TMS320DM644x DVEVM Windows CE v5.0 BSP

User's Guide

Literature Number: SPRUEV9
March 2007

Texas Instruments

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 DM644x DVEVM.

Purpose and Scope

This document provides information about the release contents of Windows CE 5.0 BSP for DaVinci-based DM644x DVEVM. The document illustrates various components that are part of this release, the procedure to install this release on to the host system, and lists the limitations of this release.

This document assumes that the user has access to Microsoft® Platform Builder 5.0 and is familiar with its usage.

Note: This release of BSP is tested on the DaVinci Rev-D EVM board.

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

Table 1. Terms, Acronyms and Descriptions

Number	Term	Description
1	APIs	Application Programmer Interface
2	ATA	AT Attachment
3	BLCOMMON	Boot Loader Common Architecture
4	BSP	Board Support Package
5	CETK	Windows CE Test Kit
6	CF	Compact Flash
7	DAC	Digital to Analog Converter
8	DHCP	Dynamic Host Configuration Protocol
9	DLL	Dynamic Link Library
10	DMA	Direct Memory Access
11	DMSoC	Digital Media System-on-Chip
12	EDMA	Enhanced Direct Memory Access Controller
13	EVM	Evaluation Module
14	FAT	File Allocation Table
15	GDI	Graphic Device Interface (Windows CE Display driver model)
16	GUI	Graphical User Interface
17	GWES	Graphics Windows Events Subsystem
18	I2C	Inter-Integrated Circuit
19	IDE	Integrated Development Environment

Table 1. Terms, Acronyms and Descriptions (continued)

Number	Term	Description
20	IOCTL	Input Output Control
21	IST	Interrupt Service Thread
22	MDD	Model Device Driver
23	MMU	Memory Management Unit
24	NTSC	National Television System Committee
25	OAL	OEM Adaptation Layer
26	OEM	Original Equipment Manufacturer
27	OSD	On-Screen Display
28	PAL	Phase-Alternating Line (Television standard)
29	PDD	Platform Device Driver
30	PIO	Programmed Input Output
31	PQ	Production Quality
32	RGB	Red-Green-Blue Digital Color Format
33	RTC	Real Time Clock
34	SDRAM	Synchronous Dynamic Random Access Memory
35	UDMA	Ultra Direct Memory Access
36	VPBE	Video Processing Back End
37	VPSS	Video Processing Sub System (Module on DMSoC)

Related Documentation from Texas Instruments

The following documents describe the BSP for DaVinci-based DM644x DVEVM.

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 DM644x 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.

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.

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

Windows, Microsoft are registered trademarks of Microsoft Corporation in the United States and/or other countries.

TMS320DM644x DVEVM Windows CE v5.0 BSP

1 Components of BSP

1.1 Bootloader

Please refer to the *TMS320DM644x DVEVM Windows CE v5.0 BSP Bootloader Users Guide* (SPRUEW1) for more information on the bootloader features.

1.2 OEM Adaptation Layer

An OEM adaptation layer (OAL) is an abstraction layer of code that logically resides between kernel and the hardware device. Physically, the OAL is linked with the kernel libraries to create the kernel executable file. The OAL facilitates communication between the operating system (OS) and the target device and includes code to handle interrupts, timers, power management, bus abstraction; generic I/O control codes (IOCTLs), and so on.

The OAL is developed using the Production Quality OAL model of Windows CE 5.0. The PQOAL model provides us with greater level of software componentization through the usage of software libraries, standard file naming conventions.

The production-quality OAL provides the following improvements over the previous OAL model:

- A common set of processor-specific components
- OAL software components
- A standard directory structure
- Conventions for BSP development

The OAL libraries are a collection of functional, static libraries that are assembled together in a modular approach, to create an OAL. The individual libraries conform to a set of APIs common across all CPU architectures. The hardware library is organized and implemented in a consistent fashion across platforms.

The OAL includes the code for the following:

- Initialization of CPU
- Initialization of OS Timer clock and RTC
- Initialization of MMU and cache
- Initialization of interrupt controller
- APIs to program/ access the interrupt controller
- Generic IO control functionality for the OEM supplied information

1.3 Drivers

This release of BSP supports the following drivers:

- I2C driver
- Serial driver
- EDMA APIs
- Audio driver
- VPBE driver ith DirectDraw extensions

- IR/Keypad driver
- Block-device driver for ATA-based hard disk and compact flash disk
- SPI driver
- DSP/BIOS link
- NDIS driver
- VPFE controller driver
- SD host controller driver
- NAND flash media driver
- USB function controller driver
- USB host controller driver

1.4 Codec Engine

This release also contains the codec engine binary build component. The following watermarked codecs are included in binary form with this BSP.

- WMA9 decoder
- MP3 decoder
- AAC decoder
- G.711 decoder and encoder
- H.264 decoder and encoder
- WMV9 decoder
- MPEG2 decoder
- MPEG4 decoder and encoder

Since these codecs are watermarked, they exhibit the following characteristics:

- All the audio decoder would introduce a periodic sine tone (audible as a beep) in the playback sequence.
- All the video decoders would introduce the TI Logo on the right-top corner of the screen.
- All the video encoders would stop encoding after 10 minutes of encoding.

1.5 DirectShow Filters

This release of the BSP supports the DirectShow transform filters for following decoders:

- WMA9 decode filter
- WMV9 decode filter
- G.711 decode filter
- MP3 decode filter
- AAC decode filter
- MPEG2 decode filter
- H.264 decode filter
- MPEG4 decode filter

The following parsers filters are also made available with this BSP:

- AAC parser filter
- AVI source filter

1.6 Catalog Components

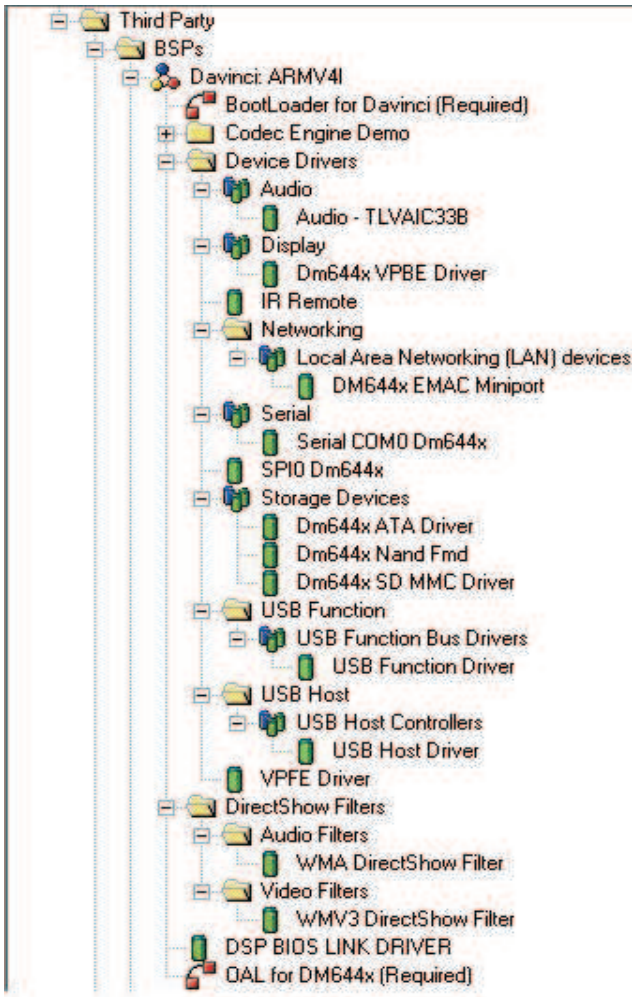


Figure 1. Catalog Entries of the BSP

2 Installation

Section 9 lists the system requirements for the installation of this release.

2.1 3.1 Installation Using the Zip Archive

This section provides the installation procedure for installing the BSP using the zip archive of BSP source code.

The following steps illustrate the procedure to install the BSP and making the BSP part of the Platform Builder IDE.

1. Extract the zip archive to $(_WINCEROOT)\PLATFORM.
2. Ensure that the $(_WINCEROOT)\PLATFORM\DAVINCI directory is created.
3. Open the DAVINCI.cec file using the CEC editor (provided by Microsoft Platform builder IDE). This file is located in the path: $(_WINCEROOT)\PLATFORM\DAVINCI
4. In the CEC editor, select the ***catalog→Add to catalog*** command.
5. In the catalog window of Platform Builder, refresh the catalog list.

Note: If the BSP was already imported to the catalog, then select Catalog>Update in Catalog command during Step 1.

2.2 Distribution Archive

The directories shown in Table 2, under the path:

$(_WINCEROOT)\Platform

are created as the part of installation procedure.

Table 2. Distribution Archive

Directory	Description
\DAVINCI\SRC\APPS	This folder has the custom apps used for testing the following: • Audio record application • Audio playback application • EDMA memory to memory transfer application • SPI test application • I2C test application • Interrupt latency test • Remote key test application • VPFE camera test application • Sample applications for DirectDraw overlay and alpha-blend • Codec engine demo applications • Sample application for codec engine using VISA APIs for encode only, decode only, and encode and decode.
\DAVINCI\SRC\BOOTLOADER\	Contains the source code necessary to create a standalone bootloader customized for the DAVINCI based DVEVM platform.
\DAVINCI\SRC\COMMON\	This folder contains the code being shared between the bootloader and the OAL. It includes assembly file for DaVinci ARM Initialization. These routines are required for implementing the Windows CE bootloader/OAL startup on the DaVinci EVM board. This directory also includes the sources for boot-part library and the Flash media driver for NAND Flash.
\DAVINCI\SRC\DM644x\	Contains the DAVINCI chip specific code libraries, which are statically linked with the OAL to produce the kernel executable. Additionally, it also contains the code for drivers that are DMSoC specific. This includes driver for UART, SPI, VPBE, EDMA and NDIS.
\DAVINCI\SRC\DRIVERS	This directory contains the source code for all the drivers integrated with the BSP for M2 Release; drivers for ATA/CF, DSPBIOSLINK, IR-remote, USB function controller, USB host controller, SDIO host controller, VPFE and Wavedev.
\DAVINCI\SRC\DShowFilters	This directory contains the source code for the transform filter implementation for the decoders bundled in this BSP. This transform filter wraps around the VISA. This directory also contains additional source/parser filters that are necessary for DirectShow framework to use these transform filters.
\DAVINCI\SRC\INC\	Contains platform specific header files.
\DAVINCI\SRC\KERNEL\	This directory contains all files and source code necessary to implement the OAL. The OAL is statically linked to the Windows CE kernel provided by Microsoft.
\DAVINCI\FILES\	Contains files providing registry settings (.reg files), configuration settings (.bib files), database information (.db files), and miscellaneous files.
\DAVINCI\TOOLS\	This directory contains the utility program for generating the header file from the binary file. This includes CCS projects for flashing the boot-loader into the NOR Flash and the CCS code for building the user bootloader.

3 BSP Configuration Files

When using the DaVinci adaptation for development, you may be required to change your platform files. The following file summaries are provided as general information. Any specific modifications that you require are detailed in later chapters of this document.

3.1 DAVINCI.bat

This file is used to set environment variables for the Windows CE build window environment. Like Setenv.bat, when WinCe.bat opens a command prompt build window, it calls DAVINCI.bat. This file is located in the $(_WINCEROOT)\PLATFORM\DAVINCI\ directory. Unlike Setenv.bat, this file is not developer-specific, but common to every developer using the board support package.

This file is relevant for both DOS command line build operations and Platform Builder IDE build operations. When called from a command line, this file specifically controls the build environment. When called from the Platform Builder IDE, this file initializes the environment.

Note: Settings made in DAVINCI.bat can/will be overridden by settings in the Platform Builder IDE.

3.2 DAVINCI.cec

DAVINCI.cec provides information that is integrated with the existing Platform Builder catalog and is used to provide additional control of the working environment within the Platform Builder IDE. This file defines the individual features and components that are used by the IDE to create a customized Windows CE kernel that can be downloaded to the target device.

4 Building BSP

This section describes the procedure to build sample workspace using the BSP.

4.1 Windows CE OS Image Build Process

This section highlights the various steps involved in building a sample Windows CE Image using the Windows CE 5.0 BSP for DaVinci. This section describes the procedure to build the image (of Mobile Handheld configuration) including various drivers supported in this release of BSP.

4.1.1 Steps Involved in Creating a Mobile Hand Held Image Configuration

The following steps illustrate procedure to build a sample Mobile Hand-Held configuration image.

1. Create a new platform by selecting the ***File→New Platform*** command. This opens the New Platform Wizard. Select the Next button.
2. Specify the name and the path for the workspace upon which the Mobile Handheld configuration is created.

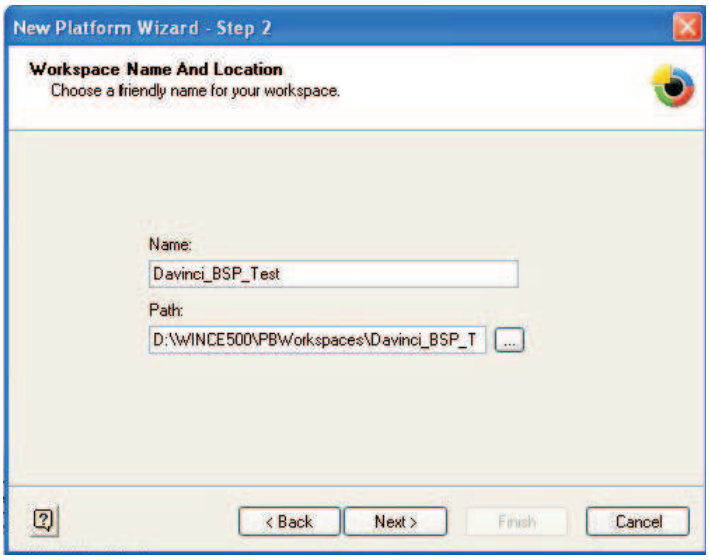


Figure 2. Workspace Name and Path Selection in Platform Wizard

3. Press *Next* after entering the name of the workspace and selecting the path for the same.
4. Select the BSP upon which the mobile handheld configuration is based. Select DaVinci BSP for building the Image.
5. Press Next to continue to the next step

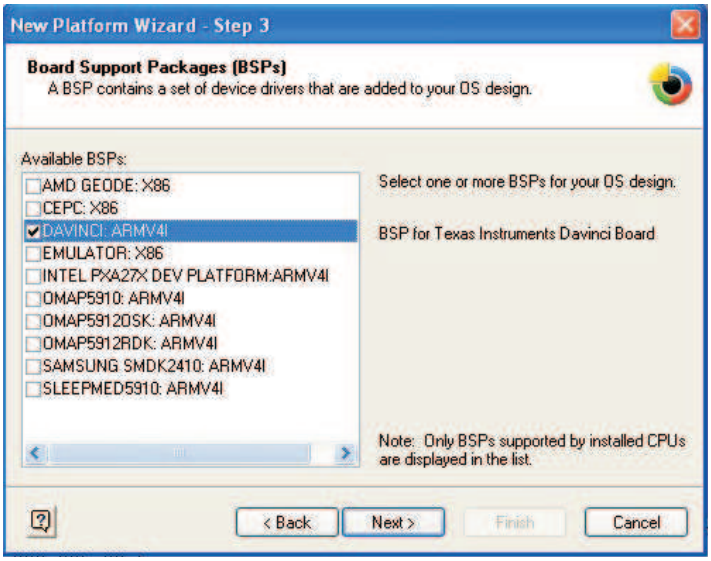


Figure 3. BSP Selection in Platform Wizard

6. Select the Design Template upon which the current workspace will be based. Select Mobile Handheld Image Configuration as shown in Figure 4.

