a tree view of audio modules that work with this processor and these are categorized into different folders. On the right hand side, I have what's called the layout window. This is where I'm going to drag and drop

audio modules and edit the audio processing. On the left hand side of the window here are four triangles. These are the four analog inputs that the EZ-KIT platform has. If I scroll over to the right hand side of the window, you'll see additional triangles on the right hand side. These six triangles are the six analog outputs that the platform provides.

So let me scroll back to the left and I'm going to design a simple stereo audio processing layout. The first thing I'm going to do is I'm going to take the first two analog inputs and I'm going to convert them to a stereo representation, so each of the triangles here represents a mono audio channel. They get combined into a single interleaved stereo format. The advantage of the stereo format is that it leads to more efficient implementation in some cases.

The next thing I'm going to do, I'm going to drag out a volume control and I'm going to expand this out a little so you can see it a little better. So there are two volume controls, these have built in loudness compensation and there are two versions. There's the mono version and if it ends in an "S" that means it's a stereo version. So I'm going to drag out the stereo version, drop it on to the layout editor and then I'll connect them together. You'll see the different colors for the mono and the stereo wires. And I'm going to continue scrolling over here and select a few more modules to add. The next thing I'm going to do, is I'm going to go find tone controls, I'm going to find a bass tone control, and a treble tone control and I'll connect these together.

And the last thing I'm going to do is to implement a peak limiter on the output. Peak limiter prevents the output from clipping or overloading the output circuitry. The peak limiters are composed of couple of modules. First of all, there's a maximum absolute value module. This takes the stereo input and computes the maximum absolute value on a sample-by-sample basis. And then I go into this module called the AGC limiter core. This module, given the maximum absolute values, computes a gain that should be applied to the output signal in order to prevent limiting. And the last thing I do, is I'm going to connect this multiplier. So this multiplier takes the gain signal from the limiter core and then applies it on a sample-by-sample basis to the stereo signal here. So these three modules together form the peak limiter. And finally, I'm going to convert this back to a mono representation. And I'm just going to give myself a little bit more space here and I'm going to increase the number of pages that I have across here. Okay now I have a little bit more room and the last thing that I'm going to do, is I'm going to take this stereo signal and go back to two mono channels. Connect this, now I have two mono signals and I'm going to connect those to my outputs. I'm connecting the first output and then connecting the second output.

So now I'm all done. I'm done instantiating the audio modules and also wiring them together. What I can also do, is I can open up, if I double click on a module, that opens up what we call an inspector. An inspector is a way of setting the audio module parameters. So you can see, for

example, I opened the inspector for the bass tone control. You'll see a couple things you can adjust. You can adjust the smoothing time, that's a time in milliseconds over which a change to the bass tone control should take effect. There's a gain here in dB. This is how many dB of bass boost you want to add. It's between plus and minus 9 dB. Then finally, there's a tone frequency. This is a frequency in hertz over which the bass tone control should operate. So for now I'm just going to leave the default settings.

Sub-chapter 3c: Generating Code

The next step is to generate code. I'm going to click on this button right here on the tool bar and generate code. When VisualAudio generates code, it runs a routing algorithm to determine what order the modules should be executed in. Next, it creates a data structure for each of the modules in the layout. It creates a list of the run order for the modules, and then finally, it writes all of this to disk.

Next, I'm going to switch over to VisualDSP++. Now each platform has an XML file that provides the capabilities of the platform to VisualAudio and then there's also an associated DPJ file. That's a VisualDSP++ project file that's used to build the executable. All the platforms provide such a DPJ file and out of the box, they can build the project and have it run on the EZ-KIT.

Sub-chapter 3d: Building the Executable

So right now I selected "Build." It's going through building all of the files, compiling the source code and when it's done it's going to link. The processor is running and I'm going to tell it to stop the running program and load the program just built. So that will load the executable that was just built and finally, I'm going to tell it to start running.

Now the audio is running in real-time on the EZ-KIT. Let's switch back to VisualAudio and try a few things. What you'll notice here, is VisualAudio, the background here is white, that means it's in design mode. In design mode you can create new audio modules and wire them together.

Sub-chapter 3e: Tuning

What I'm going to do now, is I'm going to switch to tuning mode. I'm going to click on this tool bar button. Now when in tuning mode, when I make a change to an audio module, the changes occur not only on the PC, but the changes are also sent to the DSP in real-time using the background telemetry channel in the emulator. And so you'll be able to listen to the audio and make changes in real-time.

