Texas Instruments

DM6437/C6424 DSP/BIOS PSP

User's Manual

IMPORTANT NOTICE

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

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

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

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

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

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

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

Products		Applications	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
		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

Read This First

About This Manual

This User's Manual serves as a software programmer's handbook for working with the **DM6437/C6424 DSP/BIOS PSP Version GA 1.10.03** This manual provides necessary information regarding how to effectively install, build and use **DM6437/C6424 DSP/BIOS PSP** in user systems and applications.

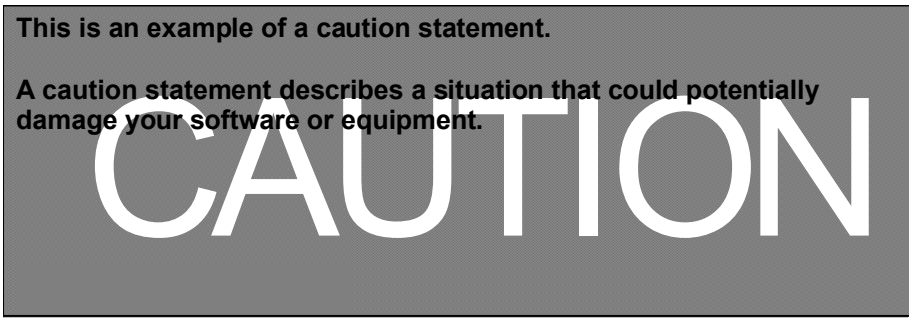
Abbreviations

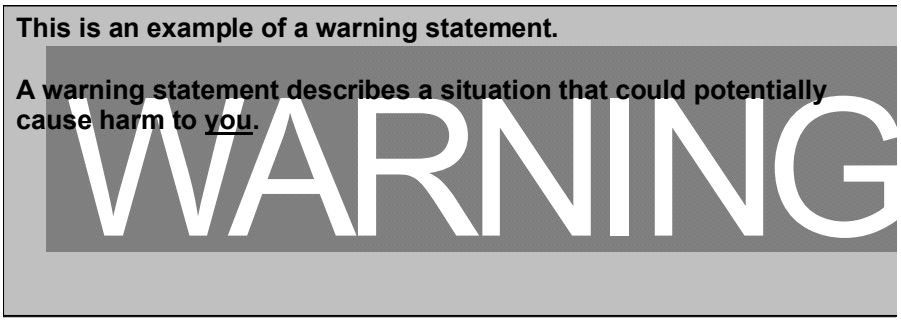
Table 1-1. Table of Abbreviations

Abbreviation	Description
PSP	Platform Support Package
DM6437/C6424 DSP/BIOS PSP	This is TI coined name for the product.
API	Application Programming Interface
IOM	Device Driver Adapter
DDC	Device driver core
LLC	Lower level controller
OS	Operating System
PAL OS	Platform abstraction layer for operating system
SOC	System On Chip

Information About Cautions and Warnings

This book may contain cautions and warnings.





The information in a caution or a warning is provided for your protection. Please read each caution and warning carefully.

Related Documentation

Internal

This is a list of documents that are TI Proprietary and Strictly Private. Exposure to audience outside TI will need due considerations and approvals from TI Legal authorities.

- DM6437 TRM and peripheral documents from PDS
- C6424 TRM and peripheral documents from PDS

Revision History

Version	Date	Revision History
1.00.00.01	17th May 2007	GA Candidate Release
1.00.01.00	22nd June 2007	GA Patch release 1
1.10.00.08	3rd December 2007	Modified for Unified release with DM648 BIOS PSP

1.11	28TH January, 2008	Updated for release 1.10.00.09 Changed the versioning of this document
1.12	May 29, 2008	Updated for release 1.10.01
1.13	Dec 22 2008	Updated for release 1.10.02
1.14	16 April 16, 2009	Updated for release 1.10.03

Contents

Table of tables

Table of figures

Chapter 1

Introduction

This chapter introduces the **DM6437/C6424 DSP/BIOS PSP** to the user by providing a brief overview of the purpose and construction of the **DM6437/C6424 DSP/BIOS PSP** along with hardware and software environment specifics in the context of **DM6437/C6424 DSP/BIOS PSP** deployment.

1.1 Overview

The DM6437/C6424 PSP is aimed at providing fundamental software abstractions to DM6437/C6424 EVM resources and plugs the same into BIOS operating systems so as to enable and ease application development by providing suitably abstracted interfaces.

1.1.1 Supported Services and features

This release of **DM6437/C6424 DSP/BIOS PSP** provides the following:

- PAL OS for BIOS
- UART, I2C, McBSP, NAND, McASP
- VPFE, VPBE, Previewer, Resizer, H3A and Histogram (Only DM6437)
- PAL SYS (VLYNQ and PCI)
- Sample applications

1.1.2 System Requirements

The following products are required to be installed for using the DM6437/C6424 DSP/BIOS PSP:

- CCS 3.3.38
- DM6437/C6424 EVM
- Code generation tools : 6.0.8
- DSP-BIOS 5.31.06 or higher for DM6437
- DSP-BIOS 5.31.06 or higher for C6424
- XDC tools 3.00.01 or higher

1.2 Driver Software Architecture

This section gives detail on the overall architecture of TI device driver.

1.2.1 Functional Decomposition

The device driver is partitioned into distinct sub-components, consistent with the roles and responsibilities already discussed in section 1.3. In the following sub-sections, each of these

functional sub-components of the device driver is further elaborated.

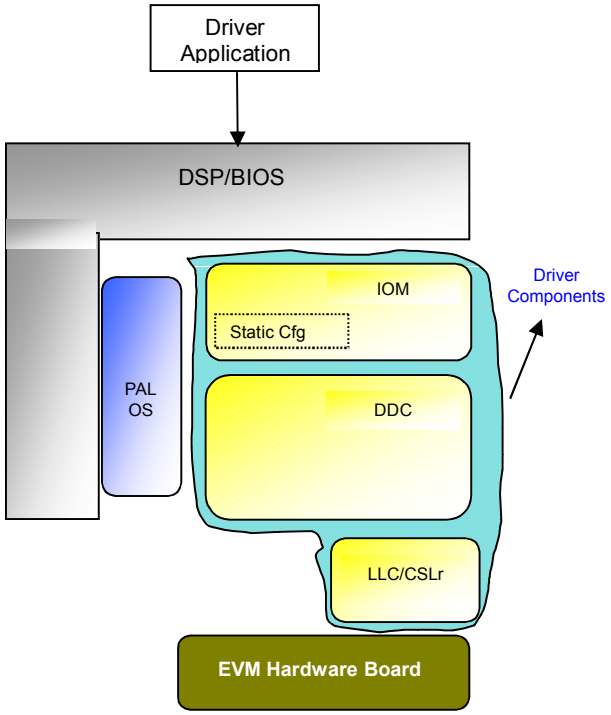


Figure 1 TI Device Driver Functional decomposition

The central portion (IOM-DDC-LLC) shown constitutes the mainline device driver component. The surrounding module PAL-OS constitute the supporting system components that facilitates the interfaces between the OS and the above mentioned device driver components. These modules do not specifically deal with device driver but assist the driver by providing OS abstraction.

1.2.2 H/W Device Specific Layer

CSL Register Layer: This is comprised of a bunch of symbolic constants (#defines) that expose the register bit-field details of the h/w along with assorted other constants such as bit-field masks, shift values and default settings.

