Presentation Title: Rapid Development of a Blackfin-based Video Application

Presenter Name: Glen Ouellette

Chapter 1: Introduction
Subchapter 1a: Agenda
Chapter 2: Video-In
Subchapter 2a: Device Driver Refresher
Subchapter 2b: Video-In Data Flow
Subchapter 2c: Video Capture Example
Subchapter 2d: Video Capture Recap
Chapter 3: Video-Out
Subchapter 3a: Video-Out Data Flow
Subchapter 3b: Video Display Example
Subchapter 3c: Video Display Recap
Chapter 4: Video-In/Video-Out Recap
Subchapter 4a: Video Pass Through
Chapter 5: MJPEG Encoding
Subchapter 5a: MJPEG Offering
Subchapter 5b: Encoding Video Data
Chapter 6: USB
Subchapter 6a: USB Offering
Subchapter 6b: USB/Host Data Transfers
Chapter 7: MJPEG Video Encoder Example
Subchapter 7a: Rapid Development Demo
Subchapter 7b: Software components
Subchapter 7c: Hardware Components
Subchapter 7d: Running the Demo
Subchapter 7e: Demo Usage/Video Playback
Video Playback

Chapter 1: Introduction

Subchapter 1a: Agenda

Hello I'm Glen Ouellette from Analog Devices Blackfin Applications Engineering Team. Today's session I'll go through Rapid Development of a video based application on Blackfin. This module goes through the Rapid Development process of a video based application running on Blackfin

using fully supported hardware and software modules. It's recommended that the user has some basic understanding or knowledge of software terminology as well as some experience in embedded systems. Also required or recommended is that the user have experience with systems services and device drivers.

The outline for the module is as follows; we'll go through a simple video-in example where we take video frame information from a ADV7183B video decoder and store it on Blackfin. We'll be using a ADV7183B device driver, which is provided with VisualDSP to do this. We'll also go through a video-out example using a ADV7179 device driver. We'll use Motion JPEG encoding to show how video information can be compressed on Blackfin. We'll also go through a USB submodule where we take video information that's been compressed and can store it on a host PC. The final submodule pulls this all together into a Motion JPG encoding application.

Chapter 2: Video-In

Subchapter 2a: Device Driver Refresher

We'll start off the Video-In with a quick refresher on device drivers and then move to a simple video capture using the ADV7183B device driver. Device drivers as you may recall use a standardized API that are independent of each driver and independent of the processor. This allows device drivers to be completely independent from each other and operate totally independent, as well as being transferable across different Blackfin processors without the application or the user having to be aware or make changes to the device driver.

Applications are responsible for providing buffers to the device driver, and by using functions such as ADI_Dev_Read and ADI_Dev_write. This allows inbound buffers to be filled with data coming from the device driver, or in the case of outbound buffers, to be sending data, the device driver sends the data from the buffer out to the device it wants it to communicate with.

The application's involvement from the device driver's perspective is rather minimal, all it does is initializes the system services and device driver. From then on each device driver in the application or in your whole system is entirely responsible for managing it's own calls of system services. Say for example in the case of a ADV7183B device driver, that device driver manages the DMA manager, the Interrupt Manager, Timer Control as well as its callbacks. For more information on Analog Devices device drivers you can go to the Analog Devices website and look in the manual section for system service and device driver help. As well as what's installed with Analog Devices VisualDSP 4.0 is, the same manuals are provided there in PDF format.

Subchapter 2b: Video-In Data Flow

So let's take a more detailed look at the ADV7183B Video-In Data Flow. Typically the video decoder chip the ADV7183B will be configured to accept some import video source. In this case we're using a DVD player and it can be in NTSC or PAL. We'll use NTSC in this example. We'll be using the BF561 EZ-Kit, and on that EZ-Kit is a ADV7183B video decoder. As well, the device driver for the ADV7183B is provided with VisualDSP and is fully supported by Analog Devices.

A typical application will take video data being decode by the ADV7183B and fill video frames on the Blackfin. Once the video frame has been filled it can then be passed on to either a different external encoder chip to be displayed locally, or processed with some type of compression or data processing application. Even more typical in a video application is to have multiple buffers. What this does is avoids over riding unprocessed or undisplayed pixels and video frames. So as we fill the first video frame, the device driver will complete filling that frame and move on to fill the second frame. Once the first video frame has been filled it can then be processed without the concern or worry of being over written by new data or pixels. And this process repeats itself.

