Presentation Title: Introduction to the NI LabVIEW™ Embedded Module for ADI Blackfin® Processors

Presenter Name: Glen Anderson

Chapter 1: Product Overview
Subchapter 1a: Introduction

Hi my name is Glen Anderson I'm a staff engineer with Analog Devices Core Development Tools Group. And today we're going to be talking about LabVIEW Embedded for Analog Devices Blackfin processor. We're going to go through a number of demonstrations today. We'll show you the product, we'll go through all the different features and really try to give you a good sense of what this product does, who this product is aimed at, and by the end we should have a good over view of LabVIEW Embedded .

First we'll start off we'll talk a little bit about just an overview of what LabVIEW Embedded for Blackfin is. I'll talk a little bit about LabVIEW the base product from which is was derived. We're going to develop some applications, there will be a number of demonstrations today. We're going to build an application, show you how to download, how to run these virtual instruments on the Blackfin. We'll look at drivers, peripheral communication using the built in driver VI's. And we'll also look at debugging capabilities, that's the ability to set breakpoints in your virtual instruments, set probes and debug interactively through the VisualDSP traditional C development environment as well.

Subchapter 1b: What is LabVIEW?

So the first thing we want to talk about before getting too deep into LabVIEW Embedded, is what is LabVIEW? LabVIEW itself is a product and it was developed by National Instruments Incorporated. NI is a company that has traditionally really been firmly cemented in the test and measurement world. Any type of signal that you might have chances are they have some sort of hardware device that will allow you to measure and test that signal. Early on what traditional test engineers had to do was they would write typically C applications to GPIB enabled devices in order to automate these devices and be able to test. The LabVIEW itself is this graphical development environment which was developed again by NI that allows engineers not only C programmers, but also engineers that really know test and measurement but not necessarily embedded development to be able to test their device whatever it might be. So LabVIEW is this graphical programming environment it's just sort of drag and drop data flow paradigm where I have these virtual instruments I can just arrange them on the screen, wire them all up and not necessarily need to know all the details of how to program a particular device be it in a osciliscope or some other signal measurement device. It's traditionally targeted to desktop environment. In recent years they have branched out into more mobile applications. They have this Compact RIO framework now, which allows you to take your measurements into more mobile

applications. They're found world wide in industrial type settings and recently in embedded applications. And that's really what we're going to be talking about today. Being a mature product LabVIEW itself has thousands of these out of box signal and mathematics processing functions and connectivity functions with various types of hardware. Really its main stay is this connectivity that it has with all sorts of different standards be it vision systems or data acquisition type systems.

Now what the folks at National Instruments have done is they've suddenly realized that there's this emerging trend that has developed where even going back to the test and measurement world where you had these measurement engineers that didn't necessarily know C but they had this nice easy to use LabVIEW environment where they could program their devices. They were really these domain experts, they knew how to test their particular application. They could use LabVIEW to do that easily. There is similarly a class of embedded developer out there, again a domain expert, who knows a lot about their particular application but not necessarily implementation details like programming in C and interrupts and registers and a lot of the things that real embedded developers tend to face. So LabVIEW Embedded is this LabVIEW graphical environment that is now expanding out to be able to target these embedded devices. And the first product it's a joint product called LabVIEW Embedded for Blackfin, it's a joint product between Analog Devices and National Instruments that now allows you to have this graphical development environment that has existed for years in the test and measurement world, you can now take that same environment and target the Blackfin processor. So we're really spreading out into a whole new territory here, this embedded development realm. The embedded module for Blackfin itself again it leverages the design capabilities and this graphical programming that people have come to know with LabVIEW. It targets the Analog Devices Blackfin processor, which is a high performance low power device. The idea here is that we want to integrate all the way from the beginning of the design cycle all the way to final production ready code without having to leave this LabVIEW environment. The fact that we have these virtual instruments there blocks which I'll demonstrate in a little bit, we have these pre-made blocks so that I don't need to reinvent the wheel, I have all this functionality out of the box, I don't need to know a lot of the details ah of how to program a Blackfin.

Subchapter 1c: Who can Benefit?