LLC Layer: The LLC (low level controller) layer forms the lower most, h/w specific under-pinning of the TI device driver.

This is comprised of a set of functions that exposes functions to set various functionalities supported by hardware. Hardware parameters for example in UART baud-rates, stop bits etc. can be set. This parameter setting is done by writing proper values to the respective hardware register.

A C-structure data type definition is provided as an as-is map of the peripheral device registers in the processor's memory map. This structure is termed the Register Overlay structure. The intended mode of usage is to initialize a pointer variable to base address of the peripheral h/w device and typecast it using structure overlay definition. This way, an otherwise un-adorned pointer assumes a strong C-type thereby making it possible to program the h/w registers by read/write to structure member elements.

It is important to note here that LLC layer *scope is limited to directed access of the underlying h/w device*. It does not depend on any specific OS and does not exploit optimizations specific to a given Compiler. In contrast to a device driver, it does not perform operations that are viewed as management of data movement over the peripheral device. It does not model state machines or protocols. Besides, LLC layer is deployed as a per-processor (single CPU) specific library of services. LLC layer philosophy mandates that services exported for one module (read peripheral h/w device) must not call the services of another module. This orthogonal rule is enforced to allow for true componentization of device drivers.

1.2.3 Device Driver Core functionality (DDC)

The DDC module for device driver, the OS abstracted portion of the driver which provides basic behavior of the driver, modeling the main functionalities and Protocols. The DDC does not directly touch the underlying h/w, it does so via the LLC layer. Likewise,

it does not make direct reference to any OS services, it glues to the OS via a well specified adaptation module termed the IOM.

By mandating certain basic interfaces and extension principles from all implemented device drivers, the DDC helps achieve, uniformity of driver API syntax/semantics across all supported devices and OS platforms.

The objective of a good driver partitionning is to sediment as much of the driver behaviour into the DDC as is practical. This calls for a IOM that is thin and efficient. Reuse and performance improvement efforts can then focus on DDC where bulk of the functionality is realized.

The DDC is modeled after object oriented style of modular software development. It specifies a base set of interfaces that standardizes the common aspects of all device drivers such as – configuration, creation, initialization & startup, termination and teardown. This base set of function can be extended by introducing operations specific to a particular device – IO Access and IO Control. DDC further encourages formation of device classes while extending the basic functionality. This way, device drivers for h/w components that are similar in data transaction models and control semantics will look alike.

The DDC on its own has no existance, it is brought to life by the IOM when the device driver is loaded by the system and removed from the system upon unloading. The IOM has the obligation to pass-on the relevant driver configuration parameters to the DDC during creation phase.

The IOM implements a set of functions that adapt the driver to OS. Likewise DDC implements a set of functions that constitute the driver functional interface.

To improve componentization of device driver, DDC comprises of C functions which makes use of the LLC Layer for implementation of certain functions that support:

- Operations to formally begin/end access to device h/w
- Operations to perform onetime setup of the H/W device such as during device driver initialization
- Operations to program the H/W device registers to change one or more of its configuration parameters
- Operations to query and infer the current state of the H/W device

1.2.4 OS Specific Device Driver Adaptation (IOM)

As discussed above, the DDC is not complete unless it is supplemented by the IOM. The IOM "adapts" the driver core to the specific OS. IOM implements aspects such as – threading

model for transaction processing, Interrupts registration and de-registration, handshaking with OS prescribed upstream/downstream data queues and threads etc., The IOM has full visibility to underlying OS services and is custom-built for a given OS.

While the IOM is primarily intended for presenting an OS manifest to the underlying DDC, it is also possible that the IOM upper-edge interface (user level) imbibes the semantics of any pre-specified Framework, if one exists. This is necessary to prevent undue overheads in system integration.

1.2.5 Platform Abstraction Layer for OS services (PALOS)

It should be clear by now that a device driver is composed of three main sub-components – the HW specific bottom-edge, the OS specific upper-edge and the central device driver core that makes no reference to any particular OS services. Memory footprint Scalability is a key consideration in deploying such a driver. In this section, we give a quick overview of suggested approach followed in TI device driver implementations.

As discussed earlier, both the IOM and DDC publish a table of function pointers for each other to use. However, it must be noticed that, not all the OS specific adaptation services are solicited by DDC in the same context. In addition, it's not practical to abstract all required OS services due to performance reasons. Therefore, we'd usually end up with a residue part of driver that must be *custom built in the IOM by directly availing the OS services.* It is this part of the IOM that is exposed through the table of functions to the DDC.

The rest of the OS services such as working with Semaphores, Mutexes, Memory buffer pools etc., are generic in nature. If these services are implemented as static functions, rolled into each IOM, memory foot print bloat occurs when there are multiple instances of device drivers in the system. To mitigate memory footprint bloat and difficulties in reuse, all generic OS abstraction services are pulled out as separate compilation units so that only one copy of these need be loaded to resolve all references across different driver instances.

This common module is termed the PAL OS and is depicted in figure below as green colored units, located inside the library to the right.

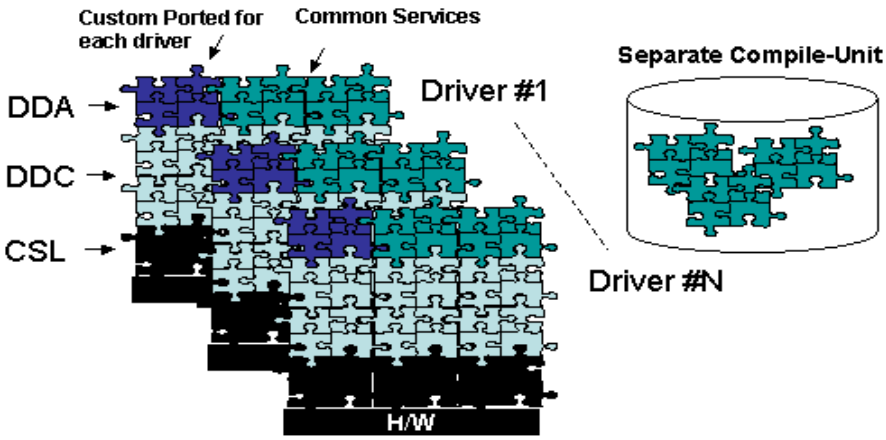


Figure 2 Genesis of PAL-OS

1.3 Design Philosophy of drivers

Central to the philosophy of TI device driver architecture is clarity in separation of roles and responsibilities for the various parts of the device driver. Rather than treat the entire device driver as a monolithic block of code, effort is made to identify the portions of the device driver that are involved in:

- Coupling or handshaking with the specific OS
- Performing primitive, directed read/write access to the h/w device
- Modeling the crux of the driver behavior – protocol, state machine etc this in itself is regardless of any given OS.

With this view, the device driver functionality can be enacted by three key roles defined here under:

- OS Specific Device Driver Adaptation (IOM)
- Device Driver Core, isolated from OS as well as H/W (DDC)
- LLC abstraction providing services to perform primitive access necessary to control/configure/examine status, of the underlying h/w device.

Since there exists a clear separation of roles and responsibilities of the three sub-components of the driver, the prescribed architecture helps in creation of robust device drivers through tested/reusable pieces. Besides, it ensures in maintaining uniform semantics for similar services, supported across different drivers, for different platforms.

The figure below further elucidates the driving philosophy in partitioning the device driver into distinct functional sub-components – IOM, DDC and LLC.

