An Independent Analysis of the

TEXAS INSTRUMENTS DIGITAL VIDEO EVALUATION MODULE (DVEVM)

By the staff of

BDTi INSIGHT • ANALYSIS • ADVICE ON SIGNAL PROCESSING TECHNOLOGY

Overview

The Texas Instruments Digital Video Evaluation Module (DVEVM) contains hardware and software that are intended to serve as:

1) *A platform for quickly evaluating the capabilities of the TMS320DM6446 chip, which is part of TI's "DaVinci" family of processing engines.*
2) *An easy-to-use, straightforward environment to enable ARM-Linux embedded systems developers (who may not have DSP expertise) to quickly begin development of applications incorporating multimedia functionality without having to program a DSP processor.*

At the heart of the DVEVM is a development board that includes a DM6466 chip and a variety of peripherals. The DM6466 chip contains an ARM9 general-purpose processor and a 'C64x+ DSP core. In addition to the development board itself, the kit includes external hardware components such as a camera and monitor, development tools, an evaluation version of MontaVista Linux, a number of off-the-shelf audio and video algorithms, and ARM source code for several demonstration applications.

In this white paper, BDTI, an independent technology analysis company, evaluates the capabilities and ease-of-use of the DVEVM.

Contents

Introduction

The Texas Instruments Digital Video Evaluation Module (DVEVM) is intended to allow engineers to quickly evaluate the TI "DaVinci" processing engine and related tools, and to facilitate rapid prototyping of video-centric systems. The DVEVM kit includes a hardware board, software components including an evaluation version of MontaVista Linux and several off-the-shelf audio and video codecs (COmpression/DECompression algorithms), as well as off-board hardware (including a video camera and monitor). The on-board DaVinci chip includes an ARM9 core and a 'C64x+ DSP processor. The 'C64x+ implements signal-processing-intensive multimedia codecs via TI-provided software, and is accessed via API calls in application software running on the ARM9. Thus, the DVEVM is intended to enable system designers to develop video-capable products without having to program a DSP processor.

BDTI recently evaluated the ease-of-use of the DVEVM package and this paper summarizes our findings. This paper is primarily written for ARM-Linux embedded systems developers interested in knowing if the DVEVM will meet their needs as a vehicle for assessing the DM644x processing capabilities and as a platform to quickly begin ARM-based application software development. No familiarity with the DVEVM, TI's DaVinci family of processors, or DSP processors in general is assumed.

Unlike many other evaluations performed by BDTI, the focus here is not on the processor or chip architecture or on the processing capabilities of the processor itself, but rather on the features and usability of the DVEVM.

About BDTI

Berkeley Design Technology, Inc. (BDTI) is widely recognized for its long history as a credible source of independent analysis, evaluation, and benchmarking of processing engines and tools targeting digital signal processing applications. In addition to detailed evaluation of the performance and architecture of DSP processors, BDTI's analysis activities have included significant focus on general-purpose microprocessor-based systems incorporating digital signal processing capabilities. This is exemplified by BDTI's groundbreaking 500-page technical report, *DSP on General-Purpose Processors*. Additionally, BDTI has completed numerous software projects targeting general-purpose processors, using a variety of operating systems including Linux and WindowsCE. For further information see http://www.BDTI.com.

Description of the DVEVM

At the heart of the DVEVM board is TI's DM6446 chip, which contains an ARM9 general-purpose processor, a 'C64x+ DSP core, and a video acceleration module. In a typical design, the ARM9 serves as the "application processor" executing Linux, the user interface, and general system control, while the 'C64x+ DSP and the video acceleration module are used for executing DSP-intensive tasks such as multimedia codecs. The DVEVM does not expose the underlying DSP implementation details to the ARM software developer; rather it treats all DSP capabilities as functions of a "black box," allowing the ARM software to invoke off-the-shelf DSP-based codecs via a common calling mechanism and an application program interface (API).

Thus, the DVEVM is most suitable for Linux-ARM developers who want to incorporate DSP-intensive codecs into their application, without needing to understand the details of the DSP architecture, programming model, software development tools, or multimedia codecs.