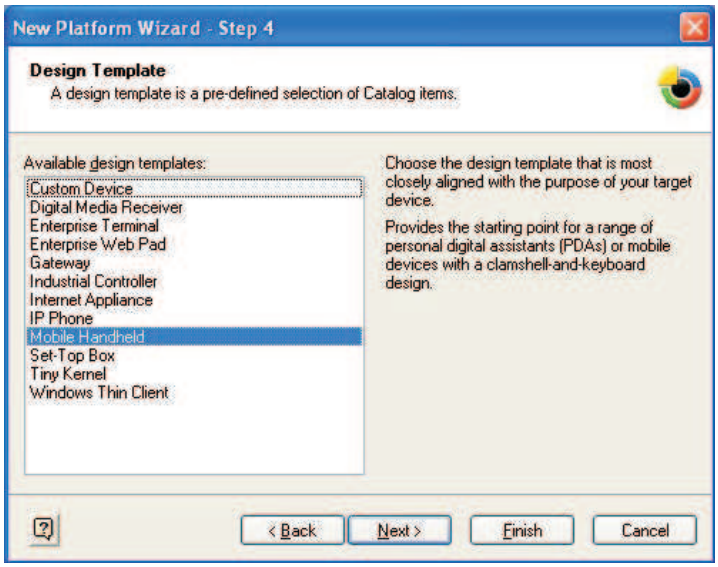


Figure 4. Design Template Selection in Platform Wizard

7. Click on the Next Button.
8. Configure the components in the workspace that is created. Select the *Windows Media Audio/MP3* only. De-select other components.

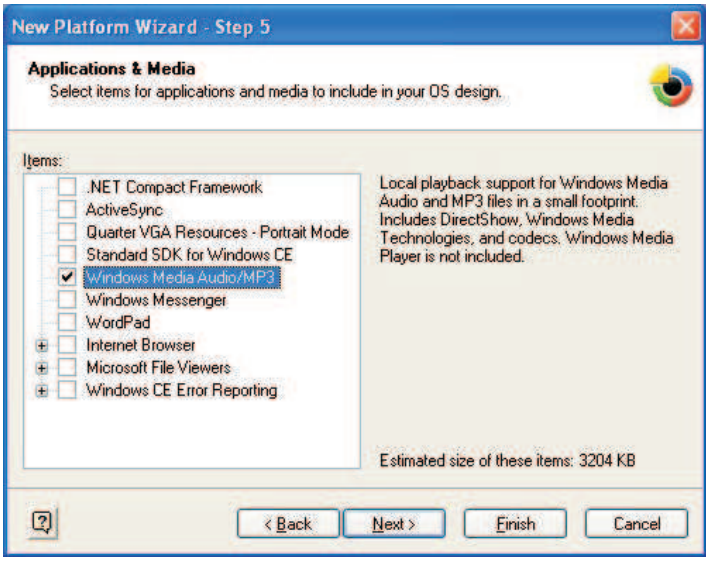


Figure 5. Application and Media Components Selection

9. Press Next.
10. Configure the Network components in the Workspace. Select the *Wired Local Area Network* components. Please select the *TCP/IP Support* component.

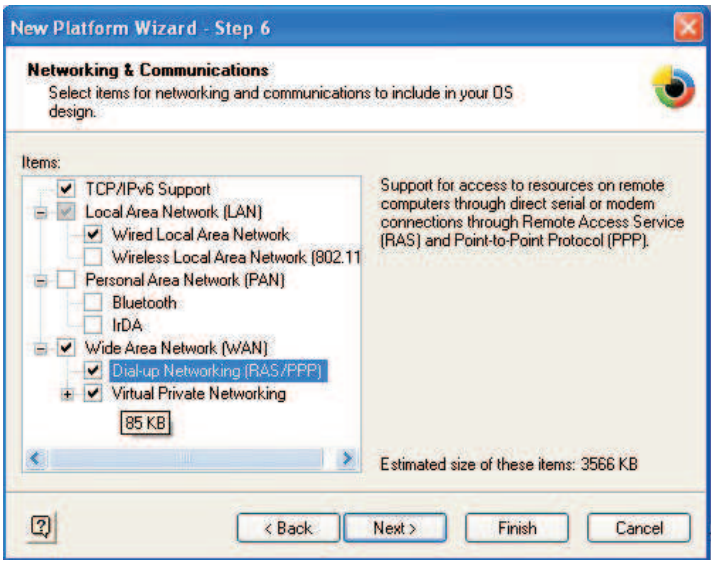


Figure 6. Networking and Communications Configuration

11. De-select other components and press Next to continue to the next step.
12. Click on Finish when the platform Wizard displays the Finish Dialog box. Now the Workspace creation is complete.

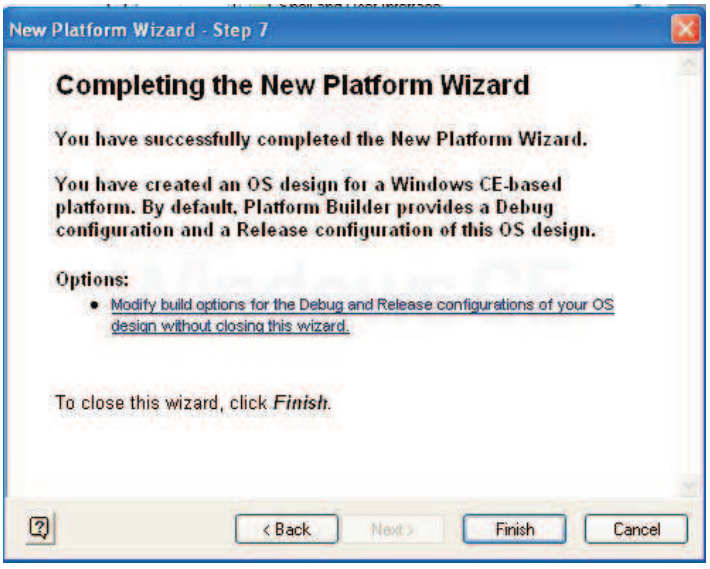


Figure 7. Environment Tab of the Project Settings Dialog Box

13. Before building the platform, select the Build Configuration from the Project Settings Dialog box.
14. Select the Build Configuration as DaVinci: ArmV4I_Release. The Build type is set to Release.
15. In the Build Options tab, select the following options as shown in Figure 8.

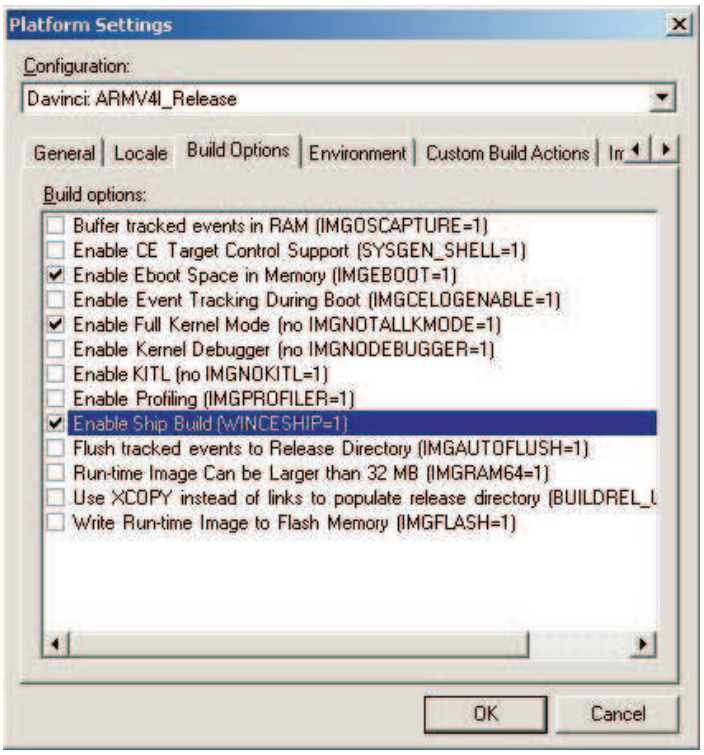


Figure 8. Build Options Tab of the Project Settings Dialog Box

16. Add the necessary DaVinci specific catalog components to the workspace from the catalog. Figure 9 shows the workspace window after including the necessary drivers.

