Motion JPEG Demo on TMS320DM6446

Zhengting He

ABSTRACT

This application report describes how to build a motion JPEG demo running on Texas Instruments DM6446 processor leveraging the JPEG codec combo and XDC tools provided with the DM6446 DVEVM/DVSDK package. The demo is derived from the motion JPEG demo running on the TI DM642 processor. Some tips on how to migrate legacy code from TI DSP only based platform such as DM642 to the new system-on-chip (SoC) platform such as DM6446 are provided.

Other than the JPEG codec combo, all other software source is provided along with this report to expedite the development cycle for users.

This application report contains project code that can be downloaded from this link: http://www-s.ti.com/sc/techlit/sprc343a.gz

Contents

List of Figures

List of Tables

Trademarks

eXpressDSP is a trademark of Texas Instruments.

Linux is a registered trademark of Linus Torvalds in the U.S. and other countries.

Windows is a trademark of Microsoft Corporation in the United States and/or other countries.

Internet Explorer is a registered trademark of Microsoft Corporation in the United States and/or other countries.

1 Demo Description

1.1 Introduction

Figure 1 shows a diagram of the motion JPEG demo running the DM6446 DVEVM board.

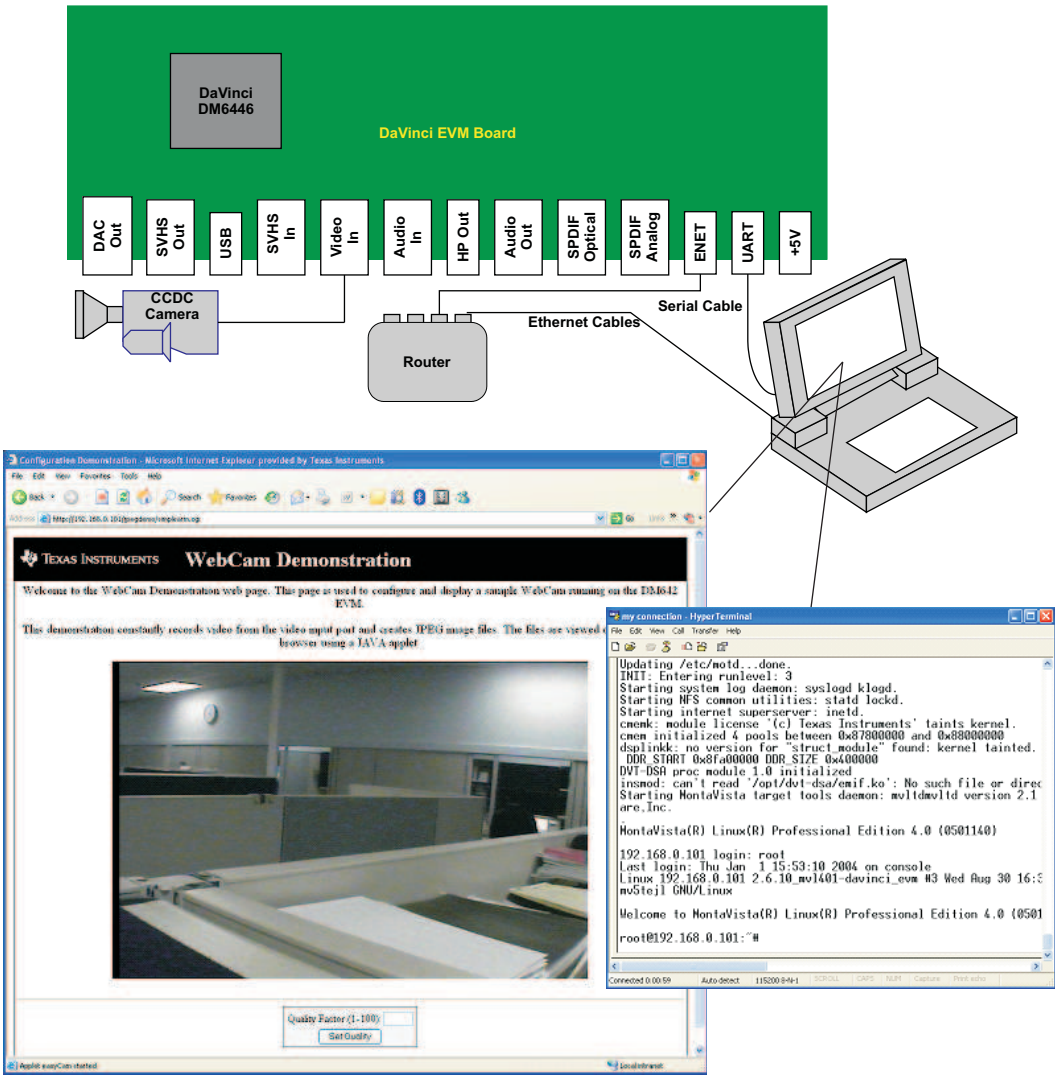

Figure 1. Motion JPEG Demo on DM6446

The CCDC camera is connected to the video in port on the DM6446 DVEVM board. The DM6446 processor captures the video signal from the video in port, compresses it by exercising the JPEG codec combo at 30 fps and writes the output JPEG image files to the Linux® file system on the DM6446 DMEVM board.