The DVEVM kit includes many of the components necessary to assess the capabilities of the DaVinci processor and get started developing multimedia applications. The DVEVM includes the following:

- TMS320DM6446 DaVinci processor-based development board including numerous peripherals
- An NTSC video camera and a microphone
- An NTSC LCD monitor and speakers
- An IR remote control used to control the demos via the on-screen menus
- A DaVinci demonstration version of MontaVista Linux
- Source code and makefiles for demo applications (loopback, capture, playback)
- DaVinci ARM tools
- A "Getting Started Guide" (roughly 70 pages long) to guide the user through the end-to-end hardware and software setup, along with pointers to more detailed documentation.

The hardware contents of the DVEVM kit are shown in Figure 1, and a block diagram of the development board is shown in Figure 2. Additional information on the DVEVM can be found at the following web sites:
http://focus.ti.com/docs/toolsw/folders/print/tmdxevm6446.html.
http://c6000.spectrumdigital.com/davincievm/revd/.

The DVEVM demonstration applications provided by TI include the following:
Encode Decode Demo:

- H.264 (Baseline Profile) video loopback (simultaneous encode and decode)

Encode Demo:

- H.264 (Baseline Profile) Video Encode, G.711 Speech Encode
- MPEG-4 (Simple Profile) Video Encode, G.711 Speech Encode

Decode Demo:

- H.264 (Baseline Profile) Video Decode, G.711 Speech Decode or AAC Audio Decode
- MPEG-4 (Simple Profile) Video Decode, G.711 Speech Decode or AAC Audio Decode
- MPEG2 Video Decode, MPEG1 Audio Decode

BDTI Evaluation Methodology

In BDTI's view, potential DVEVM users are likely to have the following attributes:

- They are looking for a way to quickly assess the capabilities of the DaVinci processing engine.
- They are looking for a development environment in which they can efficiently begin development of application software on the ARM processor to differentiate their product.
- Their product will require significant DSP processing resources for multimedia functions, particularly video, and they want to use off-the-shelf software for these components rather than developing their own.