In many applications you'll have what's referred to as multiple buffering. So you'll have multiple frames and video being written to and read from to eliminate any unwanted processing or over writing of unprocessed data. For the device driver itself typical in a video application is to use what's called 'Chaining with Loopback' and you may recall that from your experience with device drivers. This loops back automatically the first buffer to the last buffer chained in the process. It's not up to the application, the device driver manages it's own re-supply of the buffer information. So the device driver never starves for data. This is what a typical flow would look like, we would fill the first video frame and then move on to the second video frame and automatically loop back to the first frame. That's commonly referred to as 'ping-pong buffering'.

Subchapter 2c: Video Capture Example

So let's go through a simple series of programming sequence for the ADV7183B on EZ-Kit. We would initialize or the application would initialize to system services. A good practice in most applications would be to reset the ADV7183B. In this case on the EZ-Kit that's done through a programmable flag. We would then go about opening up the device driver for the ADV7183B. I've provided a little code window here so that we can see exactly how this is done. We provide the handle for the device driver, we configure the direction for inbound because we're going to be taking video information from the ADV7183B into the Blackfin, and then we'll also want to generate a PPI Callback so that we can notify our higher level application when a full buffer has been received.

The next part of the programming sequence is also as easy as the first part. We'll set up two buffers to allocate towards the video-in data. So we'll have buffer 2D data and into buffer 2D data. And those point to our two video frames, in this case video frame 0 and video frame 1. We'll have to define the width of our video elements. To optimize the use of Blackfin's bandwidth we'll employ some type of packing, in this case you'll be packing into 16 bit wide data elements. We'll set up our frame size and notify our device driver how much video information we want to bring in. In this case because it's an NTSC frame, we'll bring in 716 bytes by 525 lines. However because we're using this 16 bit wide element width we're going to divide our X count or our number of elements in a line by 2. So we'll actually have 858 elements in a line. And for our callback parameter to notify our application we're going to generate two callback indicators, the first one will be a 1 indicating that the first frame has been filled, and the next callback indicator will be a 2, indicating the second frame has been filled.

And finally because we're using chained loop back, we'll want to notify the next video frame and buffer that it has a loop back to the prior frame. Once that's been configured the device driver then only has to be enabled for it to begin operation. And this is simply done by opening the dev control and setting the data flow to true. Once the device driver has been enabled we'll begin receiving video information provided in through from say a source such as a DVD player through the ADV7183B, coming in through the Blackfin and begin filling video frame 0. This is all handled by the device driver. Again because we're using chained loopback automatically once video frame 0 has been filled, we'll begin filling video frame 1. From that point on the device driver manages itself in terms of back and forth or ping-ponging between video 0 and video frame 1.

Subchapter 2d: Video Capture Recap

So let's just recap what we have to do in that programming sequence. The first thing we did is we had to open up the ADV7183B device driver for inbound data flow, we had to configure the device driver for chained loopback and also as well because it is aware of what device is out there, what type of mode we want the ADV7183B to operate in. We had to provide video frames, in this case we're using only two data buffers. And we had to simply enable data flow. From then on the device driver operates separate from the application so the device driver will make a callback when the first frame is filled and subsequent frames are filled. That callback can be used by the application to begin the next level of processing on that data. Once the callback has been made, the callback function itself provides an extra buffer for the driver to fill and so on and so forth.

Chapter 3: Video-Out

Subchapter 3a: Video-Out Data Flow

Now let's move on to the next example or the next submodule which is Video-out. It's very similar to the Video-in case, we'll be using the ADV7179 device, which is also found on the Blackfin BF561 EZ-Kit. And we'll be displaying video frames on a local display.

In this case we'll be using the display or the video encoder ADV7179 in NTSC mode to display on a local TV. On the BF561 EZ-Kit that device is located, and we'll be using the available ADV7179 device driver, which is provided with Visual DSP.

In this case we'll take processed data or frame data in a video frame, the device driver will pass that video frame data out to the PPI to the ADV7179 where it's then encoded in NTSC and displayed on a local monitor. As was the case with the ADV7183B, output applications will use ping-pong buffering as well. The reasoning for this is similar in that you're video frame information that you're newly writing won't over write information that's being displayed, and that'll ping pong back and forth ensuring that the video information you're displaying is not being over written with new data coming in.