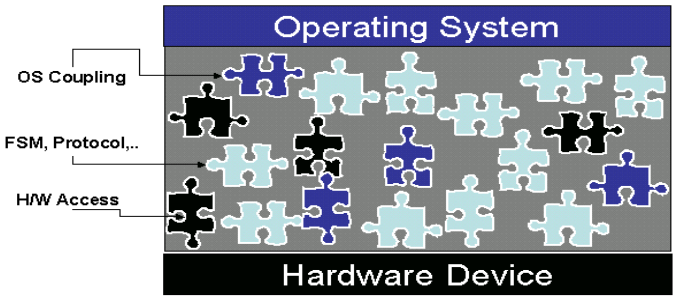


Figure 3 Monolithic Device Driver – Amorphous structure

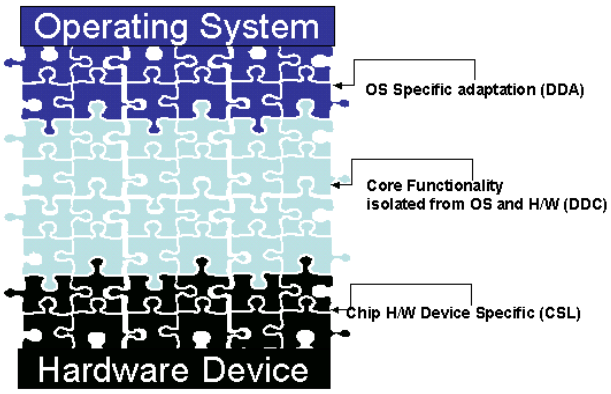


Figure 4 Layered Device Driver – Well defined structure

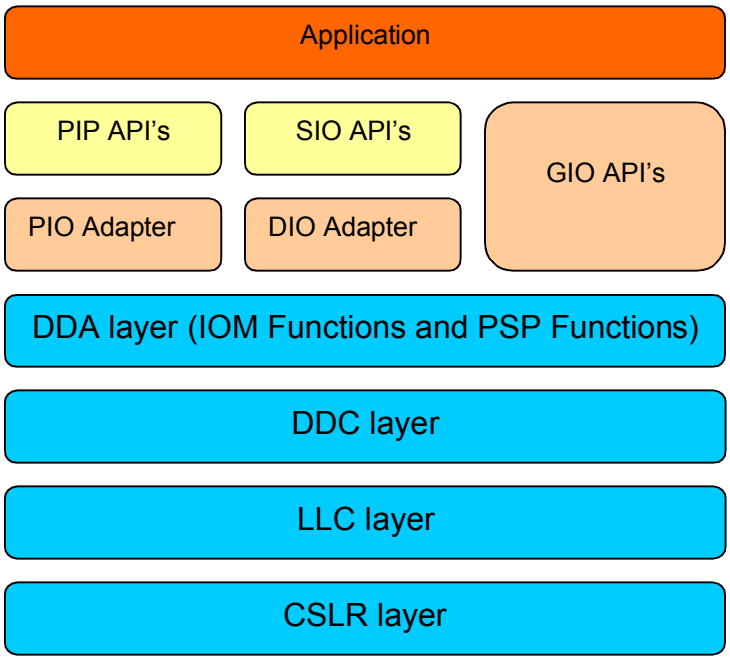


Figure 5 Device driver Architure

1.3.1 Design Goals

The following are the key device driver design goals being factored by proposed TI architecture:

- Uniformly styled drivers, promoting increased reuse and code familiarity to developers
- Minimal overheads in integrating TI device driver into any Software System
- Device driver must support Synchronous as well as Asynchronous interfaces to the user where appropriate
- Device driver must operate both with and without Interrupts capability as appropriate
- Device driver must leverage the H/W DMA capability where available to improve performance of device driver in case of block-oriented transfers.

1.3.2 Assumptions

It is assumed that TI Device Drivers are required to be fully functional in their native OS (DSP/BIOS in case of DSP side devices). Integration of TI Device Drivers into a given software system is beyond the immediate scope of the proposed architecture. However, effort has been made by TI to ease the integration via IOM interfaces discussed later in this document. The VPBE device driver is expected to perform:

- ❖ Flipping of multiple frame buffers for seamless display.
- ❖ The VPBE driver will be using only LLC layer to interact with hardware.
- ❖ The VPBE driver works in interrupt mode only.

1.3.3 Design Principles

The guiding principles for the TI Device Driver design, drawn in the context of a fore mentioned philosophy, goals and constraints can be enumerated as follows:

- ❖ Clear separation of H/W-dependent and H/W-independent parts.
- ❖ Clear separation of OS dependent and independent parts
- ❖ Consistent interfaces across same class of devices, ensured by design
- ❖ Modular design to effectively address multiple device instances – avoids needless code size penalty and also to isolate device specific driver Implementation details from usage policies.

1.3.4 Adding instance of the device driver (Example)

To have device driver included in the application, it is required to add the instance of the device driver in the TCF file of the application.

A TCI file can be used for adding the device driver instance in the TCF file. It contains following field:

1. **init function**: This function will be called before the initialization of drivers. Initializations that have to be done before driver initialization are done in this function
2. **function table ptr**: This is a pointer to a structure that contains all the function pointers to all the driver functions. This pointer should be initialized by a structure that contains function pointers to the functions the peripheral driver
3. **function table type**: Function table type which can be IOM_Fxns or DEV_Fxns
4. **device Id**: Device driver instance number
5. **device params ptr**: Pointer to a structure that will be used while initialization of drivers

The following is the example of the UART driver depicts the way a TCI file shall be made. A TCI file shall consists of the following detail:

```
bios.UDEV.create("UART0");
bios.UDEV.instance("UART0").deviceId = 0;
bios.UDEV.instance("UART0").fxnTableType = "IOM_Fxns";
bios.UDEV.instance("UART0").initFxn =
prog.extern("uartAppInit");
```

```
bios.UDEV.instance("UART0").fxnTable =
prog.extern("UARTMD_FXNS");
bios.UDEV.instance("UART0").params =
prog.extern("Uart_DevParams");
```

Field name	Description
UART0	The name of device driver instance. This name shall be given as input parameter while calling GIO_create for corresponding driver.
deviceId = 0	Refers to instance of device driver for UART0
IOM_Fxns	Type of function table
uartAppInit	Function that will be called to do the initialization required for UART driver
UARTMD_FXNS	pointer to structure containing mini-driver functions for UART driver
UART_DevParams	the configuration parameter for device driver of UART0

This TCI file, containing the driver instance, need to be imported in the TCF file in the following way:

```
utils.importFile("dm6437_uart0.tci");
```

This can be added in the TCF file by opening it text edit mode and then writing the above line in the starting. Here "dm6437_uart0.tci" is the name of the respective TCI file of the device driver that need to be included.

For this file to imported, its path needs to be listed in the projects build option. The import path can be added by adding the path of the TCI file in project's build option-DspBiosBuilder-Basic.

Chapter 2

Installation Guide

This chapter discusses the **DM6437/C6424 DSP/BIOS PSP** installation, how and what software and hardware components to be availed in order to complete a successful installation (and un-installation) of **DM6437/C6424 DSP/BIOS PSP**.

2.1 Installation and Usage Procedure

1. Install the above products as per instructions provided along with the products.
2. Install the PSP package by deflating the compressed package file in any drive (root directory only)
3. Configure UART at 115200 BAUD Rate, with 8 bit Data, No Parity, No Flow Control & Stop Bit equal to 1 on PC side.