I mentioned the word domain expert already, we can talk about who, who can benefit from LabVIEW embedded. The domain expert is a person again who really knows a lot about their particular problem that they want to solve. They know be it an algorithm or what you, they don't

necessarily know how to program an embedded device, nor do they really want to know how to program an embedded device. LabVIEW is really a great product for allowing the domain expert to stay sort of at this high level of abstraction doing what they do best which is their algorithm or their particular problem that they're solving. Embedded developers, these are the traditional C developers such as myself, there's a lot to be gained through LabVIEW in that again I don't need to reinvent the wheel, I'm going to have a faster development cycle because I have these reusable components, I have these drivers I have these algorithms, a lot of these things are provided for me out of the box through LabVIEW. Test engineers and quality engineers again we talked about how LabVIEW really comes from the test and measurement world, so as such there are companies that just have a tremendous investment in LabVIEW technology already. LabVIEW Embedded is now a way to leverage that existing investment in LabVIEW and it'll allow you to extend that into parts of the design cycle that you really couldn't target before this product. So quality engineers, you know, using a consistent set of tools if I use LabVIEW for my development phase, I use LabVIEW for my test and measurement phase, I can even use LabVIEW at the beginning of my design cycle for my prototyping stage. This consistent use of tools through out the chain of development does lead to better quality because it's not like I'm trying to spend time integrating these different tools together that weren't necessarily meant to work together in the first place.

Subchapter 1d: Architectural Overview

From a high level LabVIEW Embedded is pretty simple. You can see here I have my customer application, that's the diagram that I'm going to draw, we'll do some examples in just a little bit. LabVIEW is going to take the diagrams that I draw, under the hood it's going to generate C code, this is just standard ANSI C like a traditional embedded developer might write. Under the hood this gets fed to the VisualDSP++ tool chain, that's our Blackfin optimized compiler, assembler, linker, loader, splitter tools, but again it's all under the hood. I don't actually drop down into the VisualDSP environment, I tend to stay up in LabVIEW where I'm comfortable. You can see here the generated code gets fed into VisualDSP, that gets linked into these component drivers, which are built using the VisualDSP System Service Libraries. And it also gets linked into a number of these Blackfin optimized VI's, again VI's are Virtual Instruments. So together these are really those reusable blocks that I was mentioning earlier. And then that's all linked together using our VisualDSP++ kernel, or VDK that's the operating system under the hood that ties it all together and schedules all the different tasks to run, and then that collectively produces an application which I can then download onto my Blackfin processor.

Subchapter 1e: Demonstration Setup

What I have today is a Blackfin EZ-KIT Lite, an EZ-KIT Lite, this is Analog Device's standard evaluation platform that we have as you can see here. What I have in front of me here is a ADSP-BF 537 processor running at 600 megahertz, you can see here some of the technical specs of this particular board, 64 megabytes of SDRAM, flash memory, there's a stereo audio, analog to digital and digital to analog converters. There's a high performance USB debug agent, that's a way for me to debug my Blackfin and be able to download my code and step and run. It has integrated Ethernet and CAN, and it also has this educational laboratory virtual instrumentation suite, or Elvis, this is an NI term, which is a PCI like interface on the side of the card which I can use with a special adaptor that's supplied with the product I can connect to a full range of, of NI data acquisition boards and different types of hardware. There are also connectors on the bottom of the board for these EZ-Extender cards. There's different cards to do audio and video and Ethernet and USB and other types of peripherals depending on my application, the board's expandable to allow me to you know, really target my end application using this low cost evaluation system.

The demonstration set up that I have here is we have the computer, again I have LabVIEW and VisualDSP loaded on this, this is the LabVIEW Embedded for Blackfin product, those two together. I'm connected here using this HPUSB J TAG emulator, this is a high performance USB 2.0 emulator that allows me to download and connect to the board. I'm using this here more for speed, again this a USB 2.0 device, the product does come with this board and what I mentioned before that high speed USB debug agent, that's a USB 1.1 connection. So again out of the box you get full connectivity, it's a little slower being USB 1.1 versus USB 2.0. So that's how I'm connecting from my PC to the evaluation board, and then from the evaluation board here I have an audio line in, going into the codeec on the board. And then I have the line out of the board going to speaker that I have here, we'll be doing some audio examples in just a little bit. So that's the setup. The best thing to do I think is to let's actually start in and do some examples to show you what LabVIEW Embedded really looks like.

Chapter 2: Developing a Simple Application
Subchapter 2a: The LVE Environment

What we have here, this is the LabVIEW 7.1 Embedded Edition window, when I first start LabVIEW this is the first window that I see. If you've ever used LabVIEW before this looks very similar to what you would normally see with the desk top version with the exception of the words

