Application Report

SPRAAF6–September 2006

DaVinci System Level Benchmarking Measurements

Zhengting He, Mukul Bhatnagar, Jackie Brenner

ABSTRACT

The DaVinci™ platform offers a complete solution for many multimedia applications requiring advanced video codecs. The solution consists of a DM644x dual core architecture that offers high performance along with a rich mix of peripherals and a complete software environment. This environment includes inter-processor communication software via DSP/BIOS™ Link, and a codec framework that enables customers to use a variety of codec options.

As with any system many developers would like to understand the performance taking into account processor loading, memory usage and power consumption. In critical applications performance and power consumption analysis is often necessary to evaluate a device within a system context.

The Digital Video Evaluation Module (DVEVM) is an evaluation platform that showcases the DM644x architecture and the associated digital video and audio system software solutions. It is packaged and designed to enable users to evaluate both performance and power (energy) usage of the DM644x solution.

This document describes how the performance and power consumption can be evaluated in a system context making use of the components available with the DVEVM and Digital Video Software Development Kit (DVSDK). The performance/ CPU loading and power consumption is measured for the H.264 audio/video demonstration software that is offered with the DVEVM. This methodology can also be utilized to measure performance and power for a developer's own application software on the DVEVM within the above context.

Contents

List of Figures

List of Tables

Trademarks

DaVinci, DSP/BIOS are trademarks of Texas Instruments.

1 Introduction

The DaVinci platform offers a complete solution for many multimedia applications requiring advanced video codecs. The solution consists of a DM644x dual core architecture that offers high performance along with a rich mix of peripherals and a complete software environment including inter-processor communication software via DSP/BIOS Link and a codec framework that enables customers to use a variety of codec options. The heterogeneous multiprocessor system also runs under a Linux environment allowing many applications to be built on top of the multimedia codec base.

As with any system many developers would like to understand the performance taking into account processor loading, memory usage and power consumption. In critical applications performance and power consumption analysis is often necessary to evaluate a device within a system context.

In order to evaluate performance and understand the loading on the CPU(s), visualization of the interactions of the various hardware and software components is necessary to identify contentions and bottlenecks. Developers also want to “time-stamp” process intensive critical threads/functions to ensure they design an application with efficient resource utilization. Similarly understanding the power consumption is necessary for battery life estimates, power supply design, thermal analysis and comprehending the overall cost and complexity of a system.

The Digital Video Evaluation Module (DVEVM) is an evaluation platform that showcases the DM644x architecture and the associated digital video system software solutions. It is packaged and designed to enable users to evaluate both performance and power (energy) usage of the DM644x solution. The performance and power consumption can be evaluated in a system context making use of the components available with the DVEVM and Digital Video Software Development Kit (DVSDK). The performance/CPU loading and power consumption is measured for the audio/video demonstration software that is offered with the DVEVM. Specifically, the following three demos were used for the measurement:

- Decode demo configured as a H.264 decoder (30 fps and NTSC D1 resolution) plus an AAC decoder (160 kbps) running.
- Encode demo configured as a H.264 encoder (4Mbps, 30 fps and NTSC D1 resolution) running
- Encode/Decode demo configured as a H.264 encoder (1 Mbps, 30 fps and CIF resolution) and a H.264 decoder (30 fps and CIF resolution).

The H.264 algorithm demos were selected because they are relatively more complex and processing intensive compared to other video algorithms like MPEG4, MPEG2, etc. Therefore, the H.264 demos are an appropriate vehicle for measuring performance and power in a system context for the purposes of this application report.

The CPU loading is measured by utilizing the DM644x System on a Chip (SoC) Analyzer, which is a real-time graphical interface tool that provides system level analysis and visualization capabilities for developer's digital video and audio applications running on DM644x devices.

This application note provides the following:

- Section 2 provides a brief overview of the demonstration software that comes with the DVEVM.
- Section 3 outlines the software and hardware requirements to perform the analysis.
- Section 4 describes the methodology to calculate ARM and DSP loading making use of the DM644x SoC Analyzer and provides the loading measured on specific DVEVM demos.
- Section 5 describes the memory usage for the demo application software.
- Section 6 describes the provisions available on the DVEVM to measure power and provides power consumption data measured for some of the DVEVM demos.
- Section 7 provides a summary of the key data presented in Section 4, Section 5, and Section 6.

This document is written so that a developer can use the DVEVM/DVSDK hardware and software to reproduce the documented results to evaluate TI DM644x system solutions. This methodology can also be utilized to measure performance and power for a developer’s own application software on the DVEVM within the above context.

2 DVEVM Demonstration Software Descriptions

2.1 Demo Software Architecture

Figure 1 shows the demo software architecture running on the DaVinci processor. The codec algorithms execute on the DSP processor. The ARM side application program executes the DSP side codec via the Codec Engine. The Codec Engine calls the DSP link driver to realize the ARM-DSP communication.

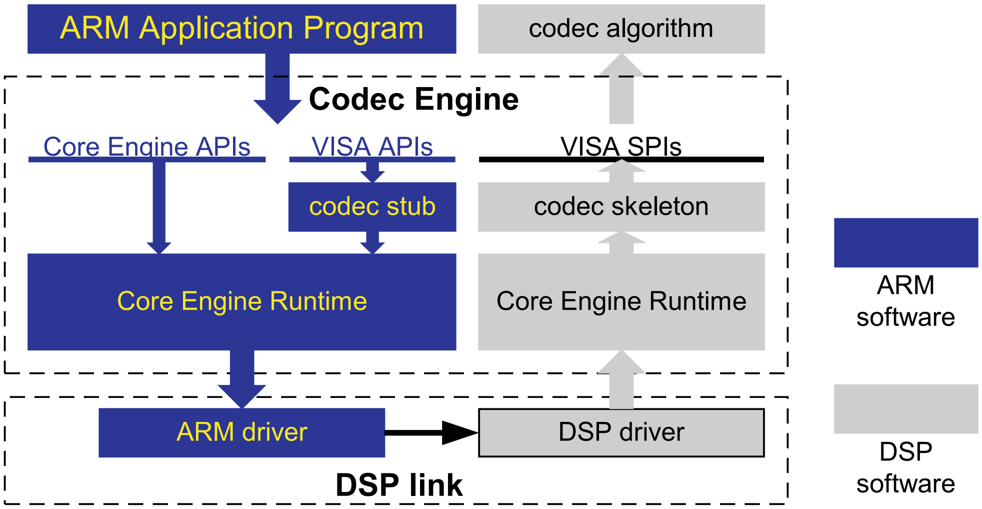


Figure 1. Demo Software Architecture

From the application developer's perspective, the Codec Engine provides a set of XDM APIs that can be used to instantiate and run eXpress DSP Algorithm Interface Standard (known as XDAIS) algorithms. XDM is used for the eXpressDSP Algorithm Interface Standard for Digital Media. An XDM algorithm is compliant with XDAIS. Additionally, it implements an extension of digital media APIs to support multimedia codecs. The XDM interfaces divide codec algorithms into four classes: Video, Image, Speech, and Audio (VISA). One set of APIs is provided per codec class.

XDM does not prevent users from creating a codec which does not belong to a VISA class. However, the developer needs to use the same methodology to create his/her own codec stub and skeleton to allow the application program to access the codec. For details on the Codec Engine, please refer to the *Codec Engine Application Developer User's Guide* (SPRUE67) [1].

2.2 Decode Demo

Figure 2 shows the block diagram of the decode demo.

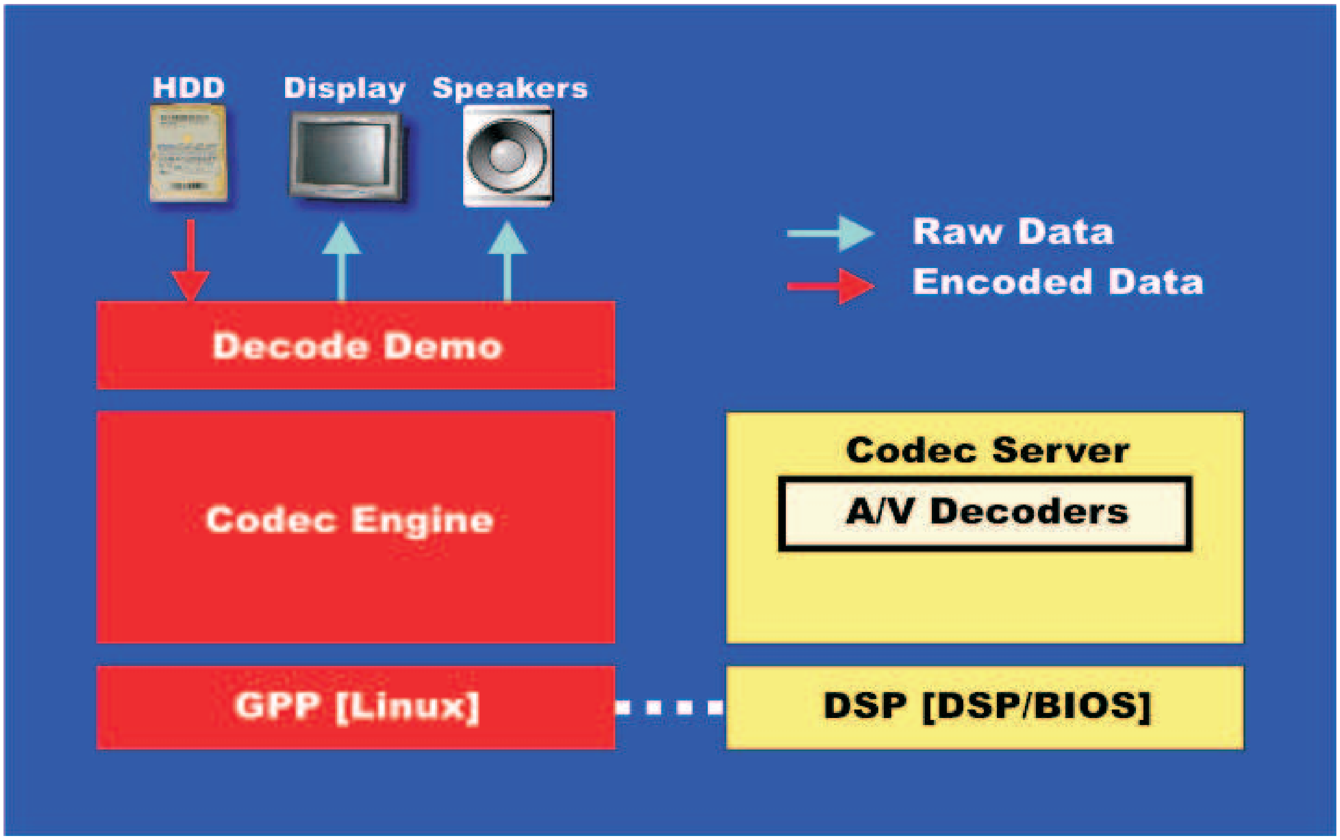


Figure 2. Decode Demo Block Diagram

This first demo utilized in this application report can simultaneously decode a video stream file and an audio or speech stream file. The files are read from the hard disk by the ARM processor and delivered to the DSP frame-by-frame via the Codec Engine. After a video frame is decoded by the DSP, the raw video data are delivered back to the ARM. The video driver on ARM will transmit the video frame to the LCD for display. Similarly, after an audio (speech) frame is decoded by the DSP, the raw audio (speech) data are delivered back to ARM which then transmits the data to the speakers.

The specific set up for the decoder benchmark data collection is described in Section 4.2.

2.3 Encode Demo

Figure 3 shows the block diagram of the encoder demo.