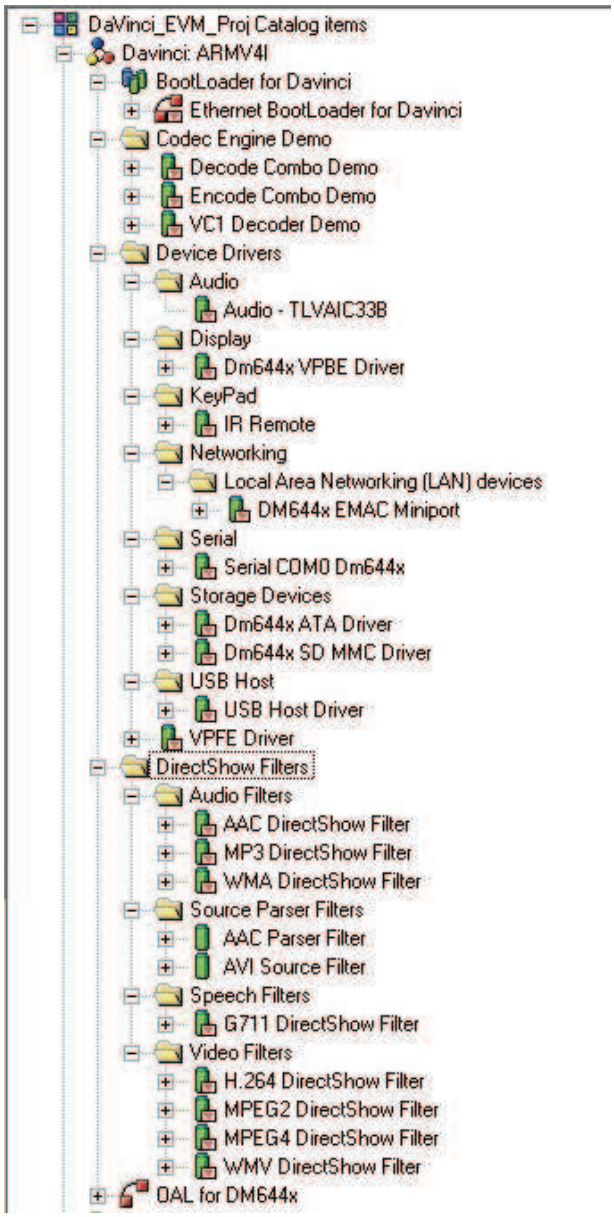


Figure 9. Workspace Window With DaVinci Components

17. Add the necessary device driver support as shown in Figure 10.

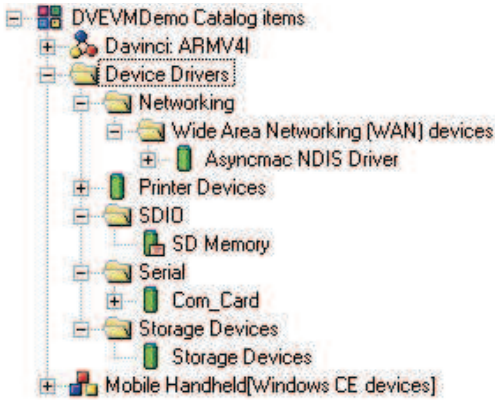


Figure 10. Workspace Window Device Drivers

18. Add the necessary Core OS components. Figure 11 shows some sample components that may be chosen.

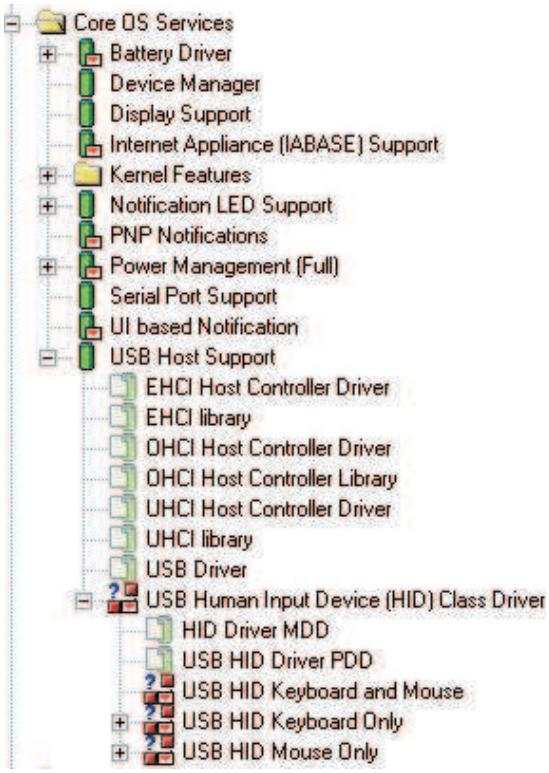


Figure 11. Workspace Window Core OS Components

19. Add the desired multi-media components. Figure 12 shows some sample components that may be chosen.

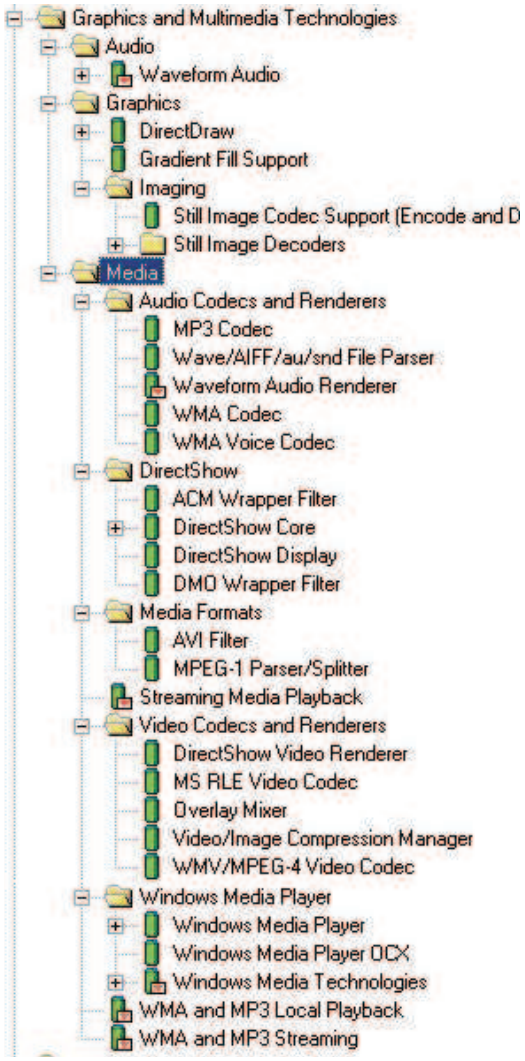


Figure 12. Workspace Window Multi-media Components

20. Select the components *DirectDraw*, *Fat File System*, *Partition Driver* and *Storage Manager Control Panel Applet* to include in the OS configuration.
21. Select ***Build OS***→*Sysgen* in order to build a release Image.
    Once the kernel image is built and the nk.bin file is created in the $(_FLATRELEASEDIR), we are ready to download the image on the board.
22. Power-up the DaVinci board and press the Enter key to enter the command mode. Type *exit* on the serial console.
23. Use the ***Target***→***Connectivity Options*** command to configure the remote connection type used to download the image.
24. In the Connectivity Options Dialog box, first add a new Device named Davinci#### and then select the Device Name.
25. For adding a new Device, click on the *Add Device* text shown on the Connectivity Options dialog box.

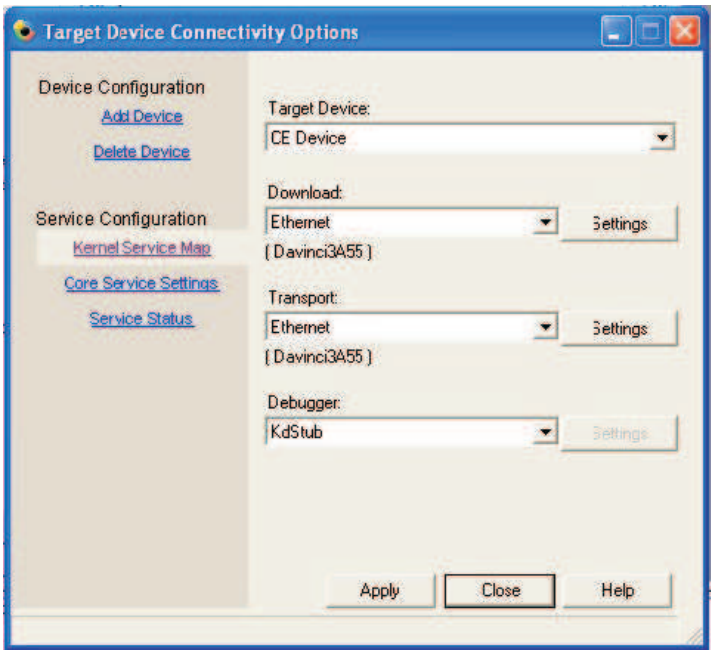


Figure 13. Target Device Connectivity Options Dialog Box

26. Once the new Device name is entered, click on the Apply button. Now configure the Device ID for this new Device.
27. Click on the *Kernel Service Map* and under the Download option, select *Ethernet* and click on the Settings Command Button.
    The boot loader would be sending BOOTME packets to a pre-determined broadcast IP Address. The name of the board should appear on the list of devices.
28. Once the Device Name is selected, click on the OK Button in the dialog box.
29. Click on Apply in the main Connectivity Options Dialog box. The download of the Image can start now.
30. Select ***Target→Attach Device*** or use the ***CTRL+SHIFT+D*** keyboard shortcut to start the download of the OS Image.
31. The download progress bar appears and starts downloading the kernel image on to the board.

5 Memory Mapping

For information on the memory map of the DMSoC, Please refer to the *TMS320DM6446 Digital Media System-on-Chip Data Manual* (SPRS283).

The following blocks of memory in SDRAM are reserved by the BSP for special purposes.

Table 3. Memory Map of OS Image

Sl. No	Start Address	Size (In Bytes)	Description
1	0x80000000	0x0001000	SDRAM region reserved for communication between Bootloader and OS image.
2	0x80001000	0x00006000	SDRAM region reserved for EMAC buffer
3	0x80007000	0x00A00000	NK kernel section in the image built for RAM
4	0x80A07000	0x05500000	RAM area reserved
5	0x85F10000	0x000E0000	SDRAM region for reserved for EMAC buffer for use in the NDIS driver
6	0x86000000	0x00500000	Region reserved for display frame-buffer
7	0x86500000	0x00020000	Region reserved for ATA driver

Table 3. Memory Map of OS Image (continued)

Sl. No	Start Address	Size (In Bytes)	Description
8	0x86600000	0x00080000	Region reserved for audio DMA buffer
9	0x86680000	0x00080000	Region reserved for SD host controller driver DMA buffer.
10	0x86700000	0x00100000	Region reserved for USB function controller driver DMA buffer.
11	0x86800000	0x00500000	Region reserved for camera (VPFE) DMA buffer.
12	0x87800000	0x00800000	Region reserved for CMEM area.
13	0x8B800000	0x04200000	SDRAM region reserved for DSP heap area for dynamic memory allocation.
14	0x8FA00000	0x00400000	SDRAM region reserved for DSP program memory
15	0x8FE00000	0x00100000	Region reserved for DSP Link memory table 0
16	0x8FF00000	0x00000080	Region reserved for DSP Link memory table 1
17	0x8FF00080	0x000FFF80	Region reserved for DSP Link memory table 2

6 Drivers

This section describes the features of the all the drivers included in this release, and identifies their source code location and CEC entries. The various build options and the registry keys are also described for every driver in their corresponding sections.

6.1 I2C Driver

Table 4. Terms and Acronyms

Number	Term	Description
1	CETK	Windows CE test kit
2	I2C	Inter-Integrated circuit
3	OAL	OEM adaptation layer
4	RTC	Real-time clock
5	APIs	Application programmer interfaces

6.1.1 Overview and Features

The I2C module is provided as a library, instead of a stream interface driver. As per Windows CE kernel architecture, RTC routines are required to be present in the OAL. Since the RTC is accessed from I2C, I2C API's are provided in OAL and the drivers can access these routines using the *KernelIoControl* routine.

6.1.2 Source Code Path

The source code for this driver implementation is located in the path:

$(_WINCEROOT)\PLATFORM\DAVINCI\SRC\DM644X\I2c

The sources file in this path builds the Library named Dm644xI2c.lib in the path:

$(WINCEROOT)\PLATFORM\DAVINCI\LIB\ARMV4I\$(WINCEDEBUG)

6.1.3 CEC Entries

Since I2C library is part of the OAL, there is no catalog entry corresponding to the I2C library. This library gets built along with the OAL by default.

6.1.4 Build Options

The default sources and the makefile is located in the path illustrated in Section 6.1.2. There are no compile time options provided to build this module. This library is built by default while building the OAL.

6.1.5 Registry File

Since this is not a driver module, there is no registry entry for I2C.

6.1.6 Driver Testing

There are no CETK tests available for the I2C. An application is provided in the *$(_WINCEROOT)\PLATFORM\DAVINCI\SRC\APPS\I2CTEST* folder to test the I2C routines. This application must be built from the command prompt. This application runs in an infinite loop and reads the current RTC value for every second and displays in the output window of the Platform Builder. To run the application, give *s I2cTestApp.exe* on CE target shell or go to ***Target→Run Programs*** and select the I2cTestApp.exe and run it.

6.2 Serial Driver

6.2.1 Overview and Features

The serial driver for this platform is a Window CE stream interface driver dynamic link library (Dm644xUart.dll), which is implemented using Windows CE driver model architectures MDD and PDD:

- Model device driver layer (MDD): a library (COM_MDD2), which draws from code under the OAK\DRIVERS tree.
- Platform-dependent driver layer (PDD): a layer implemented specifically for the DAVINCI, which is linked to the MDD library to create the driver DLL.

The following sections describe serial driver components and their functions.

The device manager uses the registry key, [HKEY_LOCAL_MACHINE \Drivers\BuiltIn\Serial] to load the driver, Dm644xUart.DLL.

As a streams model driver, the prefix registry entry under [HKEY_LOCAL_MACHINE \Drivers\BuiltIn\Serialn] defines the prefix to be used with all exported routines. The device manager uses this prefix to create the routine name, COM_Init (), and calls it to let the driver perform its specific initialization. This routine, along with all other high-level COM_xxx () routines, are located in the Microsoft-provided MDD layer. This driver supports the following features:

- Uses FIFO for transmission and reception of data
- Flow control for UART2
- Support for configuring the baud rates and configurations, parity, number of stop bits and data bits

6.2.2 Source Code Path

The source code for this driver implementation is located in the path:

$(_WINCEROOT)\PLATFORM\DAVINCI\SRC\DM644X\Serial

The sources file in this path builds the DLL named Dm644xUart.dll in the path:

$(WINCEROOT)\PLATFORM\DAVINCI\TARGET\ARMV4I\$(WINCEDEBUG)

6.2.3 CEC Entries

This driver is identified as *Serial COM0 Dm644x* under the path Catalog → Third Party → BSPs → Davinci:ARMV4I → Device Drivers → Serial → Serial COM0 Dm644x.

Including this component into the OS design ensures that the serial driver is built while building the OS image.

The following systen variables is defined when this component is included:

- BSP_USE_UART=1

6.2.4 Build Options

The default sources and the makefile are located in the path illustrated in Section 6.2.2. This sources file enables the serial driver to configure the UART in following manner:

- By default UART0 is selected by the build.
- Any other UART instance like UART1, UART2 will not be configured due to pin multiplexing issues.

However, this implementation of the driver supports all the UART (0,1,2) instances available.

6.2.5 Registry File

This driver implementation expects the configuration information to be available in the registry. Table 5 identifies the various keys corresponding to this driver and their significance. The sample registry file for configuring the serial driver is located in the path:

$(_WINCEROOT)\PLATFORM\DAVINCI\SRC\DM644x\serial\Driver\Serial1.REG

Table 5. Registry Keys for Serial Driver

Sl. No	Key Name	Value Type	Current Value	Description
1	Prefix	STRING	COM	Specifies the device prefix
2	Dll	STRING	Dm644xUart.dll	Specifies the Dll name to be loaded
3	Order	DWORD	0	Order as per which the driver needs to be loaded
4	Index	DWORD	1	Index of the driver
5	DeviceArrayIndex	DWORD	0	Hardware index used by the driver

6.2.6 Driver Testing

This implementation of the driver is tested using the serial port driver test cases provided by the CETK. All the serial port test cases are passing.

6.3 EDMA APIs

6.3.1 Overview and Features

The EDMA is implemented as a custom DLL named DM644xEDMA.dll, which is loaded and instantiated by the Windows CE system process, DEVICE.EXE. The DM644xEDMA.dll is based on the required PAL SYS EDMA API's (provided by TI) which are exported in the DM644xEDMA.DEF file. The signature and syntax for the EDMA PAL API's are maintained consistent (with the PAL SYS EDMA) with wrapper functions added to set up the EDMA controller and interrupt handling. The interrupt service thread is handled in the DM644xEDMA.dll, which takes care of setting the corresponding events based on the interrupt fired from the EDMA controller.

6.3.2 Source Code Path

The source code path for the EDMA is under $(_WINCEROOT)\Platform\DAVINCI\DM644x\EDMA. The DLL name of the EDMA driver is DM644xEdma.dll and the same would be placed under $(_WINCEROOT)\PLATFORM\DAVINCI\target\ARMV4I\$(WINCEDEBUG). The dependent libraries for EDMA driver are coredll.lib and ceddk.lib.

6.3.3 CEC Entries

There is no CEC entry for EDMA.

6.3.4 Build Options

There are no build options for EDMA.

6.3.5 Registry File

The EDMA driver has a registry entry as given in Table 6.

Table 6. Registry for EDMA Driver

Sl. No	Key Name	Value Type	Current Value	Description
1	Dll	STRING	DM644xEdma.dll	Specifies the Dll name to be loaded.
2	Index	DWORD	1	Specifies the index value.
3	Order	DWORD	1	Order as per which the driver needs to be loaded.
4	Flags	DWORD	8	DEVFLAGS_NAKEDENTRIES

6.3.6 Driver Testing

The EDMA can also be tested using a custom application which performs a memory-to-memory transfer using the PAL EDMA APIs. The application also supports channel linking and tests all the APIs required for the DMA transfer to be successful. The sources of the same is located at $(_WINCEROOT)\PLATFORM\DAVINCI\SRC\APPS\EDMATEST. The application edmatest.exe must be executed from the Run programs options provided in Platform Builder IDE.

6.4 Audio Driver

6.4.1 Overview and Features

The audio driver is implemented using Windows CE Wavedev driver model architectures involving MDD and PDD and includes the following features:

- Model device driver layer library for Wavedev (Wavemdd.lib) (MDD): provides the stream interface driver (with the WAVE MANAGER specific IOCTLs) abstraction for use by Wave API manager. The default implementation of the WaveMDD is based on the code under the path: $(_WINCEROOT)\PUBLIC\COMMON\OAK\DRIVERS\WAVEDEV\MDD.
  This audio MDD is modified in our Davinci BSP to support events from the EDMA interrupt service thread (IST).
- Platform-dependent driver layer (PDD): a layer implemented specifically for the DaVinci, which is linked to the MDD library to create the driver DLL. This layer configures the audio serial port (ASP) of DaVinci in I2S mode and configures the AIC33 audio codec.

