TMS320DM644x DVEVM Windows CE v5.0 Codec Engine Binary

User's Guide

Literature Number: SPRUEV8
March 2007

Contents

List of Figures

List of Tables

Read This First

About This Manual

This document accompanies the release of Windows® CE 5.0 BSP for DaVinci-based DVEVM.

Purpose and Scope

The codec engine is a set of libraries, which provide a standard set of APIs for application developers to run and instantiate audio/video/image/speech algorithms. The interface provided by the codec engine module is always the same regardless of whether the media algorithms execute locally or remotely.

This document provides information on the build procedure for the codec engine samples on the Windows® CE 5.0 platform. This release of the codec engine consists of:

- Codec engine core components in binary form
- Codec engine sample applications in source form

Notational Conventions

This document uses the following conventions:

- Backward slashes are used as pathname delimiters for filenames.
- Catalog->Third Party refers to the Catalog Window tree items in the platform builder IDE.
- All the shell commands are in courier new font.
- Menu commands are depicted using the following notation ***menu name > menu command***.

Terms, Acronyms and Descriptions

The following terms and acronyms will be used throughout the document and are described here for clarification.

Table 1. Terms, Acronyms and Descriptions

Number	Term	Description
1	API	Application Programmer Interface
2	CCS	Code Composer Studio™ Software (Version 2.2.1 with OSK board specific set-up installed)
3	GPP	General Purpose Processor (ARM®)
4	MMU	Memory Management Unit
5	OAL	OEM Adaptation Layer
6	OEM	Original Equipment Manufacturer
7	RCV	Raw Compressed Video
8	XDM	eXpressDSP™ Medium Software

Related Documentation from Texas Instruments

The following documents describe the BSP for DaVinci-based DVEVM.

SPRUEV8 — ***TMS320DM644x DVEVM Windows CE v5.0 Codec Engine Binary Users Guide***

Provides information on the build procedure for the codec engine samples on Windows CE 5.0 platform.

SPRUEV9 — ***TMS320DM644x DVEVM Windows CE v5.0 BSP Users Guide.***

Provides information about the release contents of Windows CE 5.0 BSP for DaVinci-based DVEVM. The document illustrates various components that are part of this release, the procedure to install this release on to the host system, and the limitations of the release.

SPRUEW1 — ***TMS320DM644x DVEVM Windows CE v5.0 BSP Bootloader Users Guide.***

Provides information about the Windows CE 5.0 bootloader for DaVinci EVM. The document illustrates various features and the build and flash procedures.

SPRUEW0 — ***TMS320DM644x DVEVM Windows CE v5.0 BSP DSP/BIOS Link Users Guide.***

Describes the usage of the DSP/BIOS Link binaries provided along with the Windows CE 5.00 BSP for the Davinci EVM platform and the integration procedures in a given Windows CE image.

SPRS283 — ***TMS320DM6446 Digital Media System-on-Chip Data Manual*** (SPRS283)

The TMS320DM6446 (also referenced as DM6446) leverages TI's DaVinci™ technology to meet the networked media encode and decode application processing needs of next-generation embedded devices.

Trademarks

Code Composer Studio, eXpressDSP, DSP/BIOS, DaVinci are trademarks of Texas Instruments.

ARM is a registered trademark of ARM Limited.

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

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

User's Guide
SPRUEV8–March 2007

Windows CE 5.0 Port on Davinci EVM

1 Codec Engine

This section explains how to build applications using the sample audio/video copy codec and the VC1 and WNA decode codecs. Examples currently contain the simple pass-through (copy) codecs implemented in the XDM algorithm standard.

1.1 Requirements

This document assumes that the sample applications are built using the XDC tools and that the XDC tools on the host development PC have been updated. If the tools are not updated, the codec engine sample applications cannot be built for the WinCE targets.

Please refer to Section 2 for the procedure to update the XDC tools.

1.2 Directory Structure

Figure 1 shows the directory structure of the codec engine binary tree. The following folders are present in the top level.

- cetools – Contains the cetools components
- examples – Contains the example codecs, servers and the sample applications of the codec engine
- packages – Contains the XDC packages of all the codec engine core components in binary form

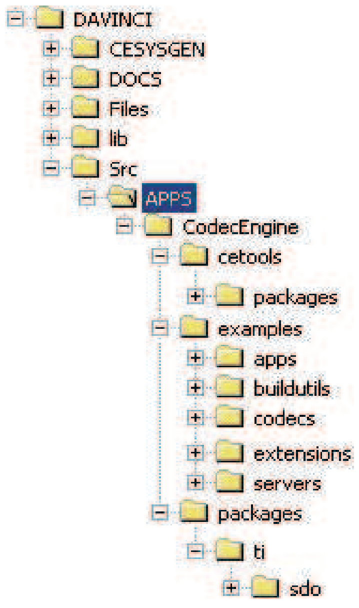

Figure 1. Codec Engine Binary Tree

1.3 Binary Tree Contents

This section provides information on the codecs, codec servers and sample applications released in the binary tree in Figure 1.

1.3.1 Codecs

The following codecs are supported in this release:

- Audio and video copy codecs
- WMA9 VC1 advanced profile decoder
- WMA9 decoder
- AAC low-complexity decoder
- G711 decoder
- G711 encoder
- MP3 decoder
- H.264 baseline profile decoder
- H.264 baseline profile encoder
- H.264 main profile decoder
- MPEG2 main profile decoder
- MPEG4 simple profile decoder
- MPEG4 simple profile encoder

Note: The codec engine binary tree contains only watermarked versions of the codecs, which limits how these codecs function.

1.3.2 Codec Servers

The following codec servers are supported in this release:

- Audio and video copy codec servers
- VC1 and WMA9 decode combo server
- VC1 decode server
- WMA9 decode server
- Decode combo codec server
- Encode combo codec server
- H.264 loopback encode-decode codec server

1.3.3 Sample Applications

Figure 2 shows the expanded view of the codec engine examples folder. The following samples can be built and executed in this binary release.