Note:

1. The XDCPATH environmental variable should be set properly to recompile the drivers/sample applications. Ensure that the XDCPATH variable should contain, BIOS packages path, XDC packages path, PSP drivers packages path, EDMA packages path.

Example:

```
z:\pspdrivers\pspdrivers_\packages;
C:\CCStudio_v3.3\bios_5_33_01\packages;
C:\Program Files\Texas Instruments\xdctools_3_10_02\packages;
C:\Program Files\Texas Instruments\edma3 lld 01 05 00\packages;
```

2. psp_xdcpaths_common.dat should contain the information about the PSP package installation directory and the EDMA installation directory.
    a. The *"pspRootDir"* string should point to the directory where the PSP package is installed. For example if the PSP packge is installed as *"z:\pspdrivers\pspdrivers_1_10_01\packages"* then the *pspRootDir* should be equal to *"z:/pspdrivers/".* Please note the forward slashes in the path.
    b. The *"pspversion"* string should contain the version number of the current PSP installation. For example, if the version of PSP package installed is 1.10.01, then *pspversion* should be equal to "1_10_01".
    c. *edma3InstallDir* should be set to point to the installation directory of the EDMA package. If EDMA driver package is installed at *"c:\ProgramFiles\TexasInstruments\",* then *edma3InstallDir* should be set to *"c:/ProgramFiles/TexasInstruments/".* Please note that forward slashes should be used here for directory paths.
    d. The first section –Section 1- should be uncommented in case of non DVSDK users and the second section –Section 2- should be commented. If users are using PSP package from inside a DVSDK installation, then the second section – Section 2- should be uncommented and first section – Section 1- should be commented.

Some sample execution steps are mentioned below:

Steps to run sample applications of EDMA, UART, I2C, NAND and VLYNQ on C6424/DM6437 EVM and VPBE and VPBE on DM6437 EVM:

1. For hardware setup, connect the UART cable to DM6437 EVM. Connect XDS 510 USB emulator to the JTAG connector on board. Switch on the power supply for board.

Note: To run VLYNQ sample, hardware setup will consist of two back-to-back connected daughter cards, which should be plugged into mini-PCI connectors of the DM6437 EVM. The daughter cards are having two ports P2 and P3. The FRC cable should be connected to P2 port at one end and P3 port at the other end. The EVM where FRC cable is connect to P2 will act as master VLYNQ device and the other will act as slave VLYNQ device.

2. Open CCS 3.3.38 setup. Select EVMDM6437_XDS510USB processor.
3. Connect to DM6437 processor.
4. Open \system\DM6437\bios\dm6437_evm\dm6437_evm_bios_sample_out.pjt . Build the image and download it to the platform using CCS.
5. Sample applications for the included driver shall automatically start following this logo.

Note: To run VLYNQ test, \system\DM6437\bios\dm6437_evm\dm6437_evm_bios_sample_out.pjt project have to run on both master and slave side. On master side this project will run as it is. But on slave side, psp_bios_vlynq_master_sample.c file have to be removed from vlynq_sample_lib.pjt library and psp_bios_vlynq_slave_sample.c have to be added. psp_bios_vlynq_master_sample.c contains the configuration of VLYNQ require for slave side. Build the code on both the sides and run the master side code first and than slave side code because master will source the VLYNQ clock.

Step to execute McASP sample application:

1. For hardware setup, connect the UART cable to DM6437 EVM. Connect XDS 510 USB emulator to the JTAG connector on board. Switch on the power supply for board.
2. Open CCS 3.3.38 setup. Select EVMDM6437_XDS510USB processor.
3. Connect to DM6437 processor.
4. Open \system\DM6437\bios\dm6437_evm\dm6437_evm_bios_sample_out.pjt .
5. Include .tci file of McASP in dm6437_evm_bios_sample_out.tcf file. Include utils.importFile("dm6437_audio_mcasp.tci"). Make sure that at the same time .tci file dm6437_audio_mcbsp.tci of McBSP is not included.
6. Build the project and load the .out file to the platform using CCS.

7. This application is echo application. This application first records the audio and then plays it back.

Step to execute McBSP sample application:

1. For hardware setup, connect the UART cable to DM6437 EVM. Connect XDS 510 USB emulator to the JTAG connector on board. Switch on the power supply for board.
2. Open CCS 3.3.38 setup. Select EVMDM6437_XDS510USB processor.
3. Connect to DM6437 processor.
4. Open \system\DM6437\bios\dm6437_evm\dm6437_evm_bios_sample_out.pjt .
5. Include .tci file of McBSP in dm6437_evm_bios_sample_out.tcf file. Include utils.importFile("dm6437_audio_mcbsp.tci"). Make sure that at the same time .tci file dm6437_audio_mcasp.tci of McASP is not included.
6. Build the project and load the .out file to the platform using CCS.
7. This application is echo application. This application first records the audio and then plays it back.

Steps for running sample application of PCI driver on DM6437 board:

1. Fit EVM DM6437 card into the PCI slot of a linux machine having linuxkernel 2.6.9(RHEL 4 AS).
2. Connect USB emulator to JTAG port of DM6437 EVM. Connect the USB cable to other PC having CCS on that.
3. Power ON the Linux machine. BIOS of linux machine should not boot.
4. Open CCS 3.3.38 on the PC and try to connect to DM6437 EVM. When the board will be connected to CCS, BIOS of linux machine will boot up.
5. Press Ctrl+Alt+F1 to go to console in linux machine.
6. Enter username and password for "root" user.
7. Copy Makefile and pcidrv.c from "pspdrivers\pal_sys\pci\docs\linuxapp" folder to "/home" directory of linux machine.
8. Go to "/home" directory by giving "cd /home" command.
9. Give "make" command. This will generate the object file of a PCI host driver for Linux. Driver will be generated by module name "pcidriver.ko" in same directory.
10. Give "insmod pcidriver.ko" command. This will insert the driver as a module.

11. This will give some print messages on Linux machine. If the print messages of linux driver for PCI are not visible on the console then give "dmesg" command to see them.
12. There will be a print message saying "Address to write from DSP = 0x518000". Note this address (0x518000).
13. Open \pspdrivers\pal_sys\pci\sample\psp_pci_bios_sample_dm6437.pjt project in CCS. Open \pspdrivers\pal_sys\pci\sample\psp_pci_bios_sample_main.c
14. Build CCS project and run the out file.
15. Out file will generate a host interrupt on Linux machine. On generation of host interrupt there will be print messages saying "Interrupt received from DSP". This indicates that DSP to Host interrupt is generated.
16. Sample application loaded on EVM DM6437 through CCS will also perform read/write test. A pop up will come to fetch the address for read write. Enter the address for read write printed on the consol of Linux host machine noted in step 12.The sample application will perform read write and print the message of success or failure on STDOUT

Steps for running VPFE raw sample application on DM6437 board:

1. For hardware setup, connect XDS 510 USB emulator to the JTAG connector on board. Switch on the power supply for board. Connect video cable from DAC-B of DM6437 EVM to TV.
2. Connect Sandwitch board on DM6437 EVM. Then connect MT9T001 Image Sensor board on Sandwitch board.
3. Open CCS 3.3.38 setup. Select EVMDM6437_XDS510USB processor.
4. Connect to DM6437 processor.
5. Open \pspdrivers\system\DM6437\bios\dm6437_evm\src\video\sample\ rawcapture\build\ dm6437_evm_rawcapture_st_sample.pjt. Build the project and load the .out file to the platform using CCS.
6. Put different objects in front on Camera and observe the black & white image of captured data on TV.