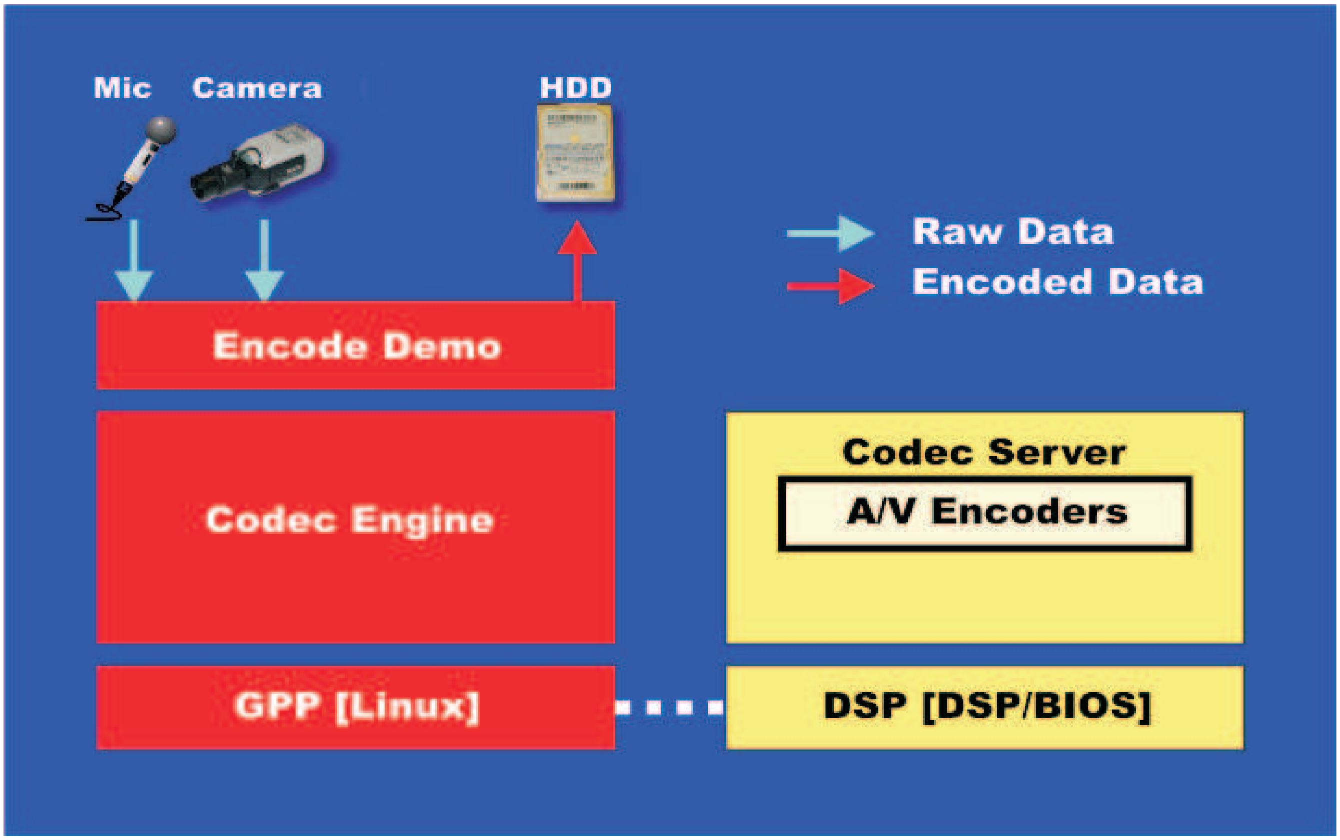


Figure 3. Encode Demo Block Diagram

The second demo utilized in this application report can simultaneously encode a video channel and a speech channel.

For the video channel, the raw video data are captured from the composite video interface to which either a CCDC camera or a DVD output can be connected. The data collected for this application report uses the camera input. The ARM processor delivers the raw video data to the DSP frame-by-frame via the Codec Engine. Meanwhile, the captured frame is also transmitted to the LCD for display by the video port driver. After a video frame is encoded, the DSP delivers the compressed frame back to ARM which writes it to the hard disk.

For the speech channel, the raw speech samples are captured from the ASP interface which connects to a microphone. Similarly, the ARM processor delivers the raw speech samples block-by-block to the DSP for encoding via the Codec Engine. After the samples are encoded, the DSP delivers the compressed data back to ARM which writes it to the hard disk.

The specific set up for the encoder benchmark data collection is described in Section 4.2.

2.4 Encode/Decode Demo

Figure 4 shows the block diagram of the Encode/Decode demo.

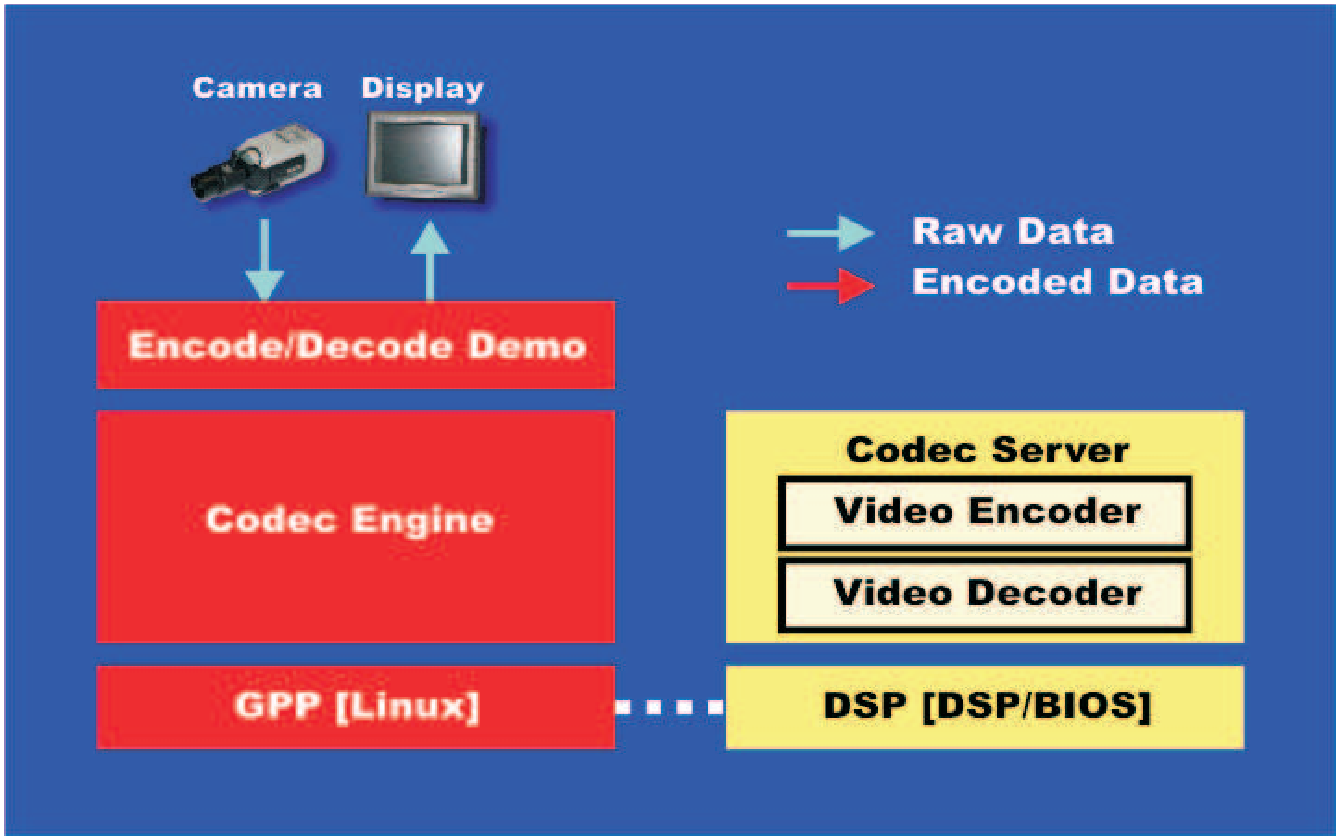


Figure 4. Encode/Decode Demo Block Diagram

The Encode/Decode demo allows you to record and playback video. The supported encode and decode algorithms are H.264 Baseline Profile. It does not encode and decode audio or speech.

The raw video data are captured from the composite video interface. The ARM processor delivers the raw video data to the DSP frame-by-frame via the Codec Engine. The DSP encodes and then decodes the frame. After that the decoded frame is delivered back to ARM. The video port driver will transmit the decoded frame to the LCD for display.

3 Necessary Equipment and Software

Figure 5 shows a diagram of the measurement setup. The measurement was performed on DaVinci DVEVM with silicon revision 1.3. The ARM, DSP and DDR2 memory were running at 297 MHz, 594 MHz and 162 MHz, respectively. The reported measurement results were measured from demo version 1.10, using DM644x SoC Analyzer version beta. The Linux kernel version is 2.6.10_mvl401-davinci-evm which can be found in file /proc/version on the target Linux system.

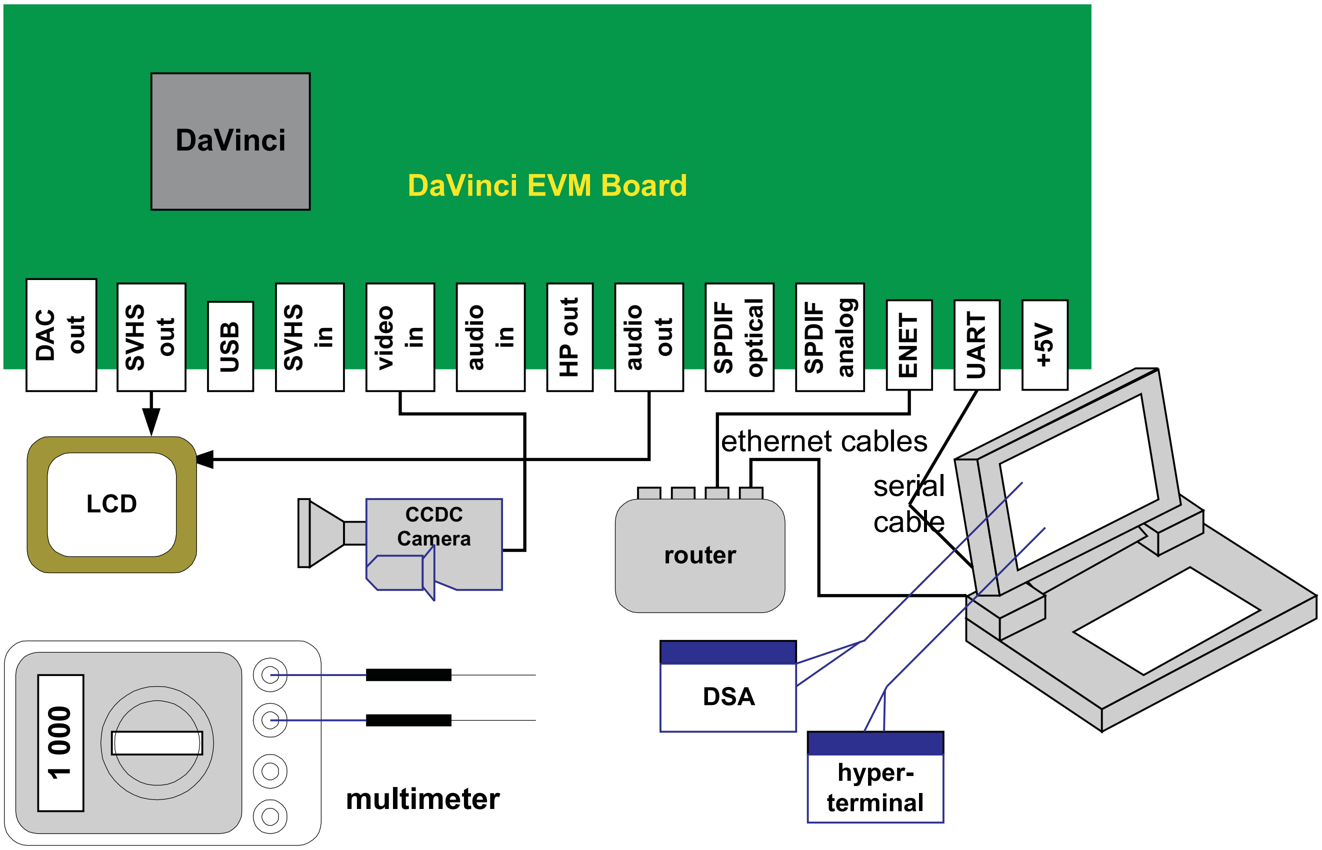


Figure 5. Measurement Setup

3.1 Equipment

The following hardware equipment is necessary to measure the power and processor loading for the three demos.

- DVEVM board
- CCDC camera which captures the input video source to execute the encode and encode/decode demos.
- LCD monitor which displays the video and audio output for the three demos.
- Two Ethernet cables: one is for the DVEVM board, and the other is for your host PC. The DM644x SoC Analyzer runs on the host PC and is used to measure demo loadings. It needs to establish an HTTP connection to the Linux HTTP server running on the target. Timestamps of events of interest will be logged to the target file system. The Analyzer pulls the log files from the DVEVM, performs some post-processing and then displays the results on the PC.
- Serial cable that connects the DVEVM UART0 port to any COM port on the host PC.
- Multimeter to measure the power for the three demos. (See Section 6.1)

3.2 Software

The following software needs to be installed on the host PC before measuring the loadings for the three demos:

- A terminal program for windows. The recommendation is to use hyperterminal which comes with default windows installation.
- DM644x SoC Analyzer which comes with the DVEVM package. It is a JAVA based program which virtually can run on any host PCs.

The Linux version on target needs to be 2.6.10_mvl401-davinci_evm or later. The *DVEVM Getting Started Guide* (SPRUE66) [2] provides instructions on how to build a Linux kernel using the default configuration. To measure the processor loadings for the demos, a new Linux kernel needs to be built with two additional modules included in the kernel: (1) the Linux Trace Toolkit (LTT) and (2) the relay file system support.

The following are the instructions for building the new kernel.

- Switch to the directory which contains the Linux kernel source. By default, it is .