Similar to the ADV7183B device driver, we'll use this in chained loop back mode. Again so that the device driver is autonomous and ensures that it's never starved for data. The first video frame will be passed through and displayed, subsequently it'll then be passed to the next video frame, video frame 1 and display that. Once it's completed displaying that frame, the device driver during the callback will load up the next video frame which is video frame 0, and that's the chained loop back method.

Subchapter 3b: Video Display Example

Programming for the Video-out is rather simple just like the Video-in case. The application will initialize system services, as was the case with the ADV7183B, it's good practice to reset the ADV7179 encoder and on this case on the EZ-Kit it's done with a programmable flag. We'll then open up the device driver using the application, we'll use the handle; ADV7179 driver handle, we'll configure the traffic direction for outbound because we're going to pass this out to the encoder chip, and we're also going to want to generate a callback so that the application's aware when the video frame has been displayed to the local television.

We'll want to provide video buffers, like in the input case we'll use two video frames, video frame 0 and video frame 1. We'll also want to use a 16 bit wide element width to optimize bandwidth utilization on the Blackfin. We'll provide the size of our video frame, 716 bytes by 525 lines, which is a standard size of an NTSC video frame. And we'll also want to generate some callback

parameters, as was the case in the input, we'll use 1 for video frame 0 and 2 for the second video frame.

So once that's been initialized the only thing that remains now to get the ADV7179 device driver operational is to set the data flow to true. Once that begins happening the device driver itself runs and starts displaying video frame 0 out through the PPI to the ADV7179 to a local monitor. When it's complete displaying frame 0 it will then begin displaying frame 1, which gives the application an opportunity to then either fill video frame 0 processed data, or leaving it as it was. And it would ping pong back and forth, notifying the applications for the callbacks when it's completed displaying a particular video frame.

Subchapter 3c: Video Display Recap

So let's recap that programming sequence. The first thing that was done was to open the ADV7179 for Outbound Data Flow. We would then set up the driver for chained-loop back mode and ITUR BT656 mode. We'll provide it with two frames of video data to send out, and then we simply enable data flow. From that point on the application and the device driver are separate tasks. The device driver will generate callbacks when it's done displaying the previously filled buffer. And that application could then acknowledge the callback and act on it appropriately either filling up that previously displayed with new data, or leaving it. The device driver once it's generated the callback will then begin displaying the next buffer in the chain. In this case we only had two video buffers, so it'll loop back or ping pong back and forth between the two buffers.

Chapter 4: Video-In/Video-Out Recap

Subchapter 4a: Video Pass Through

Okay so let's recap what we've done up to this point. First we installed the ADV7183B device driver, which got us taking video source information in through the ADV7183B and storing it to two video frames on the Blackfin. Next we went through the ADV7179 device driver where we would take video frames that were already stored on the Blackfin, display them out through the ADV7179 to a local monitor. Like I said initially these are independent drivers, however they can share and often do share the same video frame buffers. So what's allowed to happen here is use two device drivers sharing the same video buffers to get a simple video pass through as it's referred to. We'll take video frame information with ADV7183B device driver and begin filling video frame 0. Once that frame has been filled the ADV7179 device driver will then start taking that video frame information and displaying it on a local monitor. At the same time the

ADV7183B will begin filling the next frame in the chain. This continues so that they loop back and forth so you have a simple video pass through and only one frame delay between the two.

The quick tip I want to mention is that the video frames are often located in different banks is SDRAM. This eliminates latencies involved with operating SDRAM. As well on the Blackfin architectural enhancements allow us to enable DMA traffic control, this minimizes the directional turn arounds that occur in applications like this and reduce the penalties that occur with the SDRAM when you turn around the bus like that.

Beyond simple video pass through the next module in this section we'll talk about is more application based. While you can have a simple video pass through displaying information that's being stored on the Blackfin, that same video information that's being captured is often processed or compressed by an application or software CODEC. It could be a soble edge detection for lane avoidance or Motion JPEG encoding, or MPEG encoding. In this case I've shown Motion JPEG encoding. We'll take that video frame information, compress it with an encoder and store it in compressed data format.