A serial connection needs to be established between the PC and the DM6446 DVEVM board. You are able to control the DM6446 processor, boot the board, type command, etc., using a Windows™ terminal program which communicates to the DM6446 processor via this link. The recommended Windows terminal program is hyperterminal which is pre-installed with Windows XP

An IP connection is also established between the Windows PC and the DM6446 DVEVM board using the network router. You are able to retrieve and display the encoded JPEG image files on the DVEVM board via the Internet Explorer® (IE) by accessing the specific common gateway interface (CGI) program running on the board. When the CGI program is accessed, it transfers the html contents and some JAVA byte code to the Windows operating system. The html contents give the brief description to this demo. The JAVA byte code decodes the JPEG image files at 30 fps and displays them in the IE window.

The quality of the JPEG images can also be dynamically configured by typing a number in the range [1, 100] in the post form of the IE window, as shown at the end of **Figure 1**. Number “1” makes the JPEG codec to encode the video at the lowest quality and highest compression ratio. Number “100” makes it to encode the video at highest quality and lowest compression ratio.

1.2 Dataflow

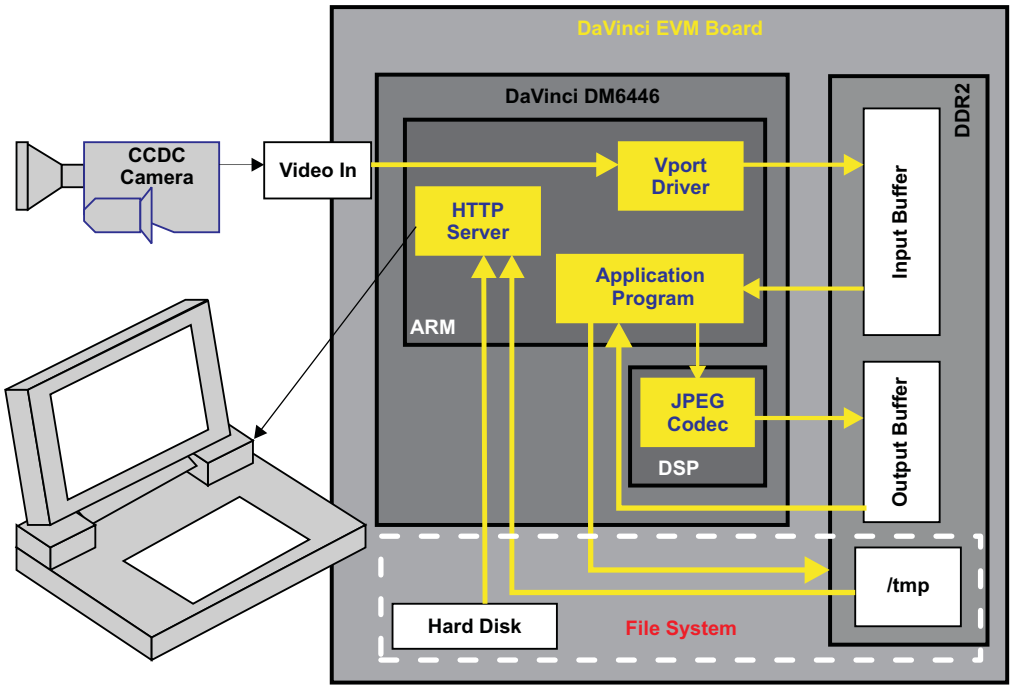

Figure 2. Dataflow of Motion JPEG Demo on DM6446

Figure 2 shows the demo dataflow. The input video signal is captured from the camera via the video input port and stored into the input buffer located in the DDR2 memory by the video port driver running on the ARM core of the DM6446 processor. The application program on ARM delivers the input data to the JPEG codec running on the DSP core. The JPEG codec encodes the input raw image and saves the compressed JPEG image to the output buffer located in the DDR2 memory. After that, the application program saves the output data to the /tmp directory which is a RAM disk mounted in the Linux file system. Storing the output images to the RAM disk allows fast access and improves system performance.

When you request to access the compressed JPEG images, the HTTP server running on the ARM core handles the request and transmits the data in the RAM disk (/tmp) via the IP link. When you request to change the image quality, the HTTP server writes the quality index to a specific file. The application program polls this file once every second to notify the JPEG codec to change the quality.

