Rapid Development of a Blackfin-based Video Application

Presented By:
Glen Ouellette
About this Module

This module discusses the rapid development process of a Blackfin® Video application using readily available and fully supported software and hardware modules.

It is recommended that users have some understanding of:

- Basic knowledge of software terminology
- Experience in embedded systems
- Blackfin System Services and Device Drivers
Module Outline

- **Video-In**
  - Refresher on Device Driver
  - Video Capture using ADV7183B Video Decoder

- **Video-Out**
  - Video Display using ADV7179 Video Encoder

- **Video Compression**
  - Overview of MJPEG offering
  - Encoding Video Data

- **USB**
  - Blackfin USB-LAN EZ-Extender
  - Blackfin-Host data transfer over USB 2.0

- **Rapid Development of a MJPEG Encode**
Outline of Video-In Sub-module

- A short Device Driver refresher
- Simple Video Capture using ADV7183B Video Decoder
Device Driver Refresher

- **Standardized API**
  - User interface is the same across all device drivers
    - Independent of driver
      - Allows buffers to be passed from driver to driver
    - Independent of processor
      - Does not require change when moving from BF561 to BF537
- **Device Drivers are Totally Independent**
- **User Application provides buffers to device driver**
  - Provided to driver via adi_dev_Read() or adi_dev_Write()
    - Inbound buffers – Filled with data received from the device
    - Outbound buffers – Contain data to send out through the device
**Device Driver Refresher (continued)**

- **Application involvement**
  - **Initialize services**
- **Independent Device drivers manage their own set of system services**
  - **Drivers call into system services as required**
    - **Video In Device Driver (e.g. ADV7183B)**
      - Calls into DMA Manager
      - Calls into Interrupt Manager
      - Calls into Timer Control
      - Calls into DCB
Additional Information on Device Drivers

- **Device Drivers and System Services Manual for Blackfin Processors**
- **Device Drivers and System Services Addendum (Sept 2005)**
Video-In Data Flows

- Video Decoder is configured to accept source input
  - NTSC or PAL
  - Example: ADV7183B with NTSC input
- Hardware
  - ADSP-BF561 EZ-Kit (Silicon Revision 0.3 or higher)
    - Has onboard ADV7183B Video Decoder
- ADV7183B Device Driver provided with VisualDSP++
- Others
Double Buffering

- Double Buffering is typical in Video Applications
  - **Benefit**
    - Pass data into a single frame while processing/displaying data elements of a previously filled frame
    - Avoids over writing unprocessed pixels/frames
  - Often Multiple Buffering (6 or more)
Dataflow Method: Chaining with Loopback

- **Chaining with Loopback**
  - Device driver automatically loops back to the first buffer after the last buffer in the chain is processed
  - Application doesn’t need to resupply buffers
    - Lower overhead
    - Device driver never “starves” for data
Video-In Programming Sequence

- **Initialize System Services**
- **Good practice to reset Video Decoder**
  - On Blackfin EZ-Kits, ADV7183B Reset is controlled via Programmable Flag
- **Open ADV7183B Device Driver**
  - ‘AD7183DriverHandle’
  - Configure for Inbound traffic from ADV7183B to Video Buffer
  - PPI_Callback_In

```c
ezErrorCheck(adi_dev_Open(DeviceManagerHandle,
    &ADIAD7183EntryPoint, // pdd entry point
    0, // device instance
    NULL, // client handle callback identifier
    &AD7183DriverHandle, // DevMgr handle for this device
    ADI_DEV_DIRECTION_INBOUND, // data direction for this device
    DMAManagerHandle, // handle to DmaMgr for this device
    DCBManagerHandle, // handle to deferred callback service
    PPI_Callback_In)); // deferred callback

/********* open AD7183 via PPI ********************/
ezErrorCheck(adi_dev_Control(AD7183DriverHandle, ADI_AD7183_CMD_OPEN_PPI, (void *)0));
```
Video-In Programming Sequence (cont’d)