Audio_copy\dualcpu	Sample application which operates the audio copy codec
avplay	Sample Application based on monolithic DLL which operates the decode combo codec server
decodeApp	Sample application based on the XDC build which operates the decode combo codec server
decodeCombo	CE monolithic DLL for the decode combo codec server
encodeApp	Sample application based on the XDC build which operates the encode combo codec server
encodeCombo	CE Monolithic DLL for the encode combo codec server
h264encdec	CE monolithic DLL for the H.264 loopback codec server

loopback	Sample application based on the monolithic DLL which operates the H.264 loopback codec server
vc1wmaApp	VC1 WMA9 sample application based on the monolithic DLL which operates the VC1 WMA9 decode combo codec server
vc1wmadecode	VC1 WMA9 CE monolithic DLL for the VC1 WMA9 decode combo codec server
Video_copy\dualcpu	Sample application which operates the video copy codec
Vplay	VC1 sample application which picks up a pre-determined video sample input with a .VC1 extension and decodes the data using the remote VC1 codec server and writes the output onto the hard disk
WmvPlayer	Sample application, which picks up a user-specified .VC1 or .RCV file video input file, decodes the stream and renders it onto the display device using DirectDraw
wmaPlayer	Sample application which picks up a user-specified .RCA file and generates an equivalent .WAV file for the given input. The application can be built to render the output to the audio device also
vc1wmadecode	Codec engine monolithic DLL build for the VC1 WMA combo decode server

Note: This release does not include the speech and image samples.

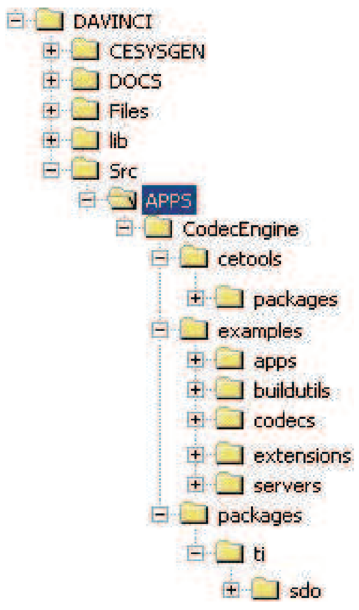

Figure 2. Codec Engine Examples

2 Updating the XDC Tools

This section provides information on the Code Composer Studio software update required to enable the XDC tools to build for Windows CE platforms.

By default, the XDC tools cannot build components for the GPP side for the Windows CE OS platform; therefore, it is mandatory that certain changes in the BIOS XDC tools files are made in order to support a build for the Windows CE OS platform.

The following steps are required to update the XDC tools:

1. Add a BIOS_INSTALL_DIR Windows environment variable. Set it to your SABIOS install location. For example,
   `BIOS_INSTALL_DIR = C:/CCStudio_v3.2/bios_5_21_01`

   Note: Be sure to use the forward slash instead of the backward slash.

2. Add the following line to your system PATH variable:
   `%BIOS_INSTALL_DIR%\xdctools;`
   For example, set
   `path=%path%;%BIOS_INSTALL_DIR%\xdctools;`
3. Create a backup copy of BIOS_INSTALL_DIR/xdctools/etc directory.
4. Create a backup copy of BIOS_INSTALL_DIR/xdctools/packages/microsoft/targets directory.
5. Create a backup copy of BIOS_INSTALL_DIR/xdctools/packages/xdc/rts directory.
6. Take the XDCTOOL_PATCH.zip provided with this release and unzip it into your local disk.
   For example, unzip XDCTOOL_PATCH.zip into c:\xdctool_patch folder.
7. Copy the updated **config.bld** file from the xdctool patch package c:\xdctool_patch\etc\config.bld into your BIOS_INSTALL_DIR/xdctools/etc directory.
8. Copy the updated **package.xdc** and the new **WINCE.xdc** from the xdctool patch package c:\xdctool_patch\packages\microsoft\targets into your BIOS_INSTALL_DIR/xdctools/packages/microsoft/targets directory.
9. Copy the updated **rts** directory from the xdctool patch package c:\xdctool_patch\xdctools\packages\xdc\rts directory into your BIOS_INSTALL_DIR/xdctools/packages/xdc/rts directory.
10. Copy the new **tisb** directory from the xdctool patch package c:\xdctool_patch\packages\tisb into the BIOS_INSTALL_DIR/xdctools/packages directory. It creates a new folder named tisb under the BIOS_INSTALL_DIR/xdctools/packages directory.
11. Update the BIOS_INSTALL_DIR \xdctools\packages\microsoft\targets\Wince.xdc file. This file contains the standard set of Windows CE include paths such as WinCE ARM cross-compiler and assembler paths that are referred by the XDC GMAKE tool.
12. Search for the keyword ***ITarget.Command cc*** in the **Wince.xdc** file. Under this section, update the **opts** field. This field contains the path for your Windows CE installation, Windows CE Public OAK, SDK paths which are referred to during the codec engine build.
13. Update the BIOS_INSTALL_DIR\xdctools\etc\config.bld file to reflect your Windows CE root installation. This file has a variable named msWinCEToolsDir, which must be pointed to during your Windows CE root installation.
14. Navigate to BIOS_INSTALL_DIR/xdctools/packages/microsoft/targets and run the following commands:
    a. Clean the targets package
    ```
    prompt> xdc clean
    ```
    b. Build the RTSC package
    ```
    prompt> xdc
    ```
    The following log shows a sample execution of the above commands.

```
C:\CCStudio_v3.2\bios_5_21\xdctools\packages\microsoft\targets>xdc clean

C:\CCStudio_v3.2\bios_5_21\xdctools\packages\microsoft\targets>xdc
making package.mak (because of package.bld) ...
generating interfaces for package microsoft.targets (because
package/package.xdc.xml is older than package.xdc
) ...
    translating VC98
    translating Win32
    translating Net32
    translating WinCE
    translating ITarget
all files complete.

C:\CCStudio_v3.2\bios_5_21\xdctools\packages\microsoft\targets>
```

15. Navigate to BIOS_INSTALL_DIR/xdctools/packages/microsoft/targets/rts and run the following commands:
    a. Clean the XDC package
       `prompt> xdc clean`
    b. Build the RTS package
       `prompt> xdc`

       The following shows a sample log of the above command execution.

```
C:\CCStudio_v3.2\bios_5_21\xdctools\packages\microsoft\targets\rts>xdc clean

C:\CCStudio_v3.2\bios_5_21\xdctools\packages\microsoft\targets\rts>xdc
making package.mak (because of package.bld) ...
generating interfaces for package microsoft.targets.rts (because
package/package.xdc.xml is older than package
.xdc) ...
all files complete.
C:\CCStudio_v3.2\bios_5_21\xdctools\packages\microsoft\targets\rts>
```

3 Using the Pre-Built Samples

This section explains how to use the pre-built sample applications that are part of this codec engine binary release.

