i.MX31 PDK 1.4 Windows CE 5.0

Reference Manual

Document Number: 926-77198
Version 1.4
10/2008

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064, Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor China Ltd.
Exchange Building 23F, No. 118 Jianguo Road
Chaoyang District
Beijing 100022, China
+86 010 5879 8000
support.asia@freescale.com

***For Literature Requests Only*:**
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-521-6274 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Contents

About This Book

Chapter 1 Introduction

Chapter 2 ATA Driver

Chapter 3
Audio Driver

Chapter 4
Backlight Driver

Chapter 5
Battery Driver

Chapter 6
Bluetooth Driver

Chapter 7
Camera Driver

Chapter 8
Chip Support Package Driver Development Kit (ARM11 CSPDDK)

Chapter 9
Display Driver

Chapter 10
Dynamic Voltage and Frequency Control (DVFC)

Chapter 11
Ethernet Boot Loader

Chapter 12
FM Radio Driver

Chapter 13
General Purpose Timer Driver

Chapter 14
Global Positioning System Driver

Chapter 15
Hantro Codecs Drivers

Chapter 16
Inter-Integrated Circuit (I^2C) Driver

Chapter 17
Keypad Driver

Chapter 18
LAN9217 Product Ethernet Driver

Chapter 19
MBX Direct3D Mobile/OpenGL ES Drivers

Chapter 20
NAND Flash Media Driver (FMD)

Chapter 21
Postfilter Driver

Chapter 22
Power Manager

Chapter 23
Power Management IC (PMIC)

Chapter 24
Secure Digital Host Controller

Chapter 25
Serial Driver

Chapter 26
Touch Panel Driver

Chapter 27
USB OTG Driver

Chapter 28
WLAN Driver

Appendix A 3-Stack
Frequently Asked Questions

Tables

Figures

About This Book

This reference manual describes the requirements, implementation, and testing for the modules included in Freescale's software development kit (SDK) for Microsoft® Windows® CE 5.0.

Audience

This document is intended for device driver developers, application developers, and test engineers who are planning to use the product. This document is also intended for people who want to know more about Freescale's software development kit (SDK) for Microsoft Windows CE 5.0.

Suggested Reading

Freescale documentation is available from the sources listed on the back cover of this manual.

- *Microsoft Platform Builder Help* may be viewed from within the platform builder application. The *Microsoft Platform Builder User Guide* can be viewed from the Microsoft Developer Network at http://msdn.microsoft.com. Visit the web site and search for the string, "Platform Builder User's Guide."
- *i.MX31 Applications Processor IC Reference Manual*
- *i.MX31 PDK Release Notes for Windows CE 5.0*
- *i.MX31 PDK Windows CE 5.0 User's Guide*
- *Microsoft Platform Builder for Windows CE 5.0 User Guide*
- *Microsoft Platform Builder for Windows CE 5.0 Help*
- *Windows CE 5.0 BSP Reference Guide (Order number RTM13)*
- *MCIMX31 and MCIMX31L Applications Processors Reference Manual*

The Freescale manuals can be also be found at http://www.freescale.com. The manuals can be downloaded directly from the web, or you can also order the printed copies. The Freescale manuals may also be provided with your PDK.

Conventions

This document uses the following conventions:

- `Courier` is used to identify commands, explicit command parameters, code examples, expressions, data types, and directives.
- **Bold** indicates the menu options or buttons the user can select. Cascaded menu options are delimited with the → symbol.
- *Italic* is used for emphasis, to identify new terms, and for replaceable command parameters.

Acronyms and Abbreviations

Table i contains acronyms and abbreviations used in this document.

Table i. Acronyms and Abbreviated Terms

Term	Meaning
API	Application programming interface
BSP	Board support package
CSP	Chip support package
CSPI	Configurable serial peripheral interface
D3DM	Direct 3D Mobile
DHCP	Dynamic host configuration protocol
DPTC	Dynamic power and temperature control
DVFC	Dynamic voltage and frequency control
DVFS	Dynamic voltage and frequency scaling
EBOOT	Ethernet bootloader
FAL	Flash abstraction layer
FIR	Fast infrared
FMD	Flash media driver
GDI	Graphics display interface
GPT	General purpose timer
I2C	Inter-integrated circuit
IDE	Integrated development environment
IPU	Image processing unit
IST	Interrupt service thread
KITL	Kernel independent transport layer
LVDS	Low-voltage differential signaling
MAC	Media access control
MMC	Multimedia cards
NLED	Notification Light Emitting Diode
OAL	OEM adaptation layer
OEM	Original equipment manufacturer
OS	Operating system
OTG	On-the-go
PMIC	Power management IC
PQOAL	Production quality OEM adaptation layer

Table i. Acronyms and Abbreviated Terms

Term	Meaning
PWM	Pulse-width modulation
SD	Secure digital cards
SDC	Synchronous display controller
SDHC	Secure digital host controller
SDIO	Secure digital I/O and combo cards
SDK	Software development kit
SDRAM	Synchronous dynamic random access memory
SIM	Subscriber identification module
SIR	Slow infrared
SoC	System on a chip
UART	Universal asynchronous receiver transmitter
USB	Universal serial bus

Chapter 1
Introduction

This Freescale i.MX31 PDK Windows CE-based product development kit (PDK) helps speed development of applications for the Freescale i.MX31 3-Stack multimedia applications processor. It includes the following:

- Software development kit (SDK), which includes tools, BSP, codecs, basic middleware, and applications
- Hardware board (i.MX31 3-Stack board)
- Documentation

This kit supports the Microsoft Windows CE 5.0 operating system, and requires the use of Microsoft Platform Builder, which is an integrated development environment (IDE) for building customized embedded OS designs. To view feature information, see the *i.MX31 PDK Release Notes*.