  /home/[YOUR_USER_ACCOUNT]/workdir/lsp/ti-davinci

  Type:

  make ARCH=arm CROSS_COMPILE=arm_v5t_le- menuconfig
- Go to Device Drivers → File systems → Pseudo filesystems → Relayfs file system support, and select it as built-in.
- Go to General Setup → Linux Trace Toolkit support and select it as built-in.
- Type:

  make ARCH=arm CROSS_COMPILE=arm_v5t_le- checksetconfig
- Type:

  make ARCH=arm CROSS_COMPILE=arm_v5t_le- uImage

4 Measuring the Processor Loading on the Demo Programs

4.1 Approach to Measure the Processor Loading

The DM644x SoC Analyzer is used to measure the processor loadings for the three demos. This section will not explain the details on how to use the analyzer, but will focus on the methodologies utilized and some common errors made by first time users. The DM644x SoC Analyzer online documentation [3] provides details on its usage.

Before booting the DVEVM board, the host PC needs to establish a serial connection with the board. The terminal program on the PC allows the user to boot and control the Linux running on the board.

After the Linux kernel has booted on the DVEVM, an HTTP connection needs to be established between the DM644x SoC Analyzer on the host PC and the DVEVM. A demo can be initiated by typing the appropriate commands in the terminal program or using the remote controller.

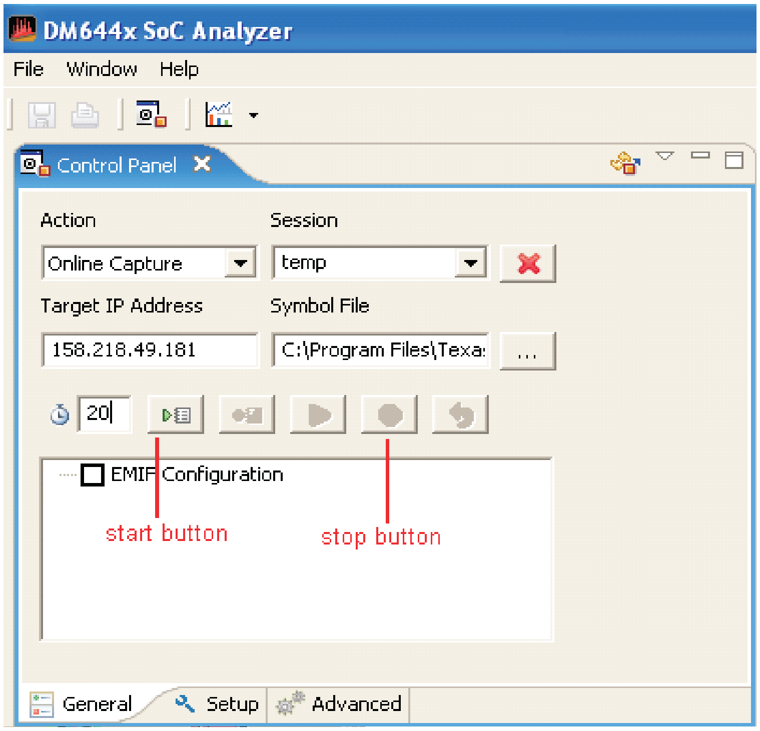
The DM644x SoC Analyzer allows the developer to control when to start/stop measurement of the loadings while the demo is running. The analyzer sends the commands to the DVEVM via the HTTP connection. While the loadings are being measured, timestamps of all events of interest are logged to files on the target file system. The typical events of interest are as follows:

- ARM process and DSP task events. For example, the event can be an ARM thread that starts pre-processing the data for the next frame.
- Events of ARM-DSP inter-processor call (IPC) made in Codec Engine. For example, the event can be the ARM issuing a command via Codec Engine to the DSP to process the next frame.
- Linux file system events, i.e., ARM writing the encoded stream to hard disk and etc.

The default location for the log files are in the /tmp directory; a RAM disk is mounted there and the data logging impacts the demo performance less when the disk I/O is avoided. The logging mechanism is based on the Linux Trace Toolkit (LTT) which needs to be enabled in the Linux kernel. LTT requires a memory buffer pinned in the Linux kernel space for logging events. Generally speaking, the more types of events the user wants to capture and/or the longer the interval the user wants to measure, the bigger the buffer needs to be. Usually it is safe to allocate two buffers with 262,144 bytes each to measure about 20 seconds.

After the measurement is performed, all the log files can be transferred to the host PC. The log files are post processed and the results will be displayed in the window. These are the processing steps:

- Launch DM644x SoC Analyzer on a windows host and open the control panel by clicking the Control Panel button as shown in Figure 6.
- Select Online Capture in the Action box.
- Input a desirable session name in the Session box.
- Input your target board’s IP address into the Target IP Address box.
- Select the appropriate symbol file in the Symbol File box for your measurement, i.e., use file encodeCombo.x64P if the encode demo is to be measured.
- Input the interval (in seconds) to be measured in the time box. In Figure 6, it is 20 seconds as an example.
- Start the demo on your target board.
- Start measuring by clicking the "start" button in the control panel.
- Wait until the timer expires or click the “stop” button to terminate the measurement. The logged data will be pulled from the target board to the windows host. After the post-processing is done by the DM644x SoC Analyzer, the results will be displayed.



Action: Online Capture

Session: temp

Target IP Address: 158.218.49.181

Symbol File: C:\Program Files\Texa

20

Figure 6. DM644x SoC Analyzer Control Panel

For further details please refer to The DM644x SoC Analyzer online documentation [3].

Some common errors which can be made when using the DM644x SoC Analyzer are listed below:

- The HTTP connection between the DM644x SoC Analyzer and the DVEVM board is not established. The DM644x Soc Analyzer will generate an error log message indicating that the web server cannot be reached. This error is typically caused by the following reasons:
  1. The http server was not started on the DVEVM.

2. The correct IP address of the DVEVM was not provided.

- The measured results cover a time interval that is smaller than expected. This can typically be caused by two reasons:
  1. Not enough memory was allocated for LTT. To remedy this either allocate a bigger buffer or reduce the number of events to be logged.
  2. The demo already terminated before event logging was stopped. To correct this, please make sure to start and stop logging while the demo is still running.
- Log files are not generated for the measurement. It is typically because the Linux kernel was not built to support LTT and/or relays. Try to follow the instructions in Section 3.2 to build a kernel which supports loading measurement.
- The results shown by DM644x SoC Analyzer are not valid. This is typically because the symbol file run by the demo does not match the one used by the SoC analyzer for data post-processing. Please make sure to use the same symbol file for both the demo and the SoC Analyzer.

4.2 Demo Configuration for Measurement

The following subsections describe the setup and measurement methodology for each of the demos used.

4.2.1 Decoder Demo Setup

The configuration for the decoder demo measurements is as follows: Linux kernel version 2.6.10_mvl401-davinci_evm, demo version 1.10 and DM644x SoC Analyzer version beta was used.

This demo executes an H.264 video decoder and an AAC audio decoder. Files davincieffect_ntsc.264 and davincieffect.aac are used as video and audio input, respectively.

The H.264 stream is NTSC D1 resolution (720x480 pixels/frame), 4 mbps, and being decoded at 30 fps.

The AAC stream is 160 kbps. Given the fact that the audio sample frequency is 48 KHz and the AAC decoder roughly produces 1K samples each frame, its frame rate can be considered as 48 fps.

The measurement starts after the demo has been run for about 15s and ends at 35s, which makes the measurement interval about 20s.

The command to start the demo from command line is as follows:

./decode -a data/sounds/davincieffect.aac -v data/videos/davincieffect_ntsc.264

4.2.2 Encoder Demo Setup

The configuration for the encoder demo measurements is as follows: Linux kernel version 2.6.10_mvl401-davinci_evm, demo version 1.10 and DM644x SoC Analyzer version beta was used.

It executes an H.264 video encoder to encode video at D1 resolution (720x480 pixels/frame), 4 mpbs and 30 fps. The input raw video frames are captured by the camera. During the measurement, a human hand waves about twice per second in front of the camera. The distance between the hand and the camera is about 1 foot.

The measurement starts after the demo running for about 5s and lasts about 20s

The command to start the demo from command line is as follows:

*./encode -v output.264 -b 4000000 -r 720x48*0

4.2.3 Encode/Decode Demo Setup

The configuration for the encode/decode demo measurements is as follows: Linux kernel version 2.6.10_mvl401-davinci_evm, demo version 1.10 and DM644x SoC Analyzer version beta was used.

This demo executes an H.264 video encoder and decoder at CIF resolution, 1 mbps and 30 fps. Frames are captured from the camera at D1 resolution with the same human gesture as described in Section 4.2.2. Before being delivered to DSP, they are cropped to CIF resolution.

The measurement starts after the demo running for about 5s and lasts about 20s.

The command to start the demo from command line is as follows:.

./encodedecode -b 1000000 -r 352x240

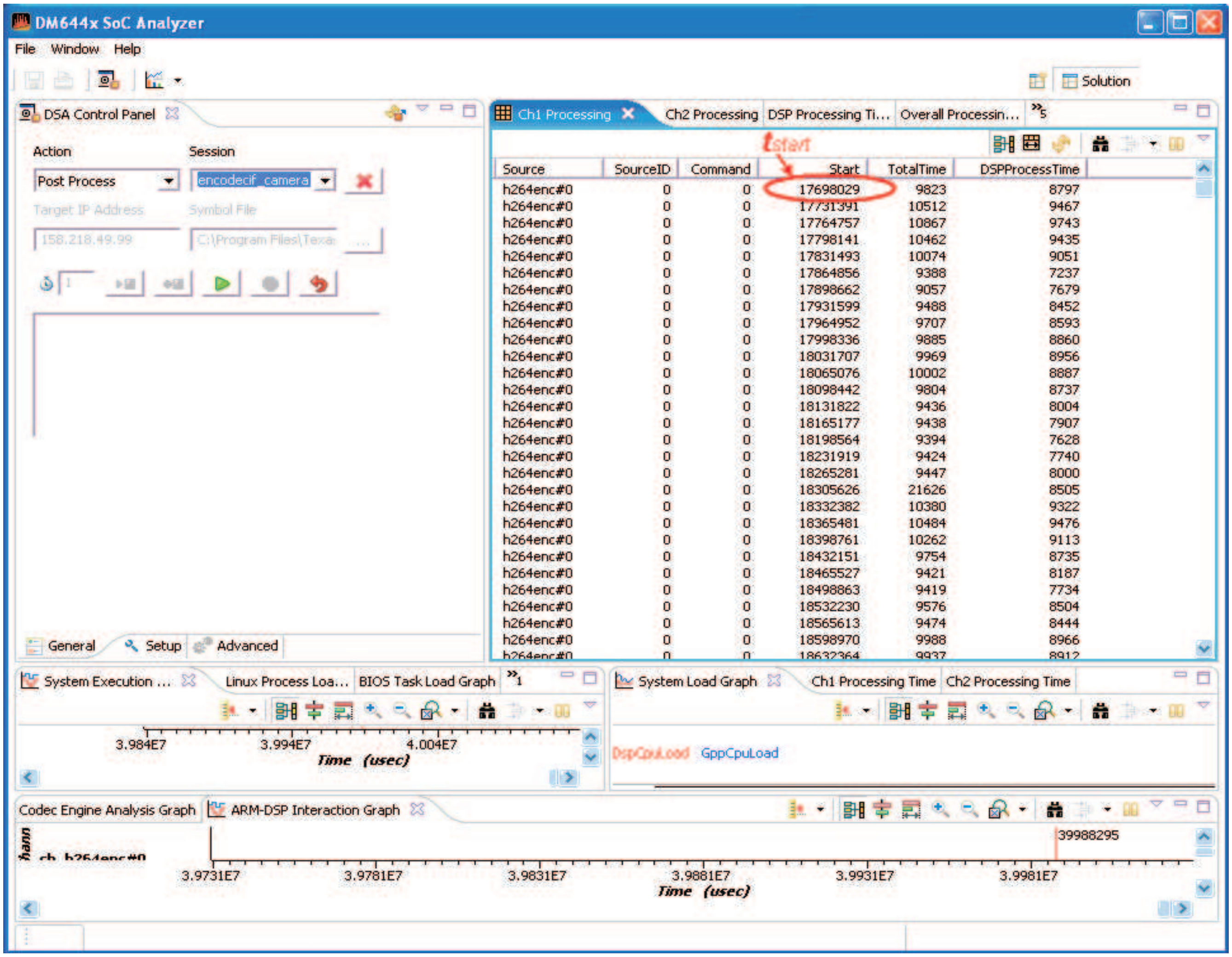
4.3 Terminology Important in Interpreting DM644x SoC Analyzer Results

The DM644x SoC Analyzer logs the timestamp (in microseconds) of all the events of interest and displays them in a GUI window. The following important information can be derived from these event timestamps.

4.3.1 Measurement Interval

Although the user can control roughly how long he/she needs to measure the demo, the actual measurement interval can be derived from the DM644x SoC Analyzer results.

Let t_{start} denote the time that the measurement starts for a demo. t_{start} can be found in the 1st row of one of the channel processing column, indicating the time when the first video (audio) frame starts to be processed. For the encode demo in which only one channel is available, t_{start} is the Start value in the 1st row of the Ch1 processing table. Figure 7 shows an example.