Steps for running VPBE sample application on DM6437 board:

1. For hardware setup, connect XDS 510 USB emulator to the JTAG connector on board. Switch on the power supply for board. Connect video cable from DAC-B of DM6437 EVM to TV.
2. Open CCS 3.3.38 setup. Select EVMDM6437_XDS510USB processor.
3. Connect to DM6437 processor.
4. Open \pspdrivers\system\DM6437\bios\dm6437_evm\src\video\sample\ vpbe\build\ dm6437_evm_vpbe_st_sample.pjt. Build the project and load the .out file to the platform using CCS.
5. Observe shrek image and a line which keeps on scrolling from up to down on TV.

Steps for running VPSS Integrated application on DM6437 board:

1. For hardware setup, connect XDS 510 USB emulator to the JTAG connector on board. Switch on the power supply for board. Connect video cable from DAC-B of DM6437 EVM to TV.
2. Open CCS 3.3.38 setup. Select EVMDM6437_XDS510USB processor.
3. Connect to DM6437 processor.
4. Open \pspdrivers\system\DM6437\bios\dm6437_evm\src\video\sample\video_integration \build\dm6437_evm_bios_vpss_integration_sample_out.pjt. Build the project and load the .out file to the platform using CCS.

There are two execution paths in application.

1) VPFE-> Previewer -> Resizer -> VPBE. This is loopback path and captured image will be displayed on TV. To enable this path, PSP_VIDEO_PATH_ENABLE macro must be undefined in psp_vpfe.h.
2) In the second path, H3A and Histogram modules' functionalities are demonstrated, and these modules will generate statistics. Application will read these statistics and will dump into files. (As file operation will take much time, for every 100 frame only one time statistics will be dump into file.). To enable this path, PSP_VIDEO_PATH_ENABLE macro must be defined in psp_vpfe.h.

Note: - To know the interpretation of statistics files of H3A and histogram, please refer Application notes of respective module.

Steps for running sample application of Previewer, Resizer, H3A and Histogram drivers on DM6437 board:

Please refer to module specific User Guide and Application note for installation steps and application usage.

2.2 Un-Installation

1. Delete the <Drive>:\psp_1_00_00_01 directory to remove DM6437 PSP package.
2. Un-install the products (listed in system requirements) as per instructions provided with the product.
3. pcidriver.ko can be un-installed from Linux machine by giving "rmmod pcidriver.ko" command on console of Linux machine (Applicable for PCI Sample Only).

2.3 PSP Component Folder

This section details the files and directory structure of the installed **DM6437/C6424 DSP/BIOS PSP** in the system. A viewgraph of the actual directory tree (as seen in the final deployed environment is inserted here for clarity.

2.3.1 Top level PSP Directory structure:

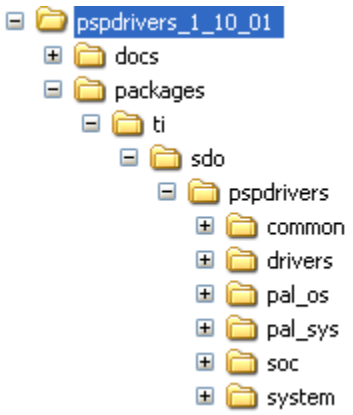


Figure 6 PSP Top level directory structure

The directories of interest (for this release) are:

\pspdrivers_1_10_00_10\docs – Documents
..\packages\ti\sdo\pspdrivers\drivers - Drivers
..\packages\ti\sdo\pspdrivers\inc – Include files used by all the drivers
..\packages\ti\sdo\pspdrivers\pal_os - PAL OS component
..\packages\ti\sdo\pspdrivers\pal_sys – PAL SYS component
..\packages\ti\sdo\pspdrivers\system – DM6437/C6424 sample Application
..\packages\ti\sdo\pspdrivers\soc – CSLRs for DM6437 SOC

2.3.2 PAL OS for BIOS Directory structure:

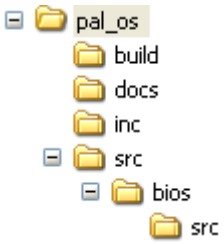


Figure 7 PAL OS directory structure

..\pal_os\inc - PAL OS header files
..\pal_os\src\bios\src - Implementation of PAL OS for BIOS
..\pal_os\docs – documents for PAL OS

2.4 CCS Projects

The following CCS projects are provided as part of the PSP package:

DM6437/C6424 Sample project for BIOS Sample Application (Executable build):

- \pspdrivers\system\DM6437\bios\dm6437_evm\build\dm6437_evm_bios_sample_out.pjt

  This project builds the sample code for UART, I2C, NAND, McASP, McBSP, VPBE and VPFE

- \pspdrivers\pal_sys\pci\sample\psp_pci_bios_sample_dm6437.pjt

  This project builds the sample code for PCI

- \pspdrivers\system\DM6437\bios\dm6437_evm\src\video\sample\video_integration\build\dm6437_evm_bios_vpss_integration_sample_out.pjt

  This project builds the integrated code for VPFE, VPBE, Previewer, Resizer, H3A and Histogram

- \pspdrivers\drivers\i2c\sample\build\dm6437_evm_i2c_st_sample.pjt

  This project builds the I2C sample application. Similarly, sample applications for UART and NAND modules can be found at \pspdrivers\drivers\<peripharal name>\sample\build\dm6437_evm_<peripharal name>_st_sample.pjt folder.

- \pspdrivers\system\DM6437\bios\dm6437_evm\src\video\sample\Previewer\build\ psp_bios_prev_st_basic_example.pjt

  This project builds the Previewer basic sample application. Similarly, sample applications for VPFE, VPBE, Resizer, H3A and Histogram modules can be found at \pspdrivers\system\DM6437\bios\dm6437_evm\src\video\sample folder.

- \pspdrivers\system\DM6437\bios\dm6437_evm\src\audio\sample\build\ dm6437_evm_audio_st_sample.pjt

  This project builds the audio sample application.

***Note**: When loading the BIOS .PJT files, ignore the warning/error indication that .CMD files are missing. The .CMD files are auto-generated (from .TCF file) when the project is built and added to the link process.*

***NOTE**: To test VPFE and VPBE Device Driver configure the setting of PLL1 and PLL2 in high speed mode.*

Chapter 3

Integration Guide

This chapter discusses the **DM6437/C6424 DSP/BIOS PSP** package usage.

3.1 Application Usage of PSP

3.1.1 Demo Application

As part of the PSP package, a demo application is provided to do the following:

1. Display the PSP logo on UART Console
2. Run the sample code for the drivers

NOTE: In case if cache is enabled and driver is configured in EDMA mode or driver uses EDMA, application must provide cache line size (current configuration of cache line size is 64 – bytes) aligned address to the driver.

NOTE: Binary files (.out) of sample application are available in "bin" folder located in its respective sample application package.

3.1.2 I2C Driver usage example

Sample code is provided to demonstrate the usage of I2C driver. This sample demonstrates usage of I2C driver by blinking LEDs on DM6437 EVM through IO expander.

3.1.3 UART Driver usage example

UART driver sample code demonstrates read/write operations of 64-bytes.

3.1.4 McASP Driver usage example

Sample code of McASP demonstrates basic audio record and playback functionality.