NOTE

Use this guide in conjunction with Microsoft Windows Platform Builder Help (or the identical *Platform Builder User Guide)*.

- To view the Platform Builder Help, click **Help** from within the Platform Builder application.
- To view the *Platform Builder User Guide*, visit: http://msdn.microsoft.com.

1.1 Getting Started

For instructions on installing the SDK, building, and downloading and running the OS image on the hardware board, refer to the *i.MX31 PDK 1.4 Windows CE 5.0 User's Guide* included with this distribution.

1.2 SDK System Architecture

Figure 1-1 shows the SDK system, which consists of tools, bootloader, BSP layer, a middleware and core OS service layer, and an application layer.

1.2.1 Tools and Bootloader

The PDK tools and boot loader consist of the following components.

Tools	Flashing tool: support image download and flashing from UART Firmware upgrade tool: support firmware upgrade with an application
EBOOT	
UBOOT	

1.2.2 BSP Layer

The BSP layer contains the following components.

ATA	GPT	Hantro Codec	SDHC
Audio DAC	Display LCD	I2C	Touch Panel
Backlight	Display TV-Out	KPP	USB
Battery	DVFC	MBX D3DM&OpenGL	Wi-Fi®
Bluetooth	Ethernet	NAND FMD	
Camera	FM Radio	PMIC	
DDK	GPS	Serial	

1.2.3 Middleware and Core OS Service Layer

The middleware and core OS service layer contains the following components.

Multimedia Codec & DMO (optional)	AAC Codec DMO, MPEG4 Codec DMO, WMA Codec, H264 Codec DMO, MP3 Codec DMO, DivX Codec DMO
Power Management	

1.2.4 Application Layer

The application layer contains the following components.

FM Radio application	TV-Out application
Camera application	ImgBurnGUI application
3D Demo application	

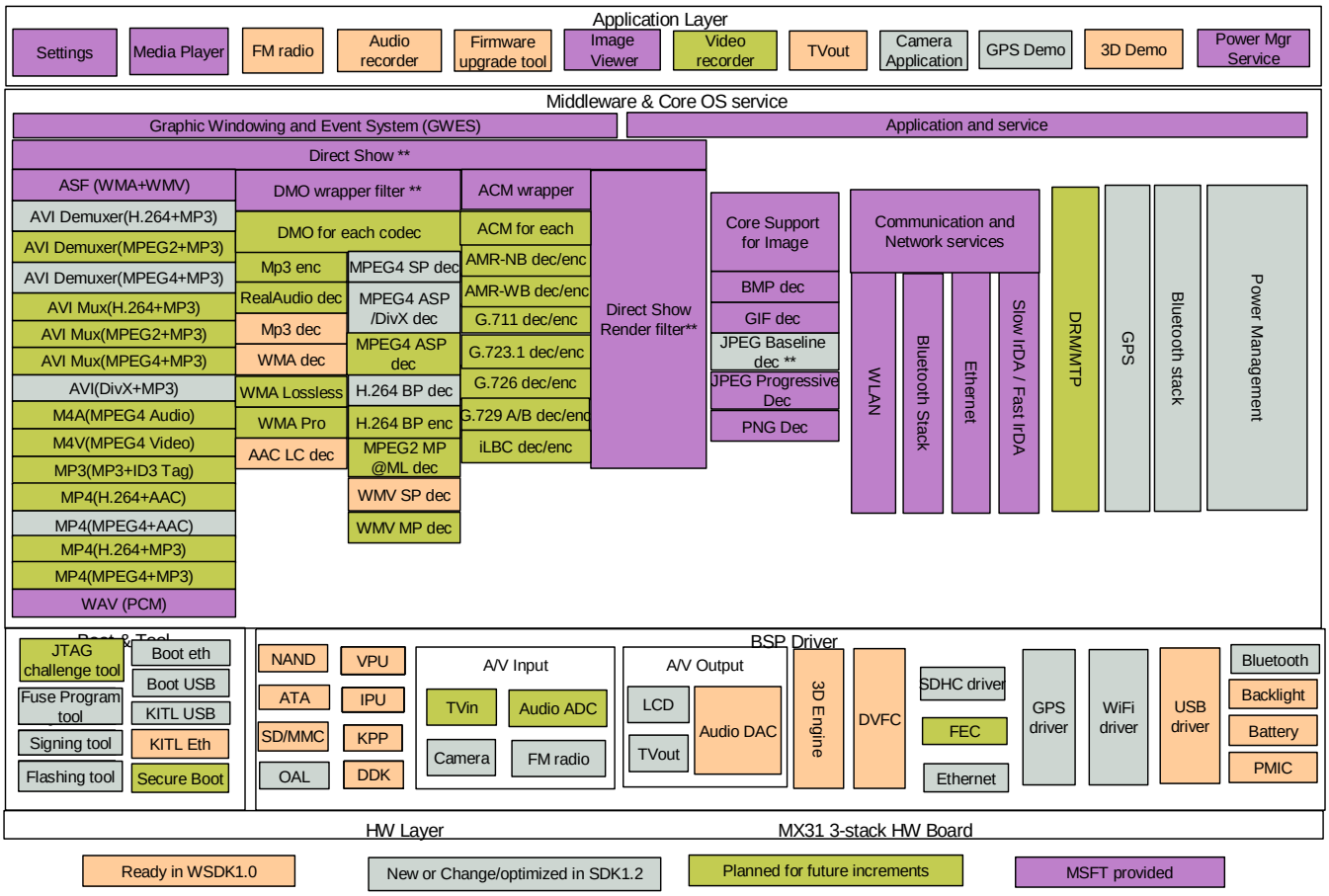

Figure 1-1. WSDK System Diagram

1.3 Windows CE Architecture

The Windows CE architecture is a variation of Microsoft's Windows operating system for minimalistic computers and embedded systems. The architecture of the operating system and sub-systems (for example, power management, DirectDraw, and so on) are described in several locations in the Platform Builder Help. For a complete description of the architecture, see the Platform Builder Help or *Platform Builder User Guide*. You may want to begin at the following location in Help:

Welcome to Windows CE 5.0 > **Windows CE Architecture**.