1.3 Software Architecture

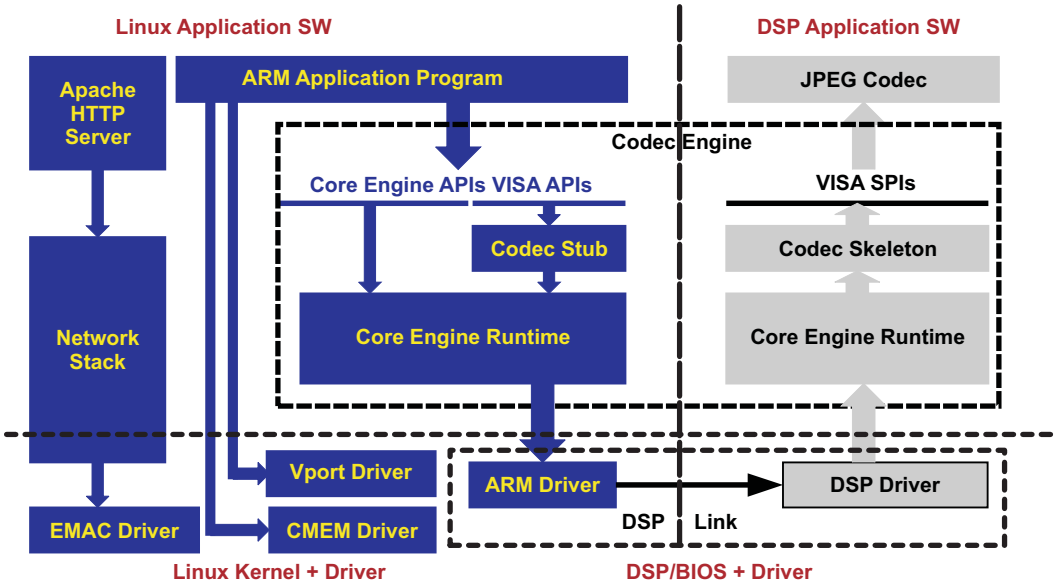

Figure 3. Software Architecture of Motion JPEG Demo on DM6446

Figure 3 shows the software architecture of the demo. The HTTP server running on ARM calls the Linux network stack to handle the HTTP requests. The network stack calls the EMAC driver to handle data sending/receiving. The HTTP server is in the Linux application (user) space and the EMAC driver is in the Linux kernel space. Several layers of the Linux network stack are in the application space while others are in kernel space.

The application program on ARM is in the Linux application space. It calls the video port driver to receive the input data. Once the video port driver is initialized, it allocates a buffer which can hold up to 3 D1 frames in the kernel space. Because of the virtual memory scheme, the Linux application cannot directly see the kernel memory addresses. Therefore, the application needs to make a mmap() call to access the input data in the buffer.

The video port driver allocates the kernel buffer in the physical address space that it can be shared by the DSP. When the ARM application notifies the DSP to encode a video frame, it does not need to copy the data to the DSP address space. Driver CMEM manages the shared memory between ARM and DSP. When the video port driver allocates the input buffer, it asks for CMEM to handle it. Similarly, when the ARM application allocates the output buffer shared by the DSP, it is also handled by CMEM driver.

From the ARM application point of view, asking DSP to encode a frame or change the video encoding quality is just a simple VISA API call. But there are a lot of tasks going on underneath the API.

- Once a VISA API is issued, pointers to any shared buffer between ARM and DSP need to be translated so that DSP program can recognize it. API arguments do not reside in the shared memory between ARM and DSP so they need to be marshaled before being copied to DSP memory. These tasks are handled by the Codec Engine on the ARM side. The image class stub handles these tasks that exercises the JPEG encoder as an image codec, specifically for this demo.
- The DSP link driver handle data copy between ARM and DSP. It has a Linux driver as the ARM portion and the DSP/BIOS driver as the DSP portion.
- Once the API arguments and buffer pointers reach DSP, appropriate codec API needs to be called or a DSP/BIOS task needs to be activated. To encode a new frame, cache also needs to be invalidated which forces the DSP core to read the new input data from external memory. These tasks are handled by the Codec Engine on the DSP side, specifically the image class codec skeleton.

2 Demo Package Contents

The motion JPEG demo package for DM6446 is compressed as an SPRC343.gz file. Untaring the package creates a folder called *mjpegdemo* in which there are 9 sub-folders: *app*, *cfgquality*, *cfgquality_pal*, *simpleweb*, *simpleweb_pal*, *config*, *dsp*, *javaapplet* and *javaapplet_pal*.