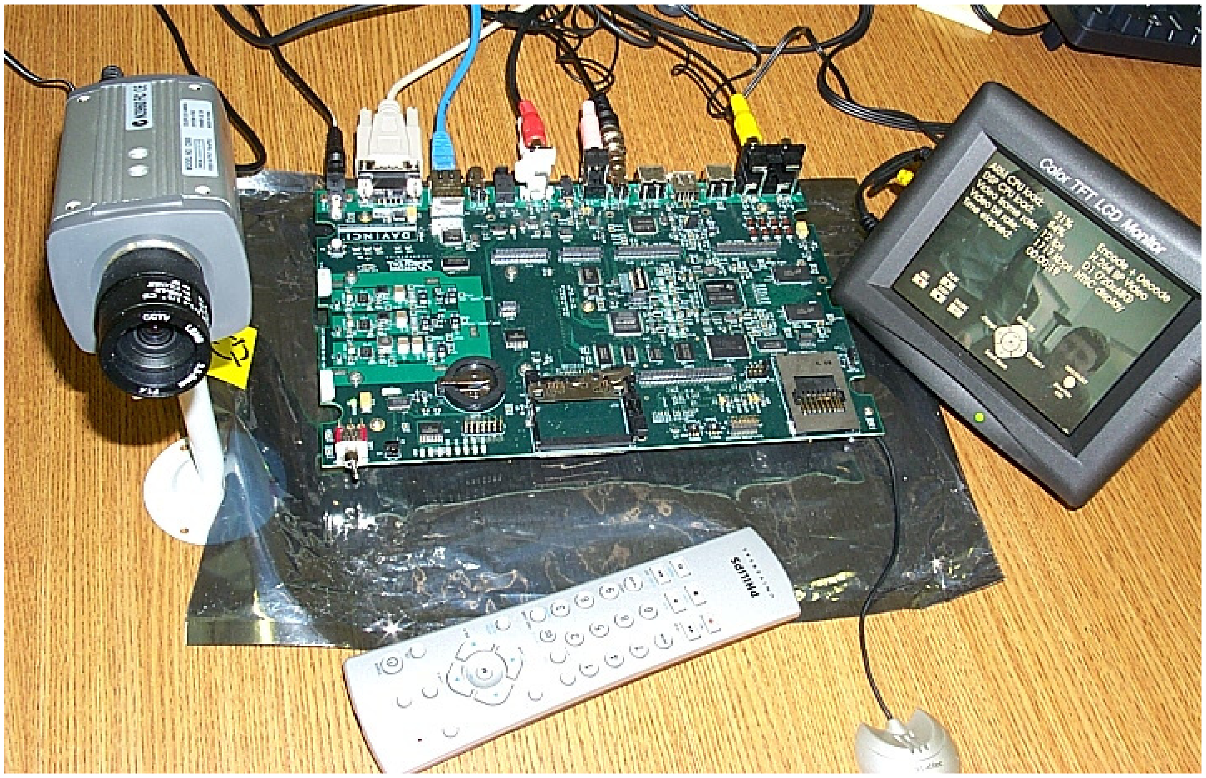


FIGURE 1. **Contents of the DVEVM kit.**

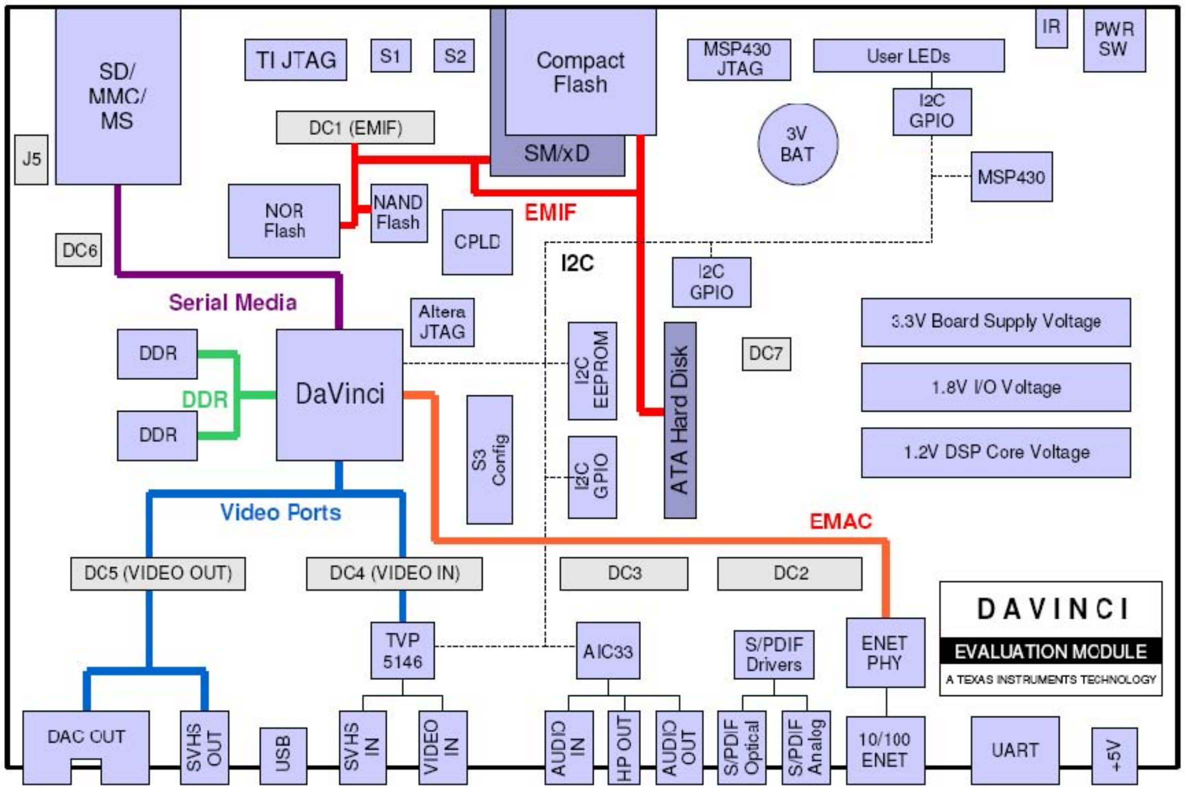


FIGURE 2. **DVEVM Block Diagram.**

BDTI's evaluation methodology, which closely followed the "DVEVM Getting Started Guide" document, is summarized in the following steps:

- Set up the board and experiment with the out-of-the-box multimedia-related demonstration programs.
- Install the tools and source code onto the host Linux machine. Review the ARM source code and documentation for the DVEVM example applications.
- Rebuild and test the multimedia codec example applications included with the DVEVM (including modifying parameters as supported by the DSP software).
- Using the TI-provided examples as a starting point, modify the code to evaluate the ease with which DVEVM users will be able to customize an application.

DVEVM Programming Model

As described earlier, the DM6446 processor contains an ARM9 CPU as well as a 'C64x+ DSP and video acceleration module. The DVEVM demo application code (e.g., system control, OS, user interface) runs on the ARM under Linux, accessing video, audio, and speech codecs residing on the DSP.

In the DVEVM, the ARM is an open software development environment—that is, TI provides the developer with source code that may be altered to modify the applications.

The 'C64x+ DSP processor, in contrast, is treated as a "black box." No source code is provided, and the user is limited to the functionality provided by the DSP software that is included with the DVEVM.

Note that the restrictions on the DSP capabilities are not due to limitations of the DVEVM hardware platform, but rather to limitations of the evaluation DSP software provided with the DVEVM. The provided DSP software is configured as a set of multimedia codec "packages." Each package provides certain capabilities (for example, H.264 encoding and decoding), which are accessed by the ARM software through TI's "Codec Engine" API.

DVEVM users who require additional multimedia codec capabilities may pursue the following options:

- Obtain evaluation versions of codecs available from TI's authorized software providers (ASPs). According to TI, these codecs use the Codec Engine APIs so they should have the same look and feel as those used for accessing the codecs provided with the DVEVM.
- Obtain TI's DaVinci Digital Video Software Development Kit (DVSDK), which includes an evaluation version of various codecs available from TI. The user can then use the DVSDKs eXpressDSP Configuration Kit to incorporate any number of available codecs and combine them into a custom executable software package.

BDTI did not obtain either of the above, and evaluated only the standard DVEVM DSP software capabilities. Further information on obtaining additional software modules is available at http://www.ti.com/digitalmediasoftware.

Out of the box, the DVEVM environment parameters are configured to boot MontaVista Linux and launch the demo application residing on the DVEVM hard drive.

TI also provides ARM source code for three demo applications that implement functionality similar to (but not exactly the same as) the demos resident on the DVEVM hard drive.

After the software and tools are installed on the host Linux machine, the user can rebuild both the MontaVista Linux kernel and ARM demo application software using supplied makefiles and instructions.

Since the user will probably perform the software development on a Linux host machine rather than on the DVEVM development board, the TI documentation pro-