- Allocate two 2D_Buffers
  - In1_Buffer2D, In2_Buffer2D
- Pointer to the data
  - Video_Frames[0] and Video_Frames[1]
- Element width
  - 16-bit wide element
- XCount, XModify, YCount, YModify
  - ITU-R BT656 NTSC Video Frame
- Callback parameter
  - For this example, ‘1’ indicates Video_Frames[0] and ‘2’ indicates Video_Frame[1]
- pNext
  - Pointer to the next Video_FRAME buffer in the chain (NULL if the last Video_FRAME buffer in chain)

```c
/****** ADV7183 Inbound Buffers **************/

In1_Buffer2D.Data = Video_Frames[0];
In1_Buffer2D.ElementWidth = sizeof(u16);
In1_Buffer2D.XCount = (1716/2);
In1_Buffer2D.XModify = 2;
In1_Buffer2D.YCount = 525;
In1_Buffer2D.YModify = 2;
In1_Buffer2D.CallbackParameter = 1;
In1_Buffer2D.pNext = &In2_Buffer2D;

In2_Buffer2D.Data = Video_Frames[1];
In2_Buffer2D.ElementWidth = sizeof(u16);
In2_Buffer2D.XCount = (1716/2);
In2_Buffer2D.XModify = 2;
In2_Buffer2D.YCount = 525;
In2_Buffer2D.YModify = 2;
In2_Buffer2D.CallbackParameter = 2;
In2_Buffer2D.pNext = NULL;
```
Then Enable Dataflow Sequence

- Start Capturing Video Data from PPI0 into Video_Frames[0] and Video_Frame[1]
  - adi_dev_Control(AD7183DriverHandle, ADI_DEV_CMD_SET_DATAFLOW, (void*)TRUE);
Programming Sequence for ADV7183B

1. Open ADV7183 driver for Inbound dataflow
2. Configure ADV7183 driver for chained loopback, ITU-R BT656 mode etc.
3. Provide 2 frame buffers for driver to fill
4. Enable dataflow
5. Driver makes callback when frame buffer is filled
6. Callback function provides next buffer for driver to fill
Outline for Video-Out Sub-module

- Simple Video Display using ADV7179 Video Encoder
Video-Out Data Flows

- **Video Encoder is configured to drive display**
  - NTSC or PAL
  - Example: ADV7179 connected to NTSC TV
- **Hardware**
  - ADSP-BF561 EZ-Kit (Silicon Revision 0.3 or higher)
    - Has onboard ADV7179 Video Encoder
- **ADV7179 Device Driver provided with VisualDSP++**
Double Buffering

- Double Buffering is typical in Video Applications
- Benefits
  - Display a single frame while filling data elements of a previously displayed filled frame
  - Avoids over displaying old pixels/frames
Dataflow Method: Chaining with Loopback

- **Chaining with Loopback**
  - Device driver automatically loops back to the first buffer after the last buffer in the chain is processed
- **Application does not need to re-supply buffers**
  - Lower overhead
  - Device driver never “starves” for data

![Diagram showing data flow](image.png)
Video-Out Programming Sequence

- Initialize System Services
- Good practice to reset Video Encoder
  - On Blackfin EZ-Kits, ADV7179 Reset is controlled via Programmable Flag
- Open ADV7179 Device Driver
  - ‘AD7179DriverHandle’
  - Configure for Outbound traffic from Video Buffer to ADV7179
  - PPI_Callback_Out

```c
ezErrorCheck(adi_dev_Open( DeviceManagerHandle,
    &ADIADV71xEntryPoint,       // Device Entry point
    0,                          // Device number
    NULL,                       // No client handle
    &AD7179DriverHandle,        // Device manager handle address
    ADI_DEV_DIRECTION_OUTBOUND, // Data Direction
    DMAManagerHandle,           // Handle to DMA Manager
    DCBManagerHandle,           // Handle to callback manager
    PPI_Callback_Out));         // deferred callback

/****** open AD7179 via PPI1 **************************/
ezErrorCheck(adi_dev_Control( AD7179DriverHandle, ADI_ADV71x_CMD_SET_PPI_DEVICE_NUMBER, (void*)1 ));
```
**Video-Out Programming Sequence – Cont’d**