Chapter 5: MJPEG Encoding
Subchapter 5a: MJPEG Offering

So let's look at an encoding application. In this particular example we're going to go through a Motion JPEG. Analog Devices has a Motion JPEG offering and we'll go through the details of that, and then we'll actually run through a sample video frame work for Motion JPEG encoding. Analog Devices Motion JPEG, JPEG SDK are stand alone encoder and decoder libraries. These are available from Blackfin or from the Analog Devices website under Code Examples. All the source code is available with the exception of the actual encoder and decoder, which are provided in library format. As well several examples are provided based entirely on Analog Devices System Services and Device Drivers. I've listed as well additional resources for back ground information on the Motion JPEG and the JPEG Libraries.

The Basics of Motion JPEG data or video encoding is showing below will typically take a video frame of 4-2-0 format of separate buffers comprised of your luminance and chrominance values. That information or video data information is then passed through the Motion JPEG encoder where it's compressed and then stored into compressed data buffers. Often times though you typically get YCrCb from such a device as the 7183B. You'll typically be at 4-2-2 interlace data,

this is where callbacks are often used to initialize or instantiate a memory DMA to separate your 4-2-2 to 4-2-0 as well as down sample.

So once we get our video full of video frame of 4-2-2 our callback would then initialize a memory DMA to separate the buffers of luminance and chrominance values. It's than rather easy for your Motion JPEG encoder to then go and compress that information and store it in compressed video data.

Subchapter 5b: Encoding Video Data

Configuring the Motion JPEG is one of the first steps that's done to use the software CODEC. I've shown here below several of the configuration options and varied those configuration options vary your performance requirements as well as your compression and quality ratios. I've shown here different video sizes so simple frames smaller frames to full size frames as well as different quality factors. What this table doesn't reveal though is the interpretation or the visual analysis that the user must have to apply to this. One saying their quality ratios, their frame sizes and their frame rates.

As well as initializing the Motion JPEG we have to initialize or allocate an output stream for the data to be moved out of the MPEG encoder to either a local buffer or up say through an application such as a USB to a local file system on a PC. And then finally we instantiate the JPEG encoder with a pointer to the new buffer.

So typical application would go something like this; we wait for a full 4-2-0 buffer, which has the separate components of Y Cr and Cb. We would then pass a buffer pointer to our Motion JPEG encoder, and then we would then just perform the Motion JPEG encode. So here we're taking through the callback we would turn on our memory DMA and separate the luminance and chorminance values. Once that's been completed we can then Motion JPEG encode and store the compressed data either locally or up through a file system through our file IO to our USB host say for example. And this would continue.

As was the case with our video-in and video-out examples, it's very typical to see an application like this double buffering. We'll also want to encode multiple frames so that we ensure that video frame information that's coming say from a ADV7183B can be memory DMA to a YCrCb buffer, isn't overriding the same location that's being now encoded and stored in compressed data

format. So this is referred to as double buffering this data to ensure that we're not storing or overriding data that hasn't yet been compressed and stored.

Chapter 6: USB
Subchapter 6a: USB Offering

Now that we've gone through the Motion JPEG we'll take a look at USB, and I think you see where this is going. We'll go through the USB EZ LAN Extender which is available from Analog Devices, and then go through a simple Blackfin to host data transfer using USB 2.0.

Analog Devices has a fully supported for software and hardware USB EZ LAN Extender, which plugs on to several of the Blackfin EZ-Kits, including the one that we're using today the ADSP-BF561 EZ-Kit. It provides the Blackfin with the highest speed USB 2.0 connectivity to a host PC for example. I provided here some sample bench marks using the Device Driver for the net 2272 PLX USB chip on various platforms of Blackfin. In this case we're using the BF561. As you can see from benchmarks that it's more than suitable for compressed video applications. Another thing to note is that on USB PC it's often the times that the operating system such as Windows often lags your USB 2.0 port, so it's essential for the application to be aware that performance and benchmark metrics can be impacted by the Windows Operating System.

Subchapter 6b: USB/Host Data Transfers

So let's go through a simple USB Blackfin to host transfer. It's more typical or often typical for the host PC to send command blocks back and forth to the host and to the Blackfin. So in this case Blackfin would receive a USB command block and then perform some function or action based on that command block. It may be to perform an IO based communication with the host, either uploading data payload to the host or doing a file IO and reading data off of the host.