This release of the codec engine binary tree provides the following pre-built sample applications:

- Audio/video copy codec samples - Sample application which uses the audio/video copy codecs.
- Decode combo codec server samples - Two sample applications are based on the decode combo codec server.
  – Avplay - sample application reads a user-specified H.264, MPEG2, MPEG4, AAC or MP3 file, decodes it, and renders the output to either the display or the audio driver.
  – decodeApp - sample application reads a user-specified H.264, MPEG2, MPEG4, AAC or MP3 file, decodes it, and writes the decoded file into the user-specified output file.
- Encode combo codec server samples - This release has one sample application which is based on the encode combo codec server.
  – encodeApp - sample application reads a user-specified Raw YUV or PCM input file, encodes it using either the H.264, MPEG4 or G711 encoder, and writes the encoded data into the user-specified output file.
- H.264 loopback codec server samples - This release has one sample application which is based on the H.264 loopback codec server
  – Loopback - sample application based on the monolithic DLL concept which basically accepts a user-specified YUV file, encodes it, decodes the encoded output,and writes the decoded YUV file back into another output YUV file.
- VC1 WMA decode combo samples - Three sample applications are based on the VC1 WMA decode combo server.

- – Vplay - sample application reads a pre-designed .VC1 file and generates the YUV equivalent of the input file.
- – wmvPlayer - sample application reads a user-specified .VC1 or .RCV file and renders the decoded YUV frame into display.
- – wmaPlayer - sample application reads a user-specified .RCA file and renders the decoded PCM output into the audio driver.
- – vc1wmaApp - sample application decodes a predefined video and audio stream and render the output to the display and audio drivers, respectively.

3.1 Using the wmvPlayer Sample

This section provides information on how to invoke the wmvPlayer application.

1. Build a regular mobile hand-held based configuration image from the platform builder. Select DirectDraw component in the image. If the video sample files are loaded into secondary storage, then select the appropriate storage components into the Image.
2. Select ***Third Party→BSPs→DaVinci→Codec Engine Demo→VC1 Decoder Demo*** catalog component.
   - This catalog item performs necessary configuration for integration of the VC1 DSP decoder, sample VC1 file and wmvPlayer executable and also the codec engine DLLs into the underlying NK.BIN. Selecting this item into the workspace during building of the NK.BIN ensures that the VC1 decoder demo application wmvPlayer.exe is merged into the image.
   - If the wmvPlayer application is invoked without specifying any input file, it defaults to SA10094_720X480.vc1. However, the user can specify custom video files through a command-line argument. To specify the command-line arguments, invoke the application through the target control shell or the command shell in the NK.BIN.
3. Once the rendering is complete, press the *CC* button on the IR remote to quit the application.

3.2 Using the Decode Combo Sample Application

This section provides information on selecting and executing the pre-built decode combo sample application.

1. Build a regular mobile-handheld configuration OS Image with KITL support. Include the Third Party > BSPs > DAVINCI > Codec Engine Demo > Decode Combo Demo Catalog component in the image.
2. This component ensures that the decode combo server (decodeCombo.x64P), decode combo monolithic DLL (decodeCombo.dll), and the decode sample application (encodeApp.exe), are copied and merged into the Windows CE OS Image.
3. This catalog component copies the required DSP/BIOS LINK DLLs into the OS image, and also copies the required codec engine registry settings into the final OS image registry file.
4. Once the OS image build is complete, boot the Davinci EVM board with the OS image.
5. After the target loads and the initialization is complete, using the Remote Tools, copy the required H.264 or MPEG2 or MPEG4 or AAC or MP3 files into the secondary storage (hard disk) or on the root folder of the target device.
6. Execute the sample application by invoking the same in the target control shell using the commands `avplay`

   This application expects the following command-line parameters:

   -d Decoder type to be instantiated. [H264, or MPEG2 or MPEG4 or AAC or MP3]

   -i Input file to be decoded with the complete path

   -w Width of the frame if the input file contains Video data

   -h Height of the frame if the input file contains Video data
7. The application launches and invokes the appropriate decoder based on the input file type specified by the user. Note that this application requires the extension name to be specified for the input files. The appropriate decoder is invoked based only on the input file extension.
8. For video input files, the application invokes the appropriate video decoder (H.264, MPEG2 or MPEG4) and for audio input file, it invokes the appropriate audio decoder (AAC or MP3).
9. The application reads the input file on a frame basis, decodes the same, and renders the output to

either to the video display or the audio driver.

3.3 Using the Encode Combo Sample Application

This section provides information on selecting and executing the pre-built encode combo sample application.

1. Build a regular mobile-handheld configuration OS image with KITL support. Please include the Third Party > BSPs > DAVINCI > Codec Engine Demo > Encode Combo Demo Catalog component in the image.
2. This component ensures that the encode combo server (encodeCombo.x64P), encode combo monolithic DLL (encodeCombo.dll) and the decode sample app (avplay.exe), are copied and merged into the Windows CE OS image.
3. Note that this catalog component also copies the required DSP/BIOS LINK DLLs into the OS image. It also copies the required codec engine registry settings into the final OS image registry file.
4. Once the OS image build is complete, boot the Davinci EVM Board with the OS image.
5. After the target loads and the initialization is complete, use the remote tools and copy the required raw video or audio file (for raw video, YUV 422 format files are required and for audio raw PCM files are required) into the secondary storage (hard disk) or on the root folder of the target device.
6. Execute the sample application by invoking it in the target control shell using the command `s encodeApp`

   This application expects the following command-line parameters:

   -i Raw video or audio input file to be encoded

   -o Output File into which the encoded output is to be stored with the extension (.264 for H.264, or .mpeg4 for MPEG4) or (.g711 for G711)

   -w Width of the frame if the input file contains video data

   -h Height of the frame if the input file contains video data

   -n Number of video/audio frames to be encoded
7. The application launches and invokes the appropriate encoder based on the output file type specified by the user. This application requires the extension name to be specified for the output files. The appropriate encoder is invoked based only on the output file extension.
8. For video output files, the application invokes the appropriate video encoder (H.264 or MPEG4) and for audio output files, it invokes the G711 Encoder.
9. The application reads the input file on a frame basis encodes the same and writes the encoded data into the user-specified output file.

4 Building the Examples

4.1 Examples Backup

This step is optional, but recommended if you plan to modify the samples in any way. Copy the entire *examples* tree out of the binary tree. It will ensure you have a backup copy of the original examples, as provided by the codec engine product.

Note: The following notation will be used throughout this document.

- `<CE_EXAMPLES_INSTALL_DIR>` - Absolute path of the examples directory or the copy you made.
- `<CE_INSTALL_DIR>` - Root directory of your codec engine installation. The original examples are in `<CE_INSTALL_DIR>/`examples.
- `<BIOS_INSTALL_DIR>` - Root directory of your DSP/BIOS™ software installation.
- `<XDC_INSTALL_DIR>` - Root directory of your xdctools installation.
- *directory/file* - Position of the file relative to the examples directory; for examples, `codecs/makefile` refers to `<CE_EXAMPLES_INSTALL_DIR>/codecs/makefile`.