3.1.5 McBSP Driver usage example

Sample code of McBSP demonstrates basic audio record and playback functionality. This code uses SIO interface to demonstrate this.

3.1.6 NAND Driver usage example

NAND sample application demonstrates basic sector read/write operation for NAND.

3.1.7 EDMA3 Driver usage example

Sample code is provided to demonstrate the usage of EDMA driver. EDMA has been tested in simple memory to memory copy on DM6437 EVM board. The application demonstrates the copying of data through DMA channel and QDMA channel.

3.1.8 PAL SYS VLYNQ Driver usage example

VLYNQ sample application demonstrates the read/write operation on back to back connect DM6437 EVMs through VLYNQ connectors. The sample first detects the VLYNQ link and than writes data on to the peer SOC and read data from the peer SOC.

3.1.9 PAL SYS PCI driver usage example

PCI driver sample application demonstrates read/write operation across PCI bus. This application has to be run on a DM6437 EVM fit in PCI slot of LINUX machine. By running a same program on LINUX machine a memory mapped area on the host side can be obtained. This memory mapped address can be used by the sample application of DM6437 EVM for read/write operation.

3.1.10 VPFE sample application (Only for DM6437)

dm6437_evm_rawcapture_st_sample.pjt : This application is loopback application which captures the raw data,convert this raw data into black & white logic and give it to display driver(VPBE).

3.1.11 VPBE sample application (Only for DM6437)

dm6437_evm_vpbe_st_sample.pjt : This applcation displays the shrek image on Video-0 window continously for 30 seconds.

3.1.12 VPSS Integrated application (Only for DM6437)

dm6437_evm_bios_vpss_integration_sample_out.pjt: This application is loopback application which demonstrates VPFE,

Previewer, Resizer, H3A, Histogram and VPBE driver functionalities.

3.1.13 VPFE-Previewer On-the-fly sample application (Only for DM6437)

There are two sample applications to demonstrate Previewer On-the-fly and Dark frame capture/subtract features:

- psp_bios_prev_st_on_the_fly_example.pjt: This application is loopback application which demonstrates VPFE, Previewer On-the-fly and VPBE driver functionalities.
- psp_bios_prev_st_on_the_fly_dfc_example.pjt: This application is loopback application which demonstrates dark frame capture and dark frame subtract functionality.

3.1.14 Previewer driver usage example (Only for DM6437)

There are two sample applications to demonstrate Previewer driver features:

- psp_bios_prev_st_basic_example.pjt: This application demonstrates basic previewing functionality. This application is tested on DM6437 EVM board. This application converts raw image in Bayer pattern to YUV4:2:2 image.
- psp_bios_prev_st_multipass_example.pjt: This application demonstrates multipass functionality to preview image of horizontal size greater than 1280 pixels. This application is tested on DM6437 EVM board. This application converts raw image in Bayer pattern to YUV4:2:2 image.

3.1.15 Resizer driver usage example (Only for DM6437)

There are three sample applications to demonstrate Resizer driver features:

- psp_bios_resz_st_downscale_example.pjt: This application demonstrates downscaling feature of Resizer. It resizes 640x480 image to 320x240 image. This application is tested on DM6437 EVM board.
- psp_bios_resz_st_upscale_example.pjt: This application demonstrates upscaling feature of Resizer. It resizes 320x240 image to 640x480 image. This application is tested on DM6437 EVM board.
- psp_bios_resz_st_multipass_example.pjt: This application demonstrates multipass functionality. It downscales 1280x640 image to 128x64 image. This application is tested on DM6437 EVM board.

3.1.16 H3A driver usage example (Only for DM6437)

There are two sample applications to demonstrate H3A driver features:

- psp_bios_af_st_example.pjt: This application demonstrates auto focus functionality of H3A. This application is tested on DM6437 EVM board.
- psp_bios_aew_st_example.pjt: This application demonstrates auto exposure and white balancing functionality of H3A. This application is tested on DM6437 EVM board.

3.1.17 Histogram driver usage example (Only for DM6437)

psp_bios_hist_st_sample.pjt: This sample application demonstrates basic functionality of Histogram. This application is tested on DM6437 EVM board.

3.2 Instrumentation Tool - SoC Analyzer usage

3.2.1 Pre-requist installations for instrumentation

1. Install "SOC analyzer" by running dvt.exe

2. Install **DVSDK** targeted for DM6437 to setup the log server. Source for this is DM6437_DVDP_setupwin32_1_01_00_11.exe and above.

3. Install setup_evmdm6437_v1.exe to install the board specific utilities and BSL for DM6437. This will create a boards\evmdm6437_v1 folder in CCS install directory.

4. Please build Debug & Release profiles before building iDebug & iRelease profiles.

5. Ensure that the following as environment variables are correctly set to current installations paths.

   Some sample settings are shown below –
   BIOS_INSTALL_DIR: C:\CCStudio_v3.3\bios_5_31_06
   BIOSDVSDK_INSTALL_DIR: C:\dvsdk_1_01_00_11
   BIOSUTILS_INSTALL_DIR: % BIOSDVSDK_INSTALL_DIR %\biosutils_1_00_02
   BSL_EVMDM6437_INSTALLDIR = C:\CCStudio_v3.3\boards\evmdm6437_v1

   Note: These settings are just indicative. Please ensure the paths match your installation setup.

3.2.2 INSTRUMENATION IN SAMPLE PROJECT

1. Include the following files in application project that uses driver library.

   - %BIOSUTILS_INSTALL_DIR%\packages\ti\bios\log\ndk\examples\evmdm6437\evmdm6437init.c
   - %BIOSUTILS_INSTALL_DIR%packages\ti\bios\log\ndk\examples\common\logserverstacksetup.c

2. Include the following Libraries in sample project of driver.

   - %BSL_EVMDM6437_INSTALLDIR%\lib\evmdm6437bsl.lib
   - %NDK_INSTALL_DIR%\packages\ti\ndk\lib\c64plus\hal\hal_ser_stub.lib
   - %NDK_INSTALL_DIR%\packages\ti\ndk\lib\c64plus\hal\hal_timer_bios.lib
   - %NDK_INSTALL_DIR%\packages\ti\ndk\lib\c64plus\miniPrintf.lib
   - %NDK_INSTALL_DIR%\packages\ti\ndk\lib\c64plus\netctrl.lib
   - %NDK_INSTALL_DIR%\packages\ti\ndk\lib\c64plus\nettool.lib
   - %NDK_INSTALL_DIR%\packages\ti\ndk\lib\c64plus\os.lib
   - %NDK_INSTALL_DIR%\packages\ti\ndk\lib\c64plus\stack.lib
   - %NDK_INSTALL_DIR%\packages\ti\ndk\lib\hal\evmdm6437\hal_eth_dm64lc.lib

- %NDK_INSTALL_DIR%\ packages\ti\ndk\lib\c64plus\hal\hal_userled_stub.lib

3. Change "Debug" build configuration as per the following.

   Add the following compiler options, include file paths and include library paths and DSP-BIOS builder settings.

   Compiler options:

   -pds238

   Include paths:

   Open Build options of sample project. Go to "Compiler" tab. Select the "Preprocessor" category. Add the following paths in "include search path(-i)" field.