Action: Post Process

Session: encodecif_camera

Target IP Address: 158.218.49.99

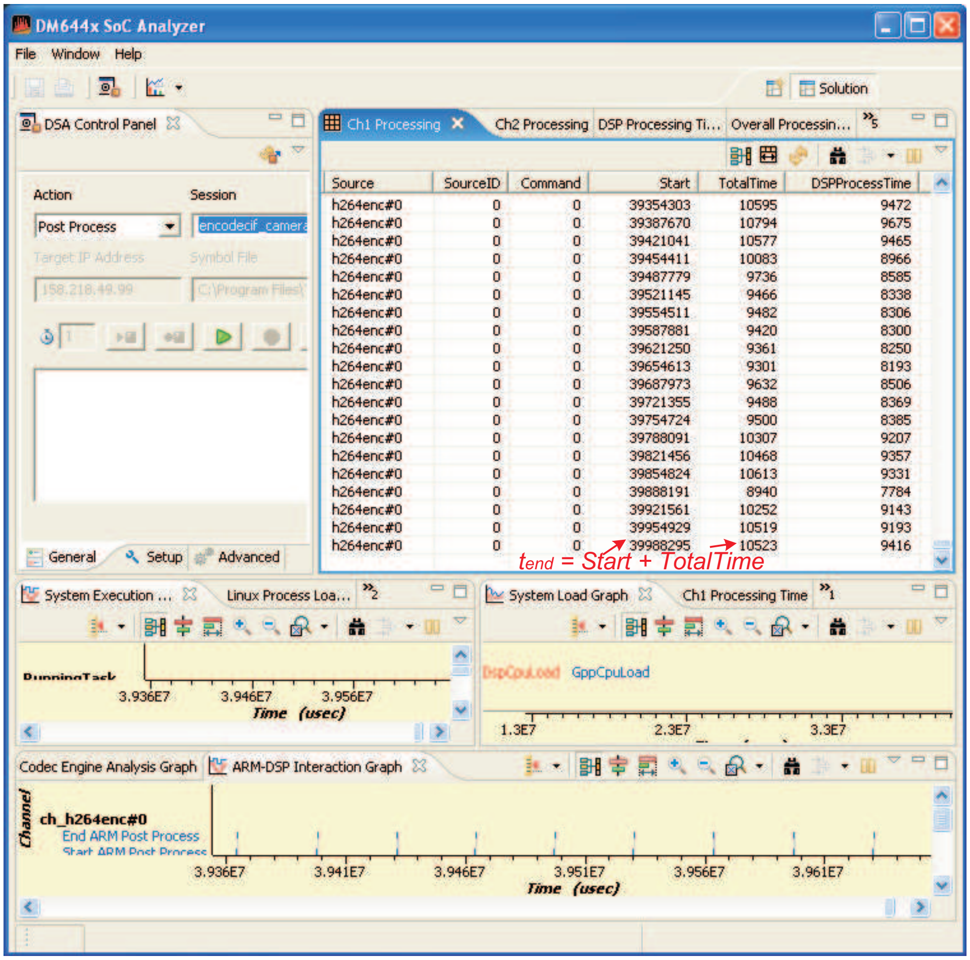
Symbol File: C:\Program Files\Texa

Source	SourceID	Command	Start	TotalTime	DSPProcessTime
h264enc#0	0	0	17698029	9823	8797
h264enc#0	0	0	17731391	10512	9467
h264enc#0	0	0	17764757	10867	9743
h264enc#0	0	0	17798141	10462	9435
h264enc#0	0	0	17831493	10074	9051
h264enc#0	0	0	17864856	9388	7237
h264enc#0	0	0	17898662	9057	7679
h264enc#0	0	0	17931599	9488	8452
h264enc#0	0	0	17964952	9707	8593
h264enc#0	0	0	17998336	9885	8860
h264enc#0	0	0	18031707	9969	8956
h264enc#0	0	0	18065076	10002	8887
h264enc#0	0	0	18098442	9804	8737
h264enc#0	0	0	18131822	9436	8004
h264enc#0	0	0	18165177	9438	7907
h264enc#0	0	0	18198564	9394	7628
h264enc#0	0	0	18231919	9424	7740
h264enc#0	0	0	18265281	9447	8000
h264enc#0	0	0	18305626	21626	8505
h264enc#0	0	0	18332382	10380	9322
h264enc#0	0	0	18365481	10484	9476
h264enc#0	0	0	18398761	10262	9113
h264enc#0	0	0	18432151	9754	8735
h264enc#0	0	0	18465527	9421	8187
h264enc#0	0	0	18498863	9419	7734
h264enc#0	0	0	18532230	9576	8504
h264enc#0	0	0	18565613	9474	8444
h264enc#0	0	0	18598970	9988	8966
h264enc#0	0	0	18632364	9937	8912

Figure 7. Find t_{start} for the Encode Demo

For the demo with two channels, such as the decode demo or the encode/decode demo, each channel has a start timestamp denoted as t_{start_ch1} and t_{start_ch2}, respectively. Similarly, t_{start_ch1} (t_{start_ch2}) is the “Start” value in the 1st row of the Ch1 (Ch2) processing table. The demo measurement start timestamp is the minimum of the two start timestamps. That is, $t_{start} = min\{ t_{start\_ch1}, t_{start\_ch2} \}$

Let t_{end} denote the time that the measurement ends for a demo. t_{end} can be found in the last row of the channel processing column, indicating the time when the last video (audio) frame was processed. It is calculated as the Start value plus the TotalTime value. Figure 8 shows how to calculate t_{end} for the encode demo.



Source	SourceID	Command	Start	TotalTime	DSPProcessTime
h264enc#0	0	0	39354303	10595	9472
h264enc#0	0	0	39387670	10794	9675
h264enc#0	0	0	39421041	10577	9465
h264enc#0	0	0	39454411	10083	8966
h264enc#0	0	0	39487779	9736	8585
h264enc#0	0	0	39521145	9466	8338
h264enc#0	0	0	39554511	9482	8306
h264enc#0	0	0	39587881	9420	8300
h264enc#0	0	0	39621250	9361	8250
h264enc#0	0	0	39654613	9301	8193
h264enc#0	0	0	39687973	9632	8506
h264enc#0	0	0	39721355	9488	8369
h264enc#0	0	0	39754724	9500	8385
h264enc#0	0	0	39788091	10307	9207
h264enc#0	0	0	39821456	10468	9357
h264enc#0	0	0	39854824	10613	9331
h264enc#0	0	0	39888191	8940	7784
h264enc#0	0	0	39921561	10252	9143
h264enc#0	0	0	39954929	10519	9193
h264enc#0	0	0	39988295	10523	9416

t_{end} = Start + TotalTime

Figure 8. Find t_{end} for the Encode Demo: t_{end} = Start+TotalTime

For the demo with two channels, each channel has an end timestamp denoted as t_{end_ch1} and t_{end_ch2}, respectively. Similarly, t_{end_ch1} (t_{end_ch2}) can be calculated as the “Start” value plus the “TotalTime” value in the last row of the Ch1 (Ch2) processing table. The demo measurement end timestamp is the maximum of the two end timestamps. That is, $t_{end} = max\{ t_{end\_ch1}, t_{end\_ch2} \}$

Let $T_{measure}$ denote the measurement interval for a demo. $T_{measure}$ can be calculated as

$$T_{measure} = t_{end} - t_{start} \qquad (1)$$

4.3.2 Processing Time for Each Frame

DM644x SoC Analyzer also displays the processing time for each video or audio frame in the corresponding channel processing table. The frame processing time consists of the following three parts:

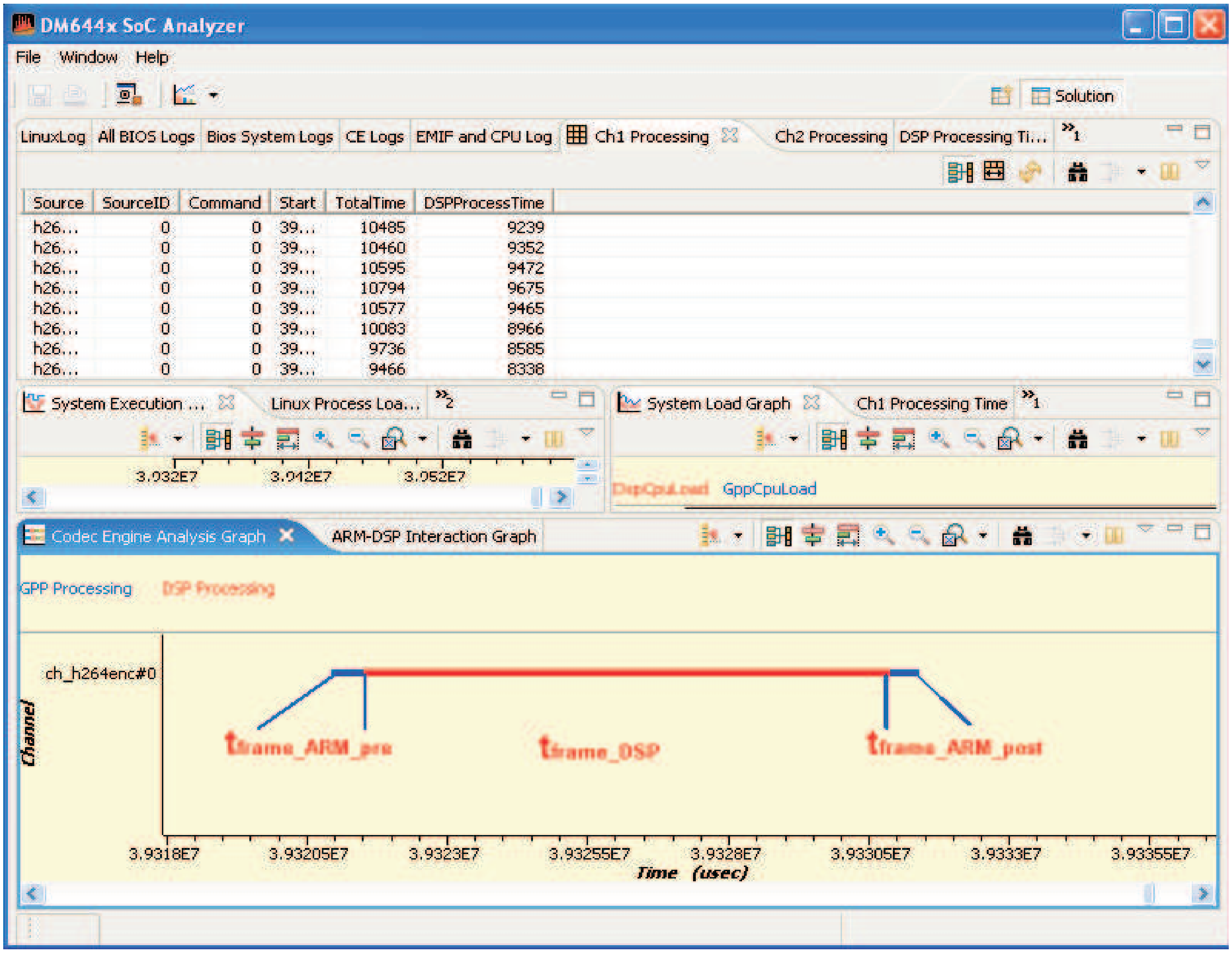
- $t_{frame\_ARM\_pre}$ which indicates the time that ARM spent on preprocessing the data before they are delivered to DSP for processing.
- t_{frame_DSP} which indicates the time that DSP spent on processing the data.
- $t_{frame\_ARM\_post}$ which indicates the time that ARM spent on postprocessing the data after receiving the processed data from DSP.

For each frame, the DM644x SoC Analyzer visualizes three parts in the Codec Engine Analysis Graph. It also shows two processing intervals in separate columns in the channel processing table: *TotalTime and DSP ProcessingTime*.

TotalTime can be expressed as *TotalTime* = $t_{frame\_ARM\_pre} + t_{frame\_DSP} + t_{frame\_ARM\_post}$.

And *DSP ProcessTime* = t_{frame_DSP}.

Figure 9 shows an example for the encode demo.