Chapter 2
ATA Driver

The Windows CE 5. 0 advanced technology attachment (ATA) driver is a block driver, used as the lower layer for such items as File Systems and USB mass storage. The ATA driver is constructed as a stream interface driver that exposes I/O control codes (IOCTL_DISK_XXX and DISK_IOCTL_XXX). The file system uses these I/O control codes to access the ATA devices.

2.1 ATA Driver Summary

Table 2-1 lists the source code location, library dependencies, and other BSP information.

Table 2-1. ATA Driver Summary

Driver Attribute	Definition
Target Platform (TGTPLAT)	3DS
Target SOC (TGTSOC)	MX31
CSP Driver Path	..\CSP\ARM\FREESCALE\<*TGTSOC*>\DRIVERS\ATA
CSP Static Library	ata_<*TGTSOC*>.lib
Platform Driver Path	..\PLATFORM\<*tgtplat*>\SRC\DRIVERS\BLOCK\ATA
Import Library	(cspddk.lib)
Driver DLL	ata_<*TGTSOC*>.dll
Catalog Item	Third Party > BSPs > Freescale <*tgtplat*>: ARMV4I > Storage Drivers > ATA device driver
SYSGEN Dependency	SYSGEN_MSPART,SYSGEN_FATFS
BSP Environment Variable	BSP_ATA

2.2 Supported Functionality

The ATA driver enables the 3-Stack board to provide the following software and hardware support:

- Provides standard Microsoft Block Storage Device API, including disk information management, formatting, block data read/write with full scatter-gather buffer support
- Supports two power management modes, full on and full off
- Supports standard bus timing mode for UDMA mode 3 (optional support for other modes, such as PIO modes 0-4, MDMA modes 0-2, and UDMA modes 0-2 and 4-5)
- Supports full sustained (media) data throughput capacity of Hitachi TravelStar C4K40 (or equivalent) at UDMA mode 3

2.3 Hardware Operation

For operation and programming information, see the chapter on the Advanced Technology Attachment (ATA) in the *MCIMX31 and MCIMX31L Applications Processors Reference Manual*.

NOTE

The i.MX31 contains an on-chip ATA controller. Data transfers on the ATA bus can take place through the following:

- CPU programmed data transfers through ATA controller registers. (Programmed I/O modes, “PIO” modes 0-4)
- “Multi-word” DMA (MDMA modes 0-2)
- “Ultra” DMA (UDMA modes 0-5)

Within the types of ATA-bus data transfer (PIO or XDMA), the various “modes” (0-*n*) refer only to specified combinations of timing parameters, as supported by industry standard hardware. The ATA DMA modes transport the data between the ATA peripheral (disk) and the system bus, through the ATA peripheral data FIFO on the i.MX31.

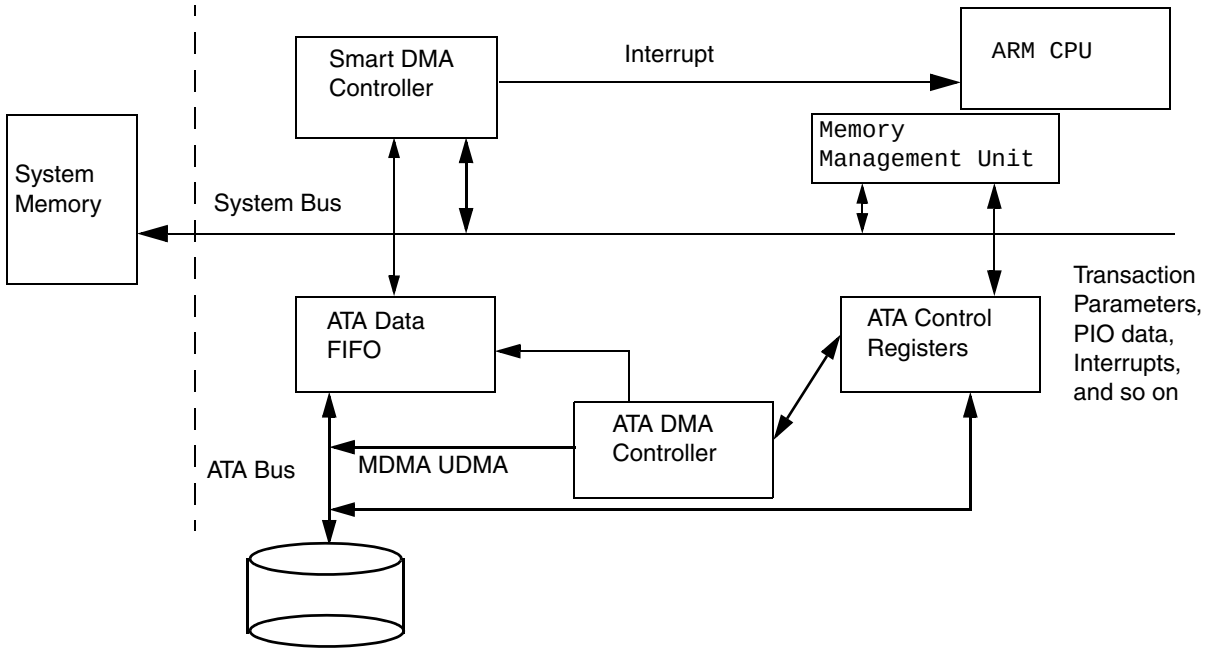

Figure 2-1. ATA Hardware Block Diagram

The i.MX31 also contains a “Smart DMA controller” (SDMA) that acts as a third-party bus mastering DMA, for transporting data between the ATA data FIFO and system memory. SDMA support is built in to the ATA driver, and is automatically configured and used when UDMA or MDMA modes are selected for data transport on the ATA bus. The default block/sector size is 512 bytes. With these sector sizes, far greater efficiency in processor/bus usage is gained by setting UDMA or MDMA modes, instead of PIO modes. The PIO modes are provided for functional compatibility with legacy hardware which may not support the fastest current data rates.