Let me start the audio here. Here you have Beethoven's Symphony Number 9. I'm going to begin and start off with the volume control and I'm going to set the gain of the volume control in dB. So as I move this, you'll hear it gets quieter and louder. You'll also hear that the changes are

without clicks, so in fact, many of the audio modules such as the tone controls and the volume controls are implemented with automatic smoothing. So you can make changes without worrying about clicks being introduced. Let me go ahead and open the treble tone control. This is the treble tone control and I can adjust the gain. You'll hear the amount of high frequencies increasing here. There's 9 dB more of high frequencies and going back down decreasing it. So you can see seamless control of the audio processing in real-time. Let me go ahead and turn the volume down a little more.

What you'll also notice, is down in the lower right hand corner of the window, there's some status information that is communicated back from the DSP. Platform status, a value of 0 means that the DSP is running in real-time. It displays a current sample rate that's 48 kHz and it also shows you the current DSP loading, in terms of the percent of the CPU dedicated to the audio processing. Right now we're about 4.37 percent of the DSP. And finally, there's the peak MIPS usage. So the peak usage recorded was about 4.47 MIPS. Just want to point out a couple of things. So this is running in real-time now on the Blackfin. Its using the optimized library of audio processing functions and you can see that a layout like this with a number of audio modules actually does not consume a large amount of processing on the Blackfin. So VisualAudio can indeed do very large processing layouts and do them efficiently. I'm going to turn off the audio and then switch back to my presentation.

So far the presentation was on the EZ-KIT development hardware. VisualAudio has the capability, also, to migrate to your target hardware. What you essentially do is you begin with a reference platform like what we have for the EZ-KIT. And, in fact, we provide all the source code for the real-time framework, real-time I/O and so forth. What you do is you take our source code and you basically write some drivers for your target hardware. So you customize it for your target hardware. Next you create this platform file, which is a text file, that describes your hardware to VisualAudio and then you can continue to use VisualAudio on your target hardware. So everything I did today, in terms of, designing a layout, generating code, building it, running it in real-time and tuning it, you can, in fact, do on your final target hardware. So there's no disconnect between the EZ-KIT and the target hardware. All the features of VisualAudio will continue to work on the target hardware.

Chapter 4: DSP Software Architecture

Sub-chapter 4a: Relationship to VDSP

Now, I'm going to talk a little bit about the underlying DSP software architecture. First of all, how are VisualAudio and VisualDSP++ related? Let me begin here on the right hand side. There's a box labeled VisualAudio designer and all the boxes on top are inputs to VisualAudio Designer. First of all, there's a set of audio module XML files. These are text files that describe the capabilities of the audio modules to VisualAudio. There's also a layout file and a system file. The

layout file contains a graphical design and the system file contains miscellaneous information. In addition, in the middle here, you'll see there's a section called "platform" and there's Platform XML file. Again, that XML file describes the capabilities of the target hardware to VisualAudio. VisualAudio takes that as input and using all the inputs, when you generate code it creates a set of C and header files that are linked in together with the VisualDSP++ project. So underneath the platform section here, you'll see the VisualAudio DPJ file. That automatically attaches the C and header file that VisualAudio generates and it also includes a bunch of platforms sources and libraries. It also links in the audio processing functions and there's also a library called the "layout support library" that allows the processing to run in real-time. VisualDSP++ takes all these as input, links them together and then creates the executable. The executable is loaded by VisualDSP++ and is running on the target hardware. So that's the relationship between VisualAudio and the VisualDSP++ development environment.

Sub-chapter 4b: Audio Module Library

Now I'm going to talk about the audio module library. In VisualAudio, the term audio module, refers to a subroutine for processing PCM audio. So there's PCM audio in/ PCM audio out. We provide 89 standard modules with the Blackfin processor and 94 standard modules for the SHARC processor. We have a wide range of standard audio processing functions. There are mixers, filters, delays, tone controls, basic math such as addition, subtraction, multiplication and so forth. There are faders, balance controls, volume controls, compressor limiters and so forth. It contains a wide number of audio modules sufficient to develop many products. All the audio modules are optimized for SIMD execution, so they're written in hand-coded assembly language and they try to take full advantage of the processing capabilities of the Blackfin processor. Some modules have separate versions from mono and stereo inputs and we do this in order to be able to further optimize the code using SIMD. For all the modules we have in the standard module pack, we also provide source code. This is very handy because you can also write your own modules and expand the collection of audio modules in the library. And the source code is very handy to serve as a starting point for a particular module you may be doing. You may be designing a new filter architecture, so you might start with an existing filter module, make the changes and include that with your library.