Source	SourceID	Command	Start	TotalTime	DSPProcessTime
h26...	0	0	39...	10485	9239
h26...	0	0	39...	10460	9352
h26...	0	0	39...	10595	9472
h26...	0	0	39...	10794	9675
h26...	0	0	39...	10577	9465
h26...	0	0	39...	10083	8966
h26...	0	0	39...	9736	8585
h26...	0	0	39...	9466	8338

Figure 9. Processing Time for Each Frame (Encode Demo Example); Note t_frame labels added for clarity

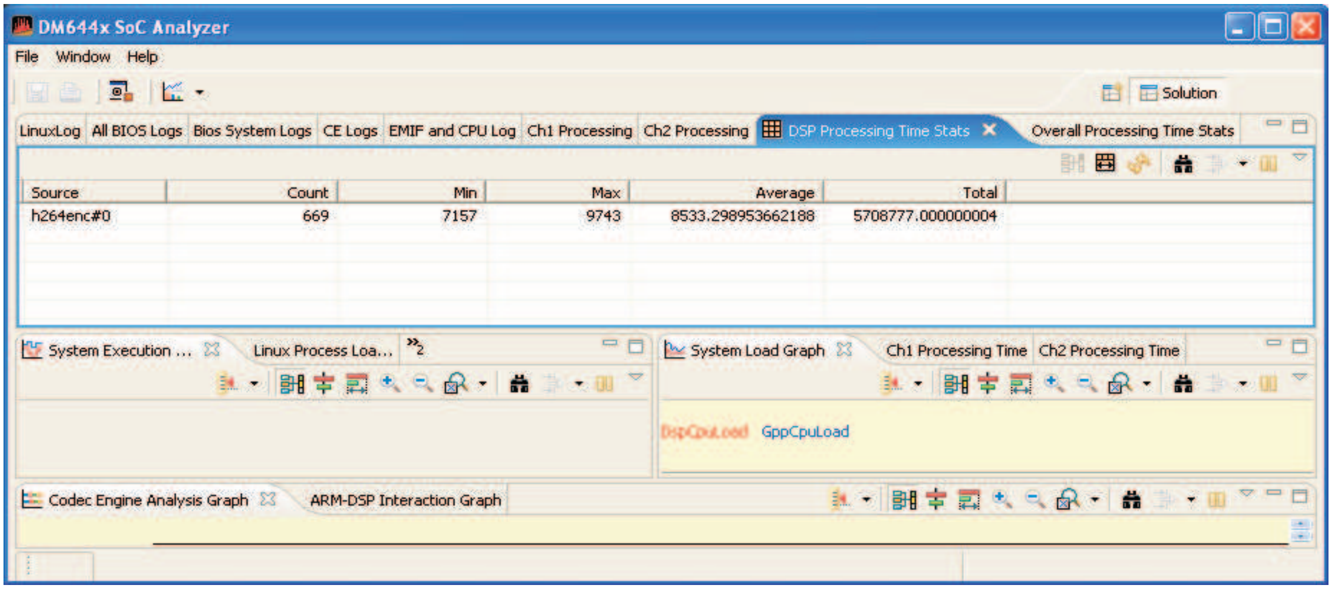
4.3.3 Processing Time Statistics

The DM644x SoC Analyzer also shows the processing time statistics for each channel. The statistics includes:

- *Count* which indicates the number of frames being processed.
- *Min* which indicates the minimum time interval to process a frame.
- *Max* which indicates the maximum time interval to process a frame.
- *Average* which indicates the average time interval to process a frame.
- *Total* which indicates the total time interval spent on processing all the frames.

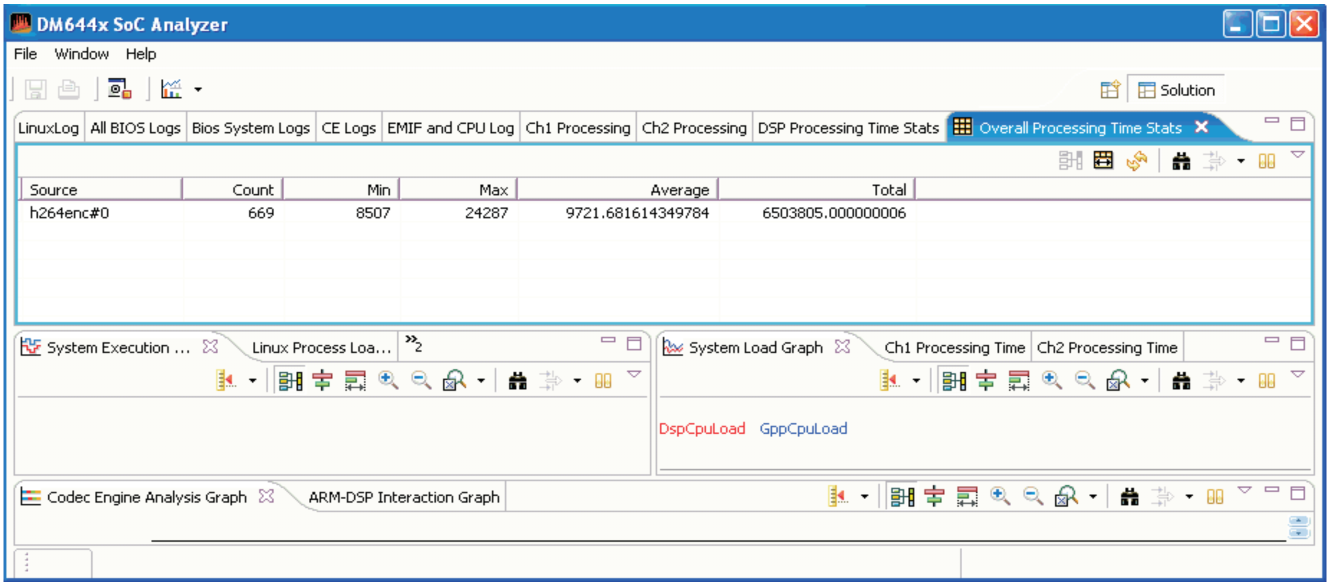
Both the DSP processing time statistics and the overall processing time statistics are shown in two separate tables. The former are calculated only based on DSP processing time (t_{frame_DSP}) while the latter are calculated including the ARM pre- and post-processing time ($TotalTime = t_{frame\_ARM\_pre} + t_{frame\_DSP} + t_{frame\_ARM\_post}$).

Figure 10 and Figure 11 show the DSP processing time statistics and overall processing time statistics for the encode demo.



Source	Count	Min	Max	Average	Total
h264enc#0	669	7157	9743	8533.298953662188	5708777.000000004

Figure 10. DSP Processing Time Statistics for Encode Demo



Source	Count	Min	Max	Average	Total
h264enc#0	669	8507	24287	9721.681614349784	6503805.000000006

Figure 11. Overall Processing Time Statistics for Encode Demo

4.4 Loading Calculation

As described previously, the DM644x SoC Analyzer records the timestamp (in microseconds) for all the events of interest and displays the measurement results in the time domain. Processor loadings as a percentage can be computed based on these timing results.

4.4.1 Min/Max/Average DSP Loading for a Video (Audio) Channel

Given the fact that the video channel is processed at 30 fps in all the three demos, the real time budget for each video frame is 33,333 µS. The min/max DSP loading for a video channel is calculated using the min/max DSP processing time for a video frame divided by the real time budget. That is,

$$L_{min\_video\_DSP} = min\{t_{video\_frame\_DSP}\} \,/\, 33{,}333\ \mu S \tag{2}$$

$$L_{max\_video\_DSP} = max\{t_{video\_frame\_DSP}\} \,/\, 33{,}333\ \mu S \tag{3}$$

where $L_{min\_video\_DSP}$ ($L_{max\_video\_DSP}$) represents the min (max) DSP loading for the video channel; and

$min\{t_{video\_frame\_DSP}\}$ ($max\{t_{video\_frame\_DSP}\}$) is the min (max) time for DSP to process a video frame which can be found in the DSP processing time statistics table as shown in Figure 10.

Let $L_{avg\_video\_DSP}$ denote the average DSP loading for a video channel. It can be calculated as the average DSP processing time for a video frame divided by the real-time budget. That is,

$$L_{avg\_video\_DSP} = t_{video\_frame\_avg\_DSP} \,/\, 33{,}333\ \mu S \tag{4}$$

where $t_{video\_frame\_avg\_DSP}$ represents the average time for DSP processing a video frame which can also be found in the DSP processing time statistics table.

The decode demo has an AAC audio channel. For this demo implementation, the frame rate is 48 fps and the real time budget is 20,833 µS. Similarly, $L_{min\_aac\_DSP}$, $L_{max\_aac\_DSP}$ and $L_{avg\_aac\_DSP}$ can be calculated as follows:

$$L_{min\_aac\_DSP} = min\{t_{aac\_frame\_DSP}\} / 20{,}833\ \mu S \tag{5}$$

$$L_{max\_aac\_DSP} = max\{t_{aac\_frame\_DSP}\} / 20{,}833\ \mu S \tag{6}$$

$$L_{avg\_aac\_DSP} = t_{aac\_frame\_avg\_DSP} / 20{,}833\ \mu S \tag{7}$$

4.4.2 Average DSP Loading

The average DSP loading for a demo is the summation of the average DSP loading for all the channels. That is,

$$L_{avg\_DSP} = \sum_{ch = i} L_{avg\_i\_DSP} \tag{8}$$

where L_{avg_DSP} is the average DSP loading for the demo, and $L_{avg\_i\_DSP}$ represents the average DSP loading for channel *i*.

For the encode demo in which only one video channel is available, $L_{avg\_video\_DSP} = L_{avg\_DSP}$.

4.4.3 Min/Max/Average Overall Loading for a Video (Audio) Channel

Compared to the DSP loading calculation which is only based on DSP processing time for each frame, the overall loading computation includes the ARM pre- and post-processing time in addition to the DSP processing time. In other words, the min/max/average DSP loading for a particular channel measures the min/max/average loading imposed only to DSP by the channel while the min/max/average overall loading measures the min/max/average loading imposed both to ARM and DSP by the channel.

For a video channel, let $L_{min\_video\_overall}$, $L_{max\_video\_overall}$ and $L_{avg\_video\_overall}$ denote the min, max and average overall loading, respectively. They can be calculated as follows:

$$L_{min\_video\_overall} = min\{t_{video\_frame\_overall}\} / 33{,}333\ \mu S \tag{9}$$

$$L_{max\_video\_overall} = max\{t_{video\_frame\_overall}\} / 33{,}333\ \mu S \tag{10}$$

$$L_{avg\_video\_overall} = t_{video\_frame\_avg\_overall} / 33{,}333\ \mu S \tag{11}$$

where $min\{t_{video\_frame\_overall}\}$, $max\{t_{video\_frame\_overall}\}$ and $t_{video\_frame\_avg\_overall}$ is the min, max and average time to process a video frame. They can be found in the overall processing time statistics table as shown in Figure 11.

Similarly, $L_{min\_aac\_overall}$, $L_{max\_aac\_overall}$ and $L_{avg\_aac\_overall}$ can be calculated as follows:

$$L_{min\_aac\_overall} = min\{t_{aac\_frame\_overall}\} / 20{,}833\ \mu S \tag{12}$$

$$L_{max\_aac\_overall} = max\{t_{aac\_frame\_overall}\} / 20{,}833\ \mu S \tag{13}$$

$$L_{avg\_aac\_overall} = t_{aac\_frame\_avg\_overall} / 20{,}833\ \mu S \tag{14}$$

4.4.4 Average ARM Loading for a Video (Audio) Channel

The average ARM loading for a channel represents the average loading imposed to ARM by the channel for each frame processing. In these demos, the pre-/post-processing work done by ARM mainly involves delivering/receiving data to/from DSP and transmitting the output data to appropriate devices for display. The real processing work is done by DSP only. Therefore, the average ARM loading can be considered as the overhead.

For a video channel, let $L_{avg\_video\_ARM}$ denote the average ARM loading. It is computed as follows:

$$L_{avg\_video\_ARM} = L_{avg\_video\_overall} - L_{avg\_video\_DSP} \tag{15}$$

Similarly, for an AAC channel, $L_{avg\_aac\_ARM}$ is computed as follows:

$$L_{avg\_aac\_ARM} = L_{avg\_aac\_overall} - L_{avg\_aac\_DSP} \tag{16}$$

4.4.5 Average ARM Loading

The average ARM loading for a demo is defined as the summation of the average ARM loading for all the channels. That is,

$$L_{avg\_ARM} = \sum_{ch=i} L_{avg\_i\_ARM} \tag{17}$$

where L_{avg_ARM} is the average ARM loading for the demo, and $L_{avg\_i\_DSP}$ represents the average ARM loading for channel *i*.

For the encode demo in which only one video channel is available, $L_{avg\_video\_ARM} = L_{avg\_ARM}$.

One thing to note is that L_{avg_ARM} represents the loading imposed to the ARM only by the demo processing work. Things that are not part of L_{avg_ARM} include the loadings imposed by other Linux processes, i.e., the HTTP server process, the shell process etc. In other words, the actual ARM loading will be higher than L_{avg_ARM}.

4.5 Loading Results

4.5.1 Decoder Demo Loading Results

Table 1 demonstrates the decoder loading measurement results.

Table 1. Loading Measurement Results for Decoder Demo

$L_{min\_H264\_DSP}$	$L_{max\_H264\_DSP}$	$L_{avg\_H264\_DSP}$
33.02% Equation 2	67.59% Equation 3	49.03% Equation 4
$L_{min\_aac\_DSP}$	$L_{max\_aac\_DSP}$	$L_{avg\_aac\_DSP}$
3.12% Equation 5	5.23% Equation 6	3.51% Equation 7
L_{avg_DSP}		
52.50% Equation 8		
$L_{avg\_H264\_ARM}$	$L_{avg\_aac\_ARM}$	
3.24% Equation 15	7.09% Equation 16	

Table 1. Loading Measurement Results for Decoder Demo (continued)

L_{avg_ARM}		
10.33%		
Equation 17		
$L_{min\_H264\_overall}$	$L_{max\_H264\_overall}$	$L_{avg\_H264\_overall}$
36.41%	79.72%	52.27%
Equation 9	Equation 10	Equation 11
$L_{min\_aac\_overall}$	$L_{max\_aac\_overall}$	$L_{avg\_aac\_overall}$
7.79%	74.53%	10.67%
Equation 12	Equation 13	Equation 14

4.5.2 Encoder Demo Loading Results

Table 2 demonstrates the encoder loading measurement results.