Embedded Eddition on this splash screen here. But one of the main differences is this execution target that we have down here. By default this says LabVIEW for Windows, which means that anything I develop in this mode is traditional LabVIEW desktop code, similar to what's been available from National Instruments for years and years. LabVIEW Embedded allows me to target other processors, in this case we have the ADSP-BF533 and the ADSP-BF537, again I'm connecting to a 537 here, so I'm going to select that. And the first thing I'm going to do is I'm just going to start off with a blank virtual instrument, or VI, so I can select that here, and I'm just going to tile these. Every virtual instrument has two parts to it, there's a front panel, which is this gray over on my left, and then there's the block diagram, which is the white window over to the right. In C terms your front panel can almost be thought of like a header file, this is where I can define the inputs to my virtual instrument and the outputs from my virtual instrument. The block diagram is really my source code, this is where I really do most of the work, be it developing an algorithm or a device driver, or whatever it is that my end application is doing. We're going to take a look at the block diagram, we'll get to the front panel in a little bit, but for right now let's take a look at where I actually develop my code. Now LabVIEW is a graphical drag and drop data flow language. That means that rather then typing code into a text editor I have these pallets, you can see here this is my function pallet. Now similar to any other language you have all the constructs that you might be already familiar with, for example here's a for loop, here's a while loop, a case structure is like an if statement, sequence structures are like curly braces in C, local variables, global variables, comments, inline C nodes and we'll talk a little bit about inline C nodes in a little bit. And then again we have numerics, this is where my addition, subtraction, multiplication, we also have much more involved blocks such as the analysis library, here's some signal processing functions, signal generation for example. Here's if I want to generate a sine wave I could drag this block and drop it there.

Subchapter 2b: Blinking an LED

The first thing I want to do is select my Blackfin palette and you can see here I can choose to view all of the different Blackfin virtual instruments that are available, so now when I right click a lot of it looks the same, you can see I still have my structures and my numerics and my signal processing over here. I also have this Blackfin pallet now, this is really what the output of this joint venture between Analog Devices and National Instruments really is, is we now have these block sets that allow me right out of the box to do a lot of pretty sophisticated and very efficient things specifically for a Blackfin and we'll talk a little bit about that. Just for example we can take a look again I'm connected to an EZ-KIT Lite evaluation board here, you can see I have an EZ-KIT palette, I can select the LED, there's a number, 6 LEDs on this particular board, and you can see

I have some blocks here, I have a block that'll let me query whether or not the LED is on or off, I can cycle LEDs, I can toggle the LEDs I can turn it on and turn it off, I'm going to toggle my LED here so you can see I can select that. And I just drag it and drop it. Again I have a 537 here so I'm going to select the 573 version of this block, and you can see it has some inputs and some outputs here. Inputs are on the left side of the block, outputs are on the right side. I have an LED number, again I have 6 LEDs here, so those are going to be LEDs I believe probably 0 through 5. I can right click here let's say I want to blink LED, toggle LED number 1 so I can just create a constant and so now when this is executed, LED number 1 should toggle, meaning if it's on it'll go off, if it's on it's going to come off. The other inputs and outputs that we have are error in and error out, this is a pretty common LabVIEW paradigm for passing error information in and out, you can almost think of it as a return value from a function. I'm not too worried about errors here being we're only toggling LEDs so I'm just going to leave those unwired for now. What I can do is lets say I want to toggle these LEDs on and off sequentially, I can take a while loop and just by surrounding this toggle LED block with my while loop I can have the LED go on and off and on and off. There's an exit condition, it's called the loop condition here in the lower right hand corner, and I can wire up conditionals to that depending on how long I want this while loop to go for. I just want this loop to go forever so I'm going to wire a constant false value to this so this is just effectively going to go on forever. If I were to run this right now it's just going to go at full speed, so we probably won't even be able to see the LEDs blinking in that case. So I want to put a little weight in here, and you can see I can go down into my time dialog and error palette and there's a wait VI here, so I'm going to put my wait, and again I'm going to create a constant, and let's say I'm just going to wait 500 milliseconds. Again my whole application up to this point is pretty simply it's just I'm going to toggle LED number 1 in a continuous loop and in between toggling I'm going to wait for roughly half a second. So that's my application simple as it may be, this will be a good place for us to start.

What I can do is I'm going to hit run here, first thing I need to do is it's saying I need to save this. So I'm going to call this LED example, and will click OK. So now I've saved the VI and it's telling me that I need an embedded project in order to run this application so I haven't created a project yet, so that's what we're going to do now. So I'm going to say okay I want to create a new project, and it comes up with a good default name being LED Example. So I'm just going to take that as my default, and so now we have this, this is the embedded project manager window. This is similar to other IDEs that you may have used in the past, this is where I can add all my virtual instruments and manage all my code in the build options. Take a little look at what's going on here, see I can create new projects, I can open existing projects, I can close the project I have open now, save, save all, these are all pretty standard. The Target menu is where a lot of the