All the audio modules use block processing. So each audio module operates on a block of data rather than sample-by-sample. And the number of samples per block is fixed and is called the TickSize and you can, in fact, adjust the TickSize using the VisualAudio designer GUI. Let me just switch back here for a moment and I'm going to go back to design mode. What you'll see right here, is this drop list shows you the set of available TickSizes for this platform and this platform supports all the way from 8 TickSizes to 2048. So depending upon your application, maybe you're trying to balance you're trying to reduce latency through the system or memory requirements, there's a trade off between TickSize and MIPS and memory usage. Now all audio

modules in the layout operate at the same TickSize, and the TickSize is adjustable through the user interface.

Block processing is a natural fit for audio decoders, which output blocks of data. For example, Dolby digital outputs blocks of 256 samples each. MP3, DTS and so forth also output different size blocks. What's nice about block processing is it yields a very efficient implementation. So we can, in fact, spend quite a bit of time optimizing the inner loops and ending up with a very efficient audio module library. It also leads to modularity, so that each audio function can be a stand alone separate subroutine.

Here's an example of the computation required by 10^{th} order IIR filter. So this is a cascade of five biquad sections. On the top you'll see the computation required for the Blackfin implementation. On the left hand side is a number of clock cycles required per audio sample process. And the X axis here is the TickSize, that is, the number of samples per block that's being processed. What you'll see is, as the TickSize increases, you get a more and more efficient implementation. So essentially, there's less overhead per module call as the TickSize increases. What you'll see, however, is a computation required quickly flattens out. So that by a TickSize of roughly 32 samples, there's very little improvement increasing the TickSize beyond that. Another thing I want to point out is you'll see that the Blackfin requires more cycles per sample than the SHARC. The reason is because the Blackfin is a native 16-bit processor and we do all the calculations in VisualAudio using double precision, so 32-bit processing. And you'll see that the SHARC is roughly 3 or 4 times as efficient as a Blackfin processor.

Sub-chapter 4c: Interconnections/Wires

Now let's talk about interconnections between audio modules or what we term as "wires". You saw today that there were mono and stereo wires. A mono wire contains TickSize audio samples. So it's a continuous block of TickSize, in this case, 32-bit fractional samples on the Blackfin. Stereo wire, on the other hand, holds interleave data and contains 2 times TickSize audio samples. So you can think about it as being interleaved left, right, left, right, and so forth. There are also a few more data types that I didn't demonstrate today. Control wires and frequency domain wires are new for the next release for VisualAudio coming out this Fall and there's also going to be a set of audio modules that use control wires and also a set of audio modules which utilize the frequency domain data types.

Sub-chapter 4d: VisualAudio Platforms

Now I'm going to talk about VisualAudio platforms. Each of the platforms is a light-weight interrupt driven real-time framework. Each platform provides double buffered DMA driven audio I/O. So that's audio I/O coming from the A/D and D/A converters on the board. There's also an interface to the VisualAudio-generated audio processing that's called the layout library. So the

platforms calls into the layout library which knows how to call the audio processing functions. Each platform also provides a separate non-real-time control thread. So this is handy for doing control within your product. So maybe you have a physical slider or a knob that's attached to your product and you want to be able to use that control for making changes to the audio module. We recommend that that's done in a non-real-time control thread. There's also an interface for tuning, so tuning allows me to control the hardware in real-time while the audio is being processed and in some cases, there's also a separate host communication library that allows you to communicate with the host microcontroller, typically over SPI. We have several application specific variants. We have one that's called a basic. It's a general purpose platform which provides PCM I/O – analog inputs, analog outputs. We also have two other variations. There's one called AVR. This one is specifically for home theater products with decoders and there's also an automotive version which introduces network interface for MOST and also sample rate conversion.