The audio driver supports Playback and Record for the different sampling rates supported by the AIC33 codec. However, the DaVinci EVM board design restricts the sampling rates for both playback and record to be consistent with each other. Currently the codec is configured as the master and the audio serial port (ASP) as the slave. The driver interfaces with the audio codec present on the DVEVM through I2C. The audio driver supports DMA mode of data transfer and the same is achieved through the EDMA PAL APIs. The base address and the size of the physical SDRAM region being used by audio driver is configurable by means of registry.

The current implementation of the audio driver supports the following modes of operation:

- This driver supports sample rates supported by AIC33 codec – Sample rates of 8 KHz, 11 KHz, 16 KHz, 22 KHz, 32 KHz, 44 KHz, 48 KHz and 96 KHz. Off these, sample rates of 8 KHz, 11 KHz, 22 KHz and 44 KHz have been validated in DVEVM.

- This implementation provides support for configuring the codec in stereo and mono mode (default being stereo mode).
- The audio codec and ASP are being configured to operate in I2S mode of operation.

6.4.2 Source Code Path

The source code path for the audio driver is under $(_WINCEROOT)\Platform\DAVINCI\SRC\Drivers\Wavedev. The DLL name of the audio driver is DM644xWaveDev.dll and the same would be placed under $(_WINCEROOT)\PLATFORM\DAVINCI\target\ARMV4I\$(WINCEDEBUG). The dependent libraries for audio driver are Dm644xI2c.lib, DM644xEDMA.lib and wavemdd.lib.

6.4.3 CEC Entry Details

This driver is identified as *Audio - TLVAIC33B* under the path Catalog → Third Party → BSPs → Davinci:ARMV4I → Device Drivers → Audio → Audio - TLVAIC33B Driver.

Including this component into the OS design ensures the audio driver is built while building the OS image.

The following sysgen variable is defined when this component is included:

- BSP_AUDIO =1

6.4.4 Build Options

This release of the audio driver has build options for the Mic and line inputs. The complier flag *MIC* supports Mic inputs for recording path and *LINE* supports Line In for recording the files from the desktop PC. The Table 7 gives an explanation of the same.

Table 7. Build Options for Audio Driver

Sl. No.	Build Flags	Description
1	MIC	This flag sets the audio driver to build with MIC input enabled.
2	LINE	This flag sets the audio driver to build with LINE input enabled.

6.4.5 Registry File

The audio driver has a registry entry as given in Table 8.

Table 8. Registry Options for Audio Driver

Sl. No.	Key Name	Value Type	Current Value	Description
1	Prefix	STRING	WAV	Specifies the device prefix for Wavedev driver. Prefix value expected by the MDD.
2	Dll	STRING	DM644xWaveDev.dll	Specifies the Dll name to be loaded.
3	Order	DWORD	3	Order as per which the driver needs to be loaded
4	Index	DWORD	1	Index of the driver
5	AudioDMAAreaBase	DWORD	86600000	Physical address of base of buffer being used by audio driver for DMA purposes. This region must be reserved in the config.bib file and must be consistent.
6	AudioDMAAreaSize	DWORD	00002000	Size of the physical memory kept reserved for audio driver DMA region.

In addition to the registry associated with the audio driver, this registry file also configures the buffer in the software mixer.

Table 9. Audio Driver Registry

Sl. No.	Key Name	Value Type	Current Value	Description
1	Buffers	DWORD	4	Number of buffers to be created at the software mixer.
2	BufferSize	DWORD	1000	Size of each buffer at the software mixer layer.

6.4.6 Driver Testing

The audio driver can be tested by a custom application provided in the BSP and the same is placed under $(_WINCEROOT)\PLATFORM\DAVINCI\Src\APPS\WAVEREC and $(_WINCEROOT)\PLATFORM\DAVINCI\Src\APPS\WAVEPLAY.

The Waverec.exe application supports recording of audio data from a line source or a MIC source based on the complier flag selected during the audio driver build. The Wavplay.exe application supports playback of audio-recorded files. The application can play only *.wav files. Another method used to validate the driver is through CETK testing. The waveform audio driver test cases is used to perform CETK testing on the audio driver.

6.4.6.1 WavRec Test Application

The WavRec.exe records a sample test.wav file for the input parameters given in the application file (user defined). The application user has to populate the necessary information like channels, bitspersample, samplerate and duration. The recorded data is stored on the target memory under the path /windows/. Currently the sample application supports 20 sec of wave data to be recorded. The sources of the same is located under $(_WINCEROOT)\PLATFORM\DAVINCI\SRC\APPS\WAVREC.

6.4.6.2 Wavplay test Application

The wavplay.exe plays out the wave file recorded by the wavrec.exe application. You can also play sample wave files by changing the audio file named under /windows/*.wav. The sources of the same is located under $(_WINCEROOT)\PLATFORM\DAVINCI\SRC\APPS\WAVPLAY.

6.5 VPBE/Display

Table 10. Terms and Acronyms

Number	Term	Description
1	CETK	Windows CE Test Kit
2	DAC	Digital to Analog Converter
3	EDMA	Enhanced Direct Memory Access Controller
4	EVM	Evaluation Module
5	GDI	Graphic Device Interface (Windows CE Display driver model)
6	GUI	Graphical User Interface
7	GWES	Graphics Windows Events Subsystem
8	NTSC	National Television System Committee
9	OSD	On-Screen Display
10	PAL	Phase-Alternating Line (Television standard)
11	RGB	Red-Green-Blue Digital Color Format
12	SDRAM	Synchronous Dynamic Random Access Memory
13	VPBE	Video Processing Back End
14	VPSS	Video Processing Sub System (Module on DMSoC)

6.5.1 Overview and Features

The Windows CE standard graphic libraries (and graphics windows and events subsystem GWES) use this driver to provide the graphical user interface (GUI). This driver provides the standard DDGPE class based DirectDraw compliant display driver supporting one mode of display with support for mouse pointer. It provides support for following type of surfaces.

- RGB 888 surface (on Video Window 0)
- YUV 4:2:2 surface (on Video window 0 and Video Window 1)
- RGB 565 surface (on OSD Window 0)

Note: The YUV 4:2:2 surfaces of destination sizes larger than 630 x 470 pixels are default given on video window 0. The YUV 4:2:2 surfaces of destination sizes smaller than 630 x 470 pixels are rendered on video window 1.

This implementation of the DirectDraw display driver configures the video window 0 in RGB 888 mode and uses this window as GDI primary surface.

The driver also supports the additional features of alpha-blending and the transparency on the OSD window 0. The driver supports the following capabilities.

- ddCaps.dwCaps
  - DDCAPS_BLT
  - DDCAPS_ALIGNSIZEDEST (Destination surface aligned to 16 pixels)
  - DDCAPS_ALIGNSIZESRC (Source surface aligned to 16 pixels)
  - DDCAPS_ALIGNSTRIDE (Stride aligned to 32 bytes)
  - DDCAPS_GDI
  - DDCAPS_OVERLAY
  - DDCAPS_OVERLAYSTRETCH (with minimum stretch factor being 1000 and maximum stretch factor being 4000)
  - DDCAPS_COLORKEY
  - DDCAPS_BLTCOLORFILL
  - DDCAPS_CANCLIP
- ddCaps.dwCaps2
  - DDCAPS2_WIDESURFACES
  - DDCAPS2_COPYFOURCC
- ddCaps.dwCKeyCaps
  - DDCKEYCAPS_SRCOVERLAY (supported only on RGB 565 surfaces with flag DDOVER_KEYSRCOVERRIDE and DDOVER_KEYSRC)
  - DDCKEYCAPS_SRCOVERLAYONEACTIVE (supported only on RGB 565 surfaces)
- ddCaps.dwFXCaps
  - DDFXCAPS_OVERLAYSTRETCHXN (stretch factor of 1000, 2000 & 4000 only)
  - DDFXCAPS_OVERLAYSTRETCHYN (stretch factor of 1000, 2000 & 4000 only)
  - DDFXCAPS_OVERLAYALPHA (Supported only on RGB565 surfaces with flag DDOVER_ALPHASRCCONSTOVERRIDE. Overlay constant bit depth of 4 bits that is being scaled to 3 bits)
- ddCaps.dwFXAlphaCaps
  - DDFXALPHACAPS_OVERLAYALPHAEDGEBLEND
- ddCaps.ddsCaps.dwCaps
  - DDSCAPS_BACKBUFFER
  - DDSCAPS_COMPLEX
  - DDSCAPS_FLIP
  - DDSCAPS_FRONTBUFFER
  - DDSCAPS_OFFSCREENPLAIN

- DDSCAPS_OVERLAY
- DDSCAPS_PALETTE
- DDSCAPS_PRIMARYSURFACE
- DDSCAPS_SYSTEMMEMORY
- DDSCAPS_VIDEOMEMORY
- DDSCAPS_VISIBLE
- DDSCAPS_STANDARDVGAMODE

On the DaVinci EVM, this display driver initializes the VPBE controller with the frame buffer residing in the SDRAM at the pre-defined addresses. The address and the range for this memory are configured using the registry entries. These memory regions are required to be reserved in CONFIG.BIB entries. The default settings reserve 5 Mbytes of SDRAM region as video memory which is being managed using the SurfaceHeap manager. After any request for creating the surface, the driver first tries to allocate memory from this region. If video memory is not available, then the memory from the system memory pool is allocated.

This driver implementation configures the VPBE controller for standard definition analog video encoders. It provides compile time support for configuring the video encoder to provide either NTSC or PAL output.

This implementation of the display driver also provides the mouse pointer support using the hardware cursor windows. However, due to the lack of support for configuring the bit-map for the hardware cursor window, setting the desired pointer shape is not supported.

This display driver advertises itself as the power management aware driver under the class PMCLASS_DISPLAY, and includes two power states:

- D0 (Full On): In this state, the primary display window along with the cursor window is enabled for display. The power to the VPSS and the DAC is enabled in this state.
- D4 (Full OFF): In this state, the primary display window and the cursor is disabled. The power to the VPSS is not disabled (for the reason that VPSS contains both the VPBE and VPFE). However, the power and clock to the DAC and video encoder are disabled.

6.5.2 Source Code Path

The source code for this driver implementation is located in the path:

$(_WINCEROOT)\PLATFORM\DAVINCI\SRC\DM644X\VPBE

These sources build the DLL named Dm644xVpbe.dll in the path:

$(WINCEROOT)\PLATFORM\DAVINCI\TARGET\ARMV4I\$(WINCEDEBUG)

This DLL statically links to the following libraries.

- COREDLL.DLL: for Windows CE system calls
- EMUL.LIB: for emulated BitBlt routines
- GPE.Lib: for graphics primitive engine
- DDGPE.Lib: for methods associated with surface heap management.
- CEDDK.Lib: for Windows CE device driver kit specific system calls.
- DM644xEDMA.Lib: for APIs programming the EDMA engine for accelerated BitBlt operations.
- dxdrvguid.lib: for DirectX driver GUIDs functions.

6.5.3 CEC Entries

This driver is identified as *DM644x VPBE Driver* under the path Catalog → Third Party → BSPs → Davinci:ARMV4I → Device Drivers → Display → DM644x VPBE Driver.

Including this component into the OS design ensures the display driver is built while building the OS image.

Including this component defines the following sysgen variables:

- BSP_DISPLAY_VPBE=1
- BSP_NODISPLAY=

- SYSGEN_DDRAW=1

6.5.4 Build Options

The default sources and the makefile are located in the path illustrated in Section 6.5.2. This sources file enables the display driver to configure the VPBE in following manner.

1. Video encoder is configured to provide NTSC mode output.
2. Video Window 0 is enabled in RGB888 mode to act as the primary display surface.

However, this implementation of driver supports the compile time configuration for following.

- Enabling/disabling the DMA acceleration for BitBlt operations.

Table 11 identifies the list of build flags supported.

Table 11. Build Options for Display Driver

Sl. No.	Build Flags	Description
1	USE_DMA_ACCEL	Defining this flag enables the code implementing the EDMA for BitBlt operations. This would require the EDMA channel being reserved for these operations.
2	NTSC_MODE	Defining this flag enables the code for configuring the video encoder of DaVinci EVM to enable the NTSC mode. If this flag is not defined, the default compilation of the driver would configure the video encoder to enable PAL mode of output.

6.5.5 Registry File

This driver implementation expects the configuration information to be available in the registry. Table 12 identifies the various keys corresponding to this driver and their significance. The sample registry file for configuring the display driver is located in the path:

$(_WINCEROOT)\PLATFORM\DAVINCI\SRC\DM644x\VPBE\VPBE.REG

Table 12. Registry Keys for Display Driver

Sl. No	Key Name	Value Type	Current Value	Description
1	FBPhysicalBase	DWORD	86000000	Physical address of the memory block reserved for video memory. The memory region needs to be reserved in the config.bib files. This key must be present for the driver to load successfully.
2	FBSize	DWORD	00500000	Size of the memory block reserved as video memory. This size must be coherent with the settings made in config.bib. This key must be present for the driver to load successfully.
3	FBBaseX	DWORD	00000090	The X coordinate of the OSD display area to be configured in the DMSoC OSD registers. This key must be present for the driver to load successfully.
4	FBBaseY	DWORD	00000014	The Y coordinate of the OSD display area to be configured in the DMSoC OSD registers. This key must be present for the driver to load successfully.
5	FBWidth	DWORD	000002E0	This key specifies the width of the horizontal line (in number of pixels) of the main display surface. The value for this key must be in such a way that the number of bytes representing one line is always integer multiple of 32. This key must be present for the driver to load successfully.
6	FBHeight	DWORD	000001e0	This key specifies the Height of the screen (in number of lines) of the main display surface. This key must be present for the driver to load successfully.

Table 12. Registry Keys for Display Driver (continued)

Sl. No	Key Name	Value Type	Current Value	Description
7	FBOffsetX	DWORD	00000000	This key specifies the offset of Display surface (Video window 0/1 or OSD window 0) relative to the BASEPX. This key must be present for the driver to load successfully.
8	FBOffsetY	DWORD	00000000	This key specifies the offset of Display surface (Video window 0/1 or OSD window 0) relative to the BASEPY. This key must be present for the driver to load successfully.

6.5.6 Driver Testing

This implementation of the driver is tested using the following CETK test cases:

- All GDI test cases
- All GDI performance test cases
- All DirectDraw test cases

All the GDI test cases are passing with the exception of:

- Alpha blend test case ID 231
- All printer tests cases in the range 1200 to 1205
- DirectDraw overlay stretch test cases 1240 and 1340

Additionally, some of the test cases may fail at specific instances. However, in other iterations, the same test cases pass. This is due to the difference in behavior of the test suites for different random numbers.