interesting stuff happens. You can see I can build my application, I can rebuild all, that deletes everything and sort of starts from scratch. I can download my finished application to the Blackfin. I can reset the Blackfin, I can run, I can debug, and we'll do both of those in just a little bit. I can set various build options, and again this brings up the build options dialog. I can set different things that tell LabVIEW how to generate code, I can tell it whether to generate this guard code which saves me from things like dividing by zero, or over-indexing an array. I can select different debug modes, these are more advanced functions here. If there's specific VisualDSP optimizations that I want to tell the compiler to make I can add those here. And if I have my own custom Blackfin hardware I can come in here and tell LabVIEW exactly what kind of hardware, what kind of Blackfin I have. I can say what the clock speed is, what my voltage is, my core voltage, my external voltage, what kind of package it is, also what kind of silicon revision it is. Now LabVIEW automatically detected that I have rev 0.2 silicon here, if I had an earlier board I would want to tell LabVIEW that so that the compiler can work around certain silicon anomalies that might be available or might be seen in revision 0.1. I'm not even going to save these, I'm just going to take the defaults as they are and we're just going to build, so I can do that by going to the target menu and selecting rebuild all. Now what LabVIEW has done here is it's taken this diagram, it's generated C code and now you can see it's actually feeding it to the VisualDSP code-generation tools, the compiler, linker and assembler in this case under the hood. Again I don't see VisualDSP any where in this process, LabVIEW is what's handling all that under the hood for me. So I've successfully built, the next step would be to down load my application on to the Blackfin so let's do that. So you can see that LabVIEW is now talking to my Blackfin again over this high performance USB emulator, it's attaching, now it's downloading the code into Blackfin. So that's done. So we've built, we've downloaded, the next thing we want to do is just try running and we'll see what happens. So I hit run and low and behold I have my LED 1 is toggling, roughly every 500 milliseconds or so all the other LEDs are off, so that's good. We stop and we think about what just happened, granted blinking LEDs isn't necessarily rocket science, however think about I would have needed to do in a traditional C development environment in order to do exactly this. We have a timer here, this 500 milliseconds, that means I would need to some how hook into the timer interrupt on the Blackfin, get out my hardware reference manual, figure out what registers I need to write to, I'd need to write an interrupt service routine, I'd need to some how maintain the state of that LED, I'd need to configure the LED whether it's in input or an output those are different configuration registers on the Blackfin. So it really required me to have a pretty in-depth knowledge of the Blackfin in that case. Doing what I just did here I had you know, no knowledge of the Blackfin in this case, all I did is I just knew I had this LED toggle and a wait block.

Subchapter 2c: Parallel Loops

That's all well and good here's where things get more powerful, let's say for example okay blinking one LED is fine, let's say I wanted to blink more than that, I can simply copy this block, and I'm just going to make a number of copies here, we'll say 4, that's a good number. So I just made 4 copies of this block, let's say I want LED 2 we're going to blink that at 750 milliseconds, LED number 3 we'll blink a solid second, and then LED 4 we'll blink at a second and a quarter. Again I'm just going to go back to the project manager here save what I've done, and we're going to build, it's telling me to save. Again it's produced the updated C code, it's fed it to the VisualDSP compiler and now it's linking my final application. That's all done, so again I'm going to download on to the Blackfin and now when I run it I see 4 LEDs, LEDs 1, 2, 3, 4 they're all blinking, it looks roughly right to me. Now if we stop and think for a minute again going to back to what I was saying earlier, we did a lot just blinking that one LED that would have been pretty tedious to do in C, now doing in that in four parallel loops all at the same time, all with different timer values, all with different LED values, see all I did is I just copied a few blocks around and built and I was all done. To do that in C is actually tremendously complex, that requires me to almost have some sort of operating system under the hood to be able to manage these parallel tasks or threads in VDK terms. And again here I didn't need to know anything about threads in this case, all I knew is I have four different tasks that I each want to run concurrently so I'll just have four different while loops all on the same diagram. Again it abstracted me from having to really understand all the low level details of what's going on in VDK and the on the Blackfin hardware. That's one example.

Chapter 3: Peripheral Communication
Subchapter 3a: An Audio Example

Now blinking LEDs, like I said, is not terribly clever real world example, what I mentioned earlier was there are a number of really powerful VI's provided in LabVIEW Embedded right out of the box. We're going to take a look at one of those now. This is an audio example, so again my set up here is I have the audio in to my evaluation board coming from my desktop PC and then I have the audio out going to my speakers. So let's take a look at this, again there's only one VI in this project, and it's considerably more complex then my blinking LED example, still no where near as complex as what this would be if I were to write it in C. Now we take a look at this, LabVIEW VI's are typically left to right, so we can start off here, you can see I allocate some buffers here. Looking at this icon I know it's an allocate buffer but other folks might not. One of the really helpful features in LabVIEW is context sensitive help. So I can bring up this context help window and just by hovering my mouse over different blocks I can get help for each of these

blocks. So it's a good way of sort of navigating my code and understanding what I'm looking at. As you can see here this is an initialize array block, I have a 512, this is just a constant, so I'm saying create an array of 512 bytes. This is coming down here I'm taking the 512 dividing by 2, so this is 256 bytes, I'm creating a couple of arrays there. Let me close this. I initialize the Blackfin audio here and you can see one of the inputs is one of the samples that I want to collect as my audio runs. And then it gives me this receive buffer and a transmit buffer, this is where while the data is coming in and going out and I'm processing the data, these are the wires that it happens on.