Table 1. Contents in mjpegdemo/app

File(s)	Description
app.c, app.h, main_native.c, smain.h	These are the source files of the ARM application program. The behavior is summarized as follows: 1. Initialize the video port driver either for NTSC or PAL format depending on the provided runtime argument. 2. Initialize codec engine and allocate all required memories. 3. Get the input image from the video port and notify the JPEG codec on DSP to encode it. 4. Write the output to the Linux file system as a JPEG file. The program is allowed to write up to 30 files in sequence (named from image1.jpg to image30.jpg) for NTSC format, or 25 files in sequence (image1.jpg to image25.jpg). Because the video is encoded at 30 fps for NTSC format or 25 fps for PAL format, the Linux file system buffers 1 second output data. 5. If 1 second of frames have been encoded since last time checking the quality index file, check the file again and re-configure the encoding quality. The name of the quality index file is Qfile.
makefile, package.bld, package.mak, package.xdc, remote.cfg	These files are used for building the application code using XDC tools which are installed with the DM6446 DVEVM/DVSDK package.
app.cgi	This is the ARM application binary code.

Table 2. Contents in mjpegdemo/simpleweb

File(s)	Description
main.c	This is the CGI source code to print out the HTML contents to the client for the NTSC format support once the client accesses it using Internet Explorer.
makefile	This file is used for building the CGI code.
simplearm.cgi	This is the CGI binary code.

Table 3. Contents in mjpegdemo/simpleweb_pal

File(s)	Description
main.c	This is the CGI source code to print out the HTML contents to the client for the PAL format support once the client accesses it using Internet Explorer.
makefile	This file is used for building the CGI code.
simplearm_pal.cgi	This is the CGI binary code.

Table 4. Contents in mjpegdemo/cfgquality

File(s)	Description
cfgquality.c	This is the CGI source code to change the encoding quality for the NTSC format support once the client accesses it using Internet Explorer. It validates the input from the client, writes the valid quality index to Qfile, and prints out the html contents to the client.
makefile	This file is used for building the CGI code.
cfgquality.cgi	This is the CGI binary code.

Table 5. Contents in mjpegdemo/cfgquality_pal

File(s)	Description
cfgquality.c	This is the CGI source code to change the encoding quality for the PAL format support once the client accesses it using Internet Explorer. It validates the input from the client, writes the valid quality index to Qfile, and prints out the html contents to the client.
makefile	This file is used for building the CGI code.
cfgquality_pal.cgi	This is the CGI binary code.

Table 6. Contents in mjpegdemo/javaapplet

File(s)	Description
easyCam.java	This is the JAVA applet source code to decode the JPEG image in NTSC format and render in client's IE window.
easyCam.class, easyCam$1.class	These files are JAVA byte code built from easyCam.java. When client accesses simplearm.cgi, the printed html contents also notifies the IE to download and execute these byte code.

Table 7. Contents in mjpegdemo/javaapplet_pal

File(s)	Description
easyCam_pal.java	This is the JAVA applet source code to decode the JPEG image in PAL format and render in client's IE window.
easyCam_pal.class, easyCam_pal$1.class	These files are JAVA byte code built from easyCam_pal.java. When client accesses simplearm_pal.cgi, the printed html contents also notifies the IE to download and execute these byte code.

Table 8. Contents in mjpegdemo/dsp

File(s)	Description
jpegencdecCombo.x64P	This is the DSP binary image which includes JPEG encoder, JPEG decoder, codec engine DSP portion, and the DSP link driver DSP portion. When the ARM application program tries to initialize the codec engine, it loads this image to DSP memory and lets DSP execute.
Folder jpegenc	This folder contains all the additional files necessary to build ARM application program to access JPEG encoder using XDC tools.
Folder jpegdec	This folder contains all the additional files necessary to build ARM application program to access JPEG decoder using XDC tools.

Table 9. Contents in mjpegdemo/config

File(s)	Description
RunInTmp	This script sets the environment to allow the ARM application program for NTSC format support to write output files to RAM disk (/tmp) while the HTTP server accesses them.
RunInTmp_pal	This script sets the environment to allow the ARM application program for PAL format support to write output files to RAM disk (/tmp) while the HTTP server accesses them.
cmemk.ko, dsplinkk.ko, loadmodules.sh	cmemk.ko and dsplinkk.ko are the CMEM and DSP link driver module, respectively. They need to be loaded to Linux kernel by executing script loadmodules.sh before running the application program.
httpd.conf	This is the configuration file to change the default setting of Apache HTTP server to support this demo. You can use this file to replace the default configuration file.
tibug.jpg	This is the TI logo image which is printed out as part of the html contents showing in client's IE window.
uImageC30	This is the Linux kernel image. In case some clients do not have the latest kernel image on their DM6446 DVEVM board, this image can be used to boot the kernel and run the demo.
Qf	This is a backup quality index file.

3 Migrating from DM642 to DM6446

This motion JPEG demo is derived from the motion JPEG demo running on TI DM642 processor. Using the motion JPEG demo, this section shows some tips on migrating legacy application code from DM642 to DM6446.