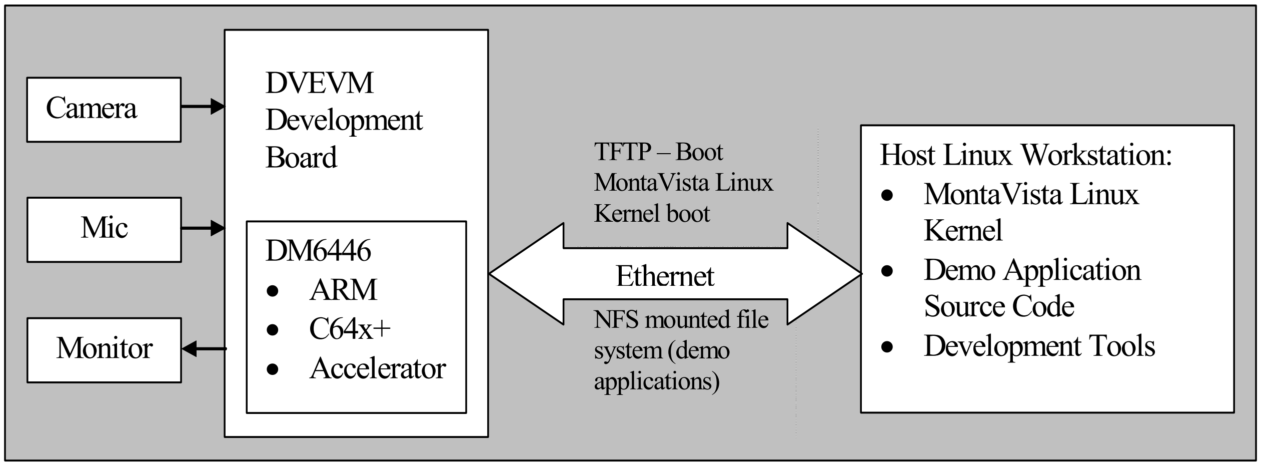

Camera
Mic
Monitor
DVEVM Development Board
DM6446
• ARM
• C64x+
• Accelerator
TFTP – Boot MontaVista Linux Kernel boot
Ethernet
NFS mounted file system (demo applications)
Host Linux Workstation:
• MontaVista Linux Kernel
• Demo Application Source Code
• Development Tools

FIGURE 3. DVEVM system-level block diagram—the DVEVM target platform and how it interacts with the host Linux workstation in a typical development environment.

vides instructions on how to set the environment variables and configure the system to perform the following (see Figure 3):

- Boot the DVEVM by loading the MontaVista Linux kernel (which can be rebuilt) stored on the host Linux machine via TFTP.
- Configure the DVEVM to NFS-mount a file system on the host Linux machine which contains the demo application source code that can be modified and rebuilt.

Once the system is configured as shown, all software development can be done on the host Linux workstation, then executed and tested on the DVEVM target development board. This leaves the preconfigured version of the demo software intact on the DVEVM hard drive, where it can later be used to verify hardware functionality if necessary.

Out-of-the-Box Experience

Getting started with a new hardware kit can be a frustrating process. You may need to write, or at least compile, a segment of code to determine if the hardware is alive and the tools are correctly installed. Then you may want to do something more complicated, such as get a real-time video or audio codec running with external devices such as a camera or monitor. To do this may require you to select and purchase a camera and monitor, then link in drivers for required peripherals into your application, as well as obtain and integrate video and audio codec software modules.

In this respect, TI's DVEVM was a pleasant surprise. Within an hour we had the board working and running the full-duplex H.264 demo application using the camera and display provided with the DVEVM package. All of the preconfigured demos, which are resident on the DVEVM hard drive, can be controlled with a remote control (included) by following the on-screen-display menu appearing after system boot. The entire process is very simple and intuitive—similar to the out-of-box experience associated with a consumer product such as a home video or audio system.

From the main menu shown in Figure 4, the Encode + Decode demo allows you to view the video captured by the camera after it is compressed and decompressed, the Encode demo records compressed audio/speech and video in a limited number of selectable formats, and the Decode demo decompresses and plays recorded audio/speech and video files. The Third Party Menu can be used to add additional demos from Texas Instruments' ASPs.

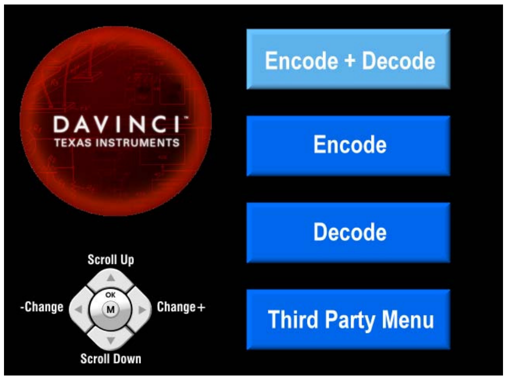
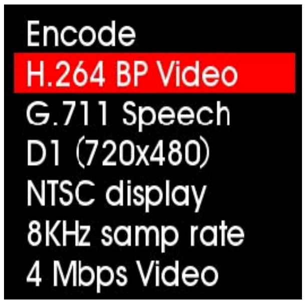


FIGURE 4. Main menu.