4.2 Edit XDC User Build Configuration File

The file <CE_EXAMPLES_INSTALL_DIR>/user.bld at the root of the Examples directory informs the XDC tools (tools which the codec engine uses to build itself, codecs, servers, etc.) where to find compilers and other tools on the user's system. Open this file in a text editor.

- Build.targets
  - The codec engine is configurable and runs in many different environments. The Build.targets array specifies which targets you want to build for. By default, the WinCE target is enabled.
- Toolchain roots - rootDir
  - C64P.rootDir - Specify where your C64 tools are. Edit the string (directory name) assigned to the C64P.rootDir variable. This is the directory that contains bin/, include/, and lib/ subdirectory (this is only necessary if *C64P* is included in your Build.targets array). This is to be updated only if building the codec server is required.
  - WinCE.rootDir – Specify where your Windows CE installation is available. Edit the string (directory name) assigned to this variable. This is the directory that contains the /PUBLIC and /SDK subdirectories of the Windows CE installation.
  - UCARM9.rootDir – Specify where your uclibc-based Arm9 tools are. Edit the string (directory name) assigned to the UCArm9.rootDir variable. This is only necessary if UCArm9 is included in your Build.targets array; it is not by default.
  - Linux86.rootDir - Specify where your native, Linux®-x86 tools are. Edit the string (directory name) assigned to the Linux86.rootDir variable. This is only necessary if Linux86 is included in your Build.targets array.

4.2.1 Edit xdcpaths for Defining Build Variables

The xdcpaths.mak file, located at the root of the examples/ directory, defines where the codec engine is installed, where BIOS is, where the XDC tools are, and where individual codec engine packages reside. Open this file in a text editor and then:

1. Specify the location of your codec engine installation:
   Uncomment the CE_INSTALL_DIR = line and specify, as the value, the absolute path to your codec engine installation directory.
2. Specify the location of the BIOS installation:
   Uncomment the BIOS_INSTALL_DIR = line and specify the absolute path to your DSP/BIOS installation directory.
3. Specify where your XDC tools are installed:
   Uncomment the XDC_INSTALL_DIR = line and specify the absolute path to your XDC tools installation directory.

   If your CE distribution does not include a cetools directory, you may also have to define the following variables:
4. Specify where your xDAIS installation directory is (note that this must be xDAIS 5.00 or greater, as the ti.xdais.dm package is not available in previous xDAIS releases):
   Uncomment the XDAIS_INSTALL_DIR = line and specify, as the value, the absolute path to your xDAIS installation directory.
5. Specify where your DSP Link installation is:
   Uncomment the DSPLINK_INSTALL_DIR = line and specify the absolute path to your DSP Link installation directory.

Note: Please do not set the path completely to DSPLINK. The build tool checks for the dsplink package starting from cetools/packages/dsplink.

6. Specify where your CMEM installation is:
   Uncomment the CMEM_INSTALL_DIR = line and specify the absolute path to your CMEM installation directory.
7. Specify where your Framework components installation is:
   Uncomment the FC_INSTALL_DIR = line and specify the absolute path to your FC installation directory. Currently, Framework components are not available for ARM or x86, so this is only necessary when building DSP-side content.

8. Each directory contains a GNU makefile which enables you to build the sample in the current directory. Top-level directories also contain a makefile which steps into subdirectories and builds all the examples under the parent directory.
9. The xdcpaths.mak file is included by the individual makefiles for all the example codecs, servers, and applications.

CAUTION

Most build troubles occur when one of the various *_INSTALL_DIR variables are incorrect. Make sure there are no extra spaces, that every individual path (segment separated by the semicolon) is correct, character for character, and the build process is very likely to go smoothly.

4.3 Audio_copy Sample Build

The following steps build an Audio_copy sample application.

1. Edit GPP audio_copy client application makefile to specify package paths for the GPP client application.
   - File apps/audio_copy/dualcpu/makefile builds the GPP client application for the DSP server above.
2. Use the GPP (ARM, on DaVinci™) application to read input audio file *in.dat*, call the DSP audio encoder codec via the codec engine, call the DSP audio decoder codec on the intermediate data, and write the resulting output data to a file, *out.dat*. Since both codecs are really just copy codecs, in.dat and out.dat should be identical.

   This makefile, is an example of a Windows CE console application building through a RTSC makefile, with several insertions in the makefile that enable inclusions of XDC-generated portions of the application.

   The audio sample application contains the following sources:

   app.cfg: RTSC configuration script that determines what codecs will be made known to the codec engine running on the ARM

   app.c: Contains the main() and the encodeDecodeFile() routine which invokes various wrapper calls present in ceapp.c

   ceapp.c: Contains the routines to initialize the codec engine, and invokes various VISA calls
3. Build the GPP audio_copy client application
   a. Build a regular mobile hand-held configuration workspace image in the platform builder IDE.
   b. Using the Build OS → Open Release Directory command, invoke a command-shell window from PB.
   c. Update the paths environment variable to include the <BIOS_INSTALL_DIR> \xdctools folder in the standard list of paths.
   ```
   path=%path%;%BIOS_INSTALL_DIR%\xdctools;
   ```
   d. Navigate to the directory <CE_EXAMPLES_INSTALL_DIR>/apps/audio_copy/dualcpu (where the makefile is located) and type:
   ```
   gmake clean
   gmake
   ```

   The preceding steps build the Audio_Copy sample application app.exe.
4. Copy the audio copy codec server, all.x64P, in.dat, and app.exe into the project's release folder.

4.3.1 Running Audio_Copy

This section assumes that a mobile hand-held configuration image is built and the resulting Image has the necessary registry settings for codec engine and DSP/BIOS LINK components.

Note: Please ensure that dspbioslink.dll and dsplinkapi.dll are merged into the workspace image either by updating the platform.bib or the project specific project.bib. Additionally, the registry settings of codec engine and DSP/BIOS LINK should also be merged into the resulting reginit.ini file.

Follow these steps to execute the Audio_copy application.

1. Power-up the evaluation module (EVM) board and download the image.
2. Once the image loads and comes up, using the remote tools, copy the audio/video copy codec server all.x64P and in.dat file into the target's root folder.

   Important: Please ensure that DSP/BIOS LINK dll is loaded at image start-up.
3. Execute the app.exe.

4.4 Video_copy Sample Build

The following steps are necessary to build a Video_copy sample application.

1. Edit GPP video_copy client application makefile to specify package paths for the GPP client application.
2. File apps/video_copy/dualcpu/makefile builds the GPP client application for the DSP server above.
3. Use the GPP (ARM, on DaVinci) application to read input video file *in.dat*, call the DSP video encoder codec via the codec engine, call the DSP video decoder codec on the intermediate data, and write the resulting output data to a file, *out.dat*. Since both codecs are really just copy codecs, in.dat and out.dat should be identical.