So from here we enter this while loop and again this is the same while loop that we had in our LED example a few moments ago but there's obviously more going on inside. When I enter the while loop, again I've initialized my audio here, I wait on this flag, this data ready flag. Now what the Blackfin is doing under the hood is it's constantly collecting the data and storing it in memory. Now I've pulled in this init audio block over here, I've passed in number of samples being 512. So what this says is when 512 samples have come in, return. So just wait here until I get a full buffer of data and then as soon as that data buffer is full, continue. So once I come out of here I can get the data, and that returns this array, I separate the data, this is in this I^2S mode, which is a standard audio format. I convert that one big buffer into separate right and left audio channels, you can see here I have right out and left out, and I'm not really doing anything with the data just yet I'm just passing it straight into this combine block, you can see BF combined audio channels, so I'm giving it the right and the left inputs and then I'm sending that as just a solid buffer coming out. And then I send that out to the chip and this block here says that I'm now done and to go back and start waiting for the next 256 samples to come in. So if I go back to my project manager, and let's build this, and let's save this, it's telling me to save that, so now I've completed the build and I'm going to again download, okay the download is complete. Now I'm just going to run and what we should hear is if I go and start playing my audio I can hear Beethoven Symphony Number 9. So I'm going to stop that.

Subchapter 3b: Front Panel Controls

I'm not doing anything with the data once I get it, so let's do something a little more interesting. The data comes in here in one big buffer, I'm separating it into right and left channel audio. So what I can do is I'm going to remove these two wires and I'm now going to goto my front panel, we really haven't talked about the front panel yet, earlier I mentioned that the front panel is where you define the inputs to your virtual instrument and you define the outputs to your VI. It also does a bit more than that, you can see the pallet here on my front panel is different from the pallet that I

saw on the block diagram, I have a number of different controls, see I have buttons they're almost graphical user interface elements here, string controls, I have charts and graphs and different types of data visualization options here. I'm just going to select a numeric control and we're just going to call this gain. You can think of this as a variable, and right now my gain is zero. Now when I drop the gain block on the front panel and my block diagram automatically appeared sort of a related block that represents it in my code, you can see it still ahs the same name gain here. Now LabVIEW has the notion of types similar to traditional programming environments. I can see that because it's orange that means that this is a double , it's a 64 bit floating point number. I can choose to change that representation and let's say I want to make this a 32 bit integer, or a long, I can select that and you can see it now turns blue which indicated it's an integer and it's an I32, which is what I want. Let's say I want to, I'm just going to multiply, the sample is coming in by a constant gain factor, so I'm going to right click, go to my numeric palette, again this is where we were looking at things like add and subtract and multiply and divide, I'm going to select multiply and wire this there and let's take the right channel audio and I'm going to wire that up to my multiply and I'm going to just wire that right to the output there. Now I want to do the same thing for my left, so I'm just going to hold down the control key and drag this multiplier, I can take my left audio and this get a little unwieldy, but let's see here, I'm just going to wire that to the left. So I'm taking my right, I'm multiplying it by my gain, sending it back, I'm taking my left channel multiplying it by that same gain factor sending it to the left, let's say I wanted to cross the the left and the right over, that would be as simple as just taking this and wiring that to the right and swapping them that way. Again it's this visual programming environment that really just makes these types of operations a lot more intuitive then they might otherwise be. I'm going to save this, we'll go back to our project.

Subchapter 3c: Runtime Tuning

Now a couple of things that we've done here in the build options that are a little bit different is I have my debug configuration which is the same as we had for the LED example and now I have this debug mode, there's a few different debug modes that I can set in my build options. For the LED example we had none because we weren't really debugging we just assumed it was going to work so we just ran. I can debug over the serial port, the EZ-KIT Lite has an RS232 port on it which I can just connect to the back of my PC. Or if I'm using this JTAG emulator or the built in USB connection that comes with the hardware I can select this non-intrusive mode, which is what I've done. We're going to select okay there. Now I'm going to save that and we'll rebuild, so now that's built I can download. Previously I've always hit this run button or alternately gone to the target menu and selected run, what I'm going to do now is instead of running I'm going to choose