The test cases 1100 and 1101 are the test cases for TEXT blit. These test cases are to be skipped in the platforms, which do not have support for keyboard. However, due to the bug in the CETK, these test cases are, by default, executed. As there is no keyboard driver, these test cases fail occasionally.

The DirectDraw CETK test cases 1240 and 1340 are failing due to a bug in the CETK. The test case is not skipping these tests based on the sub-capabilities declared by the driver.

6.5.7 Sample Applications

The following sample applications use the alpha-blend and transparency features supported by the driver. They are located under the path:

($_WINCEROOT)\PLATFORM\DAVINCI\SRC\APPS

- AlphaOnlyBug: - illustrates the usage of the alpha-blend feature of the display driver. On executing this application, the bug appears on the screen (displayed on OSD window 0) at a constant location and the appearance of the entire overlaid surface.
- Trans_AlphaBug: - illustrates the usage of the alpha-blend along with the transparency (color key). On executing this application, the bug moves around on the screen (displayed on OSD window 0). The bug appears with the black background surface that fades off and re-appears over varying location.
- TransMosquito: - illustrates the usage of transparency (colorkey) feature of the display driver. On executing this application, the bug moves around on the screen (displayed on the OSD window 0) without the black background.
- YUV_Overlay: - illustrates the usage of overlay surfaces on the YUV surface.

6.6 IR-Remote Driver

6.6.1 Overview and Features

The IR driver for this platform is a Window CE dynamic-link library (DM644xIR.dll), which is a custom implementation using Windows CE layered driver architectures MDD and PDD:

- Model Device Driver layer (MDD): a library (LayoutManager.lib) which takes care all the hardware independent functionalities
- Platform Dependent Driver layer (PDD): a layer implemented specifically for the DAVINCI, which is linked to the MDD library to create the driver DLL.

The following sections describe IR driver components and their functions.

The IR remote driver gets the data from MSP430 (triggered by the remote commander) and returns the appropriate event and scan-code to the MDD. The MDD further converts the scan codes to virtual scan key codes and posts the message to GWES.

The input system loads the keyboard driver at boot time. When the input system starts, it retrieves the name of the keyboard driver dynamic-link library (DLL) from the [HKEY_LOCAL_MACHINE\Hardware\DeviceMap\KEYBD\Drivername] registry key.

Table 13. IR-Remote Key Mapping

Remote Key	Key Code
Record	Not mapped currently
Stop	VK_CANCEL
PAUSE	VK_PAUSE
REWIND	VK_PRIOR
PLAY	VK_PLAY
FORWARD	VK_NEXT
INPUT	Not mapped currently
0	VK_F12
1	VK_F11
2	Not mapped currently
3	VK_F2
4	VK_F3
5	VK_F5
6	VK_DELETE
7	VK_F10
8	VK_F4
9	VK_LMENU
ENTER	VK_ENTER
PREV.CHAN	VK_BACK
SUBTITLE	VK_ESCAPE
CODE	Not mapped Currently
SLEEP	Not mapped Currently
MENU	Not mapped Currently
INFO/SELECT	VK_LWIN
CHAN+	VK_UP
CHAN-	VK_DOWN
VOLUME+	VK_RIGHT
VOLUME-	VK_LEFT
MUTE	VK_TAB

6.6.2 Source Code Path

The source code for this driver implementation is located in the path:

$(_WINCEROOT)\PLATFORM\DAVINCI\SRC\ Drivers \IR

The sources file in this path builds the DLL named Dm644xIr.dll in the path:

$(WINCEROOT)\PLATFORM\DAVINCI\TARGET\ARMV4I\$(WINCEDEBUG)

6.6.3 CEC Entries

This driver is identified as *IR Remote* under the path Catalog → Third Party → BSPs → Davinci:ARMV4I → Device Drivers → IR Remote.

Including this component into the OS design ensures the IR remote driver is built while building the OS image.

Including this component defines the following sysgen variables:

- BSP_USE_IR = 1

6.6.4 Build Options

The default sources and the makefile are located in the path illustrated in Section 6.6.2. This sources file enables the IR remote driver.

6.6.5 Registry File

This driver implementation expects the configuration information to be available in the registry. Table 14 identifies the various keys corresponding to this driver and their significance. The sample registry file Dm644xIR.REG used for configuring the IR remote driver is located in the path:

$(_WINCEROOT)\PLATFORM\DAVINCI\SRC\DRIVERS\IR

Table 14. Registry Keys for IR Remote Driver

Sl. No	Key Name	Value Type	Current Value	Description
1	Layout File	STRING	Dm644xIr.dll	Specifies the layout to be used
2	Layout File	STRING	US	Specifies the text used by the layout
3	DriverName	DWORD	Dm644xIr.dll	Specifies the driver which needs to be loaded
4	Order	DWORD	0	Specifies the order in which the driver has to be loaded

6.6.6 Driver Testing

This implementation of the driver is tested using a custom sample application. The sample application source is found in the following path:

$(_WINCEROOT)\PLATFORM\DAVINCI\Src\APPS\RemoteKeyTest

This sample application processes IR remote key presses and shows the virtual key code of each key press as a text string on the video display.

6.7 ATA/CF Driver

Table 15. Terms and Acronyms

Number	Term	Description
1	ATA	AT Attachment
2	CETK	Windows CE Test Kit
3	CF	Compact Flash
4	DLL	Dynamic Link Library
5	DMA	Direct Memory Access
6	FAT	File Allocation Table
7	PIO	Programmed Input Output

Table 15. Terms and Acronyms (continued)

Number	Term	Description
8	UDMA	Ultra Direct Memory Access

6.7.1 Overview and Features

The CF/ATA driver is provided as a standard Windows CE block driver and uses the Windows CE standard partition manager. It supports only the FAT file System. This implementation of the ATA driver has support for both the UDMA and multiword DMA modes of operation, which are dynamically selected to the highest mode supported by the device attached. The driver queries the device (during initialization) for the various modes supported by the device.

The same sources (DLL) may be used for both CF and hard disk configured by means of registry. The driver supports a registry key *ForcePIOMode* to make the device operate in PIO mode, even if it supports DMA.

The driver advertises itself as the power management aware driver under the class PMCLASS_BLOCK_DEVICE. It supports the following power states.

- D0 (Full On): In this mode, clocks for CF/ATA Controller and the devices are turned ON.
- D1: In this mode, the device is put into the IDLE state.
- D2: In this mode, the device is put to STANDBY state.
- D4 (Full OFF): In this mode, the clock and power is turned OFF to both CF/ATA Controller and the device

6.7.2 Source Code Path

The source code for this driver implementation is located in the path:

$(_WINCEROOT)\PLATFORM\DAVINCI\Src\Drivers\ATA

These sources build the DLL named Dm644xAta.dll in the path:

$(WINCEROOT)\PLATFORM\DAVINCI\TARGET\ARMV4I\$(WINCEDEBUG)

6.7.3 CEC Entries

This driver is identified as *Dm644x ATA Driver* under the path Catalog → Third Party → BSPs → Davinci:ARMV4I → Device Drivers → Storage Devices → Dm644x ATA Driver.

Including this component into the OS design ensures the ATA driver is built while building the OS image.

Including this component define the following sysgen variables:

- BSP_ATA = 1
- BSP_USE_CF =

Also include the following components in the workspace:

- Catalog → Core OS → Windows CE devices → File Systems and Data Store → Storage Manager → Partition Driver
- Catalog → Core OS → Windows CE devices → File Systems and Data Store → Storage Manager → FAT File System
- Catalog → Core OS → Windows CE device → File Systems and Data Store ↑ Storage Manager → Storage Manager Control Panel Applet

6.7.4 Build Options

The default sources and the makefile is located in the path illustrated in Section 6.5.2.

The default registry settings enable the ATA driver. The environment variable BSP_USE_CF must be set to enable the CF card driver. Setting this environment variable to 1, enables the registry profiles for CF card and hence causes the driver to initialize the CF memory card.

To select the CF driver, open the build release directory command prompt and execute the command *set BSP_USE_CF=1* and execute the command *makeimg*. To get back to the ATA driver, unset the variable using command *set BSP_USE_CF=* and execute *makeimg*. Alternatively, the environment variable can also set using the Platform Builder's *Platform Settings* option.

6.7.5 Registry File

This driver implementation expects the configuration information to be available in the registry. Table 16 identifies the various keys corresponding to this driver and their significance. The sample registry file for configuring the ATA driver is located in the path:

$(_WINCEROOT)\PLATFORM\DAVINCI\SRC\DRIVERS\ATA\ATA.REG

Table 16. Registry Keys for ATA Driver

Sl. No	Key Name	Value Type	Current Value	Description
1	Index	DWORD	1	Used as an index to the device
2	Dll	String	Dm644xAta	Name of the DLL
3	Prefix	String	DSK	3-letter prefix used by the driver
4	Order	DWORD	4	Order in which this dll needs to be loaded.
5	HddDmaAreaBase	DWORD	86500000	Physical address of the SDRAM region reserved for HDD driver for DMA buffers. This must be consistent with the config.bib settings.
6	Profile	String	CF or HDD	Profile to be used by this dll.
7	ForcePIOMode	DWORD	0 or 1	Forces the driver to operate in PIO mode only.
8	Iclass	String	{A4E7EDDA-E575-4252-9D6B-4195D48BB865}.	Interface class
9	Folder	String	Compact Flash" or "Hard Disk	Name of the folder to be displayed
10	DefaultFileSystem	String	FATFS	Indicates the default file system to load
11	AutoMount	DWORD	1	Indicates whether the drive needs to be mounted to boot time
12	AutoPart	DWORD	1	Flag to partitions the store with largest creatable partition automatically
13	AutoFormat	DWORD	1	Formats a store automatically when the store is unformatted
14	FileSystem	String	fatfsd.dll	File system driver currently being used.
15	PartitionDriver	String	mspart.dll	Indicates the default partition driver to load
16	MountFlags	DWORD	0	Flags on how the partition is mounted
17	Flags	DWORD	00000024	Flag used by the file system driver.

6.7.6 Driver Testing

This implementation of the driver is tested using the following CETK test cases:

- Storage device block driver read/write test
- Storage device block driver benchmark test
- Storage device block driver API test
- File system driver test

All the test cases are passing. However, the following observations were made:

- While carrying out the tests for CF memory card, one of the tests [ID 5012] fails while executing all the tests continuously. However, while executing this test case individually, it passes.
- While carrying out the block-device driver performance tests, the size of the storage needs to limited to 64 Mbytes. This limitation is due to the CETK restriction.

6.8 SPI Driver

6.8.1 Overview and Features

The SPI driver for this platform is a Window CE stream interface driver dynamic-link library (Dm644xSpi.dll), which is implemented using Windows CE driver model architectures MDD and PDD:

- MDD: (Model Device Driver layer) a library, which is custom, defined.
- PDD: (Platform Dependent Driver layer) a layer implemented specifically for the DAVINCI, which is linked to the MDD library to create the driver DLL.

The following sections describe SPI driver components and their functions.

The device manager uses the registry key, [HKEY_LOCAL_MACHINE \Drivers\BuiltIn\SPI] to load the driver, Dm644xSpi.DLL.

As a streams model driver, the *prefix* registry entry under [HKEY_LOCAL_MACHINE \Drivers\BuiltIn\SPIn] defines the prefix to be used with all exported routines. The device manager uses this prefix to create the routine name, SPI_Init (), and calls it to let the driver perform its specific initialization. This routine, along with all other high-level SPI_xxx () routines, are located in the SPI MDD layer. This driver supports the following feature:

- Custom IOCTL to configure data format, interrupts, etc.

6.8.2 Source Code Path

The source code for this driver implementation is located in the path:

$(_WINCEROOT)\PLATFORM\DAVINCI\SRC\DM644X\SPI

These sources build the DLL named Dm644xUart.dll in the path:

$(WINCEROOT)\PLATFORM\DAVINCI\TARGET\ARMV4I\$(WINCEDEBUG)

6.8.3 CEC Entries

This driver is identified as *SPI0 Dm644x* under the path Catalog → Third Party → BSPs → Davinci:ARMV4I → Device Drivers → SPI0 DM644x.

Including this component into the OS design ensures the SPI driver is built while building the OS image.

- BSP_USE_SPI=1

6.8.4 Build Options

The default sources and the makefile are located in the path illustrated in Section 6.8.2. This sources file enables the SPI driver to configure the SPI hardware in following manner.

- By default SPI will be configured in 3-pin mode.
- The SPI_USE_4PIN flag can be set for using the SPI in 4-pin mode.

6.8.5 Registry File

This driver implementation expects the configuration information to be available in the registry. Table 17 identifies the various keys corresponding to this driver and their significance. The sample registry file for configuring the SPI driver is located in the path $(_WINCEROOT)\PLATFORM\DAVINCI\SRC\DM644x\SPI\ SpiDrv.reg.

Table 17. Registry Keys for SPI Driver

Sl. No	Key Name	Value Type	Current Value	Description
1	Prefix	STRING	SPI	Specifies the device prefix
2	Prefix	STRING	Dm644xSpi.dll	Specifies the Dll name to be loaded
3	Order	DWORD	0	Order as per which the driver needs to be loaded
4	Index	DWORD	0	Index of the driver
5	DeviceArrayIndex	DWORD	0	Hardware index used by the driver

6.8.6 Driver Testing

This implementation of the driver is tested using the sample application provided. The sample application puts the SPI in loopback mode and does a write and read call.

There are no CETK test suites available for testing the SPI driver. The application used to test the SPI driver can be located at $(_WINCEROOT)\PLATFORM\DAVINCI\SRC\APPS\SPIAPP. This application puts the device in loop back mode, writes data and reads it back. All required configurations are done using custom IOCTL calls. This application needs to be built using the command line build method. To run the application give *s Spitest.exe* on CE target shell or go to Target → Run Programs and select the Spitest.exe and run it. This test application does the following:

- Writes a pre-defined data to the SPI transmit port.
- Reads the SPI port back for any data on the port.
- Compares the data read against the original data written.
- Pints the success or failure message.

6.9 NDIS Miniport Driver

6.9.1 Overview and Features

The Windows CE protocol drivers and NDIS wrapper use this particular miniport driver for providing networking capabilities to the DM644x EVM. This miniport driver implements the media access layer (MAC) sub layer of data link layer according to the OSI model.

The DM644x miniport EMAC driver is being surrounded by the NDIS wrapper and uses the NDIS library functions. This driver code implements the support functions expected by the NDIS wrapper, to use the EMAC Ethernet controller for its functionality. The NDIS wrapper forms the glue between the protocol driver and the network controller.