   The video sample application contains the following sources:

   ceapp.cfg: RTSC configuration script that determines what codecs will be made known to the codec engine running on the ARM.

   app.c: Contains the main() and the encodeDecodeFile() routine which invokes various wrapper calls present in ceapp.c.

   ceapp.c: Contains the routines to initialize the codec engine, and invokes various VISA calls.
4. Build the GPP video_copy client application.
   a. Build a regular mobile hand-held configuration workspace image in the platform builder IDE.
   b. Using the Build OS → Open Release Directory command, invoke a command-shell window from PB.
   c. Update the paths environment variable to include the <BIOS_INSTALL_DIR> \xdctools folder in the standard list of paths.
      ```
      path=%path%;%BIOS_INSTALL_DIR%\xdctools;
      ```
   d. Navigate to the directory <CE_EXAMPLES_INSTALL_DIR>/apps/video_copy/dualcpu (where the makefile is) and type:
      ```
      gmake clean
      gmake
      ```

   The preceding steps build the Video_Copy sample application app.exe.
5. Copy the video copy codec server, all.x64P, in.dat and app.exe into the project's release folder.

4.4.1 Running Video_Copy

This section assumes that a mobile hand-held configuration image is built and the resulting image has the necessary registry settings for codec engine and DSP/BIOS LINK components.

Note: Please ensure that the dspbioslink.dll and dsplinkapi.dll are merged into the workspace image either by updating the platform.bib or the project specific project.bib. Additionally, the registry settings of the codec engine and DSP/BIOS LINK should also be merged into the resulting reginit.ini file

The following steps execute the video_copy application.

1. Power-up the EVM board and download the image.
2. Once the image loads and comes up, using the remote tools, copy the audio/video copy codec server all.x64P and in.dat file into the target's release folder.

   Important: Please ensure that the DSP/BIOS LINK dll is loaded at image start-up.
3. Execute the app.exe.

4.5 Vplay Sample Build

The Vplay sample application uses a real codec. It is a sample application based on the VC1 decoder. It takes as input a .VC1 file which contains compressed raw video data, passes the same to the DSP VC1 codec server, reads the result from the DSP, and then writes the decoded output frames into a file named output.yuv onto the hard disk of the DaVinci EVM board.

In order to execute the vplay sample application, ensure that the Windows CE OS image is built with the hard disk component selected, and the storage manager component (FAT FileSystem and partition driver) are selected.

1. Edit the GPP vplay client application makefile to specify package paths for the GPP client application.
2. File apps/vplay/makefile builds the GPP client application for the DSP server above.
3. Use the GPP (ARM, on DaVinci) application to read input VC1 file *SA10094_720X480.vc1*, call the DSP VC1 decoder via the codec engine, and write the resulting output data to a file, *output.yuv*.

   The Vplay sample application contains the following sources:

   app.cfg: RTSC configuration script that determines what codecs will be made known to the codec engine running on the DSP.

   app.c: Contains the main() and invokes routines defined in svd.c, bufIo.c

   svd.c: Contains the wrapper routines to invoke the video decode algorithm, sends and receives data from the remote VC1 codec server.

   BufIo.c: Contains the file I/O related routines to read the input VC1 file.
4. Build the GPP vplay client application.
   a. Build a regular mobile hand-held configuration workspace Image in the platform builder IDE. Ensure that hard disk support is added into the image.
   b. Using the Build OS → Open Release Directory command, invoke a command-shell window from PB.
   c. Update the paths environment variable to include the <BIOS_INSTALL_DIR> \xdctools folder in the standard list of paths.
      ```
      path=%path%;%BIOS_INSTALL_DIR%\xdctools;
      ```
   d. Navigate to the directory <CE_EXAMPLES_INSTALL_DIR>/apps/vplay (where the makefile is) and type:
      ```
      gmake clean
      gmake
      ```

   The preceding steps build the Vplay sample application, vplay.exe
5. Copy the VC1 decode codec server vc1wma9decodeCombo.x64P, SA10094_720X480.vc1 and app.exe into the project's release folder.

4.5.1 Running Vplay

This section assumes that a mobile hand-held configuration image is built with hard disk support and the resulting image has the necessary registry settings for codec engine and DSP/BIOS LINK components.

Note: Please ensure that the dspbioslink.dll and dsplinkapi.dll are merged into the workspace image either by updating the platform.bib or the project specific project.bib. Additionally, verify if the hard disk driver is present in the image. The registry settings of the codec engine and DSP/BIOS LINK should also be merged into the resulting reginit.ini file.

The following steps execute the Vplay application.

1. Power-up the EVM board and download the image.
2. Once the image loads and comes up, using the remote tools, copy the VC1 codec server vc1decodeCombo.x64P and SA10094_720X480.vc1 file into the target's release folder.

   Important: Please ensure that DSP/BIOS LINK dll is loaded at image start-up.
3. Execute the vplay.exe. It opens the SA10094_720X480.vc1, decodes the same and writes the generated output into the hard disk in a file named output.yuv.

4.6 WmvPlayer Sample Build

The wmvPlayer sample application uses a real codec. It is a sample application based on the VC1 decoder. It takes as input a .VC1 file which contains compressed raw video data, passes the same to the DSP VC1 codec server, reads the result from the DSP and then renders the decoded output frames onto the display device using DirectDraw. The following steps build the WmvPlayer sample:

1. Edit GPP wmvPlayer client application makefile to specify package paths for the GPP client application.
2. File apps/wmvPlayer/makefile builds the GPP client application. The GPP (ARM, on DaVinci) application reads the user specified input VC1 file.

   The video sample application contains the following sources:

   wmvPlayer.cfg: RTSC configuration script that determines what codecs will be made known to the codec engine running on the ARM.

   bufio.cpp: Contains the file I/O routines to read the input sample VC1 file.

   svd.cpp: Contains the wrapper routines to initialize codec engine, and invoke various VISA calls for the VC1 codec server.

   WmvPlayer.cpp: Contains the WinMain() routine for the sample application. Initializes the DirectDraw infrastructure, calls the routines in bufio.cpp and svd.cpp to decode the frame and render the output onto display.
3. Build the GPP wmvPlayer client application.
   a. Build a regular mobile hand-held configuration workspace image in the platform builder IDE.

      Important: Please ensure that DirectDraw component (SYSGEN_DDRAW) is included in the Windows CE image.
   b. Using the Build OS → Open Release Directory command; invoke a command-shell window from PB.
   c. Update the paths environment variable to include the <BIOS_INSTALL_DIR> \xdctools folder in the standard list of paths.
      ```
      path=%path%;%BIOS_INSTALL_DIR%\xdctools;
      ```
   d. Navigate to the directory <CE_EXAMPLES_INSTALL_DIR>/apps/wmvPlayer (where the makefile is) and type:
      ```
      gmake clean
      gmake
      ```

   The preceding steps build the VC1 sample application wmvPlayer.exe
4. Copy the VC1 decode codec server vc1wma9decodeCombo.x64P, SA10094_720X480.vc1 and wmvPlayer.exe into the project's release folder.