The Basic Commands over view is the command block comes down as I showed on the previous slide, indicating what functions the Blackfin is to perform. And you can refer to the USB Command.H file for information of the simple command blocks structure. I've shown an example here below where you can bring in a simple command, data, how much data you want to move and different parameters in that command block that the application running on Blackfin can use to determine what function it is that the host would like to operate. A USB Command.H also contains default commands whether it be memory reads, memory writes, for the USB IO start and stop to indicate to the application what type of function it wants to perform such as the memory

and when to start and stop that function. So once the command block as been received the application will then switch based on that command block. Again I've shown a sample code window below, we will receive a command block and then based on that particular command whether it be to start, read or write, the application will perform some function.

For additional information the Blackfin USB there's schematics and LAN information on the FTP set of Analog Devices. As well Blackfin USB Firmware is provided with VisaulDSP from Analog Devices. In that same VisualDSP are C examples as well as a full device driver for bulk and I synchronous versions. Windows Host applications are also provided to allow you to run on a Windows host PC device driver and view our USB based applications on Blackfin.

Chapter 7: MJPEG Video Encoder Example
Subchapter 7a: Rapid Development Demo

So let's go through and pull all these building blocks together that I've talked about in the first four of this presentation. And we'll do a full wrap development of a Motion JPEG Video Encoder on Blackfin.

So first thing that we did was introduce the video-in component, which is a ADV7183B. We then introduced a video-out component, which use a ADV7179. We discussed the Motion JPEG where we take that video frame information, compress or separate it with memory DMA's and then compress it and encode into compressed video data buffers. And then we talked about how the USB host transactions take place with commands received from host and say in this case we'll be doing a memory or a data write up through the host to the local PC.

Subchapter 7b: Software components

So we'll go through the software structure of how a typical application will go about doing this. The software flow is fairly straight forward, the application starts off by setting up system services, we would then go and initialize our device driver for our net 2272 and this sets up the PLX chip so that we can receive commands from the host PC. Based on those commands that we receive we then perform some type of function on the parameters received within that command. Once that function is completed, say it's performing a Motion JPEG encode, will then encode until we're told to stop encoding and it will repeat or circle back to the top and wait for the next command from the host.

So let's take a closer look at how the encode actually takes place. The parameter received, or the command received from the USB host is a Motion JPEG encode, will then set up the Motion JPEG parameters as we saw in that section, and will open the remote file on a host PC. It will then go and set up our video device drivers, the ADV7183B and ADV7179. Once those have been configured we'll then have video information coming into the Blackfin as well as video information going out to be displayed on a local monitor. In this case we'll then pole a push bottom switch on the EZ-Kit say for example, waiting for us, or telling the software application when to begin Motion JPEG encoding. This could be anything in a typical application. It could be, it's completely arbitrary that we chose a push bottom, it could be saying in a security application and motion activation sensor.

Once that's been completed or that software switch has been pressed, we'll then begin encoding video frames performing IO of those compressed frames up over USB to the host PC until we're told to stop by say pressing the same switch or the motion in the case of a video security application is stopped.

Subchapter 7c: Hardware Components

So let's see what's required in order to run this. We'll have a BF561 EZ-Kit, we'll have the USB from Analog Devices, the USB EZ LAN Extender Card. We need a host PC running Windows XP in this case that has a USB 2.0 port. We need a video source, we'll use a DVD player, a video display for NTSC output, we'll also need VisualDSP installed to give us the capability to use the device drivers and system services. And we'll use the software that's provided with this presentation that allows you to rapidly do this on your own terminal.

The installation is quite easy, you will take the zip file, and extract the software into a directory in your hard drive on your PC, let's call this directory the root directory. And we'll reference it as such later in this presentation. We will connect the EZ LAN extender card to the EZ-Kit, and we'll connect the USB 2.0 port from the PC to the USB 2.0 port on the USB EZ LAN extender card. It's also important to set the dip switches on both the EZ LAN extender card and the EZ-Kit as I've shown on this slide to ensure that the USB and the Blackfin are communicating as intended.

A basic hardware overview, once we've plugged those two cards together, the EZ LAN extender card onto the EZ-Kit as shown as follows; we'll have our video source providing data to the ADV7183B, which is located on the EZ-Kit. We'll have the ADV7179 taking that same video data and doing a pass through to a local monitor. The Blackfin will be running a Motion JPEG encode,

and the USB Net 2272 will be then taking that compressed or encoded data and passing it up to a host PC where it gets stored in an AVI file format. Once it's on the host PC we can then use any media player viewer on the host PC to view that compressed data.