The driver configures the EMAC controller of the DM644x SoC to enable the channel 0 for its operations. This driver initializes the Intel PHY (on-board connected to DMSoC via MII interface) for its operation. It supports dynamic link status detection and its associated event handling. A dedicated EMACINT is provided for CPU through hardware IRQ 13.

This driver implementation requires a physical memory of size 0x000E0000 to be reserved for its operation. The base address of this physical memory region (reserved for NDIS buffers) may be changed by means of the registry. However, this needs to be synchronized with the value configured in the config.bib file.

6.9.2 Source Code Path

The source code for this driver implementation is located in the path:

$(_WINCEROOT)\PLATFORM\DAVINCI\SRC\DM644X\EMACMiniport

These sources build the DLL named Dm644xEmacMiniport.dll in the path:

$(WINCEROOT)\PLATFORM\DAVINCI\TARGET\ARMV4I\$(WINCEDEBUG)

This DLL statically links to the following libraries:

- NDIS.lib: Microsoft provided NDIS library.
- COREDLL.lib: Basic operating system (OS) module that provides core functionality to other modules.
- CEDDK.lib: For Windows CE device driver kit specific system calls.

6.9.3 CEC Entries

This driver is identified as *DM644x EMAC Miniport* under the path Catalog → Third Party → BSPs → Davinci:ARMV4I → Device Drivers → Networking → Local Area Networking (LAN) devices → DM644x EMAC Miniport.

Including this component into the OS design ensures the EMAC miniport driver is built while building the OS image.

Including this component defines the following environment variables:

- IMGNOKITL = 1
- BSP_NOETHER =
- BSP_NOSHAREETH=1
- BSP_NIC_DM644XEMAC =1
- IMGNOSHAREETH =1

6.9.4 Build Options

The default sources and the Makefile are located in the path illustrated in Section 6.9.2.

This sources file enables the EMAC miniport driver to configure it in following manner.

- NDIS_MINIPORT_DRIVER is defined to receive appropriate miniport structure information.
- This EMAC mini-port driver conforms to NDIS 5.1 specification.

6.9.5 Registry File

This driver implementation expects the configuration information to be available in the registry. Table 18 identifies the various keys corresponding to this driver and their significance. The sample registry file for configuring this driver is located in the path:

$(_WINCEROOT)\PLATFORM\DAVINCI\SRC\DM644x\EMACMiniport\ DM644xEmacMiniport.reg /

Table 18. Registry Entries for EMAC NDIS Miniport Driver

Sl. No	Key Name	Value Type	Current Value	Description
1	DisplayName	STRING	DM644x EMAC Miniport Driver	Specifies the Name.
2	Group	STRING	NDIS	Specifies the group to which it belongs.
3	InterruptNumber	DWORD	0D	Hardware IRQ line of EMAC controller to CPU.
4	InterruptNumber	DWORD	85F10000	Physical address of the SDRAM region reserved for buffers for NDIS operation. This must correspond to the address kept reserved in config.bib file.

6.9.6 Driver Testing

This implementation of NDIS mini-port driver is verified for its functionality using the custom methods. Currently, the driver is being tested in the following manner.

1. Ping from host to the DEVEM board. This has been verified for ping packet lengths of up to 30,000 bytes.
2. Ping from DVEVM to any other host PC on the network. This has been verified for ping packet lengths of up to 1472 bytes. However, in the debug builds, this has been verified for up to 5000 bytes.
3. Telnet to the DEVEM using the other host PC. This is confirmed by executing commands on the Telnet

command.

4. FTP files of varying sizes from the DEVEM to the host PC. The file sizes of up to 112 Mbytes have been verified.
5. FTP files of varying sizes from the host PC to the DVEVM. The file sizes of up to 112 Mbytes have been verified.

One card and Two card CETK tests are being carried out using the procedure illustrated in the URL: http://msdn2.microsoft.com/en-us/library/ms836786.aspx .

6.10 DSP/BIOS Link Driver

Please refer to *TMS320DM644x DVEVM Windows CE v5.0 BSP DSP/BIOS Link Users Guide* (SPRUEW0), for more details on this driver.

6.11 VPFE Driver

6.11.1 Overview and Features

The driver for VPFE is implemented using the stream interface driver architecture. Currently the driver utilizes only CCDC subsection of the VPFE module.

The device manager uses the registry key, [HKEY_LOCAL_MACHINE \Drivers\BuiltIn\Camera] to load the driver, Dm644xVpfe.DLL.

As a streams model driver, the prefix registry entry under [HKEY_LOCAL_MACHINE \Drivers\BuiltIn\CAMn] defines the prefix to be used with all exported routines. The device manager uses this prefix to create the routine name, CAM_Init (), and calls it to let the driver perform its specific initialization. This routine, along with all other high-level CAM_xxx () routines, are located in the VPFE driver.

The following sections describe VPFE driver components and their functions.

6.11.2 Source Code Path

The source code for this driver implementation is located in the path:

$(_WINCEROOT)\PLATFORM\DAVINCI\SRC\Drivers\VPFE

These sources build the DLL named Dm644xVpfe.dll in the path:

$(WINCEROOT)\PLATFORM\DAVINCI\TARGET\ARMV4I\$(WINCEDEBUG)

6.11.3 CEC Entries

This driver is identified as *VPFE Driver* under the path Catalog → Third Party → BSPs → Davinci:ARMV4I → Device Drivers.

Including this component into the OS design ensures the VPFE driver is built while building the OS image.

Including this component defines the following sysgen variables:

- BSP_VPFE_FUNCTION=1

6.11.4 Build Options

There are no compile time switch options supported in this release of the driver.

6.11.5 Registry File

This driver implementation expects the configuration information to be available in the registry. Table 19 identifies the various keys corresponding to this driver and their significance. The sample registry file for configuring the VPFE driver is located in the path:

$(_WINCEROOT)\PLATFORM\DAVINCI\SRC\Drivers\VPFE\ Dm644xVpfe.reg

Table 19. Registry Keys for VPFE Driver

Sl. No	Key Name	Value Type	Current Value	Description
1	Prefix	STRING	CAM	Specifies the device prefix
2	Dll	STRING	Dm644xVpfe.dll	Specifies the Dll name to be loaded
3	Order	DWORD	10	Order as per which the driver needs to be loaded
4	Index	DWORD	1	Index of the driver
5	CameraDmaBufferBase	DWORD	86800000	Physical address of the base of SDRAM region reserved for VPFE driver DMA buffer. This must be consistent with the config.bib settings.
6	CameraDmaBufferSize	DWORD	00400000	Size of the SDRAM region reserved for VPFE DMA buffer.

6.11.6 Driver Testing

This implementation of the driver is tested using the sample application provided. The sample application obtains the captured frame from VPFE driver and displays it using Video1 of VPBE subsystem.

There are no CETK test suites available for testing VPFE driver. The application used to test the VPFE driver can be located at $(_WINCEROOT)\PLATFORM\DAVINCI\SRC\APPS\VPFE. All required configurations are done using custom IOCTL calls. This application needs to be built using the command line build method. To run the application give *s vpfeTest.exe* on CE target shell or go to Target → Run Programs and select the vpfeTest.exe and run it. This test application does the following.

1. Maps the memory allocated for driver operation (memory for Video1 frame buffer and buffers for storing the captured frames).
2. After call to CreateFile of CAM1, the application initializes the driver. This includes configuring the video mode, registering the video capture buffers with the driver using call to DeviceIoControl with IOCTL_CAM_REGISTER_BUFF. After this call, app should not access the video capture buffers (Driver will be using it).
3. Calls DeviceIoControl with IOCTL_VIDEO_START_STREAMING for starting capture
4. Waits for a captured frame using call to DeviceIoControl with IOCTL_CAM_DEQUEUE_BUFF. When this call unblocks, the app understands that a new frame is available and the buffer in which the frame is available is passed to the application. Now the app can access buffer.
5. Copies the frame from capture video buffer to frame buffer memory.
6. Calls DeviceIoControl With IOCTL_CAM_ENQUEUE_BUFF, which informs the driver that the app does not need the buffer any more and the driver can use it. After this call, the app should not access that buffer.

6.12 7.12 SD Host Controller

Table 20. Terms and Acronyms

Number	Term	Description
1	SDHC	Secure Digital Host Controller
2	SD	Secure Digital Card
3	MMC	Multimedia Memory Card
4	SDIO	Secure Digital Input Output

6.12.1 Overview and Features

The secure digital (SD) card host controller driver controls the host controller hardware and conforms to a host controller software interface that the bus driver uses to communicate and set operating parameters. The host controller driver uses an API set exported by the bus driver for registering and unregistering. The SD card stack uses a dynamic architecture that allows host controller drivers to register or unregister at any time. The host controller driver interface provides a hardware abstraction layer between the bus driver and the host controller implementation.

The DaVinci SD host controller driver requires the services of the SD bus library, SD host controller library, and the SD memory library to generate a working SD host controller driver. The main features of the SD host controller driver are that it supports both the SD and the MMC cards. This driver supports configuring the SD bus in both 4-bit mode for the SD card and in 1-bit mode for the MMC card. The driver implementation also supports the hot-plugging feature. This is more of a storage medium used for copying any data content. Once the driver comes up, the SD card or MMC card can be seen as driver (normally seen as storage card folder) when the windows explorer is opened.

6.12.2 Source Code Path

The source code for this driver implementation is located in the path:

$(_WINCEROOT)\PLATFORM\DAVINCI\SRC\DRIVERS\SDHC

. These sources build the DLL named Dm644xSDIO.dll in the path:

$(WINCEROOT)\PLATFORM\DAVINCI\TARGET\ARMV4I\$(_WINCEDEBUG)

6.12.3 CEC Entries

This driver is identified as *DM644x SD MMC Driver* under the path Catalog → Third Party → BSPs → Davinci:ARMV4I → Device Drivers → Storage Devices → DM644x SD MMC Driver. Including this component into the OS design ensures the SDHC driver is built while building the OS image.

Including this component defines the following sysgen variables:

- BSP_SDMMC=1

6.12.4 Build Options

The default sources and the Makefile is located in the path illustrated in Section 6.12.2. This sources file enables the SD MMC card driver to configure in following manner.

Table 21 identifies the list of build flags supported.

Table 21. Build Options for SD/MMC Card Driver

Sl. No	Build Flags	Description
1	SDIO_GPIO_WORKAROUND	Defining this flag enables the code implementing for work around for proper functioning of the SD card driver with IR included in the image. Both the IR and SDMMC use the same MSP interrupt.

6.12.5 Registry File

This driver implementation expects the configuration information to be available in the registry. Table 22 identifies the various keys corresponding to this driver and their significance. The sample registry file for configuring the SD\MMC driver is located in the path:

$(_WINCEROOT)\PLATFORM\DAVINCI\SRC\DRIVERS\SDHC\SDHC.REG

Table 22. Registry Keys for SD/MMC Card Driver

Sl. No	Key Name	Value Type	Current Value	Description
1	Order	DWORD	21	Order in which the DLL loads
2	Dll	String	DM644xSDIO.dll	Name of the DLL
3	Prefix	String	SHC	Prefix used by the driver
4	IClass	Multi_Sz	A32942B7-920C-486b-B0E6-92A702A99B35	Interface class
5	DmaPhysicalBase	DWORD	86680000	Memory reserved of DMA area.

6.12.6 Driver Testing

This implementation of the driver is tested using the following CETK test cases:

- All storage device block driver read/write test cases
- All storage device block driver benchmark test cases
- All storage device block driver API test cases
- File system driver test

All the above said test cases are passing. While some of the test cases may fail at specific instances, in other iterations, the same test cases pass.

The test cases for Benchmark sometimes throw out unexpected results, such as the Clientside.exe unloading, and tests not completing. However, this scenario needs to thoroughly test to conclude the problem.

6.13 NAND Flash Media Driver

6.13.1 Overview and Features

The NAND flash media driver (FMD) is a device driver that performs the actual input and output of data to NAND flash memory device. This FMD contains all of the device-specific code necessary for read, write and erase commands to the flash memory device. The FMD is linked with the Flash abstraction layer (FAL) to create a block driver that a file system such as FAT can use. The FMD is also linked with the boot-loader so that the boot loader can create partitions and flash a run-time image into a specific partition (BINFS).

The following sections describe NAND FMD driver components and their functions.

The device manager uses the registry key, [HKEY_LOCAL_MACHINE \Drivers\BuiltIn\FlashDisk] to load the driver, Dm644xNand.dll.

6.13.2 Source Code Path

The source code for this driver implementation is located in the path:

$(_WINCEROOT)\PLATFORM\DAVINCI\SRC\COMMON\NAND

These sources build the DLL named Dm644xNand.dll in the path:

$(WINCEROOT)\PLATFORM\DAVINCI\TARGET\ARMV4I\$(WINCEDEBUG)

6.13.3 CEC Entries

This driver is identified as *Dm644x Nand Fmd* under the path Catalog → Third Party → BSPs → Davinci:ARMV4I → Device Drivers → Storage Devices.

Including this component into the OS design ensures the NAND FMD is built while building the OS image.

Including this component defines the following sysgen variables:

- BSP_NAND_DRIVER = 1
- SYSGEN_STOREMGR = 1
- SYSGEN_FATFS = 1
- SYSGEN_MSPART = 1
- SYSGEN_STOREMGR_CPL = 1

6.13.4 Build Options

The default sources and the Makefile are located in the path illustrated in Section 6.13.2.

6.13.5 Registry File

This driver implementation expects the configuration information to be available in the registry. Table 23 identifies the various keys corresponding to this driver and their significance. The sample registry file for configuring the NAND FMD is located in the path:

$(_WINCEROOT)\PLATFORM\DAVINCI\SRC\COMMON\NAND\DLL\ Dm644xNand.reg.