- **Allocate two 2D_Buffers**
  - Out1_Buffer2D, Out2_Buffer2D
- **Pointer to the data**
  - Video_Frames[0] and Video_Frames[1]
- **Element width**
  - 16-bit wide element
- **XCount, XModify, YCount, YModify**
  - ITU-R BT656 NTSC Video Frame = 1716 Bytes per line by 525 lines per Frame
- **Callback parameter**
  - For this example, ‘1’ indicates Video_FRAME[0] and ‘2’ indicates Video_FRAME[1]
- **pNext**
  - Pointer to the next Video_Frame buffer in the chain (NULL if the last Video_Frame buffer in chain)

```c
/*------- AD7179 Outbound Buffers--------------*/
Out1_Buffer2D.Data = Video_Frames[0];
Out1_Buffer2D.ElementWidth = sizeof(u16);
Out1_Buffer2D.XCount = (1716/2);
Out1_Buffer2D.XModify = 2;
Out1_Buffer2D.YCount = 525;
Out1_Buffer2D.YModify = 2;
Out1_Buffer2D.CallbackParameter = 1;
Out1_Buffer2D.pNext = &Out2_Buffer2D;

Out2_Buffer2D.Data = Video_Frames[1];
Out2_Buffer2D.ElementWidth = sizeof(u16);
Out2_Buffer2D.XCount = (1716/2);
Out2_Buffer2D.XModify = 2;
Out2_Buffer2D.YCount = 525;
Out2_Buffer2D.YModify = 2;
Out2_Buffer2D.CallbackParameter = 2;
Out2_Buffer2D.pNext = NULL;
```
Then Enable Dataflow Sequence

- Start Displaying Video Data from Video_Frames[0] and Video_Frames[1] into PPI1
  - adi_dev_Control(AD7179DriverHandle, ADI_DEV_CMD_SET_DATAFLOW, (void*)TRUE));
Programming Sequence for ADV7179

1. Open ADV7179 driver for Outbound dataflow
2. Configure ADV7179 driver for chained loopback, ITU-R BT656 mode etc.
3. Provide 2 frame buffers for driver to send out
4. Enable dataflow
5. Driver makes callback when frame buffer has been sent out
6. Callback function provides next buffer for driver to send out
Recap: Video-In and Video-Out

- Installed ADV7183B Device Driver
  - Video Decoder is configured to accept source input
- Installed ADV7179 Device Driver
  - Video Encoder is configured to drive display
- Independent Drivers, but can share Video Frame Buffers
Recap: Rapid Development of Video Pass Through

- Reuse Video Frame buffers to create Simple Video Pass Through
  - Only 1 Frame Delay between input and output

Quick Tips:
- Map video frames into different banks of SDRAM to avoid latencies
- Configure DMA traffic control to optimize unidirectional traffic and reduce SDRAM bus turnaround penalties
Recap: Beyond Pass Through - Video Processing

- Simple Video Pass Through is common
  - E.g. Back-up Video Camera, Side-view Mirrors, Back seat monitor
- Often, Inbound Frames are processed/compressed by software codecs
  - E.g. Sobel Edge detection, MJPEG Encoding, MPEG Encoding
Outline for Encoding Video Data Sub-module

- Overview of the MJPEG offering
- Encoding Video Data
Overview of the MJPEG offering

- JPEG/MJPEG SDK includes stand alone Encoder and Decoder Libraries
  - Access to all source code, except the actual encoder and decoder algorithm (libraries provided)
  - Examples based entirely on ADI’s System Services and Device Driver libraries
- Other Resources
  - ADI, “JPEG Encoder Library Developer's Guide”
  - ADI, “JPEG Decoder Library Developer's Guide”
  - ADI, “MJPEG AVI Library Developer’s Guide”
MJPEG Encoding Video Data Basics

- An Inbound buffer is processed/compressed by MJPEG codec
  - MJPEG Codec Encodes a 4:2:0 Video buffer comprised of Y, Cr, Cb Buffers

- Memory DMA is often used to separate and down sample 4:2:2 interlaced image to a progressive YCrCb 4:2:0 format
  - Down sampling the Cr and Cb components by two in the vertical direction
  - Callback typically initiates Memory DMA action
Initializing MJPEG Codec

Configure MJPEG Codec
- JPEG_FRAME_WIDTH, JPEG_FRAME_HEIGHT, JPEG_QUALITYFACTOR, etc.