4.6.1 Running WmvPlayer

This section assumes that a mobile hand-held configuration image is built and the resulting image has the necessary registry settings for codec engine and DSP/BIOS LINK components.

Note: Please ensure that the dspbioslink.dll and dsplinkapi.dll are merged into the workspace image either by updating the platform.bib or the project specific project.bib. Additionally, the registry settings of the codec engine and DSP/BIOS LINK should also be merged into the resulting reginit.ini file.

The following steps execute the WmvPlayer.

1. Power-up the EVM board and download the image.
2. After the image initialization is complete and the Windows CE shell comes up, use the Tools → Remote File Viewer command to view the folders/files on the target.
3. Copy the VC1 codec server vc1decodeCombo.x64P and SA10094_720X480.vc1 file into the target's root folder.

   Important: Please ensure that DSP/BIOS LINK dll is loaded at image start-up.
4. Copy the wmvPlayer.exe into the target's root folder.
5. Close the remote file viewer window and invoke the wmvPlayer.exe application from the Windows CE target shell. The syntax for invoking this application is:
   ```
   wmvPlayer <source_vc1_filename.vc1>
   ```
   For example to invoke SA10094_720X480.vc1 placed in root folder, enter:
   ```
   s wmvPlayer SA10094_720X480.vc1
   ```
   Note that this sample application supports playing only compressed raw VC1 file stream and does not support decode of regular WMV9/VC1 files. The output of this video sample contains three football players.

   The SA10094_720X480.vc1 test file is a sample file containing data worth 29 video frames. This demo app is a Win32 application, which checks for the WM_CLOSE message.
6. Press the *CC* button on the IR remote to shutdown the wmvPlayer application. The Windows CE shell appears back after this is done.

4.7 WmaPlayer Sample Build

The wmaPlayer sample application uses the WMA audio decoder. It is a sample application based on the WMA9 decoder. It takes as input a WMA file converted in raw compressed audio (RCA) format. The application reads the raw audio payloads from the user specified RCA file, invokes the audio decoder VISA routines to decode the same. The decoded output can either be written into a output WAVE file or the same can be written by invoking the Windows CE Wave API manager routines.

1. Edit the GPP wmaPlayer client application makefile to specify package paths for the GPP client application.
2. File apps/wmaPlayer/makefile contains the build rules to build the wmaPlayer sample application.

   The associated package.bld provides instructions on the various source modules to be built in order to generate the executable.

   The wmaPlayer package.bld contains the following sources:

   wmaPlayer.cfg: RTSC configuration script that determines what codecs will be made known to the codec engine running on the ARM.

   audioDecoder.c: Contains the audio decoder related routines. It contains all the routines to read the input RCA file, initialize the audio decoder algorithm, invoke the decoder process routines.

   wmaPlayer.c: Contains the WinMain() routine for the sample application. Initializes the audio decoder codec engine, invokes the routines defined audioDecoder.c to read the user specified Input file and perform the audio decoding.
3. Build the GPP wmaPlayer client application.
   a. Build a regular mobile hand-held configuration workspace Image in the platform builder IDE.

      Important: Ensure that the audio driver and waveform audio [SYSGEN_AUDIO] is included in the Windows CE image.

b. Using the Build OS → Open Release Directory command, invoke a command-shell window from PB.

c. Update the paths environment variable to include the <BIOS_INSTALL_DIR> \xdctools folder in the standard list of paths.
```
path=%path%;%BIOS_INSTALL_DIR%\xdctools;
```

d. Navigate to the directory <CE_EXAMPLES_INSTALL_DIR>/apps/wmaPlayer (where the makefile is) and type:
```
gmake clean
gmake
```

The preceding steps build the WMA audio sample application wmaPlayer.exe.

4.7.1 Running the wmaPlayer

This section provides information on using the audio decoder sample application. This section assumes that the steps mentioned in Section 2 are completed before performing the procedure mentioned in this section.

1. Build a mobile-handheld configuration based Windows CE Image derived from the DaVinci BSP catalog component.
2. Ensure that the audio driver and the waveform audio component are selected in the image.
3. Build the audio decoder sample application using the procedure mentioned in Section 4.7.
4. Boot the DaVinci EVM board with the Windows CE image.
5. Using the Remote tools, copy the <CE_EXAMPLES_INSTALL_DIR>/servers/vc1wmadecode/vc1wma9decodeCombo.x64P DSP codec image file onto the device.
6. Copy the <CE_EXAMPLES_INSTALL_DIR>/apps/wmaPlayer/wmaPlayer.exe into the device.
7. Copy a sample RCA file <CE_EXAMPLES_INSTALL_DIR>/apps/wmaPlayer/welcome.rca file onto the device.
8. Invoke the wmaPlayer executable from the target shell window. The syntax is given below:
   ```
   s wmaPlayer welcome.rca
   ```
   By default, the sample application is built to generate an output WAVE file for the given input RCA File. The wmaPlayer.exe generates an output.wav file in the root folder of the target device.

   The application is a Windows application which checks for the input Windows messages.
9. Press *CC* or the ESC button on the IR remote to close the application.

4.7.2 Audio Rendering

The wmaPlayer sample application can also be built to render the generated output into the audio device present in the OS image. For generating this version of the sample application, please follow the steps given below:

1. Follow the steps mentioned in Section 4.7 to set up a build shell for the sample application. In the header file examples\apps\wmaPlayer\AudioDecoder.hpp, search for the line containing the following:
   ```
   #define WAVE_OUTPUT_ENABLE      1
   ```
2. Comment out this line and then perform a gmake clean and gmake command. The built executable can be copied into the target device.
3. Now the application invokes a secondary thread which performs the WaveAPI manager calls to submit a filled WAVEHDR structures. Invoke the wmaPlayer.exe as specified in Step 1 of Section 4.7.1.
4. Once the event object notification is received, it checks for further data to be decoded from the input file, performs the decode call, and submits another set of WAVEHDR structures to the WAVE API manager.
5. Repeat this process until the input file is completely decoded or the audio decoder algorithm returns ERROR.

   The application is a Windows application which checks for the input Windows messages.
6. Press *CC* or the ESC button on the IR remote to close the application

4.8 Decode Server Samples

This release of the codec engine has two sample applications for verifying the working of the decode combo codec server. This section provides information regarding these sample applications, the build procedure for them, and the necessary configuration. The sample applications provided with this release are:

decodeApp – This is a console-based application which basically accepts a user specified file (H.264, MPEG4, MPEG4AAC, MP3 or G.711). The application instantiates the appropriate decoder and writes the decoded data into pre-designated output files.