to debug, and this is going to do a couple of different things. So now you notice that my front panel has now popped to the front and it's no longer this grid that I normally I see on it is gone, it's a solid gray. I have this abort button is now there, I have a pause button, so I'm actively debugging my application and I can interactively set different values for this gain, let me go back here and just make sure that my music is playing. Now if I select let's say a gain value of 1 I can hear the music playing again. Let's say I want to go back and select a gain value of 2, gets louder, so on and so forth. So it's almost like a volume control in that case. Now gain is one thing, this could just as well be filter parameters, let's say I'm developing a filter algorithm in LabVIEW I can have all my different filter parameters here and I can actually tune them on the fly really running on hardware and I can actually see the effects that the different values have as I'm running in real time, which is really one of the powerful things in LabVIEW that again is really difficult to do in traditional development environments where there just isn't that type of interactivity. I'm going to stop debugging here and go back.

Subchapter 3d: Device Drivers

Now the last thing we want to touch on before we leave this audio example is let's go back and take a little closer look at what's really going on here. We touched a little bit about you know, this init audio block, this wait on flag, the get buffer, a lot of these are available if I look at my pallet under this Blackfin, you see I have devices, there's a common pallet here which allows me to open a device and close a device and get the data to and from a device. But one of the real powerful features of LabVIEW Embedded for Blackfin are all these other device blocks. For example I'm using the 1854 and 1871, these two blocks here these are the ADC and DAC found on the 537 EZ-KIT that I'm using to stream my audio in and out of. So it's really as simple as I can call open I can read some data, I can write some data, I can set different parameters, let's say there's a volume register on one of the chips, I can set that using this control method. So these are drivers that I didn't have to write Analog Devices it already provided these for me out of the box, it's just one less thing that I need to worry about, it's one of these reusable components that I was talking about earlier. Now let's say I have a device that isn't represented in this pallet, some other codec for example, we have these block sets that give you access to all the on chip peripherals on the Blackfin, the PPI the SPI, the SPORT and the UART on the Blackfin in this case, so I could actually let's say I had my own codec and it was physically on my hardware connected to the PPI, I could use these low level PPI blocks and actually implement my driver for my device where ever that device is, purely in LabVIEW and I don't need to drop into a C environment to do any of this. I can use these basic building blocks, which do a lot of the complex parts of driver development like handling DMA and interrupts and all that sort of thing. So again

this is really an area where this out of box power is really a pretty wonderful thing. There are blocks here where I can control the power of the Blackfin, I mentioned that the Blackfin is a really low power high performance part, so here I can actually tell it, I can retrieve the frequency, I can tell it to run at a particular frequency, I can tell it to go as fast as it possibly can given a certain voltage, let's say that there was a wireless or hand held device, I can use that to really maximize my battery power.

And then we have these analysis libraries. Up here we saw that LabVIEW be it LabVIEW for Windows or LabVIEW for Blackfin has these built in signal processing blocks. Again here's my signal generation, here's some time domain functions, different filters, Butterworth, Chebyshev, these are all blocks that LabVIEW users have had for years and years and years, and there's a lot of existing code that takes advantage of these blocks. Having these available makes porting existing applications from LabVIEW Windows to LabVIEW Blackfin very easy. However these being very very portable they're not necessarily tuned for the Blackfin, so that's where Analog Devices has stepped up and has implemented these Blackfin analysis libraries, and you can see here here's some signal processing, here's some filters, here's a complex FIR, here's a regular FIR, IIRs, different types of filters there. These are filters that have been hand tuned by the development tools group at Analog Devices to really take advantage of the architecture of the Blackfin and to really get as much performance as possible out of there. So while I can use the existing LabVIEW blocks for portability and really just prototype and get up and running quickly that way, for maximum performance I also have the option of moving to these Blackfin specific libraries which are guaranteed to really work very well on my Blackfin.

Chapter 4: Debugging Capabilities
Subchapter 4a: Breakpoints and Probes

Let's take a look now at debugging. I'm going to save and close my project. One of the one of the things that you just can't escape with any type of development environment is the fact that there are going to be bugs in whatever code that you write. It's just one of those facts of life. So LabVIEW actually has some really extensive capabilities built in to allow you to really get down deep and see what your application is doing, whether it's doing something wrong or maybe it's doing the right thing but not performing the way you would expect it to. There are a lot of tools built into LabVIEW to really allow you to see this. So here's a debug example, again we'll just open up the front panel, see I have a couple of elements here on my front panel, we can look at the code underneath and it's not a very complex diagram but it does do a few things. So again reading left to right I initialize an array in this case its dimension size is 5 so it's an array of 5