   -i"%NDK_INSTALL_DIR%\packages\ti\ndk\inc"
   -i"%NDK_INSTALL_DIR%\packages\ti\ndk\example\tools"
   -i"% BIOSDVSDK_INSTALL_DIR%\boards\evmdm6437_v1\include"
   -i"%BIOSUTILS_INSTALL_DIR%\packages\ti\bios\log\ndk"
   -i"%BIOSUTILS_INSTALL_DIR%\packages\ti\bios\log\support"
   -i"%NDK_INSTALL_DIR%\packages\ti\ndk\inc"
   -i"%BIOS_INSTALL_DIR%\packages"
   -i"%BSL_EVMDM6437_INSTALLDIR%\include"
   -i"%BIOSUTILS_INSTALL_DIR%\packages"

   "DspBiosBuilder" Settings

   Open Build options of sample project. Go to "DspBiosBuilder" tab. Select the "Basic" category. Add the following paths in "import path" field.

   %NDK_INSTALL_DIR%\packages\ti\ndk\inc\tci;
   %BIOSUTILS_INSTALL_DIR%\packages\ti\bios\utils;
   %BIOSUTILS_INSTALL_DIR%\packages\ti\bios\log\support;
   %BIOSUTILS_INSTALL_DIR%\packages\ti\bios\log\ndk\examples\common"

   "Linker" Settings:

   Open Build options of sample project. Go to "Linker" tab. Select the "Library" category. Add the following paths in "Search Path(-i)" field.

   -i"%BIOS_INSTALL_DIR%\packages\ti\rtdx\lib\c6000"
   –i"%BIOSUTILS_INSTALL_DIR%\packages\ti\bios\log\ndk\lib"
   -i"%BIOSUTILS_INSTALL_DIR%\packages\ti\bios\log\support\lib"
   -i"%BIOSUTILS_INSTALL_DIR%\packages\ti\bios\utils\lib"

   Add the following paths in "Incl. Libraries(-l)" field.

   -l"logsupport.a64P"
   -l"logservercgi.a64P"
   -l"utils.a64P"

4. Create "iDebug" build configuration. Keep the compiler options, include file paths, include library paths and DSP-BIOS builder settings same as "Debug" configuration.

   Add the following lines in pjt of sample project to create "iDebug" configuration.

   Config="iDebug"
   ["Compiler" Settings: "iDebug"]

5. Repeat step 3 and 4 for release mode to generate iRelease configuration.

6. Do the following changes in TCF file of application project.

   - Remove "trace" object of LOG to avoid multiple definitions.
   - Add following lines in application project's TCF file:

     ```
     bios.GBL.CALLUSERINITFXN = 1;
     bios.GBL.USERINITFXN = prog.extern("dm6437_init");
     ```

     This will set "dm6437_init" as a global initialization function. This function will do the initialization required for network stack setup and read MAC address of EVM DM6437 board

   - Add the following lines at the start of application project TCF file.

     ```
     utils.importFile('ndk.tci');
     utils.importFile('Load.tci');
     utils.importFile('LogTrack.tci');
     utils.importFile('logserverexample.tci');
     ```

   - Increase the buffer size of "LOG_system" buffer = 262144, "DVTEvent_Log" buffer = 65536 and change the segment of both these buffers to DDR2 instead of IRAM setting following values in the TCF file:

     ```
     bios.LOG.instance("LOG_system").bufLen = 262144;
     bios.LOG.instance("LOG_system").bufSeg = prog.get("DDR2");
     ```

     Do the same for "DVTEvent_Log".

     ```
     bios.LOG.instance("DVTEvent_Log").bufSeg = prog.get("DDR2");
     bios.LOG.instance("DVTEvent_Log").bufLen = 65536;
     ```

7. Do the following changes in main sample application of project.

   - Add the following code in the main function(starting point) of application project. This is for initializing log server.

     ```
     LogAux_init();
     LogTrack_init();
     TRC_disable(TRC_LOGCLK);
     ```

   - Include the following files in main sample application of driver.

```
#include <LogServerCgi.h>
#include <LogAux.h>
#include <LogTrack.h>
```

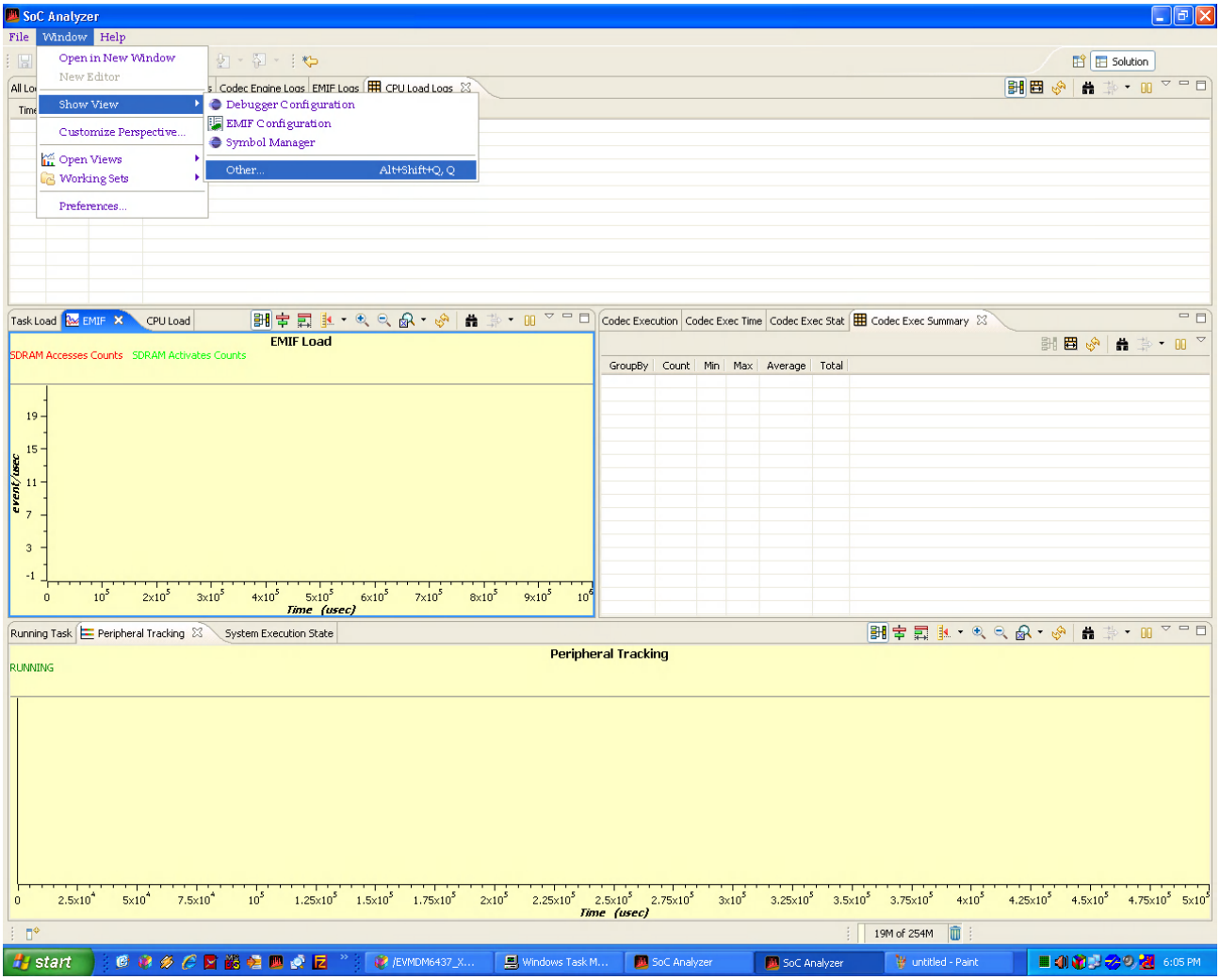
- Add the following code in file containing main function of application project.

```
extern LOG_Obj logTrace;
extern LOG_Obj logTest;

/*
 *  logTracePrd
 *
 *  This fxn runs periodically and continuously updates the
 *  logTrace log.
 *
 */
Void logTracePrdfxn()
{
   LOG_printf4(&logTrace, "logTrace data: %c %c %c %c\n", 'p', 'q', 'r', 's');
}

/*
 *  logTestPrd
 *
 *  This fxn runs periodically and continuously updates the
 *  logTest log.
 *
 */
Void logTestPrdfxn()
{
   LOG_printf4(&logTest, "logTest data: %c %c %c %c\n", 'h', 'i', 'j', 'k');
}
```