3.1 Motion JPEG Demo on DM642

Figure 4 is the dataflow diagram of the motion JPEG demo running on DM642 which is a DSP only environment. The input video signal is captured from the camera via the video input port and stored to the input buffer located in the SDRAM by the video port driver. The JPEG codec task reads the input frame directly, encodes it and saves the compressed JPEG image to the output buffer located in the SDRAM. There is also a DSP version HTTP server running. When you request to access the compressed JPEG images, the HTTP server handles the request and transmits the data in the output buffer via the IP link. When you request to change the image quality, the HTTP server writes the new quality index number to the SDRAM. When the JPEG codec starts to encode the next frame, it is able to read the updated number.

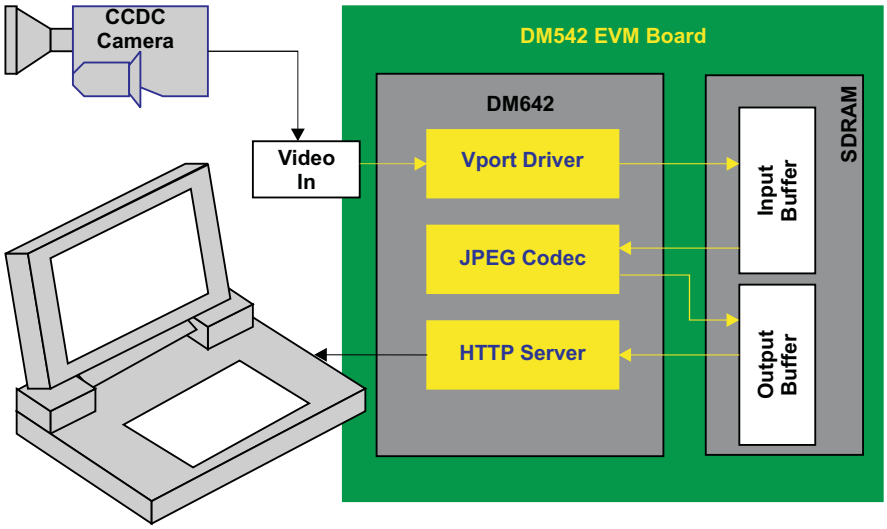

Figure 4. Dataflow of Motion JPEG Demo on DM642

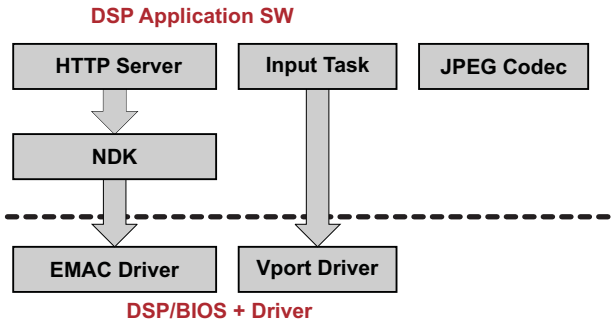

Figure 5. Software Architecture of Motion JPEG Demo on DM642

Figure 5 shows the software architecture of DM642 version demo. The HTTP server handles the HTTP requests and is implemented based on the Network Development Kit (NDK). The NDK is a TI library that emulates the behavior of Berkeley socket (BSD). NDK calls the EMAC driver to handle data sending/receiving.

The input task calls the video port driver to receive the input data. When a new frame is received, it notifies the JPEG codec to encode it.

3.2 DM642 Demo vs. DM6446 Demo

The pros of the DM642 version over the DM6446 version are:

- For the DM642 version demo, the video port driver runs on DSP. Because TI DSP does not differentiate application space and kernel space, this application can access the input buffer directly.
- For DM6446 version demo, the ARM application needs to deliver the frame to DSP and wait for it to finish encoding. The sequence involves address translation, argument copying, and cache invalidation by the Codec Engine, and data transfer by DSP link, as described in Section 3.1. For the DM642 version, activating the JPEG codec task to encode a frame when a new frame is captured is simply a DSP/BIOS semaphore post operation.
- For the DM642 version demo, an HTTP server is implemented based on NDK. The server task can retrieve the output images from the output buffer directly without saving it as a separate file as in the DM6446 version.

The cons of the DM642 version are:

- The HTTP server needs to be implemented by the user. For new TI DSP users, getting familiar with NDK and creating an HTTP server extends the development cycle, and may even be error prone. While for the DM6446 version, mature open source software such as apache http server, tiny http server, etc. can be deployed directly.
- Video port driver and HTTP server execute on DSP and compete for MIPS with the JPEG codec. For the DM6446 version, offloading these two tasks to the ARM core leaves room for DSP to do more complicated multimedia processing.

3.3 Porting DM642 Demo to DM6446 Platform

To port the DM642 version motion JPEG demo to DM6446 platform, the main tasks are as follows:

- Using the Linux video port driver running on DM6446 ARM core
- Using the Linux Apache HTTP server running on DM6446 ARM core
- Making the JPEG codec to be XDM compliant

3.3.1 Using the DM6446 Linux Video Port Driver

The APIs of the DM6446 video front end port driver are in compliance to second generation of video for Linux standard (V4L2) and different from the APIs of the DM642 video port driver. But the new APIs are also easy to use. To access the video data which is in the kernel space, the application needs to call the Linux API mmap(). The source code *app.c* shows the example on how to call the DM6446 video port driver.