elements of type, again this is orange so I know that's a double. I pass this array of 5 doubles into my while loop, I rotate an array and again I can just press Control H and see this context help. I shift the array by 1, I replace an element with a randomly generated number, so this is going to generate a number between 0 and 1, I multiply that by this factor of X which is on my front panel, and then I pass that to this replace array subset. So effectively I'm just generating a random number, I'm shifting my array, I'm inserting this new number at the beginning of the array and then down here I calculate the mean of the array and then I output that to this mean block here. Again you can see it, this is my X and this is my mean, so there's effectively one input and one output to this particular example. So let's say I really want to see what's going on here. I can choose to set a breakpoint, let's say I actually want to step through my application. Now LabVIEW doesn't require me to drop down into the C environment because again this is really all C code under the hood, but I don't need to drop down to that level of detail if I don't want to. I can go here and again in my build options I have this non-intrusive mode selected, so I'm going to be able to debug. I'm going to download, it's going to tell me I need to build it, so we'll do a build, okay so now we've downloaded, again I just select Debug from my Target menu or from my tool bar up here, I'll select from the toolbar this time. Now I've run and LabVIEW is now flashing this initialized array block where I set my breakpoint indicating that it's halted there, I now have a lot of the options that you would find in a traditional development environment. I can step over my code, I can step out of this entire VI, I can choose to continue sort of like a run, or I can just abort debugging all together. I'm going to choose to step over this initialize array, and you can see that brings me to my while loop, I want to step in to the while loop to see what's going on in here, and you can see I can step a few times, now I'm at my rotate 1-D array block, step one more time, so now I'm at my multiply. You can see I can set what we call probes here. I actually want to see the value, what number am I actually multiplying, so I can choose to set some probes. You can see that probe number 5 is on my value of X and that's a value of 5, which is right for my front panel. And this random number that was generating using my random number block looks like a valid number between 0 and 1 to me, so that's how I can tell what's there. Now if I were to step over this I can see that 5 times 0 well .83, 4.15 looks about right. So this allows me to really get in and I can analyze really what's going on in my code. I can choose to set another breakpoint here on my mean, I can choose to continue and now every time I hit continue it's going to stop at this mean block because that's where my breakpoint is going to go around the loop once, stop at mean, then when I hit continue it'll do the same thing again. So it really does let me get in there and see what's going on inside my application.

Subchapter 4b: Synchronized Debugging

One of the really important things about LabVIEW Embedded is that there's really the sense of investment in traditional technologies that I think that a lot of companies have. And LabVIEW actually goes to great lengths to allow you to kind of reuse a lot of that. Let me stop debugging. Let's take a quick look at some of the features that really let me get even lower level, here you can see I was stepping through my diagram I can look at the wires, I can step into different VI's and step over and step out of. Let's say that there's something going on under the hood, even at the C level that I might need to really go in and investigate. LabVIEW allows you to actually go down into the C level and debug seamlessly there. So I'm going to save this. One thing I can do is go into my target configuration, now this is a dialog I really haven't looked at, this is where I tell LabVIEW how I'm communicating to my Blackfin, in this case I'm selecting this 537 EZ-KIT Lite via HP-USB Ice, which is how I'm connected, and I have these debug options here. These allow me to really just control different debugging features. This front panel update period allows me to set how often LabVIEW goes out to that target and queries a particular value let's say for a probe window. I can tell LabVIEW to allow me to probe arrays and clusters, a cluster is like a structure in C. One of the really neat things that I can do is debug using the VisualDSP++ IDDE, now I mentioned VisualDSP a couple times earlier and how it's really the underlying technology that takes the C code and generates this optimized Blackfin application. The VisualDSP IDDE or Integrated Development and Debug Environment is the application in VisualDSP that allows a traditional C developer to step through their code. So what I can here, let's say I want to get done into the C level, I can choose debug using the IDDE and select okay, now the next thing I want to do is launch the IDE, okay so here's the VisualDSP IDDE, now there are a lot of things here that are familiar in traditional C development environments, a project group for example, I have memory windows and register windows, I'm actually connected to this Blackfin now and if I wanted to I could choose to go and look at accumulators and different types of registers and memory and I can run and step into and all that. I'm going to keep the same debugging example here, I don't need to rebuild, all I'm going to do now is just, let's go back and make sure we still have our breakpoint on this initialized array, I'm going to download. What's happening here is rather than talking to the Blackfin directly, LabVIEW is now loading the code into the VisualDSP IDDE so now when I choose to debug, you can see I'm halted here, in the IDE you can see my "Debug Example.c" which corresponds to "Debug Example.vi" here, you can see the actual C code and in this case it's right before this array initialize block here. So now I'm in my C I can choose to look at memory windows, I can open up some register windows, I can look at expression windows, evaluate different things there, the value of N step is 1 in this case, so I can get as low level as I need to. I can even step my C code a couple times here and then I can go back and forth into LabVIEW, minimize this, and I can still step over, let's see if I can show both of these at the same time. So you can see my C code here and my LabVIEW diagram here, and

I can step there and you can see that the C code actually is keeping lock step with my diagram. So that's one way that LabVIEW allows you to get into a lower level realizing that you know, there are times when it may be necessary.