<table>
<thead>
<tr>
<th>Horizontal</th>
<th>Vertical</th>
<th>Frames/Sec</th>
<th>Quality Factor</th>
<th>Compression Ratio</th>
<th>Cycles/Pel</th>
<th>Mhz Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>176</td>
<td>144</td>
<td>30</td>
<td>60</td>
<td>10.3</td>
<td>53.89</td>
<td>41</td>
</tr>
<tr>
<td>176</td>
<td>144</td>
<td>30</td>
<td>40</td>
<td>13.1</td>
<td>49.08</td>
<td>37</td>
</tr>
<tr>
<td>640</td>
<td>480</td>
<td>30</td>
<td>60</td>
<td>14.0</td>
<td>43.68</td>
<td>403</td>
</tr>
<tr>
<td>640</td>
<td>480</td>
<td>30</td>
<td>40</td>
<td>19.2</td>
<td>38.7</td>
<td>357</td>
</tr>
</tbody>
</table>

*For more details and other benchmarks look into the documentation (Developer Guides) that comes with the software product brief for JPEG/MJPEG SDK

Allocate the output streambuffer
- StreamBuffer_Obj = JPEG_MemAlloc_NEW(3 * Input_Width * Input_Height, MEM_TYPE_DATA);
  - Typical application would streaming output to local file system (i.e. fprintf) or remote host file system.

Instantiate JPEG Encoder
- IJpegEnc = JPEG_Encoder_NEW(&IImageParam);
Encode Video

◆ Step 1: Wait for full YCrCb 4:2:0 Buffer

◆ Step 2: Pass YCrCb 4:2:0 Buffer Pointer
  ● JPEG_McuBuffer_CONFIG(mcu_ex, MCU_POINTER_C1, (unsigned int)lInputBuffer);

◆ Step 3: Perform MJPEG Encode
  ● JPEG_EncodeSequentialImage(lJpegEnc, &NumBytes);

◆ To encode next frame, go to Step 1

◆ Without Double Buffering, frame being encoded is also being overwritten with new data – result is corrupted image
Encode YCrCb 4:2:0 Frame with Double Buffering

- To encode multiple frames, double buffering is used
  - Ensures previous frame which is being encoded isn’t corrupted with next frame data
Outline for USB Sub-module

- Blackfin USB-LAN EZ-Extender
- Simple Blackfin-Host Data Transfers over USB 2.0
Blackfin USB-LAN EZ-Extender Hardware

- Adds USB 2.0 High speed connectivity to Blackfin
  - PLX NET2272 USB 2.0 High Speed Controller
- USB-LAN EZ-Extender plugs on to ADSP-BF533, ADSP-BF537, and ADSP-BF561 EZ-KITs
# Blackfin USB-LAN EZ-Extender Benchmarks

<table>
<thead>
<tr>
<th>Processor</th>
<th>OUT -&gt; from host MBytes/sec</th>
<th>IN &lt;- to host MBytes/sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>BF533</td>
<td>20.0</td>
<td>25.1</td>
</tr>
<tr>
<td>BF537</td>
<td>20.3</td>
<td>25.0</td>
</tr>
<tr>
<td>BF561</td>
<td>16.3</td>
<td>20.0</td>
</tr>
</tbody>
</table>

- **Test Conditions**
  - Using the provided host application, bulk host driver, and firmware
  - Test system: P4 2.0 GHz, 768 MB RAM, USB 2.0 host controller, WinXP Pro SP2, no other USB devices on the bus
  - Measure on Windows and takes into account complete transaction time from host application point of view
Simple Blackfin-Host Transfers Via NET2272 USB

- Receive USB command block from the Host indicating what function for Blackfin to perform

- Perform IO Based on Command using File I/O or Stdio
  - Upload Data payload to Host PC
  - Download Data payload from Host PC
Basic Commands Overview

- Application exchanges command blocks with the host
  - Indicating what functions to perform
  - Refer to USBCMD.h for default USBCB command block structure

Example:

```c
typedef struct _USCB
{
    ULONG ulCommand;       // command to execute
    ULONG ulData;          // generic data field OR 1st Parameter
    ULONG ulCount;         // number of bytes to transfer OR 2nd Parameter
    ULONG ulPar3;
    ULONG ulPar4;          // 3rd Parameter
} USCB, *PUSCB;
```
Basic Commands Usage

*USBCMD.h also contains default Commands*

```c
enum _USB_COMMAND
{
    NO_COMMAND,        // nothing to do here...
    GET_FW_VERSION,    // get the firmware version
    QUERY_SUPPORT,     // query for support
    QUERY_REPLY,       // query reply
    LOOPBACK,          // run loopback on the device
    MEMORY_READ,       // read from specified memory on the device
    MEMORY_WRITE,      // write to specified memory on the device
    USBIO_START,       // run USB IO on this device
    USBIO_STOP,        // stop USB IO on this device
    USBIO_OPEN,        // open file on host
    ...
};
```