Table 23. Registry Keys for NAND FMD

Sl. No	Key Name	Value Type	Current Value	Description
1	Index	DWORD	1	Specifies the device prefix
2	Dll	STRING	Dm644xNand	Specifies the Dll name to be loaded
3	Prefix	STRING	DSK	Specifies the device prefix
4	Order	DWORD	0	Order by which the driver gets loaded
5	Profile	STRING	FlashDisk	Profile to be used by this dll.
6	IClass	STRING	{A4E7EDDA-E575-4252-9D6B-4195D48BB865}	Interface class
7	Folder	STRING	NAND Flash	Name of the folder to be displayed
8	DefaultFileSystem	STRING	FATFS	Indicates the default file system to load
9	AutoMount	DWORD	1	Indicates whether the drive needs to be mounted to boot time
10	AutoPart	DWORD	1	Flag to partitions the store with largest creatable partition automatically
11	AutoFormat	DWORD	1	Formats a store automatically when the store is unformatted
12	FileSystem	STRING	fatfsd.dll	File system driver currently being used.
13	PartitionDriver	STRING	mspart.dll	Indicates the default partition driver to load

6.13.6 Driver Testing

This implementation of the driver is tested using the following CETK test cases:

- Storage device block driver read/write test
- Storage device block driver benchmark test
- Storage device block driver API test
- Flash memory read/write and performance test

All the test cases are passing. However, the following observation was made.

- The CETK for storage block driver benchmark may overwrite the file system information (if any existing). This could lead to some test cases failing randomly. If the read/write happens in that sector, then the test case may fail saying media is write-protected.

6.14 USB Function Controller Driver

Table 24. Terms and Acronyms

Number	Term	Description
1	CETK	Windows CE Test Kit
2	EVM	Evaluation Module
3	SDRAM	Synchronous Dynamic Random Access Memory
4	USB	Universal Serial Bus
5	OTG	On-The-Go
6	MSC	Mass Storage Class
7	RAS	Remote Access Server
8	BOT	Bulk-only Transport
9	CBW	Command Block Wrapper
10	CSW	Command Status Wrapper

6.14.1 Overview and Features

Windows CE 5.0 provides well-defined driver architecture to support the USB function controller and the various class drivers on top of it.

USB function controller drivers are layered drivers and consist of an MDD and a PDD. The PDD abstracts a USB function controller and the MDD provides a function driver with the ability to configure and access the underlying USB function controller.

The PDD Layer abstracts the USB controller, supports configuration of the USB function hardware and provides an event-based notification to the MDD layer.

The USB function controller driver, MDD, exposes a bus interface, which is implemented in the MDD. This allows USB function client drivers to be loaded as stream interface drivers. The MDD layer exposes a stream-driver interface and maintains the context information for each underlying PDD.

On the DaVinci EVM, the USB function driver initializes the USB controller in device mode and configures the USB controller in full-speed mode. It supports the access to the registers using both Indexed and flat addressing modes.

6.14.2 Source Code Path

The source code for this driver implementation is located in the path:

$(_WINCEROOT)\PLATFORM\DAVINCI\SRC\Drivers\USBFN

These sources build the DLL named Dm644xUsbFn.dll in the path:

$(WINCEROOT)\PLATFORM\DAVINCI\TARGET\ARMV4I\$(_WINCEDEBUG)

6.14.3 CEC Entries

This driver is identified as *USB Function Driver* under the path Catalog → Third Party → BSPs → Davinci:ARMV4I → Device Drivers → USB Function Bus Drivers → USB Function Driver.

Including this component defines the following sysgen variables:

- BSP_USB_FUNCTION=1
- BSP_NOUSB=
- SYSGEN_USBFN_STORAGE=1

6.14.4 Build Options

The default sources and the makefile are located in the path illustrated in Section 6.14.2. This sources file enables the USB function driver to configure the USB controller in following manner:

- USB controller access with index addressing mode
- USB controller access with flat addressing mode

Table 25. Build Options for Function Controller Driver

Sl. No	Build Flags	Description
1	USB_EARLY_HANDSHAKE	Defining this flag enables the sending of USB control handshakes from the USB interrupt handler routines itself rather than waiting for the MDD layer to invoke the ControlStatusHandShake routine. This flag is defined by default for better response times during USB transactions.
2	MUSB_FLAT_REG_MODEL	Defining this flag causes the driver to address the USB controller registers in flat addressing mode. By default, this flag is not defined and hence the driver uses indexed addressing mode.
3	CPPI_DMA_SUPPORT	This flag is defined by default and enables the CPPI DMA support for both transmit and receive paths in the driver.

6.14.5 Registry File

This driver implementation expects the configuration information to be available in the registry. Table 26 identifies the various keys corresponding to this driver and their significance. The sample registry file for configuring the display driver is located in the path:

$(_WINCEROOT)\PLATFORM\DAVINCI\SRC\DM644x\USBFN\DM644xUsbFn.REG

Table 26. Registry Keys for USB Function Controller Driver

Sl. No	Key Name	Value Type	Current Value	Description
1	InterfaceType	DWORD	0	Type of the interface.
2	InterfaceType	STRING	UFN	Prefix used by the driver
3	Dll	STRING	Dm644xUsbFn.dll	Name of the driver DLL
4	Order	DWORD	10	Load Order of the USB function driver.
5	Priority256	DWORD	0x64	Priority of the interrupt service thread maintained by the USB function driver
6	MemBase	MULTI_SZ	01C64000, 01C64400	Specifies a range of memory addresses used by the USB function driver to access the hardware registers.
7	MemLen	MULTI_SZ	300, 300	Specifies the length of the above memory ranges that needs to be virtually mapped by the driver for register access.
8	Irq	DWORD	0C	Specifies the physical interrupt number of USB on the platform
9	BusIoctl	DWORD	2a0048	Specifies the default IOCTL routine to be invoked during startup of the driver
10	Iclass	MULTI_SZ		Specifies the Iclass interface for the driver
11	TxDmaBuffer	DWORD	86700000	This key defines the SDRAM memory region which is utilized by the USB function driver as DMA buffer area.
12	TxDmaBuffSize	DWORD	4000	This key defines the SDRAM memory region length which is utilized by the USB function driver.

6.14.6 USB Mass Storage Client Driver Registry Settings

Table 27. Registry for MSC Client Driver

Sl. No	Key Name	Value Type	Current Value	Description
1	Dll	STRING	usbmsfn.dll	Davinci EVM generic mass storage
2	InterfaceSubClass	DWORD	06	Interface sub-class code used by the client driver
3	InterfaceProtocol	DWORD	50	Interface protocol code reserved for USB mass storage
4	DeviceName	STRING	DSK1:	Name of the block device which is to be presented as USB mass storage device
5	FriendlyName	STRING	Mass Storage	Friendly name of the client driver
6	idVendor	DWORD	0451	Vendor ID for the USB mass storage. This is set to Texas Instruments vendor ID
7	Manufacturer	STRING	Texas Instruments	Name of the manufacturer
8	idProduct	DWORD	0xFFFF	Product ID for the USB mass storage
9	Product	STRING	Davinci EVM Generic Mass Storage	Name of the USB mass storage device as seen by the host PC
10	bcdDevice	DWORD	0	Device ID used by the USB mass storage

6.14.7 Driver Testing

The Windows CE 5.0 Test Kit does not include any test cases for the USB function driver.

6.14.7.1 USB Mass Storage

The mass storage driver has been tested with the RAMDISK driver and hard disk on the DVEVM by these methods:

- Transfer of files of varying sizes from host PC to the USB mass storage.
- Transfer of files of varying sizes from USB mass storage to host PC.

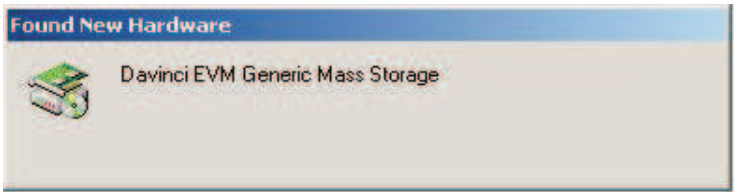
On certain Windows 2000 host PCs, the USB mass storage device is detected during the initial plug-in sequence and also the driver is automatically detected and the drive-loaded.

However, on those host PCs which do not have the recent service packs installed, the host Windows take you through a series of steps to recognize the *new hardware*. This was observed during the initial testing.

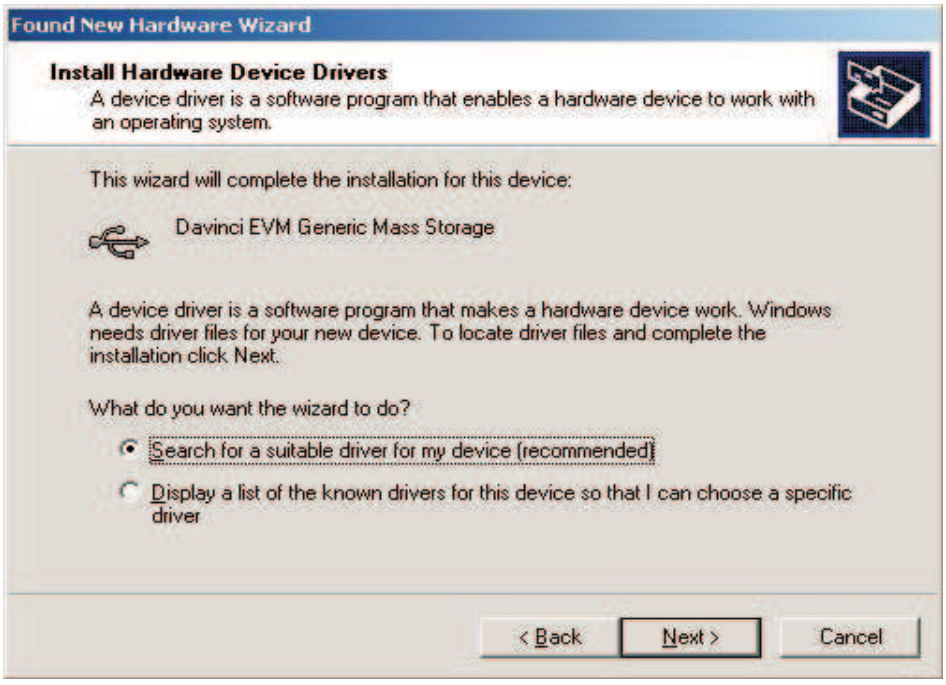
6.14.7.2 USB Mass Storage Hardware Detection on Host PC

This section explains the steps involved in setting up the USB mass storage on the host PC. When the DaVinci EVM, loaded with USB driver is plugged into the USB port on the host PC, we observe the following dialog on the host.

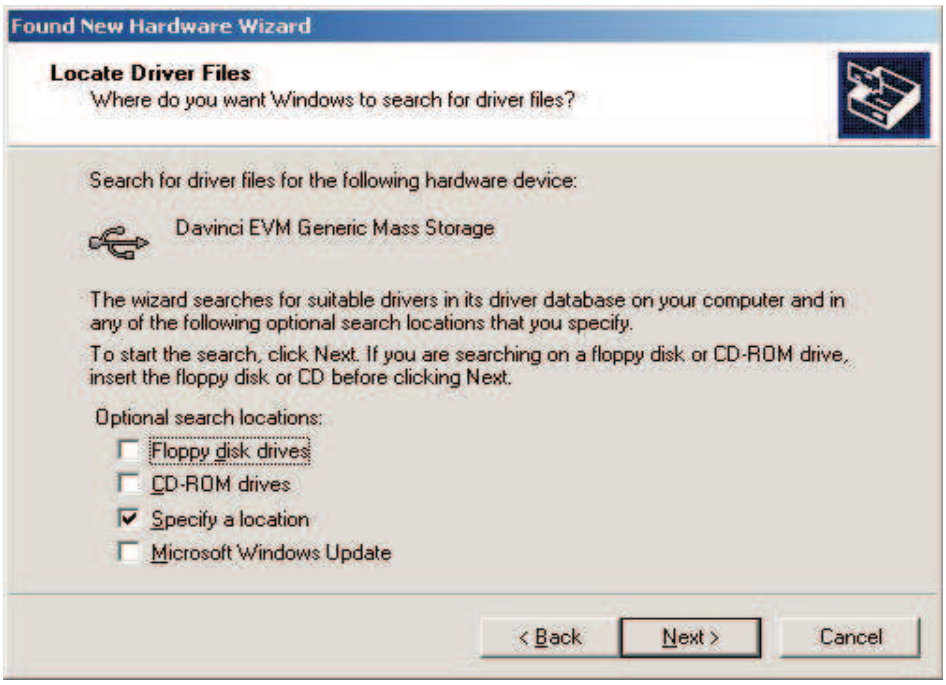
The host declares the Davinci EVM as *New USB Hardware* and prompts you to point to the appropriate driver.



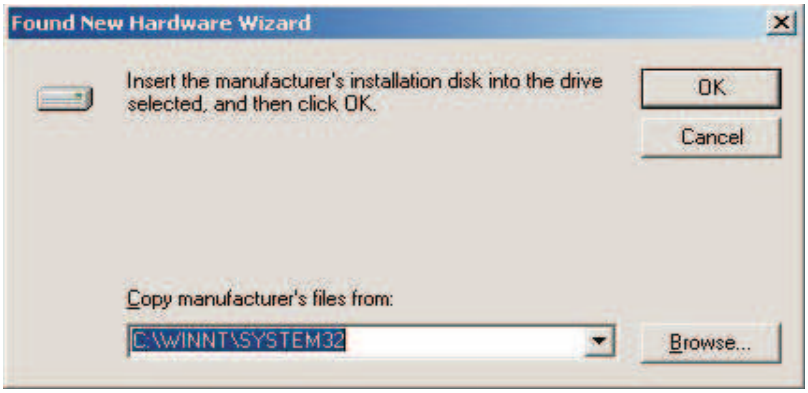
1. Press the Next button to continue with the hardware installation process.



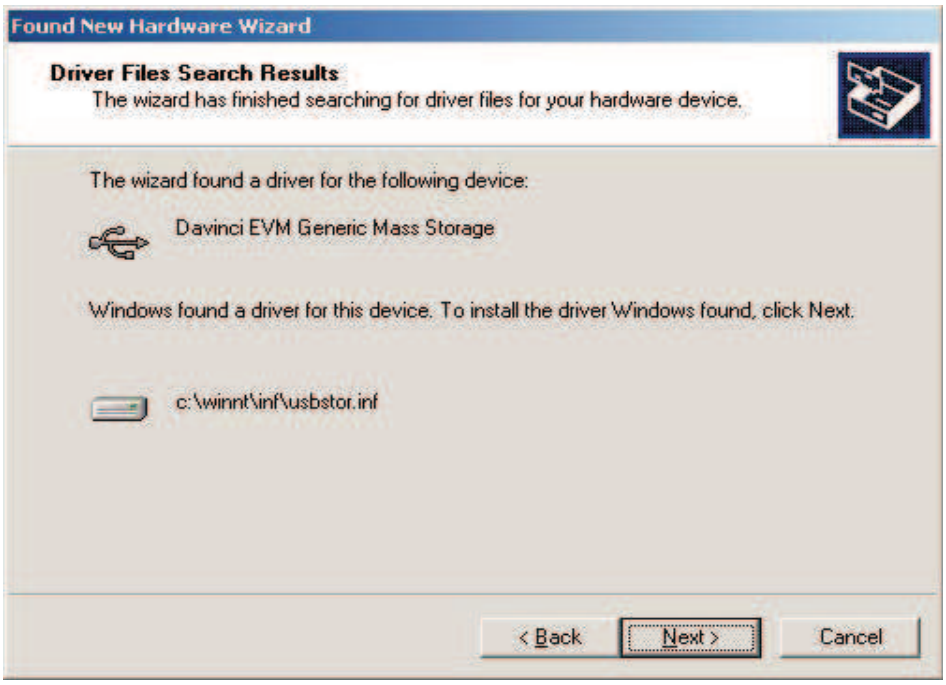
2. As shown in the *New Hardware Wizard*, select the Search for a suitable driver option and press the Next button.