The appropriate ATA-specific mode (PIO, MDMA, or UDMA) must be selected based on the capabilities of the specific attached ATA peripheral.

2.3.1 Conflicts with other Peripherals and Catalog Options

2.3.1.1 Conflicts with SoC Peripherals

On the i.MX31 processor, the ATA has signals that can conflict with the CSPI device (CSPI1), USB Host 1 port, and PWM module, depending on configuration. See below for details of the current implementation.

2.3.1.2 Conflicts on the 3-Stack Board

Due to CSPI1 device, USB Host 1 port and PWM module are not supported on the 3-Stack board. The ATA as implemented for the 3-Stack board has no pin conflicts with the CSPI device (CSPI1), USB Host 1 port, and PWM.

2.3.2 Cabling

No ribbon cable is needed, because an ATA socket is available on the 3-Stack board.

2.4 Software Operation

2.4.1 Application / User Interface to ATA drives

The ATA device exports a standard streams interface to the Windows File System. Application-level access to ATA disks is through the Windows File System, using functions, such as CreateFile() and CloseHandle().

The File System, or user software that requires block device access to the ATA, does so through the standard Windows CE Block Device IOCTLs. These provide functions to acquire disk information and to read and write blocks (disk sectors) of data.

2.4.2 ATA Driver Configuration

You configure the driver into the BSP build by dragging the catalog item. Doing so defines the environment variable/configuration option: BSP_ATA.

Configuration for the ATA is then provided through registry settings imported from the following file:

```
platform.reg
```

These settings can be modified to select timing and transfer mode, and if necessary the device prefix and index.

2.4.2.1 Transfer Mode and Timing

The mode by which data is transported on the ATA bus (TransferMode) is configured by a registry settings, as defined in Section 2.4.4, “Registry Settings.”

The ATA bus timings are based on the i.MX31 clock, as defined in the *MCIMX31 and MCIMX31L Applications Processors Reference Manual*. The driver requires a clock period of 15 nanosec (66.6 MHz).

2.4.2.2 Prefix and Index

The default device prefix is “DSK” and Index is “1”. These items are important when configuring a storage device as source for the USB Mass Storage client. The USB Mass Storage client (function) driver's default registry configuration in the following file sets the source block device as “DSK1”:

```
PUBLIC\Common\OAK\FILES\common.reg
```

When no Index is configured for the ATA block device, the bus enumerator assigns an index according to the order of block device loading. When the removable storage is attached to USB host ports (as mass storage class), or when RAMDISK is included, the index assigned to these other block devices can influence any index automatically assigned by the bus enumerator.

2.4.2.3 IOMUX and Pinout

The internal i.MX31 ATA signals can be multiplexed to a choice of pins on IC, as described for the IOMUX in the *MCIMX31 and MCIMX31L Applications Processors Reference Manual*.

The ATA driver supports two hardware configurations, which defines other on-chip peripherals and devices with which an ATA device can be simultaneously configured within the BSP.

2.4.2.4 Defaults

The default mode for the ATA is transfer mode UDMA mode 3 for the i.MX31, as selected by the default file supplied for the build:

```
platform.reg
```

Default IOMUX configuration supports hardware mode 1.

2.4.3 Power Management

The ATA supports two power management modes, ON (D0) and OFF (D4). These modes are managed through the standard Windows Power Manager. The power manager uses IOCTL_POWER_SET to switch the disk's power state, according to the inactivity settings configured in Power Manager.

For standard block drivers, PowerUp and PowerDown functions are called by the Device Manager.

The primary method for limiting power consumption in the ATA module is to gate off all clocks to the module when those clocks are not needed. This is accomplished through the DDKClockSetGatingMode function call.

The clock is turned on during the initialization process and turned off after initialization is completed. Data transfer operations are handled in the DSK_IOCTL function to process the IOCTL calls from the File System. The ATA driver turns ON the clock and enables the ATA module before processing any data transfer. After the block of data has been processed, the ATA module is disabled and the clock is turned OFF.

2.4.3.1 PowerUp

The PowerUp function, which is called by Device Manager, sets a flag to indicate that power is up.

2.4.3.2 PowerDown

The PowerDown function, which is called by Device Manager, ensures that volatile data is stored in RAM, and sets a flag to indicate when power is down.

2.4.3.3 IOCTL_POWER_SET

This IOCTL handles the request to change disk power state (D0 or D4), and is called by power manager (or SetDevicePower() API).

2.4.4 Registry Settings

The ATA driver settings are taken from `platform.reg`, which can be customized for each particular build. These registry values are located under the registry key:

```
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\ATA_MX31]
```

The values should be defined under the registry key in `platform.reg`. You can qualify these with the BSP_ATA system variable for configurable catalog item support.

Table 2-2. Registry Settings

Value	Type	Content	Description
Dll	sz	ata_mx31.dll	Driver dynamic link library
IClass	sz	“{A4E7EDDA-E575-4252-9D6B-4195D48BB865}” GUID for a power-manageable block device	
TransferMode	dword	08	PIO mode 0
		09	PIO mode 1
		...	...
		0C	PIO mode 4
		20	MDMA mode 0
		21	MDMA mode 1
		22	MDMA mode 2
		40	UDMA mode 0
		...	...
		45	UDMA mode 5
IOMuxMask	dword	040007f8	GPR mask for IOMUX setting. Derived from the i.MX31 hardware manual and devices configured in combination
IOMuxVal	dword	00000318	GPR value for IOMUX setting, as above.