Table 2. Loading Measurement Results for Encoder Demo

$L_{min\_H26\_DSP}$	$L_{max\_H264\_DSP}$	$L_{avg\_H264\_DSP}$
74.63%	103.10%	87.54%
Equation 2	Equation 3	Equation 4
$L_{avg\_DSP} = L_{avg\_H264\_DSP}$		
87.54%		
Equation 8		
$L_{avg\_H264\_ARM} = L_{avg\_ARM}$		
3.20%		
Equation 15		
$L_{min\_H264\_overall}$	$L_{max\_H264\_overall}$	$L_{avg\_H264\_overall}$
77.73%	106.36%	90.74%
Equation 9	Equation 10	Equation 11

4.5.3 Encode/decode Demo Loading Results

Table 3 demonstrates the encode/decode loading measurement results.

Table 3. Loading Measurement Results for Encode/Decode Demo

$L_{min\_H264enc\_DSP}$	$L_{max\_ H264enc\_DSP}$	$L_{avg\_H264enc\_DSP}$
26.51%	36.10%	31.88%
Equation 2	Equation 3	Equation 4
$L_{min\_H264dec\_DSP}$	$L_{max\_H264dec\_DSP}$	$L_{avg\_H264dec\_DSP}$
11.75%	17.24%	15.44%
Equation 2	Equation 3	Equation 4
L_{avg_DSP}		
47.28%		
Equation 8		
$L_{avg\_H264enc\_ARM}$	$L_{avg\_H264dec\_ARM}$	
3.20%	3.04%	
Equation 15	Equation 15	

Table 3. Loading Measurement Results for Encode/Decode Demo (continued)

L_{avg_ARM}		
6.24% Equation 17		
$L_{min\_H264enc\_overall}$	$L_{min\_H264enc\_overall}$	$L_{avg\_H264enc\_overall}$
29.69% Equation 9	39.29% Equation 10	35.08% Equation 11
$L_{min\_H264dec\_overall}$	$L_{max\_H264dec\_overall}$	$L_{avg\_H264dec\_overall}$
14.69% Equation 9	20.22% Equation 10	18.48% Equation 11

4.6 Reproducing the Loading Measurement Results

For the same video decoder implementation, the DSP loading mainly depends on the bit rate and the amount of motion in the input video stream. For the decode demo, users can easily reproduce the results shown in Section 4.2.1 as long as the same audio/video input file are used and the measurement is carried roughly from 15s to 35s.

Generally speaking, the DSP loading for video encoding depends on various factors such as the amount of motion in the captured frames, the background texture, brightness, noise and etc. Therefore, for the encode and encode/decode demos, the user may get different results since the human gesture (literally hand waving) captured by the input camera will not likely be exactly the same. For example, waving the hand faster and closer to the camera produces more motion and increases the DSP loading. To reproduce the results for the encode and encode/decode demos shown in Table 2 and Table 3, please follow the instructions in Section 4.2.2 and Section 4.2.3 .

4.7 Loading Results Analysis

4.7.1 Overhead Sources

Most results show that the video processing overhead on ARM takes about 3% of the time budget per frame. The demo implementation makes an ARM-DSP Inter-Processor Call (IPC) every 33 ms to process a video frame. Therefore, the rule of thumb is that every time an IPC is made from ARM to DSP, a 1 ms overhead is introduced.

Note that the DM644x SoC Analyzer itself adds overhead to the loading since it needs to collect the traces on both the ARM and DSP processors. These tasks are not trivial since they involve string parsing/formatting, timestamp reading and writing all the trace data to files. Currently, the overhead added by the DM644x SoC Analyzer itself is about 400 µs for each audio/video frame.

4.7.2 Explanation of Greater than 100% Loading

Sometimes the loading result is more than 100%, i.e., $L_{max\_h264\_overall}$ in Table 2. It does not represent the actual loading of the ARM or DSP since processor loading can never be larger than 100%. It means that the processing time for a frame is more than the real-time budget. An occasional and small amount of real-time deadline violation has little impact if the subsequent frames are processed within the real-time budget and if the data buffering scheme allow all the buffered frames to fit into the time budget as a whole. However, if several frames violate the real-time deadline in sequence, frame drops may be noticed by the user.

4.7.3 H.264 Encoder Loading vs. H.264 Decoder Loading

It can be observed that for the same frame rate and resolution, i.e. 30 fps and D1 in the measurement setup, the DSP loading of H.264 encoder is significantly higher than that of H.264 decoder. Figure 12 shows the block diagram of an H.264 encoder and an H.264 decoder to help illustrate the differences.

The H.264 encoder processes each frame macroblock by macroblock (16 × 16 pixels), similar to other video encoding standards. It has a forward path and a reconstruction path. The forward path encodes a frame into bits. The reconstruction path generates a reference frame from the encoded bits.

In the forward path, each macroblock can either be encoded in intra mode or inter mode when the current frame (F(n)) is presented for encoding. In either case, the difference between the current macroblock and the prediction macroblock (M) is computed. The difference is further passed through the DCT, quantization (Q), recorder and entropy encode module to become encoded bits. In inter mode, M is found in other encoded frames by the ME (motion estimation) module. In intra mode, M is formed from samples in the current frame.

In the reconstruction path, D (which is the difference between the current macroblock and M after passing module Q) is passed through the inverse quantization (IQ) and inverse DCT (IDCT) module. Then it is added with M and further filtered by a loop filter to produce the reference frame F’(n). The purpose of the reconstruction path is to ensure that the encoder and decoder will use the identical reference frame to create the prediction macroblock M. Otherwise the error between the encoder and decoder will accumulate.

It is apparent in Figure 12 that the reconstruction path in the encoder contains more modules than in the decoder. Another difference is the entropy encode module in encoder vs. the entropy decode module in decoder. In reality, these two modules require very similar DSP loadings. Therefore, we can conclude that H.264 encoder has higher loading compared to H.264 decoder since additionally it needs to perform ME, DCT, quantization and inter vs. intra mode decision.

The exact loading difference between an H.264 encoder and an H.264 decoder depends on the implementation. The tradeoff is mainly related to the ME module. An ME requiring higher/lower loading may find better/worse prediction macroblock M to achieve better/worse video quality.

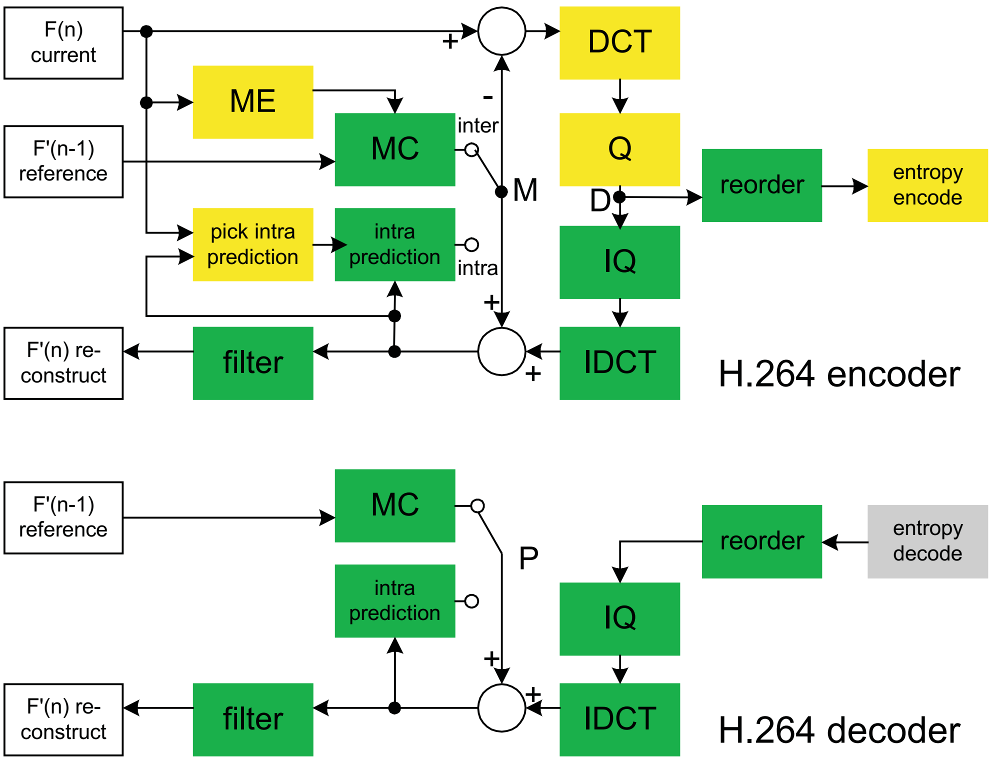


Figure 12. H.264 Encoder and Decoder Block Diagram

4.7.4 H.264 Encoder/Decoder Loading at Different Resolution

The size ratio between a CIF frame and a D1 frame is approximately ¼. However, it is noticeable that the H.264 encoder loading in the encode/decode demo (at CIF resolution) is higher than ¼ of the H.264 encoder loading in the encode demo (at D1 resolution). Similarly, the H.264 decoder loading in the encode/decode demo is higher than ¼ of the H.264 decoder loading in the decode demo.

The reasons can be explained as follows:

- In the encode/decode demo where the H.264 encoder and decoder execute simultaneously, the cache miss penalty can downgrade their performance, compared to the case when each of them executes alone. For details on cache performance analysis, please refer to *TMS320 C6000 DSP Cache User's Guide* (SPRU656) [4].
- Some modules such as ME, entropy encode/decode in the H.264 encoder/decoder do not scale linearly on frame size.

5 Demo Memory Usage

This section describes the memory usage by the three demos. The types of memory used are described below:

- *ARM static memory* includes the code (.text) section and the data section. The static memory usage information for an ARM demo binary can be found by running the Linux host utility arm_vt5_le_size included in the DVEVM package. For example, assuming the DVEVM package has been installed on the Linux host and the demo binaries, to find the ARM static memory usage for the encode demo, switch to the directory containing the demo binaries and type command "arm_vt5_le-size encode".
- *ARM program stack* is automatically allocated by Linux. This information can be found in the status file for the corresponding process. For example, if the demo process id is 1000, its stack size can be found in file "/proc/1000/status".
- *DSP static memory* includes the code section and the data section. They can be found in the generated memory map file of the corresponding DSP program.
- *Demo dynamic memory* is limited in this application note to the memory allocated by the audio/video processing algorithms and related drivers. Dynamic memory allocated by other processes, i.e. interface process, is not taken into account. The reported dynamic memory usage information is collected from the demo source code and includes the following 3 types:
  - Dynamic memory allocated and used by the ARM application program. The memory allocated by the interface program is not counted.
  - Dynamic memory allocated by the ARM application program but shared both by the ARM and DSP. This memory includes the input/output buffers for the corresponding DSP codecs.
  - Dynamic memory allocated by various related drivers in the Linux kernel space. The related drivers are the video capture driver, video display driver and OSD driver.

One thing to note is the DSP codec may allocate additional memory for its own purpose, i.e., allocating memory for a task stack etc. Such memory allocation is codec specific and will not be reported in this document. For codec specific memory footprint information on the DSP, please refer to the H.264 codec data sheets.

5.1 Decoder Demo Memory Usage

Table 4. ARM Static Memory Usage for Decoder Demo

Code	Data	Total
121,707 bytes	11,320 bytes	133,027 bytes

Table 5. ARM Stack Memory Usage for Decoder Demo

ARM Program Stack Size
84K bytes

Table 6. DSP Static Memory Usage for Decoder Demo

Code	Data	Total
555,264 bytes	553,480 bytes	1,108,744 bytes

Table 7. Dynamic Memory Usage for Decoder Demo

Name	Bytes	Type	Comments
readBuffer for H.264 decoder	3,145,728	ARM-DSP share	This buffer is 3Mbytes and stores the encoded video data.
frameBuf for H.264 decoder	2,488,320	ARM-DSP share	This buffer can hold up to 3 decoded video frame in PAL D1 size (YUV 4:2:2 data). The data are copied to displayBuf for display.
DisplayBuf for video	2,488,320	Driver	This buffer is allocated in kernel by the video display driver. It can hold up to 3 D1 frames. The decoder demo uses all of them.
OSDframeBuf	1,658,880	Driver	This buffer is allocated by the OSD driver in kernel. It can hold up to 2 D1 frames.
OSDframeBuf	184,320	Driver	This buffer is allocated by the OSD driver in kernel used for its transparency functionality.
OSDframeBuf	61,440	ARM-DSP share	This buffer is 60K bytes and stores the encoded audio data.
rawBuffer	10,240	ARM-DSP share	This buffer is 10K bytes to store up to 5 blocks of decoded audio samples.
Total	9,965,568		The total allocated memory is about 9.5M bytes for this demo configuration.