Subchapter 4c: Reusing Code

One of the other mechanisms that LabVIEW Embedded for Blackfin gives you for linking into existing C code is what we call the Call Library Function Note rather, that's available in this advanced pallet. Let's say for example I have my own code in a library already that I've developed under VisualDSP, I can choose to drop one of these on, double click it, and that allows me to configure things. Let's say "My Library.dlb", DLB is the extension for VisualDSP Library, let's say I'm going to keep with the addition theme here we'll say the function name is MyAdd". I can say that the return type is a numeric type 32 bit integer, I can add a parameter and you can see the function prototype in C is available down along here, so I've added an argument called arg1 and let's say I'm doing to change that to X, which I did in my inline C node, and I'm going to add another parameter and we'll call that Y. When I click okay see the block actually changes where I have my inputs and this one is called X and this one is called Y, and here's my output which is labeled result over here. So this is another way where as I actually typed the C in here, from here I can link to existing code that I already have.

The third way that LabVIEW allows me to preserve my investment and traditional tools that I may have is in the Embedded Project Manager directly I have the ability to add files, I can add VI's here, I can also add external, you see I can actually add .C and .lib or .dlb files directly to my project. So let's say I had a library or I had a .C file, or even an assembly file, let's say I had an algorithm that I purchased which was written in assembly and I had a .asm file. I could ass this directly to my project and then using this call library function node I can call that as well. So LabVIEW really makes it easy to use these traditional tools that you may already have whether they be algorithms or drivers whatever that code may do. So that's another very important part of LabVIEW.

Chapter 5: Conclusion

Subchapter 5a: Product Details

We've talked a lot about LabVIEW itself and the different features it has, debugging, drivers, the analysis libraries. We talked a little bit about what you actually get when you purchase LabVIEW Embedded for Blackfin. The product itself is available both from Analog Devices and National Instruments jointly. You can go to either company to purchase the product. The product is supported through National Instruments, now this is actually really nice in that you have this one unified support structure through National Instruments where you can go for questions or help with your particular application. And National Instruments, let's say it turns out to be an issue that is best handled by Analog Devices, through our collaboration NI and ADI have developed a lot of these internal links now where we can seamlessly you know, triage those types of problems without having you go to different companies and have to go through different support mechanisms. So fully supported by National Instruments. With the package itself you actually get quite a bit, you effectively get a full seat of National Instruments LabVIEW. You get full seat of Analog Devices' VisualDSP++, you get this LabVIEW Embedded module for ADI Blackfin, that's the software that sits on the top of LabVIEW Embedded that gives you those optimized libraries and debug capability. You get this ADSP-BF537 EZ-KIT Lite that I've been demonstrating today. Again that comes with the built in USB debug agent so you don't need to purchase for example this high performance external JTAG ice. You get all the cabling, you get headphones, things that normally come in our EZ-KIT Lite evaluation packages. You get this data acquisition adaptor which again plugs on to this NI Elvis interface, it's says PCI interface on the edge of the card, which then allows you to connect to a wide range of National Instruments data acquisition hardware. And you also have the availability of automatic software updates through National Instruments as new releases come out and so if you need support that mechanism is available as well.

Subchapter 5b: Summary

In conclusion we've talked a lot about LabVIEW Embedded now, really the key points are faster time to market, we believe that with this higher level easier to use data flow language, by extracting the implementation from the problem itself it allows us to get to market faster. It allows us to adapt to changes and varying market pressures much faster um because of this drag and drop paradigm. We can incorporate these advanced features and functions into our product much quicker as new algorithms and new drivers are produced for LabVIEW Embedded, those are made available to you so that as your needs change in the future you have that there. And then this common consistent use of tools throughout the product life cycle is really an important

part of the whole package where traditionally LabVIEW already has a firm footing in the prototyping stage and the test and measurement stage, we can now take that into the actual development phase in the middle and this consistent use of tools from beginning of design to product release is really powerful and the quality is much greater for having that consistent tool set.

For more information on LabVIEW Embedded you can visit the National Instruments website at NI.com you can see more information on LabVIEW at NI.com/LabVIEW. You can see more information on VisualDSP++ at the link provided, that's our CrossCore tools website, or you can visit Analog.com. If you have any questions you can click the ask a question button at the bottom of your screen. I hope you found today's module informative, and thanks for joining in.

End of file.