3. Select the *Specify a location* option and press the Next button.

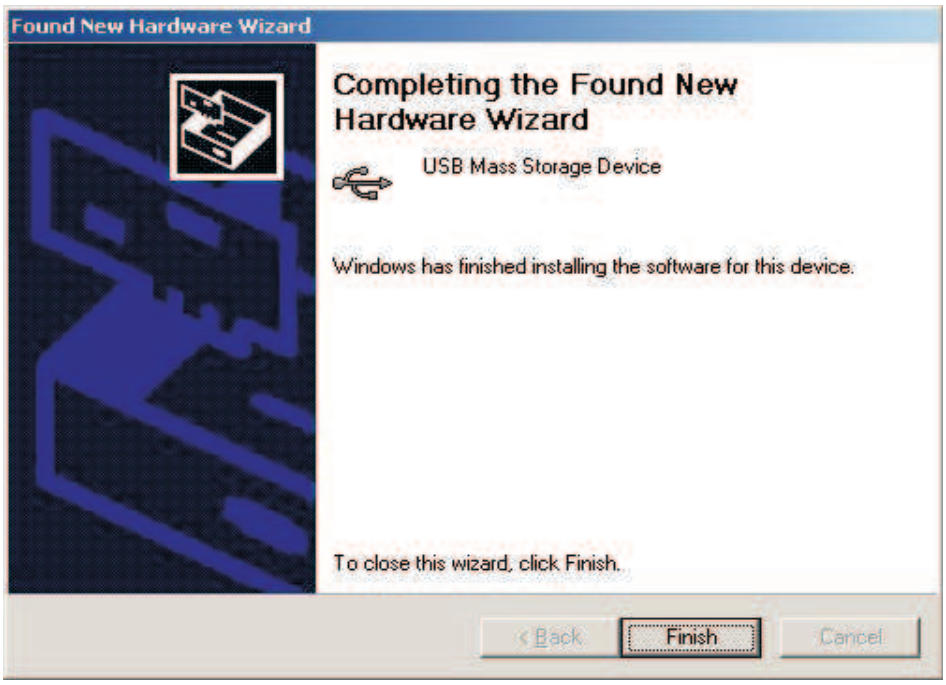


4. Point to your WINNT\SYSTEM32 installation path to pick up the default Microsoft USB drivers and press OK button.

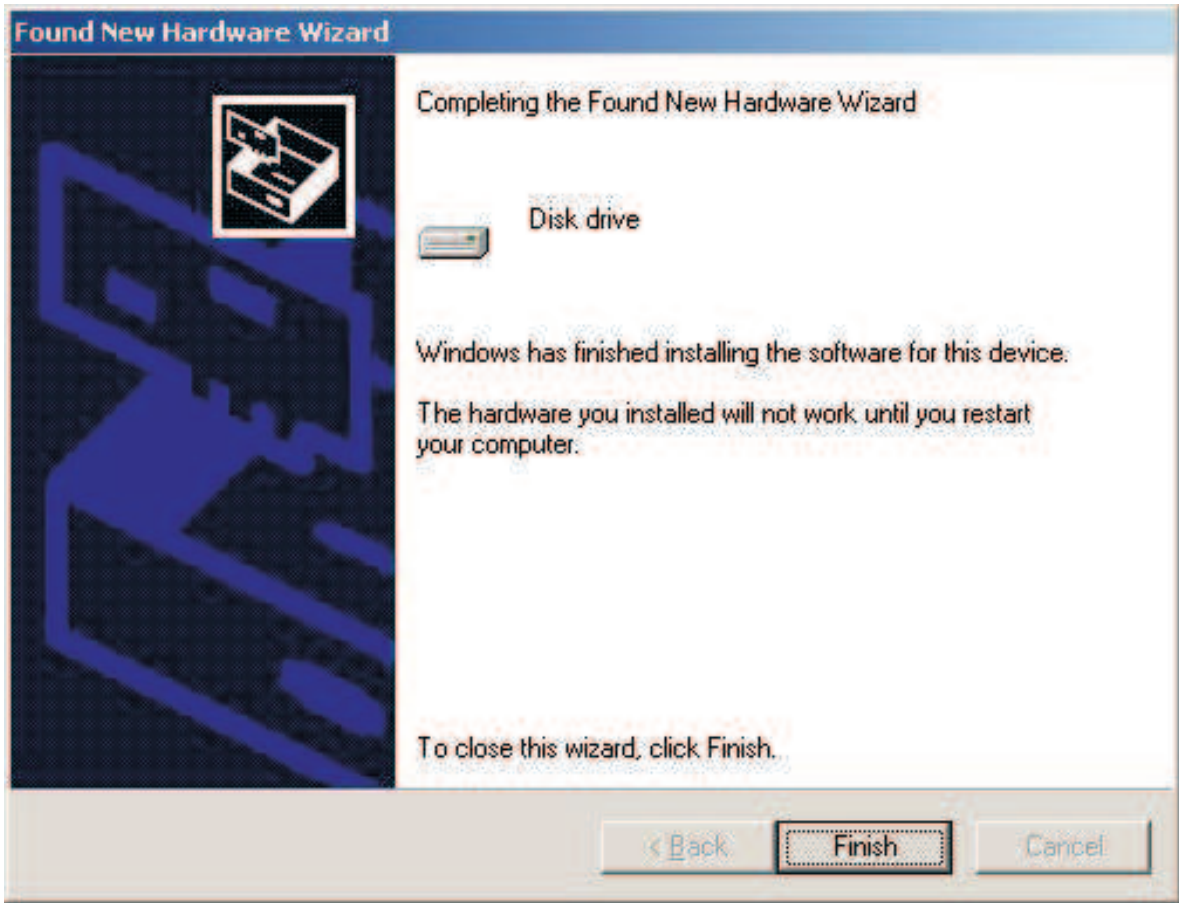


5. The installation process detects the default Microsoft host module usbstor.inf from your Windows 2000 installation.

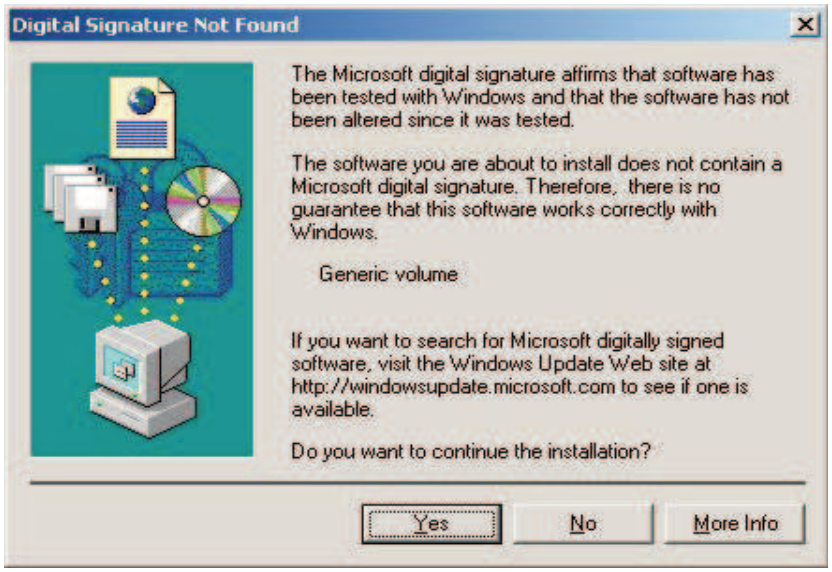
6. Since the usbstor.inf does not have any reference to our USB mass storage device, the above dialog will be displayed. Press the Yes button to continue with the installation.



7. Press the Finish button to complete the installation process.



8. Press the Finish button again, which now installs the USB disk drive on the host PC.



9. On certain PCs, another dialog box appears prompting for the *Generic Volume*. Press the Yes button and Finish button later, to complete the volume mount.

6.14.7.3 USB Mass Storage Conformance Tests

Since there are no standard Windows CE test kit tests for the USB function controller, this release of the USB mass storage has been tested with the *USB Mass Storage Compliance* tests as specified by the USB organization.

The USB organization has the *Universal Serial Bus Mass Storage Class Compliance Test Specification* document at the following URL: http://www.usb.org/developers/devclass_docs/MSC-compliance-0_9a.pdf.

Some of the most important test cases mentioned in the above link are executed against this USB mass storage driver. Please note that this would require the USB Sniffer to be installed in the host PC.

Table 28. Mass Storage Conformance Tests

Sl. No	Test Description	Test Status	Remarks
1	Device must have a serial number (if the device supports a BOT interface, bInterfaceProtocol = 0x50).	Pass	This information can be retrieved from the device descriptor sent by the device to the host.
2	Devices must support at least one data interface (InterfaceClass = 0x08).	Pass	This information can be retrieved from the interface descriptor sent by the device to the host.
3	Data interface SubClass code must be valid (0x01 – 0x06).	Pass	This information can be retrieved from the interface descriptor sent by the device to the host.
4	Data interface protocol code must be 0x50 (BOT interface), or 0x00 or 0x01 (CB/CBI interface).	Pass	This information can be retrieved from the interface descriptor sent by the device to the host.
5	Data interface protocol code must be 0x50 (BOT interface), or 0x00 or 0x01 (CB/CBI interface).	Pass	This information can be retrieved from the interface descriptor and the EndPoint descriptor sent by the device to the host.
6	The CB/CBI device must not be high-speed device.	Pass	This information can be retrieved from the device descriptor sent by the device to the host.
7	The CSW must be 13 bytes long.	Pass	This information can be retrieved from the command status response sent by the device to host.
8	The CSW signature must be 0x53425355.	Pass	This information can be retrieved from the command status response sent by the device to host.
9	The tag field of a CSW must match the tag of the associated CBW.	Pass	This information can be retrieved from the command block wrapper sent by host and command status wrapper sent by device to host.

Table 28. Mass Storage Conformance Tests (continued)

Sl. No	Test Description	Test Status	Remarks
10	The CSW status value must be 0x00, 0x01, or 0x02.	Pass	This information can be retrieved from the command status response sent by the device to host.
11	If the CSW status value is 0x00 or 0x01, the residue must be less than or equal to the transfer length.	Pass	This information can be retrieved from the command status response sent by the device to host.
12	The interface must respond properly to Get_Status requests at all times.	Pass	The status responses sent by the device to the host can verify this information.
13	The interface shall be capable of responding to commands without failure immediately following configuration.	Pass	The same can be verified from the enumeration logs seen on the debug Window

6.15 USB Host Controller Driver

Table 29. Terms and Acronyms

Number	Term	Description
1	CETK	Windows CE Test Kit
2	EVM	Evaluation Module
3	USB	Universal Serial Bus
4	OTG	On-The-Go
5	MSC	Mass Storage Class

6.15.1 Overview and Features

Windows CE 5.0 provides well-defined driver architecture to support the USB host controller and the various class drivers on top of it.

6.15.2 Source Code Path

The source code for this driver implementation is located in the path:

$(_WINCEROOT)\PLATFORM\DAVINCI\SRC\Drivers\USBH

These sources build the DLL named DM644xUsbh.dll in the path:

$(WINCEROOT)\PLATFORM\DAVINCI\TARGET\ARMV4I\$(_WINCEDEBUG)

6.15.3 CEC Entry

This driver is identified as *USB Host Driver* under the path Catalog → Third Party → BSPs → Davinci:ARMV4I → Device Drivers → USB Host → USB Host Controller →b USB Host Driver.

Including this component into the OS design ensures the display driver is built while building the OS image.

Including this component defines the following sysgen variables:

- SYSGEN_USB_HID=1
- SYSGEN_MSPART= 1
- SYSGEN_STOREMGR=1
- SYSGEN_USB_HID_CLIENTS=1
- SYSGEN_USB_STORAGE=1
- SYSGEN_STOREMGR_CPL=1
- SYSGEN_FATFS=1

- BSP_USB_HOST=1

6.15.4 Build Options

This driver does not support any compile time configuration options.

6.15.5 Registry File

The sample registry file for configuring the display driver is located in the path:

$(_WINCEROOT)\PLATFORM\DAVINCI\SRC\Drivers\USBH\Dm644xUsbh.REG

Table 30. USB Host Controller Driver Registry

Sl. No	Key Name	Value Type	Current Value	Description
1	Order	DWORD	6	Order in which the DLL loads
2	Dll	String	Dm644xUsbh.dll	Name of the DLL
3	Prefix	String	HCD	Prefix used by the driver
4	Index	DWORD	1	Instance index for the USB Host controller driver

6.15.6 Driver Testing

The USB host driver supports only the following devices:

- USB keybaord
- USB mouse
- USB mass storage

7 DirectShow Filters

Please refer to Section 7 for more details on using the DirectShow filters and the related components.

8 Known Issues/Caveats

The following issues are prevalent in this release of the board support package.

- This version of BSP does not provide support for the KITL connection over serial connection.
- This release of the BSP uses the UART port as a debug port and does not allow the serial port to be used by the applicaton.
- The EVM board has only one external USB port. This port may be either configured as the USB host port or as the USB function port. Therefore, any OS configuration would have either a USB host controller driver or USB function controller driver, but not both.
- The EMIF bus on the DVEVM is being shared by the following peripherals. Since there is a difference in the initialization sequence for each of these peripherals, only one of the following drivers may be included in the given OS configuration.
  - ATA hard disk driver
  - Compact flash controller driver
  - NAND flash media driver

9 References

A. Microsoft Platform Builder 5.0 On-Line Help
B. DaVinci Evaluation Module – Technical Reference; Rev.A; January 2006.

IMPORTANT NOTICE

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

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

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

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

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

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

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

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