5.2 *Encoder Demo Memory Usage*

Table 8. ARM Static Memory Usage for Encoder Demo

Code	Data	Total
117,109 bytes	11,088 bytes	128,197 bytes

Table 9. ARM Stack Memory Usage for Encoder Demo

ARM Program Stack Size
84K bytes

Table 10. DSP Static Memory Usage for Encoder Demo

Code	Data	Total
341,248 bytes	479,769 bytes	821,017 bytes

Table 11. Dynamic Memory Usage for Encoder Demo

Name	Bytes	Type	Comments
encodedBuf for H.264 encoder	691,200	ARM-DSP share	This buffer is used to store the encoded data. It can hold up to 1 uncompressed D1 frame.
captureBuf for video	2,488,320	Driver	This buffer is allocated in kernel by the video capture driver. It can hold up to 3 video frames in PAL D1 size (YUV 4:2:2 data). The demo only uses 2 of them.
capCtrlBuf	16	ARM only	This buffer is used by ARM only for controlling the captureBuf.
displayBuf for video	2,488,320	Driver	This buffer is allocated in kernel by the video display driver. It can hold up to 3 D1 video frames. The demo uses all of them.
OSDframeBuf	1,658,880	Driver	This buffer is allocated by the OSD driver in kernel. It can hold up to 2 D1 frames.
OSDtransBuf	184,320	Driver	This buffer is allocated by the OSD driver in kernel used for its transparency functionality.
Total	7,511,056		The total allocated memory is about 7.2M bytes for this demo configuration.

5.3 Encode/Decode Demo Memory Usage

Table 12. ARM Static Memory Usage for Encode/Decode Demo

Code	Data	Total
114,000 bytes	11,040 bytes	125,040 bytes

Table 13. ARM Stack Memory Usage for Encode/Decode Demo

ARM Program Stack Size
84K bytes

Table 14. DSP Static Memory Usage for Encode/Decode Demo

Code	Data	Total
381,088 bytes	468,545 bytes	849,633 bytes

Table 15. Dynamic Memory Usage for Encode/Decode Demo

Name	Bytes	Type	Comments
encodedBuf for H.264 encoder	1,658,880	ARM-DSP share	This buffer is used to store the encoded data. It can hold up to 1 uncompressed D1 frame.
captureBuf for video	2,488,320	Driver	This buffer is allocated in kernel by the video capture driver. It can hold up to 3 video frames in PAL D1 size (YUV 4:2:2 data). The demo uses all of f them.
capCtrlBuf	24	ARM only	This buffer is used by ARM only for controlling the captureBuf.
frameBuf for H.264 decoder	506,880	ARM-DSP share	This buffer can hold up to 3 decoded video frame in CIF size (YUV 4:2:2 data). The data are copied to displayBuf for display.
displayBuf for video	2,488,320	Driver	This buffer is allocated in kernel by the video display driver. It can hold up to 3 D1 video frames. The demo uses all of them.
OSDframeBuf	1,658,880	Driver	This buffer is allocated by the OSD driver in kernel. It can hold up to 2 D1 frames.
OSDtransBuf	184,320	Driver	This buffer is allocated by the OSD driver in kernel used for its transparency functionality.
Total	8,985,624		The total allocated memory is about 8.6 M bytes for this demo configuration.

6 Power Measurements on the Demos

6.1 Terminology

The overall average power consumption for a complementary metal-oxide semiconductor (CMOS) circuit is the sum of static and active power consumption.

$P_{total} = P_{static} + P_{active}$

The static power consumption, as the name suggests, is the component that is independent of the processor activity. This is the power dissipated when the transistors are not switching; it is essentially the “leakage” current, due to the gate current and source to drain current elements of the transistors and additionally due to DC currents in some analog components. The static power consumption is primarily the function of both the supply voltage and the operating temperature within a given process technology. The static power consumption increases as the temperature and/or supply voltage is increased.

The static power consumption is defined as follows:

$P_{static} \sim V \times I_{leakage}$

$\mathrm{P}_{\mathrm{static}} = \mathrm{V} \times \mathrm{I}_{\mathrm{leakage}}\ (\mathrm{V,temp})$

where V is the supply voltage and is the $I_{leakage}$ current at that voltage and a given temperature.

The active or dynamic power consumption is attributed to processor activity. It is the power dissipated in CMOS circuits due to switching of the device core and I/O which result from charging and discharging the node capacitances (mostly gate and wire capacitance). The active power component is largely independent of the operating temperature; it is primarily a function of the supply voltage and switching frequency, and is calculated as follows:

$Pactive \sim Core(A \times CV^2 f) + I/O(N \times C_{io}\ V_{io}^2\ f_{io})$

where C is node capacitance, V is the core supply voltage, f is the switching frequency and A is the fraction of gates actively switching. Similarly, C_{io} is pin/pad capacitance, V_{io} is the I/O supply voltage, f_{io} is the toggle rate for the I/O pins and N is the number of bits/pins switching.

6.2 DM644x Power Supply Pins and Description

The DM644x has different operating voltages for core and I/O. The voltage supplies can be broadly divided into three categories, namely:

- Core supply
- 1.8 V I/O supply
- 3.3 V I/O supply

Within these categories, there are dedicated power pins on the DM644x for supplying voltage to a particular power domain, module, peripheral I/O etc, please refer to the DM6446 data manual *TMS320DM6446 Digital Medial System-On-Chip* (SPRS283) [5].

Table 16 provides the list of the power pins on the DM644x, along with the description and pin numbers.

Table 16. Power Supply Terminal Function

Type	Name	Description	Pin No.
Core Supply	C_{VDD}	Core supply voltage. Powers the “Always On” domain. The majority of DM644x modules (including ARM subsystem , peripherals etc) lie in the “Always On” power domain, and are powered by these pins	F15, K12, M12, L11, M10, L10, K10, L9, L8, M8
	C_{VDDDSP}	DSP subsystem (SS) core supply voltage. Powers the “DSP” power domain. The DSP subsystem including the c64x+ and VICP are powered by these pins	J13, H12, H11, J11, K11, J10, H10, J9, K9, K8, H8
	V_{DDA_1P1V}	DAC analog core supply voltage	P16

Table 16. Power Supply Terminal Function (continued)

Type	Name	Description	Pin No.
1.8 V I/O Supply	DV_{DD18}	1.8 V I/O supply voltage. These pins are for all the 1.8 V peripheral I/O. Most of the peripherals on DM644x have 1.8 V i/o (except for EMAC, MDIO, MMC/and GPIOV33)	N5, G15, F14, J15, H14, K14, M14, L13, G9, F8, E7, G7, J7, L7, F6, H6, K6, M6
	DV_{DDR2}	1.8 V DDR2 I/O supply voltage. These pins are for exclusively for the DDR2 I/O	T5, P6, N7, P8, N9, R9, P10, N11, R11, P12, N13, R13, P14, R15
	DDR_VDDLL	1.8 V supply voltage for DDR2 Digital Locked Loop (DLL)	T10
	V_{DDA_1P8V}	DAC 1.8 V Analog I/O supply voltage	R18
	$PLLV_{DD18}$	1.8 V Power supply voltage for the PLL1 (System PLL) and PLL2 (clocks for DDR2 EMIF and optional VPBE clocks)	M2
	MXV_{DD}	1.8 V supply voltage for MX oscillator	L5
	$M24V_{DD}$	1.8 V supply voltage for M24 (USB) oscillator	F16
	USB_V_{DD1P8}	1.8 V I/O supply voltage for the USB PHY	H17
3.3 V I/O Supply	DV_{DD33}	3.3 V I/O supply voltage. These pins are for all the 3.3 V peripheral i/o. i.e. EMAC, MDIO, MMC/ and GPIOV33.	F10, F11, F12, F13
	USB_V_{DDA3P3}	3.3 V analog voltage supply for USB PHY	J19

6.3 Power Measurements on the DM644x DVEVM

The DM644x DVEVM board is designed such that it allows you to conveniently measure the power consumption for both the DM644x SoC and the board itself. For additional details on the board, please refer to the Davinci EVM technical reference guide [6], on Spectrum Digital support website. For almost all important power pins on the DM664x, there are test points with current sensing resistors in between.

These current sensing resistors are designed for low ohmic value (low resistance) and tight tolerance (±1 percent) so as to minimize power consumption. As a result, these resistors are used to monitor the current in a circuit and translate the amount of current in that circuit into a voltage that can be easily measured and monitored (e.g., using a multimeter).

By measuring the voltage drop across these resistors at the given test points on the board, you can deduce the amount of current fed to the various power pins on the DM644x and calculate the overall power consumption.

Table 17 provides the list of test points that were used to measure the power consumption for the demos on the DM6446 DVEVM. For ease of use and reference in this document, a naming convention was added (first column) to refer to the power consumption at these test points.

Figure 13 shows the location of various test points on the DM644x DVEVM.

Table 17. List of Test Points to Measure DM644x Power Consumption on the DVEVM

Name	Test Points	Resistor	Description
			Core Supply Measurements
P_{TOT_CORE}	TP35-TP44	R59 (0.025)	DM644x total core supply voltage. The voltage drop at this test point gives the total core power consumption, this includes the consumption by the “Always On” Domain, the DSP Subsystem domain and DAC analog core power consumption (CV_{DD} + CV_{DDDSP} + V_{DDA_1P1V}) .
			1.8 V I/O Supply Measurements
P_{PLL}	TP19-TP8	R35 (0.22)	The voltage drop at this point gives the power consumed by the by PLL1 and PLL2 ($PLLV_{DD18}$).
P_{IO_18}	TP21-TP11	R37 (0.025)	The voltage drop at this test point gives the power consumption for sum of the 1.8V I/O, the DDR2 I/O, System oscillator and USB Oscillator (DV_{DD18} + DV_{DDR2}. + MXV_{DD} + $M24V_{DD}$)
P_{DDR_DLL}	TP39-TP40	R53 (0.22)	The voltage drop at this test point gives the power consumed by the DDR2 DLL (DDR_VDDLL).

Table 17. List of Test Points to Measure DM644x Power Consumption on the DVEVM (continued)

Name	Test Points	Resistor	Description
P_{DAC_IO18}	TP53-TP54	R63 (0.22)	The voltage drop at this test point gives the power consumed by the DAC analog I/O (V_{DDA_1P8V}).
P_{USB_IO18}	TP46-TP47	R60 (0.22)	The voltage drop at this test point gives the power consumption for the 1.8 V I/O of the USB PHY (USB_V_{DD1P8}).
			3.3 V I/O Supply Measurements
P_{IO_33}	TP32-TP22	R36 (0.025	The voltage drop at this test point gives the power consumption for all the 3.3 V I/O (DV_{DD33})
P_{USB_IO33}	TP48-TP49	R61 (0.22)	The voltage drop at this test point gives the power consumed by the analog portion of the USB PHY (USB_V_{DDA3P3}) .

Note: There are dedicated test points for measuring CV_{DD} (TP41-TP42) and CV_{DDDSP} (TP30-TP31). However, there is an internal shorting-switch between the Always On and DSP power domains, controlled by a system module register, Chip Shorting Switch Control (CHP_SHRTSW) register. This switch needs to be closed (DSPPWRON = 1) before enabling the DSP power domain in ARM boot modes (in DSP self boot mode DSPPWRON = 1, by default). Once the switch is closed/shorted for DSP operations, the two supply voltage rails are internally shorted and the current is shared between the two rails. Thus for conditions where DSPWRON = 1, it is not beneficial to monitor the CV_{DD} and CV_{DDDSP} power consumption separately. The DAC analog core supply power consumption can also be separately (V_{DDA_1P1V}) measured at TP61-TP55, but is negligible as compared the consumption seen on the other pins. Therefore, the total core power consumption can be measured at TP35-TP44.