3.3.2 Using the Apache HTTP Server on DM6446

The developer needs to use the Linux Apache HTTP server running on DM6446 ARM core instead of programming his/her own server to handle HTTP requests. Because the Apache server has been ported to DM6446 and provided with DM6446 DVEVM/DVSDK as a default option, the main work is to configure the Apache server to support running the demo cgi programs. The configuration step is shown in Section 4.

3.3.3 Make JPEG Codec section 4XDM Compliant

The legacy JPEG codec running on DM642 is compliant to TI eXpressDSP™ Algorithm Interface Standard (xDAIS). Algorithm compliant to xDAIS requests memory usage through xDAIS interface so that different algorithms can be easily integrated together without worrying about corrupting each other's memory.

From the application developer's perspective, the Codec Engine provides a set of xDM APIs that can be used to instantiate and run xDAIS algorithms. xDM stands for the eXpressDSP Algorithm Interface Standard for Digital Media and can be considered compliant with and a super set of xDAIS. Additionally, it implements two additional APIs to support multimedia codecs: process and control. The process API triggers the codec algorithm to process the input data. The control API has a broad range of usage such as inquires the codec status, dynamically configure the codec, etc. For the motion JPEG demo on DM6446, process is called to encode a raw video frame to a JPEG file, and control is called to dynamically change the encoding quality.

The xDM interfaces divide codec algorithms into four classes: video, image, speech, and audio (VISA). For each class, one set of APIs and data structures (struct in C language) are provided. The class specific data structures typically define the parameters which are common for this class of codec at the beginning of the structure. xDM defines an "extended" parameter in almost each structure to allow users to add their own special parameters there. xDM also allows users to create a codec which does not belong to a VISA class. When any extended parameters are added in data structure, or a completely new codec class is created, 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 [3] *Codec Engine Application Developer User's Guide* (SPRUE67).

To convert the DM642 version JPEG codec to be XDM compliant, the main work are as follows.

- Implement the process and control call.
- Make the codec recognize the image class data structures of xDM. Because this JPEG codec does not have any special parameters to be added, the existing image class stub/skeleton can be used.

4 How to Run

4.1 Necessary Hardware and Software

- DM6446 DVEVM board
- Windows PC with Internet Explorer installed
- CCD camera (NTSC or PAL resolution) connected to the DM6446 board
- Network router connecting Windows PC and DM6446 DVEVM board
- JAVA runtime environment (JRE) pre-installed on your Windows PC. If you do not have JRE pre-installed, please download and install it from http://www.java.com/en/download/manual.jsp.
- Serial cable connecting your Windows PC and DM6446 board
- Terminal program on Windows PC, i.e., hyperterminal

4.2 Configuration Steps Before Running the Demo

This section shows the preparation steps before running the demo. They only need to be done once.

1. Boot your DM6446 DVEVM board. Put the demo package *SPRC343.gz*. into a temporary directory on the hard drive of your DM6446 DVEVM board. In the rest of this section, it is assumed it is in the /opt directory.
2. Unzip the package by typing *tar -xzf SPRC343.gz*. You will see the /opt/mjpegdemo folder is created. There are 9 sub-folders as described previously:

```
cd /opt

tar -xzf sprc343.gz
```

3. Configure the Apache server on your DM6446 board as follows:
   a. Create a directory */var/www/html/jpegdemo* which is the demo directory on your DM6446 board.

```
mkdir /var/www/html/jpegdemo
```

   b. Add the following lines in file */etc/apache/httpd.conf* on your DM6446 board:

```
Alias /jpegdemo/ "/var/www/html/jpegdemo" <Directory "/var/www/html/jpegdemo"> Options
+ExecCGI </Directory> AddHandler cgi-script .cgi
```

   Or simply replace the default *httpd.conf* file with the one provided.

```
mv /etc/apache/httpd.conf /etc/apache/httpd.conf.bk
cp mjpegdemo/config/httpd.conf /etc/apache/
```

4. Copy the ARM application binary to the demo directory.

```
cp app/app.cgi /var/www/html/jpegdemo
```

5. Copy the JPEG codec combo to the demo directory.

```
cp dsp/jpegencdecCombo.x64P /var/www/html/jpegdemo
```

6. Copy the JAVA byte code to the demo directory.

```
cp javaapplet/*.class /var/www/html/jpegdemo
cp javaapplet_pal/*.class /var/www/html/jpegdemo
```

7. Copy the CGI program which prints out the html contents to the demo directory.

```
cp simpleweb/simplearm.cgi /var/www/html/jpegdemo
cp simpleweb_pal/simplearm_pal.cgi /var/www/html/jpegdemo
```