Table 2-2. Registry Settings

Value	Type	Content	Description
InterruptDriven	dword	01 (00)	enable interrupt driven I/O use for PIO or MDMA/UDMA modes (disable interrupt; not used normally)
DMA	dword	00 01	disable DMA (always disable for PIO mode) enable DMA (always enable for MDMA or UDMA modes)
IORDYEnable	dword	01	enable Host IORDY for PIO mode 3 and 4

As indicated in the above table, the following settings should be combined:

For PIO modes:

```
"InterruptDriven"=dword:01      ; 01-enable interrupt driven I/O, 00-disable
"DMA"=dword:00                  ; disable DMA
"TransferMode"=dword:0c         ; 08-PIO mode 0, ..., 0C-PIO mode 4
"IORDYEnable"=dword:01          ; enable Host IORDY for PIO mode 3, 4
```

For MWDMA modes:

```
"InterruptDriven"=dword:01      ; enable interrupt driven I/O
"DMA"=dword:01                  ; enable DMA
"TransferMode"=dword:20         ; 20-MWDMA mode 0, ..., 22-MWDMA mode 2
"IORDYEnable"=dword:01          ; enable Host IORDY for PIO mode 3, 4
```

For UDMA modes:

```
"InterruptDriven"=dword:01      ; enable interrupt driven I/O
"DMA"=dword:01                  ; enable DMA
"TransferMode"=dword:43         ; 40-UDMA mode 0, ..., 45-UDMA mode 5
"IORDYEnable"=dword:01          ; enable Host IORDY for PIO mode 3, 4
```

The following standard registry entries are also to be included for the ATA device under the above key:

Table 2-3. Standard Registry Entries

Value	Type	Content	Description
Prefix	sz	“DSK”	Device identifier (combined with Index for DSK1 for example)
Index	dword	1	Instance of ATA drive.
Order	dword	10	Early, to allow file system loading
DoubleBufferSize	dword	10000	128 sectors
DrqDataBlockSize	dword	200	Each data request is one sector, always 512 bytes
WriteCache	dword	01	disk internal cache is enabled within drive
LookAhead	dword	01	disk read-ahead to internal is enabled within drive
DeviceId	dword	00	primary device ID
HDProfile	sz	“HDProfile”	Storage Manager profile to be used in GetDeviceInfo (see below)

In addition to these values, the ATA uses the HDProfile information from the StorageManager registry keys. The following are default/sample values:

```
[HKEY_LOCAL_MACHINE\System\StorageManager\Profiles\HDProfile]
"Name"="ATA Hard Disk Drive"
"Folder"="Hard Disk"
```

```
[HKEY_LOCAL_MACHINE\System\StorageManager\Profiles\HDProfile\FATFS]
"EnableCacheWarm"=dword:00000000
```

2.4.5 DMA Support

The ATA driver supports the DMA (the default mode) and non-DMA transfer modes. The ATA supports three transfer types:

- UDMA works in the DMA operation mode
- MDMA works in the DMA operation mode
- PIO works in the Non-DMA operation mode

To change the transfer mode, change the value of “TransferMode” from the registry. When the ATA driver operates in SDMA, it always uses the scatter gather method. Though the BSP_SDMA_SUPPORT_ATA flag is present in the `bsp_cfg.h` file, it does not control whether SDMA is used.

The driver does not allocate or manage DMA buffers internally. All buffers are allocated and managed by the upper layers, the details of which are given in the request submitted to the driver. For every request submitted to it, the driver attempts to build a DMA Scatter Gather Buffer Descriptor list for the buffer passed to it by the upper layer.

For the driver to attempt to build the Scatter Gather DMA Buffer Descriptors, the upper layer should ensure that the buffer meets the following criteria.

- Start of the buffer should be a cache-line (32 byte) aligned address.
- Number of bytes to transfer should be cache-line (32 byte) aligned.

2.5 Unit Test

The ATA driver is tested using the Storage Block Device test cases, which are included in the Windows CE 5.0 Test Kit (CETK). There are no custom CETK test cases for ATA driver. The Storage Device test cases used to test ATA driver includes the following:

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

2.5.1 Unit Test Hardware

In order to run the ATA driver unit tests, you need the i.MX31 board and the attached Hitachi hard disk C4K40. Other drives supporting up to UDMA mode 3 may be used.

2.5.2 Unit Test Software

Table 2-4 lists the software required to run the unit tests.

Table 2-4. Software Requirements

Requirements	Description
Tux.exe	Tux test harness, which is needed for executing the test.
Kato.dll	Kato logging engine, which is required for logging test data.
Tooltalk.dll	Library required by Tux.exe and Kato.dll. Handles the transport between the target device and the development workstation.
fsdtst.dll	Test .dll file used to perform File System Driver Test cases.
disktest.dll	Test .dll file used to perform Storage Device Block Driver API test cases.
rw_all.dll	Test .dll file used to perform Storage Device Block Driver Benchmark tests.
rwtest.dll	Test .dll for various read/write options, including multi-threading and various block sizes.

2.5.3 Building the Storage Device Tests

The Storage Device Tests come pre-built as part of the CETK. No steps are required to build these tests. The following files are included among the required CETK files:

- `fsdtst.dll`
- `disktest.dll`
- `rw_all.dll rwtest.dll`

The CETK files are located in the following location:

```
\Program Files\Windows CE Platform Builder\5.00\cepb\wcetk\ddtk\armv4I
```

2.5.4 Running the Storage Device Tests

CAUTION

These CETK tests destroy any information residing on the hard disk.