Let me talk about the basic and AVR platforms in a little more detail. The basic platform is kind of the entry level VisualAudio platform. This is, in fact, the easiest one to understand and often a good starting point for products. It's targeted at PCM-based products; that is products with analog inputs, analog outputs and no audio decoders. We've divided the platform into two parts. There's a common core framework and there are platform specific drivers. So if you're migrating this basic platform to your target hardware you have to update the platform-specific drivers. So you have to write drivers for your particular A/D and D/A converters. It provides double buffered DMA driven, block-based audio I/O. On the diagram here, you can see that the multi-channel audio codec's or the S/PDIF transceiver comes in through the serial port. It's set up to receive a block of audio data and when a block of data has been received, an interrupt is generated that triggers the audio processing. We run through the audio processing layout, create the audio outputs and then these are sent back out through the serial ports. The audio processing layout executes at interrupt level and finally, there's a separate thread for tuning host communication and for that user control code which executes at non-interrupt level. So the way I think about it is the user control code is always running except when it's being interrupted by the audio processing. This segregation into two different threads is really useful because it ensures that anything you do in the control code thread won't starve the real-time audio processing thread.

Now let's look at the AVR platform. Starting on the top left here, there's a S/PDIF input. This is typically for compressed audio such as that coming from a DVD player. It's received through the serial port interface and then it goes through a bitstream detector. Bitstream detector is a module which looks at the incoming data stream and determines, is it uncompressed PCM audio or is a compressed format, such as Dolby Digital or DTS? If a compressed format is detected, the appropriate audio decoder is allocated and the audio decoder is called. So this might, for example, be Dolby Digital operating in 5.1 mode. This will create six audio outputs that go to the

audio processing layout. Within the audio processing layout, you can design your audio processing. Maybe you'll add volume controls, some spectral processing, EQ's, limiters and so forth. And then the output of the audio processing goes out through the serial ports out to multi-channel D/A converters. So this is a typical AVR platform setup. You also see the two boxes here. This is again, the real-time portion that's triggered by the serial port interrupt. This runs in real-time and there's a separate user control thread which uses all the residual computation. Again, within the control thread, you do product control, host interface, and also, the tuning interface.

Now let me talk a little bit about how the real-time platform interfaces to the layout. On the left hand side, I'm going to start with the real-time audio I/O. So the platform essentially buffers up the audio into individual audio buffers. There's one audio buffer per mono input to the audio processing layout. So in the example before, we had a total of four inputs and six audio outputs. So there are four inputs here. Four buffers, one for each audio input. Next you call in to the VisualAudio layout support library. This works in conjunction with the audio module render functions. These are the real-time functions for processing data and the layout support library also looks at the audio modules data structures generated by the VisualAudio Designer application. So it runs through the list of audio modules to render, calls the appropriate real-time functions, processes the audio, and then returns it back in place into these input output buffers, and then these buffers are passed back into the real-time portion. What's nice about this sort of clean interface is you can, in fact, use the VisualAudio layout support library and the generated code from VisualAudio with your own platform and so forth. So maybe you have an existing audio processing platform. All you need to do is ensure that the data is buffered up into separate buffers and then you call into the VisualAudio layout support library and you can add real-time audio processing capabilities to your audio product.

Chapter 5: Conclusion

Sub-chapter 5a: Summary

This concludes my presentation on VisualAudio. In summary, VisualAudio accelerates the development of embedded audio applications. An intuitive graphical user interface allows audio processing to be easily designed and configured. Today's demo utilized the Blackfin 537 EZ-KIT. VisualAudio supports both the Blackfin and SHARC families of processors and also works with many different EZ-KIT development platforms. VisualAudio also generates very efficient code, in terms of MIPS and memory usage. This was an introductory training module and there's a separate training module which covers the VisualAudio environment in more depth, discusses advanced user interface features, writing custom audio modules and also interfacing to external design applications.

Sub-chapter 5b: Additional Information

Additional information on VisualAudio can be found in a number of places. First of all, a free download is available from the VisualAudio product page shown here. This includes the VisualAudio Designer application, the EZ-KIT platforms, the audio module libraries, and also full documentation. Additional examples and tutorials can be found at the VisualAudio Developers website shown here. Specific technical questions can be sent to the support email address shown here and finally, you can also click the "Ask a question" button.

This wraps up my presentation of VisualAudio. I'd like to just thank you for your time and attention today. Thanks again.