8. Copy the CGI program which handles encoding quality change to the demo directory.

```
cp cfgquality/cfgquality.cgi /var/www/html/jpegdemo
cp cfgquality_pal/cfgquality_pal.cgi /var/www/html/jpegdemo
```

9. Copy the other required files in *config* sub-folder to the demo directory.

```
cp config/RunInTmp /var/www/html/jpegdemo
cp config/RunInTmp_pal /var/www/html/jpegdemo
cp config/tibug.jpg /var/www/html/jpegdemo
cp config/Qf /var/www/html/jpegdemo
cp config/loadmodules.sh /var/www/html/jpegdemo
cp config/*.ko /var/www/html/jpegdemo
```

4.3 Executing the Demo

The following are steps to run the demo once you have completed the steps in Section 4.2.

4.3.1 Executing the Demo for NTSC Format

1. Make all the HW connections.
   a. Hook the CCD NTSC camera to your DM6446 board.
   b. Connect your Windows PC and DM6446 board using the network router.
   c. Connect your Windows PC and DM6446 board using a serial cable.
2. Launch the terminal program on your Windows PC and boot your DM6446 board.
3. Switch to the demo directory.

```
cd /var/www/html/jpegdemo
```

4. If it is already running, stop the tiny http server on your DM6446 board because it conflicts with the Apache HTTP server.

```
killall thttpd
```

5. Start the Apache server.

```
httpd
```

6. Run the script *RunInTmp* to allow the application program to write output files to the RAM disk (/tmp) while the Apache HTTP server can still access the files.

```
./RunInTmp
```

7. Load the DSP link and CMEM modules to kernel.

```
./loadmodules.sh
```

8. Start the application program.

```
cd /tmp
./app.cgi
```

9. Launch Internet Explorer on your Windows operating system and access URL *http://[DM6446_board_IP_address]/jpegdemo/simplearm.cgi* where DM6446_board_IP_address is the IP address of your board.
10. You should be able to see the videos playing in your Inernet Explorer window.
11. You can change the video quality by typing number in range [1,100] in the IE Window.

4.3.2 Executing the Demo for PAL Format

1. 1. Make all the HW connections.
   a. Hook the CCD PAL camera to your DM6446 board.
   b. Connect to your Windows PC and DM6446 board using the network router.
   c. Connect to your Windows PC and DM6446 board using a serial cable.
2. Launch the terminal program on your Windows PC and boot your DM6446 board.

3. Switch to the demo directory.

```
cd /var/www/html/jpegdemo
```

4. If it is already running, stop the tiny http server on your DM6446 board because it conflicts with the Apache HTTP server.

```
killall thttpd
```

5. Start the Apache server.

```
httpd
```

6. Run the script *RunInTmp_pal* to allow the application program to write output files to the RAM disk (/tmp) while the Apache HTTP server can still access the files.

```
./RunInTmp_pal
```

7. Load the DSP link and CMEM modules to kernel.

```
./loadmodules.sh
```

8. Start the application program, providing the *-PAL* argument .

```
cd /tmp
./app.cgi -PAL
```

9. Launch Internet Explorer on your Windows operating system and access URL *http://[DM6446_board_IP_address]/jpegdemo/simplearm_pal.cgi* where DM6446_board_IP_address is the IP address of your board.
10. You should be able to see the videos playing in your IE window.
11. You can change the video quality by typing number in range [1,100] in the IE window.

5 How to Re-Compile

5.1 Necessary HW and SW

In addition to the HW and SW necessary to run the demo, a Linux development host (or Linux virtual machine running on VMWare) with latest DM6446 DVEVM/DVSDK package being installed are required to re-compile the demo code.

5.2 Configuration Steps Before Re-Compiling the Demo

1. Install and configure the DVEVM packages by following the steps in the [1] *DVEVM Getting Started Guide* (SPRUE66). Make sure that you can write a simple program and compile and run it on the DM6446 board.
2. Install and configure the DVSDK packages by following the steps in the [2] *DVSDK Getting Started Guide* (SPRUEG8). Make sure that you can compile the codec engine examples.

3. Log in as the user to your Linux host. Put *SPRC343.gz* into a temporary directory (/tmp) and untar it. You will see the folder /tmp/mjpegdemo is created with 6 sub-folders as described before.

```
cd /tmp
untar -xzf sprc343.gz
```

4. Create a folder in *[DVEVM_INSTALL_DIR]/[codec_engine_x_xx]/examples/apps/jpegencdec*, which is the demo development directory. *[DVEVM_INSTALL_DIR]/[codec_engine_x_xx]* is the directory where the codec engine package for the DVEVM/DVSDK is installed. In the rest of the section, it will be referred to as /home/user/dvevm_1_10/codec_engine_1_02/.

```
mkdir /home/user/dvevm_1_10/codec_engine_1_02/examples/apps/jpegencdec
```