Launch the tests from a command line or from the Windows CE Target Control window in Platform Builder. The following command line runs the file system driver test:

```
tux –o –d fsdtst -x 1001-1010,5001-5031 -c "-p HDProfile –zorch”
```

This command performs file system tests that cover all of the required File System API functions. Excluded are those tests that manipulate disk partitions.

The command line option HDProfile refers to the registry setting used to establish storage device profile information to the Storage Manager:

```
[HKEY_LOCAL_MACHINE\System\StorageManager\Profiles\HDProfile]
"Name"="ATA Hard Disk Drive"
"Folder"="Hard Disk"
```

NOTE

The command line option `-zorch` is case sensitive (the help message within the `test .dll` is not correct) and is used to confirm over-writing of all information on the hard disk.

The command line for running the Storage Device Block Driver API Test is:

```
tux –o –d disktest –c "-p HDProfile -zorch"
```

The command line for running the Storage Device Block Driver Read/Write Test is:

```
tux -o -d rwtest -c "-p HDProfile -zorch"
```

The command line for running the Storage Device Block Benchmark Test is:

```
tux -o -d rw_all –x 1006 –c "-p HDProfile -zorch"
```

This test includes only the benchmark test for 128 contiguous sectors. The test reads and writes all sectors of the drive in 128 block (64 kbyte) chunks. When drive read-ahead is enabled, this allows the drive to provide maximum sustained data rate from the media, to ensure that the ATA driver supports the same.

It is not necessary for all drive sectors to be tested, but the pre-compiled test does not have options to limit the portion tested, and all components are not publicly available for test customization. The test takes about four hours to execute on a 40 gigabyte drive. Tests using smaller contiguous chunks take even longer, and are not necessary for driver characterization.

For detailed information on the Storage Device CETK test cases, go to **Debugging and Testing** > **Tools for Debugging and Testing** > **Windows CE Test Kit** > **CETK Tests**, and then view the following topics:

- File System Driver Test
- Storage Device Block Driver API Test
- Storage Device Block Driver Read/Write Test
- Storage Device Block Benchmark Test

2.6 Basic Elements for Driver Development

This section describes the basic elements for driver development in the <TGTPLAT> BSP.

2.6.1 BSP Environment Variables

The variable BSP_ATA is set to enable the ATA device driver.

2.6.2 Mutual Exclusive Drivers

There are no mutually exclusive drivers because the ATA as implemented for the 3-Stack board has no any pin conflicts with the CSPI device (CSPI1), USB Host 1 port, and PWM.

2.6.3 Dependencies of Drivers

Table 2-5 lists the Microsoft defined environment variables used in the BSP.

Table 2-5. Environment Variables

Variable	Description
SYSGEN_FATFS	Set to support FAT16 file system
SYSGEN_STOREMGR	Set to support storage manager
SYSGEN_STOREMGR_CPL	Set to support storage manager in control panel
SYSGEN_MSPART	Set to support partition driver

2.7 Block Device API Reference

This section describes the primary interface to the ATA block device, which is through the standard Windows CE Block Device IOCTLs. Application-level access to ATA disks should be through the Windows File System.

For reverse compatibility, the deprecated DISK_IOCTL* are also supported, but not documented here. For information, see the Windows CE 5.0 Help.

The driver also supports the standard XXX_Init, XXX_Deinit, XXX_Open and XXX_Close routines, as used by Device Manager and the bus enumerator to load the driver. When the registry settings for ATA are correct, these functions are handled automatically, and need no further documentation here.

2.7.1 IOCTL_DISK_DEVICE_INFO

The DeviceIoControl request returns storage information to block device drivers.

Parameters

lpInBuffer [in] Pointer to a STORAGEDEVICEINFO structure.

nInBufferSize [in] Specifies the size of the STORAGEDEVICEINFO structure.

lpBytesReturned [out] Pointer to a DWORD to receive the total number of bytes returned.

2.7.2 IOCTL_DISK_GET_STORAGEID

The DeviceIoControl request returns the current STORAGE_IDENTIFICATION structure for a particular storage device.

Parameters

hDevice [in] Handle to the block device storage volume, which can be obtained by opening the FAT volume by its file system entry. The following code example shows how to open a PC Card storage volume.

```
hVolume = CreateFile(TEXT("\Storage Card\Vol:"),
GENERIC_READ|GENERIC_WRITE, 0, NULL, OPEN_EXISTING, 0, NULL);
```

lpOutBuffer	[out] Set to the address of an allocated STORAGE_IDENTIFICATION structure. This buffer receives the storage identifier data when the IoControl call returns.
nOutBufferSize	[out] Set to the size of the STORAGE_IDENTIFICATION structure and also additional memory for the identifiers. For ATA disk devices, the identifiers consist of 20 bytes for a manufacturer identifier string, and also 10 bytes for the serial number of the disk.
lpBytesReturned	[out] Pointer to a DWORD to receive the total number of bytes returned.

2.7.3 IOCTL_DISK_GETINFO

This DeviceIoControl request returns notifies the block device drivers to return disk information.

Parameters

lpOutBuffer	[out] Pointer to a DISK_INFO structure.
nOutBufferSize	[out] Specifies the size of the DISK_INFO structure.
lpBytesReturned	[out] Pointer to a DWORD to receive the total number of bytes returned.

2.7.4 IOCTL_DISK_GETNAME

This DeviceIoControl request services the request from the FAT file system for the name of the folder that determines how users access the block device. If the driver does not supply a name, the FAT file system uses the default name passed to it by the file system.

Parameters

lpOutBuffer	[out] Specifies a buffer allocated by the file system driver. The device driver fills this buffer with the folder name. The folder name must be a Unicode string.
nOutBufferSize	[out] Specifies the size of lpOutBuffer. Always set to MAX_PATH where MAX_PATH includes the terminating NULL character.
lpBytesReturned	[out] Set by the device driver to the length of the returned string and also the terminating NULL character.