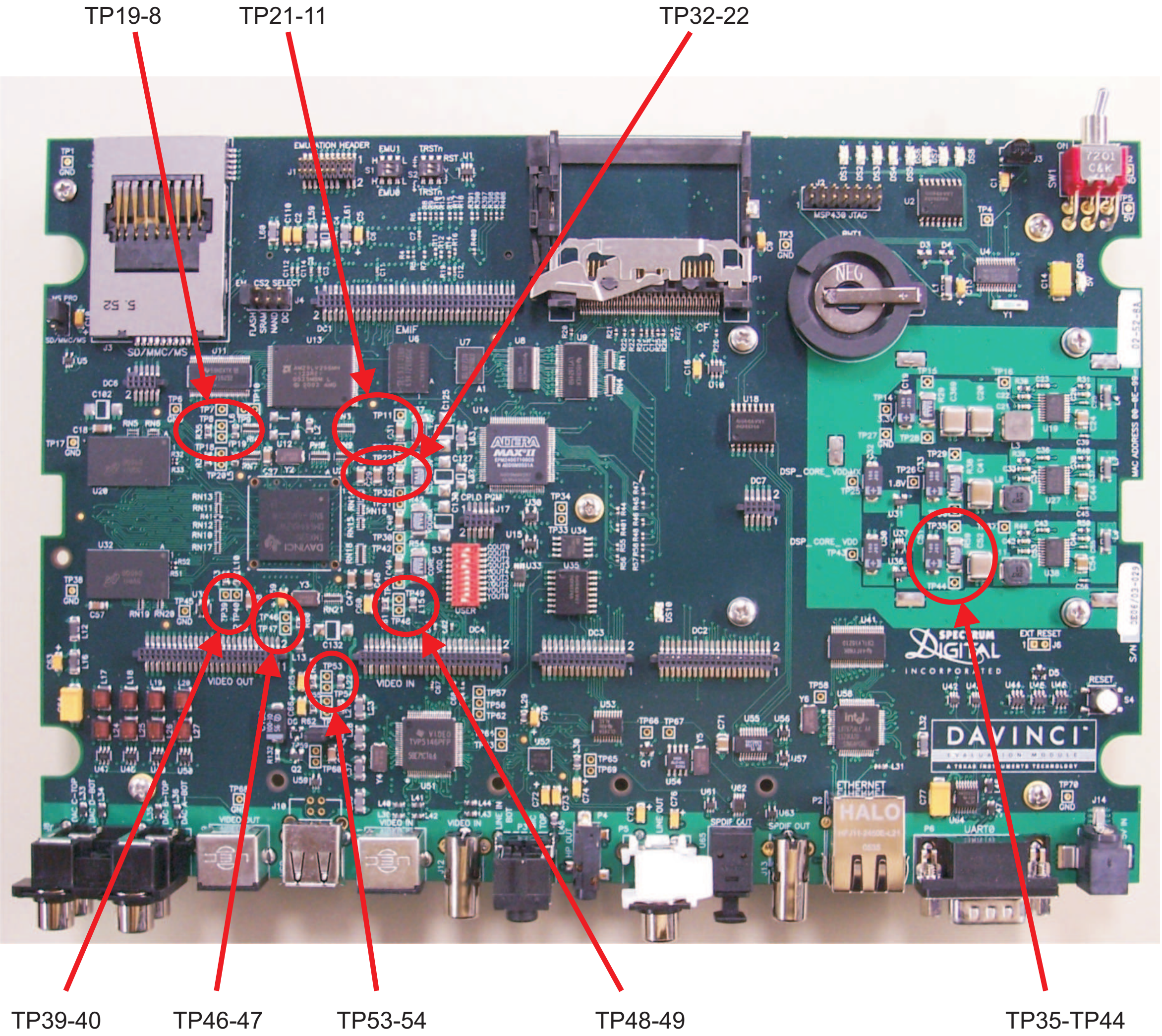


Figure 13. Test Points for Power Measurements on DM644x DVEVM

6.4 Results

Power measurements were performed on the demos. All numbers reported in the following tables were collected under these conditions:

- Platform: DM6446 DVEVM (Silicon Revision 1.3)
- Temperature: Room Temp
- Operation Frequency: C64x+: 594 MHz, ARM: 297 MHz, DDR2: 162 MHz
- Operating Core Voltage : CV_{DD}/CV_{DDDSP} = 1.2 V
- Multimeter: Fluke 79 III True RMS Multimeter, to measure voltage drop across the shunt resistors between various test points on the DVEVM.

All power measurements are reported in milliWatts and calculated as follows:

$$Px = (\frac{Vvd}{Rs}) \times V$$

Where *Vvd* is the voltage drop (in mV) measured across the test point, *Rs* is value of the shunt resistor at the test point and *Vx* is the supply voltage.

6.4.1 H.264 Decode Demo

Table 18 provides the power consumption measured at each test point.

Table 18. H.264 Decode Demo Power Measurements at Individual Test Points on the DVEVM

Description	Core Power 1.2 V	1.8 V IO Power					3.3 V IO Power	
	P_{TOT_CORE} (mW)	P_{PLL} (mW)	P_{IO_18} (mW)	P_{DDR_DLL} (mW)	P_{DAC_IO18} (mW)	P_{USB_IO18} (mW)	P_{IO_33} (mW)	P_{USB_IO33} (mW)
Decode Demo	686	76	58	10	8	37	13	12

6.4.2 H.264 Encode Demo

Table 19 provides the power consumption measured at each test point.

Table 19. H.264 Encode Demo Power Measurements at Individual Test Points on the DVEVM

Description	Core Power 1.2 V	1.8 V IO Power					3.3 V IO Power	
	P_{TOT_CORE} (mW)	P_{PLL} (mW)	P_{IO_18} (mW)	P_{DDR_DLL} (mW)	P_{DAC_IO18} (mW)	P_{USB_IO18} (mW)	P_{IO_33} (mW)	P_{USB_IO33} (mW)
Encode Demo	782	76	58	11	8	37	13	12

6.4.3 Encode/decode Demo

Table 20 provides the power consumption measured at each test point.

Table 20. Encode/Decode Demo Power Measurements at Individual Test Points on the DVEVM

Description	Core Power 1.2 V	1.8 V IO Power					3.3 V IO Power	
	P_{TOT_CORE} (mW)	P_{PLL} (mW)	P_{IO_18} (mW)	P_{DDR_DLL} (mW)	P_{DAC_IO18} (mW)	P_{USB_IO18} (mW)	P_{IO_33} (mW)	P_{USB_IO33} (mW)
Encode Demo	686	76	50	11	8	37	13	12

6.5 Analysis/Observations

Following are some analysis and observations for the power measured for the various demos:

1. The total power on these demos essentially vary due to differences in ARM and DSP CPU loading. This is seen on P_{TOT_CORE} and IO activity on the DDR2, and on P_{IO_18} based on image resolution CIF/D1, buffering schemes, reads versus writes (i.e., VFE writes to the DDR2 , VBE reads from DDR2) .
2. The average core power consumption is highly dependent on both the cycle efficiency (instructions per cycle) and the relative time the CPU(s) spends in active processing threads versus idle threads (also equivalent to the loading measurements). The total core power consumption observed for decode and encode/decode demo is nearly identical; the core power consumption is higher for the encode demo. Based on the results and analysis in Section 4.6 and Section 4.7, the power measurement data shows variations with the CPU loading on ARM and DSP; the loading values are nearly identical for the decode and encode/decode demo, but higher for the encode demo.

Note: A high CPU loading does not necessarily imply high power consumption, because the power consumption also depends on the instructions executed per cycle. For processor architectures with a high degree of parallelism, the instructions per cycle can have a bigger impact on power consumption. As an example a C64x+ CPU spending 90% of time in active processing threads but executing control code which is only one or two instructions per cycle would consume much less power than if the C64x+ CPU was spending 70% of time executing code like an FIR filter where six to seven instructions per cycle are executed. In general, when analyzing application power like the DVEVM demos, there are several functions/kernels being executed with different cycle efficiencies and memory usage. Therefore, the total power is dependent both on processor loading and the types of algorithms being executed.

3. The total core power is also affected by whether or not the algorithm makes use of the VICP. The VICP is used for H.264 encode but not for H.264 decode.
4. The power consumption on the PLL and DDR2 DLL is dependent only on the configuration of PLL1 and PLL2, therefore for a fixed frequency of operation for the DSP/ARM and DDR2, the consumption on these power rails do not vary with activity.
5. There is finite power consumption observed on the USB power pins P_{USB_IO18} and P_{USB_IO33} , even though the USB module is not used in these demos. This power consumption is close to negligible if the USB module and PHY were appropriately powered down via the power sleep controller (PSC) and the USB PHY Control register. The USB module is clocked but idled.
6. There is a finite I/O power consumption observed on 3.3V I/O pins. The demo does not actively make use of any 3.3 V I/Os (except for EMAC that is used for DSA data logging to host), and the finite power consumption seen on these pins is primarily due to the pull-up/pull down configuration of the 3.3 V I/O on the DVEVM.

6.6 General Considerations

1. The primary intention for this application note is to familiarize end user with the provisions available on the DVEVM to measure power consumption. The total power consumption measured could be slightly different based on factors like leakage on the device, measuring conditions, version of DVEVM demos, etc.
2. The reported power consumption data can have some degree of error (less then ±2-5%).
3. All measurements were performed on a nominal device.
4. For higher accuracy in measurements one could run the same tests under more stringent conditions, e.g.
   a. Accurately control the case temperature of device and remove inaccuracies in results due to self heating etc.
   b. Make use of current probes.
   c. Incorporate the setup to monitor the actual supply voltage at all test points.
   d. Average the readings over several runs.

7 Summary

The following section summarizes the key results for CPU loading, memory usage and power consumption for the H.264 audio/video demonstration software used in this application report.

7.1 H.264 Decode Demo

- Average ARM and DSP CPU loading

Table 21. Average ARM and DSP Loading for the H.264 Decode Demo Including AAC

	Average DSP Loading	Average ARM Loading
Decode Demo	52.50%	10.33%

- Static Memory Usage by ARM and DSP (see Section 5.1 for dynamic memory usage)

Table 22. Static Memory Usage for ARM and DSP for the H.264 Decode Demo Including AAC

	Code	Data	Total
ARM	121,707 bytes	11,320 bytes	133,027 bytes
DSP	555,264 bytes	553,480 bytes	1,108,744 bytes

- Total power consumption

Table 23. Total DM644x Power Consumption for the H.264 Decode Demo

	1.2 V Core Power (mW)	Total 1.8 V IO Power (mW)	Total 3.3 V IO Power (mW)	Total Power (mW)
Decode Demo	686	189	25	900

7.2 H.264 Encode Demo

- Average ARM and DSP CPU loading

Table 24. Average ARM and DSP Loading for the H.264 Encode Demo

	Average DSP Loading	Average ARM Loading
Decode Demo	87.54%	3.20%

- Static Memory Usage for ARM and DSP (see Section 5.2 for dynamic memory usage)

Table 25. Static Memory Usage for ARM and DSP for the H.264 Encode Demo

	Code	Data	Total
ARM	117,109 bytes	11,088 bytes	128,197 bytes
DSP	341,248 bytes	479,769 bytes	821,017 bytes

- Total power consumption

Table 26. Total DM644x Power Consumption for the H.264 Encode Demo

	1.2 V Core Power (mW)	Total 1.8 V IO Power (mW)	Total 3.3 V IO Power (mW)	Total Power (mW)
Encode Demo	782	190	25	997

7.3 H.264 Encode/Decode Demo

- Average ARM and DSP CPU loading

Table 27. Average ARM and DSP Loading for the H.264 Encode/Decode Demo

	Average DSP Loading	Average ARM Loading
Encode/Decode Demo	47.28%	6.24%

- Static Memory Usage for ARM and DSP (see Section 5.3 for dynamic memory usage)

Table 28. Static Memory Usage for ARM and DSP for the Encode/Decode Demo

	Code	Data	Total
ARM	114,000 bytes	11,040 bytes	125,040 bytes
DSP	381,088 bytes	468,545 bytes	849,633 bytes

- Total power consumption

Table 29. Total DM644x Power Consumption for the Encode/decode Demo

	1.2 V Core Power (mW)	Total 1.8 V IO Power (mW)	Total 3.3 V IO Power (mW)	Total Power (mW)
Encode Demo	686	182	25	893

8 Conclusion

This application note discusses ways to analyze performance and power using the hardware and software provided with the DM644x DVEVM/DVSDK. The demonstration software that is available with the DVEVM was used as an example to evaluate the performance and power consumption.

For performance, both the individual loading for each codec channel and the overall loading for the demo were calculated for ARM and DSP processors making use of DM644x SoC Analyzer. Analysis was also given to help understand the results. Additionally, both the static and dynamic memory usage was reported for each demo for both the ARM and DSP .

For power consumption, the various core and I/O power supply pins on DM644x were listed and corresponding test points on the DVEVM that allow measuring the power consumption were highlighted. The power consumption was measured for the above mentioned demos along with an analysis section to understand the measured data.

The methodology outlined in this application note can be applied by developers to evaluate and understand performance and power consumption requirements in a system context for their own DM644x based application software.

Evaluating performance and power consumption in a system context is critical at both the device selection stage and at the actual application development stage. This application note highlights how the DM644x hardware and software development resources allow both the assessment and monitoring of performance and power consumption with a methodical approach and tools to enhance the user experience. This enables system developers to improve performance and optimize utilization of critical resources in their own system reducing time to market and development costs.

9 References

1. *Codec Engine Application Developer User's Guide*, SPRUE67.
2. *DVEVM Getting Started Guide*, SPRUE66.
3. DSA online documentation available from the help menu in the DM644x SoC Analyzer tool.
4. *TMS320 C6000 DSP Cache User's Guide*, SPRU656.

5. *TMS320DM6446 Digital Media System-On-Chip* SPRS283.
6. DaVinci EVM Technical Reference Guide (Spectrum Digital)

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
Low Power Wireless	www.ti.com/lpw	Telephony	www.ti.com/telephony
		Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

Mailing Address: Texas Instruments
Post Office Box 655303 Dallas, Texas 75265