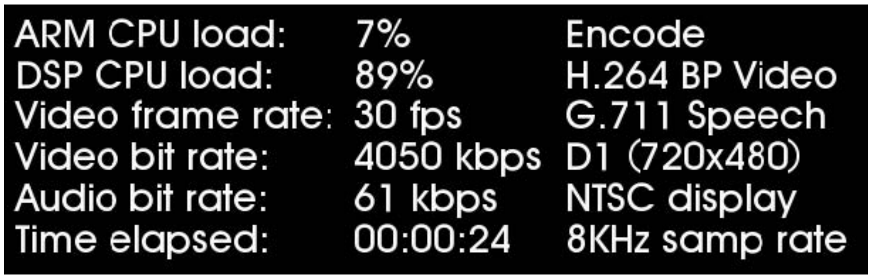
After selecting a demo, a settings screen, as shown in Figure 5, appears allowing the user to configure a limited set of codecs and parameters.



```
Encode
H.264 BP Video
G.711 Speech
D1 (720x480)
NTSC display
8KHz samp rate
4 Mbps Video
```

FIGURE 5. Settings screen.

While the demo is running, information on the demo configuration and coarse processor utilization statistics can be displayed using a toggle button on the remote control.



```
ARM CPU load:      7%          Encode
DSP CPU load:      89%         H.264 BP Video
Video frame rate:  30 fps      G.711 Speech
Video bit rate:    4050 kbps   D1 (720x480)
Audio bit rate:    61 kbps     NTSC display
Time elapsed:      00:00:24    8KHz samp rate
```

FIGURE 6. Demo configuration and coarse processor utilization statistics.

An example of this information is shown in Figure 6. These statistics should be sufficient to give the user a general sense of the processors' utilization (e.g., the ARM9 is shown as being lightly loaded, while the 'C64x+ DSP is shown as 89% utilized, and thus has little headroom available). More accurate processor utilization statistics would require the use of the DM644x SoC analyzer, which is included with TI's (separate) Digital Video Software Development Kit (DVSDK).

The demos can also be invoked via command line, by connecting the DVEVM UART to a COM port on your PC and launching a hyperterminal session.

TI provides ARM source code for a set of command-line demo applications that implement very similar functionality to these menu-driven applications. This source code can be rebuilt and modified as detailed below.

Configuring the DVEVM and Host Linux Machine

The process of rebuilding the provided ARM source code for the demo applications and developing new applications would typically be done on a host Linux machine as described in the "DVEVM Programming Model" section above. This section describes our findings when configuring the DVEVM and host Linux machine for this purpose.

We came across several minor documentation-related issues when configuring our DVEVM and Host Linux system, including:

- The Linux system requirements were not mentioned in the documentation, so we had to make some tweaks to our configuration as we went along.
- Several installation instructions that the documentation said should take "several minutes" in fact took over 30 minutes (this is probably due to the fact that we were using a virtual Linux machine that was slower than typical).
- Occasionally, relevant information or commands were not mentioned the first time they would have been useful. For example, the hyperterminal settings for connecting the PC COM port to the DVEVM target UART are not mentioned in the section that described how to run the demos from the command line, where we first needed them.

The above issues were inconveniences, but didn't block our progress. However, we did come across more problematic documentation errors.

When mounting the NFS file system, the "Getting Started Guide" contains a command sequence that does not work as shown in the manual. An email to TI technical support was quickly answered with a supplemental document showing an updated syntax that did resolve the issue, but we were initially frustrated while unsuccessfully trying the incorrect command.

Once the NFS file system was successfully mounted, we were able to write, compile, and run a simple "Hello World" program in a matter of minutes.

Then we rebuilt the Linux kernel on our Linux workstation and attempted to have the DVEVM boot using the new Linux kernel via TFTP. Again the "Getting Started Guide" instructed us to execute a command sequence that did not work for us; we later discovered that the command sequence that was specified is incompatible with some switches/routers, including ours. We resolved this issue by looking at the DaVinci Linux open source web site, where we discovered that other DVEVM users had experienced similar problems and a recommendation was posted. This recommendation resolved the issue, and then we were able to successfully boot the DVEVM with our newly compiled MontaVista Linux kernel via TFTP and run the new demo applications via the NFS mounted file system.