2.7.5 IOCTL_DISK_READ

This DeviceIoControl request services FAT file system requests to read data from the block device.

Parameters

lpInBuffer	[in] Pointer to a SG_REQ structure.
nInBufferSize	[in] Specifies the size of the SG_REQ structure.
lpBytesReturned	[out] Pointer to a DWORD to receive total bytes returned. Set to NULL if you do not need to return this value.

2.7.6 IOCTL_DISK_SETINFO

This DeviceIoControl request services FAT file system requests to set disk information.

Parameters

lpInBuffer	[in] Pointer to a DISK_INFO structure.
nInBufferSize	[in] Specifies the size of DISK_INFO.
lpBytesReturned	[out] Pointer to a DWORD to receive total bytes returned.

2.7.7 IOCTL_DISK_WRITE

This DeviceIoControl request services FAT file system requests to write data to the block device.

Parameters

lpInBuffer	[in] Pointer to an SG_REQ structure.
nInBufferSize	[in] Specifies the size of SG_REQ.
lpBytesReturned	[out] Pointer to a DWORD to receive total bytes returned.

See the sr_status member of SG_REQ for write status. ERROR_SUCCESS indicates write success.

2.7.8 IOCTL_DISK_FLUSH_CACHE

This DeviceIoControl request issues the ATA FLUSH CACHE command to the disk.

Parameters	[No parameters]
Return Value	ERROR_SUCCESS: flushed okay
	ERROR_GEN_FAILURE: Failed to send flush command. Either write caching was not enabled on the device, or command was aborted.

Chapter 3
Audio Driver

The audio driver module for the MC13783 PMIC (`wavedev_MC13783.dll`) provides audio playback and recording functions. This module performs audio playback using the MC13783 PMIC Stereo DAC, and recording using the MC13783 Voice codec.

NOTE

For the 3-Stack board, the SSI channel for the Voice codec is not connected on the CPU engine board. So only the playback function is supported. The recording function is not supported, and is disabled in the driver.

For information about accessing the application with the audio driver using the methods and functions associated with WaveOut functionality, see the Platform Builder Help topic:

Windows CE Features → Graphics and Multimedia Technologies → Audio → Waveform Audio → Waveform Audio Application Development

3.1 Audio Driver Summary

Table 3-1 lists the source code location, library dependencies, and other BSP information.

NOTE

You should include the recommended catalog items in the OS design in order to provide a fairly comprehensive audio playback capability using the Windows Media Player application. Decide which audio codecs to include or exclude based on the specific functional requirements and degree of audio support needed.

Table 3-1. Audio Driver Summary

Driver Attribute	Definition
Target Platform (TGTPLAT)	3DS
Target SOC (TGTSOC)	MX31
MXARM11 CSP Driver Path	..\CSP\ARM\FREESCALE\MXARM11\DRIVERS\WAVEDEV ..\CSP\ARM\FREESCALE\PMIC\MC13783\SDK
IC-specific CSP Driver Path	N/A
CSP Static Library	mxarm11_wavedev.lib pmicSdkCsp_MC13783.lib
Platform Driver Path	..\PLATFORM\<*tgtplat*> \SRC\DRIVERS\PMIC\MC13783\PDK ..\PLATFORM\<*tgtplat*>\SRC\DRIVERS\WAVEDEV\MC13783
Import Library	N/A

Table 3-1. Audio Driver Summary(Continued)

Driver Attribute	Definition
Driver DLL	wavedev_MC13783.dll
Required Catalog Items	Third Party→ BSPs → Freescale *<tgtplat>* → Device Drivers → MC13783 Audio Driver
Recommended Catalog Items These catalog items are all located under the following: **Core OS → Windows CE devices →Graphics and Multimedia Technologies**	Audio → Waveform Audio Media → Audio Codecs and Renderers → MP3 Codec Media → Audio Codecs and Renderers → MPEG-1 Layer 1 and 2 Audio Codec Media → Audio Codecs and Renderers →MS ADPCM Audio Codec Media → Audio Codecs and Renderers → Wave/AIFF/au/snd File Parser Media → Audio Codecs and Renderers → Waveform Audio Renderer Media→ Audio Codecs and Renderers → WMA Codec Media → Windows Media Player → Windows Media Player Media→ Windows Media Player → Windows Media Player OCX Media→ Windows Media Player → Windows Media Technologies
SYSGEN Dependency	N/A
BSP Environment Variables	BSP_AUDIO_MC13783=1 BSP_PMIC_MC13783=1

NOTE

The selection and use of the Windows Media Player and the various software codecs is beyond the scope of the audio driver and is not discussed further in this document. For information about these items, see the Platform Builder Help topic:

Windows CE Features → Graphics and Multimedia Technologies

3.2 Supported Functionality

The audio driver enables the 3-Stack board to provide the following software and hardware support:

- Conforms to the Microsoft audio driver architecture as defined for Windows CE and all related operating systems
- Supports any Freescale MXARM11-based platform that is compatible with the MC13783 PMIC
- Uses double-buffered DMA operations to transfer audio data between memory and the SSI FIFO
- Supports two power management modes, full on and full off
- Minimizes power consumption at all times using clock gating and by disabling all audio-related hardware components that are not actively being used

3.3 Hardware Operation

Figure 3-1 displays the hardware components and the default configuration that is used for both audio playback and recording. For operation and programming, see the chapters in the *MCIMX31 and MCIMX31L Applications Processors Reference Manual* for the SSI, Serial Clock PLL, SDMA, Audio MUX, and IO MUX components.

For technical details for the MC13783 audio components, see the *MC13783 Power Management and Audio Circuit User's Guide*. These components include the Stereo DAC, the Voice codec, the available audio input/output paths, and the supported amplifier/mixer configurations.