Avaplay – This is a console-based application which accepts a user-specified file (H.264 MPEG4, MPEG4AAC, MP3 or G.711) as input. The application instantiates the appropriate decoder and renders the output to the display or the audio device.

4.8.1 DecodeApp Sample

The decodeApp sample is an application for testing the decode codec combo. The application reads files in different compression formats, decodes the clip, and writes the result (video/audio) to the appropriate video and files. DecodeApp supports the following content:

- Video - H.264,MPEG4, MPEG2
- Audio - AAC,MP3
- Speech- G711

The sample application contains the following source files:

decodeApp.c - Source file which contains the WinMain() routine and invokes the various routines which internally invokes the appropriate decoders.

decodeApp.h – Header file for the routines defined in decodeApp.c

makefile – Makefile which contains the build rules for building the application

package.bld – Package file which contains the RTSC specific configuration

package.xdc– XDC file for the RTSC-based build

ecodeApp.cfg – Configuration file which declares the names of the decode combo server, the decoders

Please follow the procedure mentioned below for building the decodeApp application.

1. Build a regular mobile hand-held configuration Workspace Image in the Platform Builder IDE
2. Using the Build OS →Open Release Directory command, invoke a command-shell window from PB
3. Update the paths environment variable to include the <BIOS_INSTALL_DIR> \xdctools folder in the standard list of paths
   ```
   path=%path%;%BIOS_INSTALL_DIR%\xdctools;
   ```
4. Update the XDC_PATH variable to include the imports folder in the standard list of paths
   ```
   set XDCPATH=$(XDC_PATH);C:\wince500\platform\davinci\src\ce-d16\imports
   ```
5. Navigate to the directory <CE_EXAMPLES_INSTALL_DIR>/apps/decodeApp (where the makefile is) and type
   ```
   gmake clean
   gmake
   ```

These steps build the decodeApp.exe in the apps/decodeApp folder.

4.8.2 AvPlay Sample

The avplay sample is an application for testing the decode codec combo. The application reads files in different compression formats; decodes the clip and renders the result (video/audio) to the appropriate video/audio files.

The Avplay supports the following content:

- video - H.264,MPEG4, MPEG2
- Audio - AAC,MP3

- Speech- G711

  This application is based on the decode combo monolithic DLL and uses the Windows PB tools for compilation. The sample application contains the following source files:
- audioDecoder.cpp – Source file which contains the audio and speech-related routines
- audioDecoder.hpp - Header file for the routines defined in audioDecoder.cpp
- avplay.cpp – Main source file which contains the WinMain() routine and the corresponding helper routines
- bufio.cpp – Contains the file I/O related routines
- bufio.h – The header file for the routines defined in bufio.cpp
- svd.cpp – Contains the video decoder wrapper routines. These wrapper routines are invoked by the routines defined in avplay.cpp
- svd.h – Header file for the routines defined in svd.cpp
- makefile – PB build makefile
- sources – PB build file which contains the list of all the source files to be built and the include, library paths

Please follow the procedure mentioned below for building the Avplay application.

- Build a regular mobile hand-held configuration Workspace Image in the Platform Builder IDE.
- Using the Build OS →Open Release Directory command, invoke a command-shell window from PB
- Update the path's environment variable to include the <BIOS_INSTALL_DIR> \xdctools folder in the standard list of paths.
  ```
  path=%path%;%BIOS_INSTALL_DIR%\xdctools;
  ```
- Update the XDC_PATH variable to include the imports folder in the standard list of paths.
  ```
  set XDCPATH=$(XDC_PATH);C:\wince500\platform\davinci\src\ce-d16\imports
  ```
- Navigate to the directory <CE_EXAMPLES_INSTALL_DIR>/apps/avplay (where the sources is) and type
  ```
  Build -c
  ```

These steps build the avplay.exe under the obj/ARMV4I/ [retail | debug] depending on the build configuration selected.

4.9 Encode Server Samples

This release of the codec engine has one application for verifying the working of the encode combo codec server. This section provides information regarding these sample applications, the build procedure for them, and the necessary configuration.

The sample application provided with this release is:

encodeApp - This is a console-based application which basically accepts a user-specified file (YUV or PCM) and encodes it into the format as specified by the user. The application instantiates the appropriate encoder and writes the encoded data into a user-specified output file.

4.9.1 EncodeApp Sample

The encodeApp sample is an application for testing the encode codec combo. The application expects the user to specify an input file (YUV or PCM) file and encodes the same and writes the encoded data into the user specified output file.

This application supports the following content:

- Video - H.264 and MPEG4 Encoding
- Speech- G711 Encoding

The sample application contains the following source files:

- encodeApp.c – Source file which contains the WinMain() routine, invokes the various routines which internally invokes the appropriate encoder routines.
- encodeApp.h – Header file for the routines defined in encodeApp.c
- makefile – Makefile which contains the build rules for building the application

- package.bld – Package file which contains the RTSC-specific configuration
- package.xdc – XDC file for the RTSC-based build
- encodeApp.cfg – Configuration file which declares the names of the encode combo server, the decoders

Follow the procedure mentioned below for building the encodeApp application.

1. Build a regular mobile hand-held configuration Workspace Image in the Platform Builder IDE.
2. Using the Build OS>Open Release Directory command; invoke a command-shell window from PB.
3. Update the paths environment variable to include the <BIOS_INSTALL_DIR> \xdctools folder in the standard list of paths.
   ```
   path=%path%;%BIOS_INSTALL_DIR%\xdctools;
   ```
4. Update the XDC_PATH variable to include the imports folder in the standard list of paths.
   ```
   set XDCPATH=$(XDC_PATH);C:\wince500\platform\davinci\src\ce-d16\imports
   ```
5. Navigate to the directory <CE_EXAMPLES_INSTALL_DIR>/apps/encodeApp (where the makefile is) and type
   ```
   gmake clean
   gmake
   ```

These steps build the encodeApp.exe in the apps/encodeApp folder.

4.10 Loopback Server Samples

This release of the codec engine has sample application for verifying the working of the H.264 loopback combo codec server. This section provides information regarding this sample application, the build procedure for it, and the necessary configuration.

The sample application provided with this release is:

loopback - This is a console based application which basically accepts a user-specified file [YUV] and performs an encode-decode cycle on the input file for a specified number of frames. The application also allows the user to specify the output YUV file into which the decoded data is stored back. Users can verify if the encode-decode cycle was valid by manually comparing the input YUV and the output YUV File.