We view the problems we encountered while configuring our system to be primarily a reflection of the relative immaturity of the DVEVM documentation and the complexities that arise when networking with equipment from various manufacturers. The two problems described above were the only significant issues we came across when following the 70-plus page "Getting Started Guide."

Furthermore, we found TI technical support to be quite responsive.

Demo Application ARM Source Code

TI provides source code for the ARM demo applications, makefiles and documentation on how to build them, and a "readme" file that nicely describes how to use them. We were able to build and test the demo applications in a matter of minutes by following the instructions in the "Getting Started Guide."

The code is structured as a multithreaded application using standard POSIX Pthreads. The demos are composed of the following threads:

- Control thread – General tasks such as managing the on screen display and handling user interface commands.
- Video thread – Invoke the video encoder and decoder, and manage the data they supply and consume.
- Audio thread – Invoke the audio encoder and decoder, and manage the data they supply and consume.

Although relatively simple, the ARM demo software is structured in a manner representative of a realistic application, and performs a significant task (e.g., video compression and decompression). The demos are small enough and organized in such a way as to allow a new user to read and understand the general flow in a relatively short time, particularly since the functionality closely follows that of the out-of-the-box menu driven demo applications. While the source code itself is logically organized and readable, the comments are fairly sparse and there is no software design document describing the overall software architecture.

As an example, the following is a code fragment from the video thread in the video loopback (Encode + Decode) demo:

```
DBG("videoThrFxn: Entering video main loop.\n");
   while (!getQuit()) {
       if (waitForFrame(captureFd) == FAILURE) {
           breakLoop(THREAD_FAILURE);
     }
       /* Don't play if the demo is paused */
       if (!getPlay()) {
           usleep(PAUSE);
           continue;
     }
       /* Dequeue a frame buffer from the capture device driver */
       if (ioctl(captureFd, VIDIOC_DQBUF, &v4l2buf) == -1) {
           if (errno == EAGAIN) {
               continue;
        }
           ERR("VIDIOC_DQBUF failed (%s)\n", strerror(errno));
           breakLoop(THREAD_FAILURE);
     }
       /* Encode the buffer using H.264 */
       frameSize = encBufSize;
       if (encodeVideoBuffer(hEncode, vidBufs[v4l2buf.index].start,
           captureSize, encBuf, &frameSize) == FAILURE) {
           breakLoop(THREAD_FAILURE);
     }
       /* Issue capture buffer back to capture device driver */
       if (ioctl(captureFd, VIDIOC_QBUF, &v4l2buf) == -1) {
           ERR("VIDIOC_QBUF failed (%s)\n", strerror(errno));
           breakLoop(THREAD_FAILURE);
     }
       /* Decode the buffer using H.264 */
       dst = displays[workingIdx];
       if (decodeVideoBuffer(hDecode, encBuf, frameSize, dst,
           captureSize, &framesDropped) == FAILURE) {
           breakLoop(THREAD_FAILURE);
     }
       /* Increment statistics for OSD display */
       incFrames(); incVideoBytesEncoded(frameSize);

       /* Flip display buffer and working buffer */
       flipDisplayBuffers(fbFd, displayIdx);
   }
```

The ARM software interface to the DSP is via TI's Codec Engine, on which TI provides extensive documentation. From the application developer's perspective, the codec engine is a common set of APIs that allow instantiation and execution of compliant algorithms in a uniform manner.

For example, the `encodeVideoBuffer()` function from the above code fragment performs little more than make the following Codec Engine API call:

```
VIDENC_process(hEncode,&inBuf-
Desc,&outBufDesc,&inArgs,&outArgs);
```

The process of adding new codecs offered by TI ASPs that are compliant with the Codec Engine requirements follows the same methodology as invoking the codecs and APIs in the demo software. The fact that the Codec Engine shields the ARM-Linux software developer from the underlying DSP complexities is likely to result in a reduction in development time and a reduction in the required level of expertise for the application-side software developer.