*Once Command has been received, switch on the command received*

- `switch( pusbcb->ulCommand )`

**Example:**

```c
switch( pusbcb->ulCommand )
{
    case USBIO_START:        PerformIo( USB_DvHandle ); break; // perform IO over USB
    case MEMORY_READ:        ReadMemory( USB_DvHandle, (u8*)pusbcb->ulData, pusbcb->ulCount ); break; // read memory
    case MEMORY_WRITE:       WriteMemory( USB_DvHandle, (u8*)pusbcb->ulData, pusbcb->ulCount ); break; // write memory
    default:                 exit(1); break; // unsupported command
}
```
Additional Information on Blackfin USB

- Hardware schematics and layout data available on ftp site

- Blackfin USB firmware
  - Ships with latest VisualDSP++
  - C language examples - loopback and IO redirection
  - C language NET2272 driver – bulk and isochronous versions

- Windows host application
  - Ships with latest VisualDSP++
  - C/C++ example built with Microsoft Visual Studio .NET
Rapid Development of a MJPEG Video Encoder
Rapid Development with Common Components

- Video-In
- Video-Out
- MJPEG Encoder
- USB Host
**Software Flow**

1. **Setup System Services** ('main.c')
2. **Setup NET2272 Device** ('main.c')
3. **New USB command?** ('main.c')
   - NO
   - YES
4. **Identify MJPEG Coding Algorithm and Coding_Action** ('main.c')
5. **Perform MJPEG action and send Encoded image over USB** ('main.c')
6. **Setup MJPEG Parameters and OPEN remote file** ('MJPEG_Encode.c')
7. **Setup Video Devices ADV7183B and ADV7179** ('MJPEG_Encode.c')
8. **Press SW6 on EZ-KIT to START?** ('MJPEG_Encode.c')
   - NO
   - YES
9. **MJPEG Encode Video Frames as they become available** ('MJPEG_Encode.c')
10. **Perform IO of compressed MJPEG image data over USB** ('MJPEG_Encode.c')
11. **Press SW6 on EZ-KIT to STOP?** ('MJPEG_Encode.c')
   - NO
   - YES
12. **CLOSE remote file** ('MJPEG_Encode.c')

---

- **NET2272 Device Driver**
- **ADV7183B Device Driver**
- **ADV7179 Device Driver**

**ADV71783B Device Driver**

**ADV7179 Device Driver**

SW6 is arbitrary. For example, could be motion activated sensors which kicks off MJPEG encoding.

**MJPEG LIBRARY**

- SW6 is arbitrary. For example, could be motion activated sensors which kicks off MJPEG encoding.

---

**NET2272 Device Driver**

**ADV7183B Device Driver**

**ADV7179 Device Driver**

**MJPEG LIBRARY**
What’s Required

◆ Hardware
  ● ADSP-BF561 EZ-Kit (Silicon Revision 0.3 or higher)
  ● ADSP-USB_EZ_LAN Extender Card
  ● PC
    ◆ Running WinXP
    ◆ Native USB2.0 port is best
      – If you do not have, use a PCMCIA plug-in card. Transfer rate will degrade by ~30%
  ● DVD player (NTSC) for input
  ● Display (NTSC) for output

◆ Toolset
  ● VisualDSP++ 4.0 with December 2005 update and Emulator
    ● If you don’t want to recompile, use the provided *.dxe executable or burn the provided *.ldr image into flash memory

◆ Software
  ● ZIP file
    ● Posted with this presentation
Installation [1/4]

- Copy or extract the Software into a directory on your hard-disk (let’s call it “ROOT” directory)

- Connect Extender Card onto ADSP-BF561 EZ-kit

- Connect USB2.0 port on PC to USB2.0 port **ON THE EXTENDER CARD**

- Set DIP switches according to table on next Slide
## Installation [2/4]