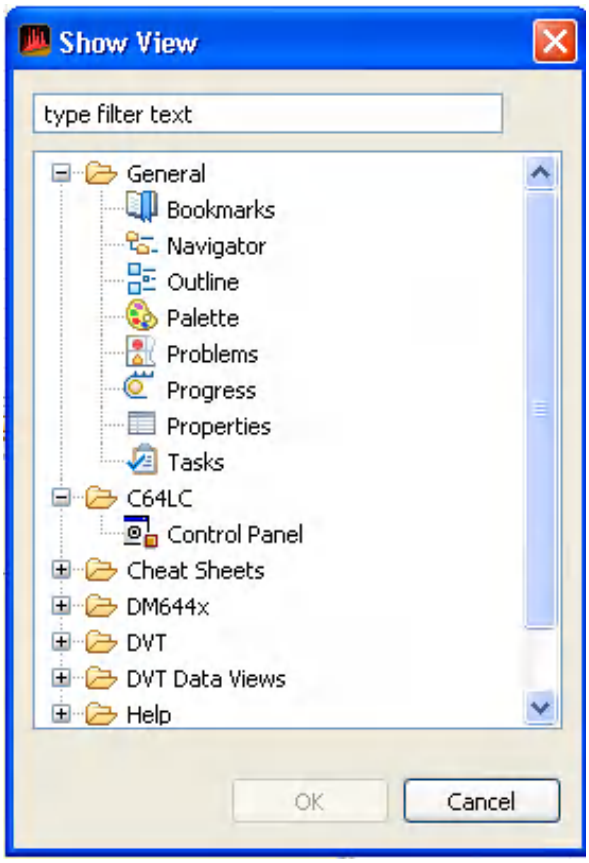
8. Build the application project in iDebug or iRelease mode and run the out file. This will print the IP address of EVM DM6437. Make a note of this IP address. Make sure that network cable is plugged in the EVM and connected to proper LAN port.

3.2.3 ANALYZING INFORMATION

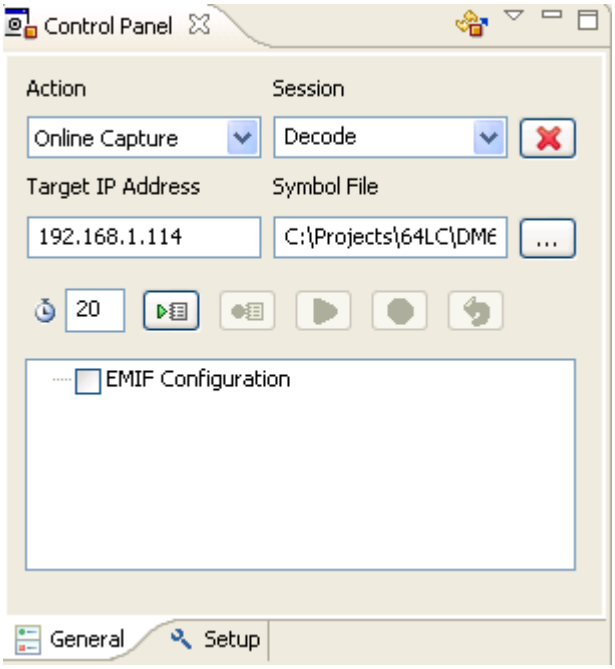
1. Open SOC Analyzer. At the first use, the SoC Analyzer will start-up with a blank screen. In order to capture and visualize data, certains settings have to be done for SOC Analyzer tool. The settings have to be done in Control panel of SOC Analyzer. To open control panel of SOC Analyzer go to Window->Show View->Other.. as shown in following figure.



Select the appropriate Control Panel for the platform.



2. In control panel tab, set "Action" to "Online Capture" and set "Session" to "Decode". Set the IP address note while running the application out file in "Target IP Address" field and set the path of OUT file that is running on EVM DM6437 in "Symbol File" field as show in the following figure. Set the "Number of Seconds to Capture" filed to "60" which is set to "20" in the following figure.



3. Click on the "Start Capturing Logs on Target"  button. By pressing this button SOC analyzer will start capturing data.

4. To see various graphs go to Window->Open Views select the appropriate option.

NOTE: The CPU occupancy observed in the logs will be 20-30% due to sample application behavior of the log server example available with BIOS Utils.

3.3 Interrupt configurations in TCF file

Following lines need to be there in tcf file for enabling hardware interrupt.

bios.ECM.ENABLE = 1;

ECM configuration – following settings needs to be made manually to reflect these settings available in soc.h.

```
bios.HWI.instance("HWI_INT7").interruptSelectNumber = 0;
bios.HWI.instance("HWI_INT8").interruptSelectNumber = 1;
bios.HWI.instance("HWI_INT9").interruptSelectNumber = 2;
bios.HWI.instance("HWI_INT10").interruptSelectNumber = 3
```

Chapter 4

Resource Guide

This chapter discusses the **DM6437/C6424 DSP/BIOS PSP** package resources.

Drivers	EDMA Resources	Event Combiner Used
h3a (Only for DM6437)	None	Yes
histogram (Only for DM6437)	Yes (1 EDMA Channel)	Yes
i2c	None	Yes
mcasp	Yes (2 EDMA Channels plus 4 Param Entries)	Yes
mcbsp	Yes (2 EDMA Channels plus 4 Param Entries)	Yes
nand	Yes (One EDMA Channel)	Yes
previewer (Only for DM6437)	None	Yes
resizer (Only for DM6437)	None	Yes
uart	Yes (2 EDMA Channels)	Yes
vpbe (Only for DM6437)	None	Yes
vpfe (Only for DM6437)	None	Yes
vlynq	None	Yes
pci	None	Yes
edma3	NA	Yes

All event combiner interrupts are tied to IRQ – 7,8,9 & 10 HWI Interrupts.

These can be changed in the sample TCF files & also refelect the changes in the file "soc.h" available at "<install path>\pspdrivers\soc\dm6437\dsp\inc".

PIN Mux Details:

Sample Pin Mux settings have been provided in all sample application for each peripheral. The details of PIN MUX is available as part of DM6437 Technical Reference Manual.

TCI Files:

All sample TCI files are available in respective Driver Sample Directories. These TCI files does the creation of Devices & Register IOM Function Tables. A standalone application & TCF file has been provided along with the package which indicates the minimum required TCI files to be used.

An example for Audio Applications; the required TCI files are dm6437_i2c0.tci & dm6437_audio_mcbsp.tci as indicated in the TCF file "dm6437_evm_audio_st_sample.tcf".