Let's go through the connections to the EZ-Kit. Using J6 on the BF561 EZ-Kit you would connect your video-in source from a DVD player in this example. You would also connect to the upper middle, your connection to your TV display. You will then power your player and your display to ensure that you have a video source and a destination for your video information.

Subchapter 7d: Running the Demo

You can then apply power to the EZ-Kit, fire up VisualDSP, in the case where you'd like to flash the BF561 EZ-Kit with the image that's provided with this example, you can then load up the image to our flash programmer, load the image file. Once that's completed you could exit VisualDSP and reset the EZ-Kit. Once the EZ-Kit has been reset, it should then look like a USB device to the Windows XP Operating System. Windows should detect the new device, and you can point Windows to where the host driver is for the USB device. This step only needs to be done once because from now on Windows will retain that information of what the device is.

Once the EZ-Kit is running you'll have a video flow going into the EZ-Kit, you have it connected to a local display. You can then go and fire up a DOS prompt on your PC. And I'm going to go ahead and do that. Once you've navigated to where that root director is, you would then want to navigate two directories lower to where your host application is. This is the application that's going to send the commands to the Blackfin to indicate what type of function it wants it to perform. In this application again it's Motion JPEG encoding. There are a couple simple commands that you can run to indicate how the device is connected and what mode you want the device to run in. For example typing 'Hostapp -a' will let you know how many Blackfin EZ-Kits or USB devices are connected to the PC. So let's do that. Here it tells that I have one USB EZ LAN extended connected to the host PC.

A few other things to note; when this application runs it'll be storing the compressed information on your PC up through USB. That video information is stored based on your Root directory in the images/motionjpg folder. Those files will be stored in the AVI format and when we get to the end of this session we'll go through how to view those compressed data streams.

The application uses file IO to communicate with the host PC. So a command would come down from the host indicating a Motion JPEG encode, compressed data would then be written back up to the host PC with FileIO. Instructions provided to make it very easy for you to follow.

Subchapter 7e: Demo Usage/Video Playback

So let's go through the usage now but before we actually start to capture the information. On your host PC as well is an encoder spec. This is a text file in the working directory, which indicates to the Motion JPEG what type of compression you'd like to perform. It specifies the size of your video display and the quality factor as well. For example it could be a 352 by 288 with a quality factor of 60, or a full size frame with a quality factor of 70. Once you've set up the type of frame you want to capture, the size of image you'd like to capture and the quality ratio, you can then go and run the encoder command to indicate the Blackfin to begin encoding and storing compressed data up on the PC. To set it for encoding you would type: 'hostapp -m e'. This sets the application running on the Blackfin from Motion JPEG and encoding. Once that's been configured you can then go and begin running the encoder. So let's go and run that.

Now I've configured the encoder with a Motion JPEG application for running and a Motion JPEG with encode. And now what I'll do is actually kick off the application. You can see here that the application is now running on the Blackfin, this is simple Printf or text IO with the host PC, letting it know what function Blackfin is performing. We've gone through, we've interpreted the frame size information in this case 352 by 288 and the encoding quality factor, which is set to maximum of a 100. The file it's going to write to is; display Motion JPEG 1. As well now we see that we're sitting here waiting for SW6 to be pressed. Again SW6 is arbitrary it could be some other application with a motion sensor. But in this simple demo we're running on EZ-Kit SW6 is easy. Press SW6 to begin recording, we press SW6 to stop recording. So let's go and press SW6 to begin.

You can see here we're getting indication that we're recording frames, 10 frames at a time as each time it flips over. And so now we'll actually recording video frames. We're compressing them and storing them up through the USB to the host PC. These are being stored in your root directory Motion JPEG folder in the format of an AVI file. I think that should be good, we can stop recording now. We've captured 686 frames at 30 frames per second. We can exit out of this easily by hitting Q to escape. So now that we've done this we've actually captured video coming in from our DVD source, being encoded with Motion JPEG on Blackfin, and being stored by USB up to the host PC. So now let's play back the video that was encoded on the Blackfin processor.

END OF DEMOSTAGE

Video Playback

This was the video that was passed in to the processor by a DVD player and encoded with Motion JPG and then sent up over the USB to the PC. Simply double clicking on the file will open up Windows Media Player and we can view the compressed out put.

All of the topics I've covered today are available from Analog Devices. Most of which are shipped with VisualDSP or contained in the zip file that's associated with this session. If you have additional questions please don't hesitate to click the 'ask a question' button below. Thank you very much.

End of Recording