<table>
<thead>
<tr>
<th></th>
<th>Setting for this demo</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>EZkit DIP switches</strong></td>
<td></td>
</tr>
<tr>
<td>SW3</td>
<td>All off except #6</td>
</tr>
<tr>
<td>SW9</td>
<td>All off except #1 and #6</td>
</tr>
<tr>
<td>SW2</td>
<td>All off</td>
</tr>
<tr>
<td>SW1</td>
<td>All off</td>
</tr>
<tr>
<td><strong>Extender card DIP switches</strong></td>
<td></td>
</tr>
<tr>
<td>SW1</td>
<td>All on except #4</td>
</tr>
<tr>
<td>SW2</td>
<td>All on except #1</td>
</tr>
</tbody>
</table>
Hardware System Overview (MJPEG Encoder)

- ADV7183B
- ADV7179
- BF561 (MJPEG Encoder)
- USB NET2272
- ADSP-BF561 EZ-KIT

Simple Video Pass Through

Stack EZ-USB Extender onto ADSP-BF561 EZ-KIT

USB 2.0 to Host PC
- Compressed Video flows from Blackfin to Host PC
- *.AVI files get stored to local hard drive
Installation [3/4]

- Connect the NTSC display to the “Video Out” Connector J6 on ADSP-BF561 EZ-Kit

- Connect the NTSC DVD player to the “Video In” Connector J6 on ADSP-BF561 EZ-Kit

- Power DVD Player and Display
Installation [4/4]

- Power the EZ-Kit
- Start VisualDSP++
- Load the Image File into flash memory
  - Use VisualDSP’s “flash programming tool”
  - Image file is “\ROOT\codec_blkfn\jmjpeg_app\BF561\bin”
- Exit VisualDSP++ and reset the EZ-Kit

- Windows should now detect a new USB Device
  - Follow the prompts to install the device driver
    - Specify the path to the device driver
  - Device Driver is located in “\ROOT\codec_host\hostdriver” directory
    - This step needs to be done only once
    - Windows will remember device driver next time
**USAGE [1/3] – General**

- Open a DOS prompt Terminal on your PC
  - Generally under Start->All Programs->Accessories
- Change Directory to “ROOT\codec_host\hostapp” directory
  - With DOS’s “cd” command (if necessary, switch to the hard-disk drive by typing “C:”)

- The “hostapp.exe” application is the main control for the Demo Application that is running on the DSP.

- A few tests:
  - Type “hostapp –h” and press “enter” - what happens?
  - Type “hostapp –a” - will tell you if and how many DSPs are connected
  - Type “hostapp –v” - more information about the application

- If the “hostapp” detects a device, you are ready to proceed!
A few general explanations:

- **Open an Explorer Window at**
  - “ROOT\images”
    - Call this the “work directory”
  - “ROOT\images\MJPEG”
    - Target directory where encoded MJPEG *.AVI files will be stored

The application uses File I/O to upload Payload to the Host PC

- **DISPLAYs information about the encoding/decoding process**
- **Follow instructions on the terminal CAREFULLY**
  - DO NOT TYPE in the terminal unless instructed to
  - Accidentally press a key at the wrong time will abort the program. Just press the reset button on the EZ-kit
Open the specification file in the work directory
- `mjpeg_encoder_spec.txt`

Each line specifies a target MJPEG (*.avi) file to be encoded from a Video source
- Type the name: for instance `mymovie`
- Type the size (Horizontal Vertical): for instance `320 240`
- Type the encoding quality factor (0 [least] – 100 [max])

Repeat for as many recordings/files you like to take
Example:
- `DISPLAY_MPEG_TEST1 352 288 60`
- `DISPLAY_MPEG_TEST3 640 480 70`

Save the file

Run ENCODER from the Host PC terminal:
- Type `hostapp –m e` - Sets the DSP application to (m)jpeg (e)ncoding
- Type `hostapp –u` - Starts encoder application

NOW (as instructed) press button on the EZ-kit
- Press to start recording
- Press to stop recording
- Press ENTER on the keyboard to move to the next file
- Repeat for each file you specified in the steps above

Play the files!
- Find files specified above in MJPEG directory in the work directory
- Play the files by double-clicking on them
Additional Information

- **Blackfin USB firmware**
  - Ships with latest VisualDSP++
  - C language examples - loopback and IO redirection
  - C language NET2272 driver – bulk and isochronous versions

- **Windows host application**
  - Ships with latest VisualDSP++
  - C/C++ example built with Microsoft Visual Studio .NET

- For questions, click “Ask A Question” button or send an email to Processor.support@analog.com