5. Switch to the development directory.

```
cd /home/user/dvevm_1_10/codec_engine_1_02/examples/apps/jpegencdec
```

6. Copy everything in *app* subfolder here

```
cp /tmp/mjpegdemo/app/* .
```

7. Copy folder *javaapplet* and *javaapplet_pal* here.

```
cp –r /tmp/mjpegdemo/javaapplet/ .
cp –r /tmp/mjpegdemo/javaapplet_pal .
```

8. Copy folder *simpleweb* and *simpleweb_pal* here.

```
cp –r /tmp/mjpegdemo/simpleweb/ .
cp –r /tmp/mjpegdemo/simpleweb_pal .
```

9. Copy folder *cfgquality* and *cfgquality_pal* here.

```
cp –r /tmp/mjpegdemo/cfgquality/ .
cp –r /tmp/mjpegdemo/cfgquality_pal .
```

Note: if your codec server directory already contains the jpeg encoder and decoder package, skip step 10 and 11.

10. Copy the XDM package for the JPEG encoder to the codec server directory. Your codec server directory is /home/user/dvevm_1_10/codec_servers_1_00. Type the following command in one line.

```
cp –r /tmp/mjpegdemo/dsp/jpegenc
/home/user/dvevm_1_10/codec_servers_1_00/packages/ti/sdo/codecs
```

11. Copy the XDM package for JPEG decoder to the codec server directory. Type the following command in one line.

```
cp –r /tmp/mjpegdemo/dsp/jpegdec
/home/user/dvevm_1_10/codec_servers_1_00/packages/ti/sdo/codecs”
```

5.3 Re-Compile the Demo

5.3.1 Re-Compile the ARM Application Code

- To compile the code, type the following:

```
cd /home/user/dvevm_1_10/codec_engine_1_02/examples/apps/jpegencdec
gmake
mv app_remote.x470MV app.cgi
```

The last step is necessary because the script *RunInTmp* assumes that the name of ARM application binary is *app.cgi*. To run this binary, it is necessary to transfer it to directory /var/www/html/jpegdemo” on your DM6446 board.

5.3.2 Re-Compile the CGI Code

- Type the following to compile the code for NTSC format support:

```
cd /home/user/dvevm_1_10/codec_engine_1_02/examples/apps/jpegencdec/simpleweb
make
```

The output code is *simplearm.cgi*. To run the generated binary, it is necessary to transfer it to the directory /var/www/html/jpegdemo” on your DM6446 board.

- Type the following to compile the code for PAL format support:

```
cd /home/user/dvevm_1_10/codec_engine_1_02/examples/apps/jpegencdec/simpleweb_pal
make
```

The output code is *simplearm_pal.cgi*. To run the generated binary, it is necessary to transfer it to the directory /var/www/html/jpegdemo” on your DM6446 board.

5.3.3 Re-Compile the JAVA Applet Code

- Type the following to compile the JAVA source code for NTSC format support:

```
cd /home/user/dvevm_1_10/codec_engine_1_02/examples/apps/jpegencdec/javaapplet
javac easyCam.java
```

The output JAVA byte code are *easyCam.class* and *easyCam$1.class*. To run the generated JAVA code, it is necessary to transfer them to directory /var/www/html/jpegdemo” on your DM6446 board.

- Type the following to compile the JAVA source code for PAL format support:

```
cd /home/user/dvevm_1_10/codec_engine_1_02/examples/apps/jpegencdec/javaapplet_pal
javac easyCam_pal.java
```

The output JAVA byte code are *easyCam_pal.class* and *easyCam_pal$1.class*. To run the generated JAVA code, it is necessary to transfer them to directory /var/www/html/jpegdemo" on your DM6446 board.

5.3.4 Re-Compile the Quality Configuration Code

- Type the following to compile the code for NTSC format support:

```
cd /home/user/dvevm_1_10/codec_engine_1_02/examples/apps/jpegencdec/cfgquality
make
```

The output code is *cfgquality.cgi*". To run the generated binary, it is necessary to transfer it to directory /var/www/html/jpegdemo" on your DM6446 board.

- Type the following to compile the code for PAL format support:

```
cd /home/user/dvevm_1_10/codec_engine_1_02/examples/apps/jpegencdec/cfgquality_pal
make
```

The output code is *cfgquality_pal.cgi*. To run the generated binary, it is necessary to transfer it to directory /var/www/html/jpegdemo" on your DM6446 board.

6 References

1. *DVEVM Getting Started Guide* (SPRUE66)
2. *DVSDK Getting Started Guide* (SPRUEG8).
3. *Codec Engine Application Developer User's Guide* (SPRUE67).

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 TI 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. Information of third parties may be subject to additional restrictions.

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.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

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
RFID	www.ti-rfid.com	Telephony	www.ti.com/telephony
Low Power Wireless	www.ti.com/lpw	Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

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