Description of Modifications made by BDTI

After testing and reviewing the source code for the example applications provided by TI, we made modifications to the code to create our own applications.

There is no ARM-Linux debugger provided with the DVEVM, although the open source GNU Data Display Debugger (DDD) can be used. If you upgrade to the DVSDK, the MontaVista DevRocket IDE debugger is included, and GreenHills offers an IDE that supports both the ARM and DSP simultaneously. Since the amount of software development we performed was relatively modest and we started with functional demo applications provided by TI, we simply used "printf()" statements for our debugging.

Prior to receiving the DVEVM and the accompanying documentation, we had planned, as part of our evaluation, to create a simple application supporting audio/video loop-back, play-back, and capture. We felt that this application would be large enough to credibly assess the DVEVM, yet simple enough to be accomplished with a reasonable level of effort. We were pleased to find that the DVEVM provides source code for these exact demo applications. So, rather than creating an all-new application, we created our own application by making relatively minor modifications to TI's code, including the following:

- In the video loop-back demo we added saving the encoded and decoded frames to the hard disk drive. This required mounting the HDD, which the TI documentation described. We modeled our implementation on the Encode demo source code, which includes saving the compressed bitstream to the HDD, and did not encounter any problems adding this functionality.
- In the Encode (capture) demo we added the capability to display the video data in real-time simultaneously with capturing it to the HDD. Note that this functionality is supported in the "out-of-box" remote control driven Encode demo that the DVEVM boots from its hard drive at start up. However, this capability was not included in the command-line Encode demo source code provided. We modeled our design based on the Decode demo source code, and no problems were encountered.
- On the video loopback demo, we adjusted the amount of delay (i.e., latency) between the encoder and decoder by adding an adjustable depth "delay buffer." This was implemented by writing the encoded data to the HDD, then waiting a specific number of frames before reading, decoding, and displaying it. Thus, this functioned as a simple initial implementation of a "jitter buffer" as might be used in a video-over-IP system. Again, no problems were encountered beyond the normal debugging process.

Next we wanted to create more elaborate applications that would use the DSP (via the Codec Engine) in ways not already supported in the demo applications provided by TI. For example, we considered the following:

- Adding a second instance of an encoder to the encode demo. This would support simultaneously capturing an incoming video stream in both H.264 and MPEG4 formats, for example.
- Decoding a compressed bitstream from the HDD using one type of codec, and then encoding the uncompressed decoder output video stream using a different type of video codec and saving the data to the HDD (i.e., a simple transcoding application).

At this point, however, we discovered that the DSP code provided with the DVEVM only allows certain codec combinations: essentially, the only combinations supported are the ones for which TI provides demos. Prior to a relatively detailed examination of the DVEVM capabilities, these restrictions were not clear to us. A user who wanted to implement a different set of codec combinations would need to either:

- Obtain the DVSDK from TI and utilize the eXpress-DSP Configuration Kit to combine the desired codec combinations into a custom "package."
- Work with a TI ASP to obtain a DSP evaluation package supporting the codecs required for their particular needs.

We did not proceed on either of these options. Further information on obtaining additional software modules is available at www.ti.com/digitalmediasoftware.

Conclusions / Highlights of our Analysis

For most application developers, we feel that the DVEVM will provide the capabilities necessary to:

- Determine if the DM644x DaVinci processor is likely to be a fit for their application
- Quickly get a feel for the DM6446 ARM development environment
- Begin prototyping multimedia applications on the ARM

Although some ARM-Linux developers will require supplemental DSP software beyond that included with the DVEVM, and we uncovered some errors in the DVEVM documentation, we were generally impressed with the quality of the DVEVM hardware, software, documentation, and technical support—particularly the user-friendly initial out-of-the-box experience. We feel that the DVEVM delivers on TI's stated mission: allowing typical ARM-Linux developers of applications incorporating video and multimedia to obtain an initial assessment of the DM644x DaVinci tools and processor capabilities, and quickly get started prototyping applications on the ARM without having to get bogged down with the details of programming the DSP.