4.10.1 Loopback Sample

The loopback sample is an application used for testing the H.264 loopback codec combo. The application accepts a user-specified YUV file; it also expects the user to specify the dimensions of each frame in the input YUV file and also the number of frames for which the encode-decode cycle is to be repeated.

The application then reads individual frames from the input YUV file, encodes each frame and decodes the encoded data back, and writes the output of the H.264 decoder into another output YUV file.

This application is based on the decode combo Monolithic DLL and uses the Windows PB tools for compilation.

The sample appliction contains the following source files:

- app.c - Source file which contains the WinMain() routine. It also contains wrapper routines to invoke the H.264 encoder and decoder VISA routines.
- App.h - Header file for the routines defined in app.c
- makefile - PB build makefile
- sources - PB build file which contains the list of all the source files to be built and the include, library paths

Follow the procedure below for building the Avplay application:

- Build a regular mobile hand-held configuration workspace image in the Platform Builder IDE
- Using the Build OS>Open Release Directory command; invoke a command-shell window from PB.
- Update the paths environment variable to include the <BIOS_INSTALL_DIR> \xdctools folder in the standard list of paths.
  ```
  path=%path%;%BIOS_INSTALL_DIR%\xdctools;
  ```
- Update the XDC_PATH variable to include the imports folder in the standard list of paths.

```
set XDCPATH=$(XDC_PATH);C:\wince500\platform\davinci\src\ce-d16\imports
```

- Navigate to the directory <CE_EXAMPLES_INSTALL_DIR>/apps/loopback (where the sources is) and type

  ```
  build -c
  ```

These steps build the loopbackApp.exe under the obj/ARMV4I/ [retail | debug] depending on the build configuration selected.

4.10.2 Loopback Sample Integration

This section provides This section provides information on integrating the loopback sample application into the Windows CE OS Image. Please ensure that the loopback application is built as per the instructions specified in Section 4.10.1.

Follow this procedure to integrate the h.264 loopback application into the Windows CE OS image.

1. Build a regular mobile hand-held configuration Workspace Image in the Platform Builder IDE.
2. Copy the <CE_EXAMPLES_INSTALL_DIR>/apps/h264encdec/cedll/dll/h264encdec.dll into the project’s release folder.
3. Update the project.bib file of the Project with the settings below. The settings must be included in the FILES section of the project.bib below

   ```
   H264encdec.dll   $(_FLATRELEASEDIR)\ H264encdec.dll   NK SHM
   ```

   This ensures that the h264encdec.dll monolithic DLL is integrated into the Windows CE OS image.
4. Perform a MAKEIMG command from the Platform builder IDE.
5. Download the Windows CE OS Image onto the target device.
6. Using the Remote tools, copy the <CE_EXAMPLES_INSTALL_DIR>/servers/loopback/loopbackCombo.x64P into the root folder of the target device.
7. Copy the loopbackApp.exe from apps/loopback/obj/ARMV4I/retail folder into the root folder of the target device.
8. Copy a Raw YUV File into the root folder of the target device.
9. Invoke the loopbackApp.exe using the syntax below:

   ```
   s loopbackApp i <Input Raw YUV File > -w <Width of each frame in pixels>-h
   <height of each frame in pixels> -n <number of frames to encode-decode>
   ```
10. The sample application generates an equivalent output YUV file named “out.yuv”. Please check for this file once the program completes the execution.
    The out.yuv file should be identical to the input raw YUV file specified by the user.

5 CMEM Settings Customization

This section provides information on customizing the settings of the contiguous cemory (CMEM) allocator used by the codec engine.

CMEM settings may be modified in the following scenarios:

- The user wants to extend the existing codec engine sample applications with a different set of memory configuration.
- The user has his/her own set of sample applications utilizing the codec engine.

5.1 Default CMCM Settings

This section provides information regarding the default CMEM settings as given in the codec engine binary tree. The default settings are given below:

Sl. No	Memory Range	Remarks
1	0x87800000 – 0x88400000	Memory Size of 0xC00000 which is 12MB of which only 10 MB is currently mapped into Buffer Pools
2	Buffer Pool 0 Eight 810KB Buffers	8 x 810KB = 6635520 bytes which is 6.328 MB The number 810 KB is arrived by taking the maximum frame size for a PAL D1 Resolution frame buffer which is (720 x 576 x 2) bytes This block is used by most of the video encode/decode sample apps
3	Buffer Pool 1 One 50.7KB Buffer	1 x 50.78 = 50.78KB
4	Buffer Pool 2 Four 1MB Buffers	4 x 1 MB = 4MB

The <CE_INSTALL_DIR>/packages/ti/sdo/ce/winceutils/cmem/ce_dll.reg registry file contains the CMEM settings. Whenever any of the codec engine demo applications are selected into the OS image, the above default settings will be utilized by executing the sample/client applications.

5.2 CMEM Customization

This section provides information on updating the ce_dll.reg for a different set of memory configuration. As mentioned in the previous section, the ce_dll.reg registry file contains the CMEM memory settings. The following shows a sample ce_dll.reg file contents:

```
 [HKEY_LOCAL_MACHINE\Software\TexasInstruments\CodecEngine]
    "Dll"="ce_dll.dll"
    "PhysMemStart"=dword:87800000
    "PhysMemEnd"=dword:88400000
    "CE_TRACE"=""
    "CE_TRACEFILE"="gt_log.txt"
    "CE_TRACEFILEFLAGS"="a"
    "NumPools"=dword:3

; First Pool table
; Creation of Twenty 4KB Buffers

[HKEY_LOCAL_MACHINE\Software\TexasInstruments\CodecEngine\Pool0]
    "BufferSize"=dword:CA800
    "BufferCount"=dword:8

; Second Pool table
; Creation of Ten 128KB Buffers
[HKEY_LOCAL_MACHINE\Software\TexasInstruments\CodecEngine\Pool1]
    "BufferSize"=dword:CB20
    "BufferCount"=dword:1

; Third Pool table
; Creation of Two 1024KB Buffers
[HKEY_LOCAL_MACHINE\Software\TexasInstruments\CodecEngine\Pool2]
    "BufferSize"=dword:100000
    "BufferCount"=dword:4
```

Update the value under the PhysMemEnd registry key if the memory range needs to be extended beyond the current value.

Note: There are certain sections of the DDR memory reserved for the DSP dynamic heap allocations. This memory section starts from 0x8B800000 DDR address for the DVEVM board. Please ensure that the PhysMemEnd does not overlap with this section.

For modifying the buffer sizes of any pool, update the BufferSize and the BufferCount registry keys in the ce_dll.reg file.

Once the changes are performed in the ce_dll.reg file, regenerate the Windows CE OS image which contains the codec engine components. The resulting Windows CE OS image will contain the updated CMEM settings.

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