NOTE

On the 3-Stack board, the connection between the IOMUX module of the i.MX31 and Voice codec of PMIC is not available.

For information about the routing of the various audio-related signal lines, see the schematics for the platform and the MC13783 PMIC (which may be in the form of an add-on daughter card).

For information about changing or fine-tuning the hardware configuration for audio playback and recording, see the Audio Driver Compile-time Configuration Options section below.

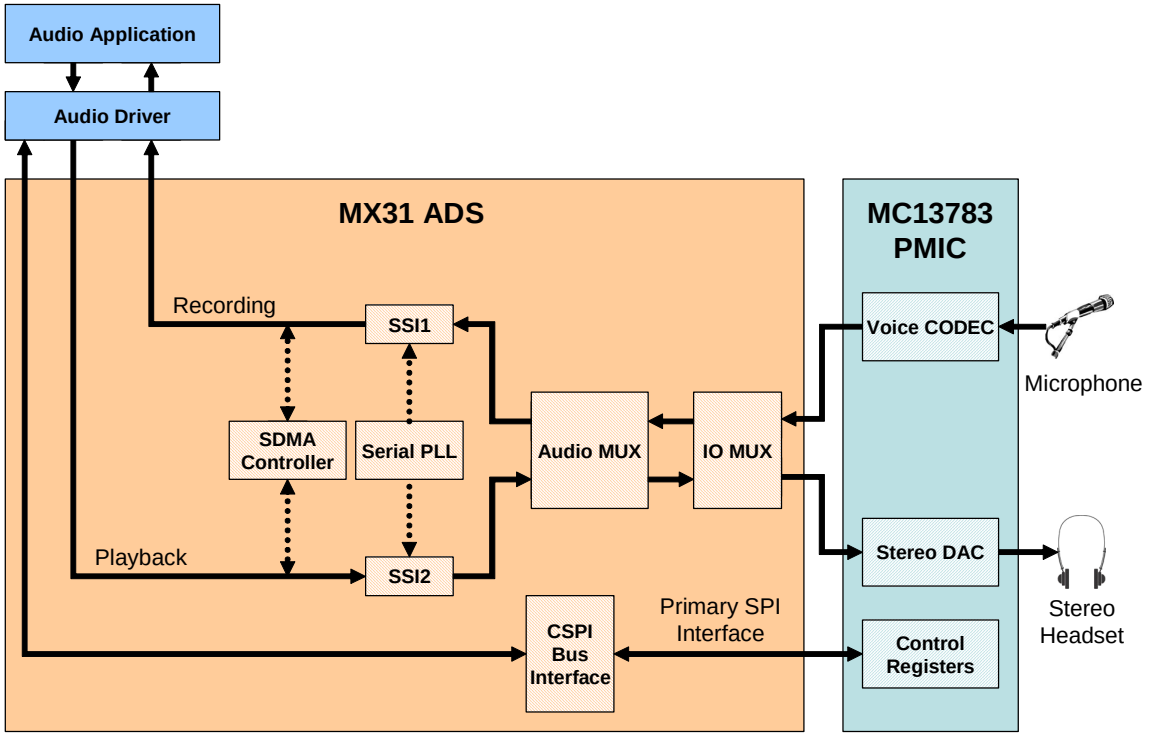

Figure 3-1. Audio Playback and Recording Hardware Components

3.3.1 Audio Playback

The following hardware configuration steps are performed prior to each playback operation (based on the default audio driver configuration):

- SSI1 is configured for time-slotted network mode using 4 timeslots per frame and a sampling rate of 44.1 kHz. The first two timeslots transmit the left and right audio channel data words, respectively. Each audio data word is 16 bits long. SSI1 is also configured to operate in master mode using the Serial PLL (for the i.MX31) clock signal to generate the appropriate framesync and bitclock signals. Note that clock gating scheme is used with SSI1 whereby all clock signals to SSI1 are disabled until SSI1 is actually being used. This helps to minimize power consumption when audio playback is not being performed.
- The SSI1 transmitter watermark level is set to support SDMA transfers during audio playback.
- The MC13783 Stereo DAC is then also configured for time-slotted network mode using 4 timeslots/frame and a 44.1 kHz sample rate but operating in slave mode. The first two timeslots are also used to receive the left and right audio channel data words, respectively, to match the SSI1 configuration. If necessary, the required MC13783 audio components are also powered on or re-enabled at this time. Normally, the MC13783 audio components that are not actively being used are kept in a power-off or disabled state so as to minimize power consumption.
- The Digital Audio MUX is configured to connect the internal port 1 (which is assigned to SSI1) with external port 4 (which is used to communicate with the Stereo DAC). At the same time, the appropriate IO MUX pins are also configured so that the Audio MUX external port 4 signals can actually be routed off-chip to the MC13783.
- The SDMA channel is fully configured to support 16-bit data transfers between the application's memory buffers and the SSI1 TX FIFO0. The SSI1 TX FIFO0 is prefilled with audio data at this point along with the DMA buffers.
- Finally, the SSI1 transmitter is enabled, which begins the transmission of the audio data stream.

The hardware repeatedly performs the following functions while audio playback is being performed:

- The SSI issues a new DMA request when the transmitter's FIFO0 level reaches the empty watermark level. The SDMA controller then refills FIFO0 using data from the DMA buffers until the DMA buffer has been emptied.
- An interrupt is generated when a DMA buffer is emptied and this interrupt is handled by the audio driver. The audio driver refills the DMA buffer and returns it to the SDMA controller for processing.
- Due to the double-buffering scheme, the SDMA controller simply uses the other DMA buffer to continue refilling the SSI1 transmitter FIFO0 while the previous DMA buffer is being refilled.

The following hardware changes are made at the completion of each playback operation:

- When the entire audio stream is transmitted, there is no more data available to refill the empty DMA buffers. Therefore, the output DMA channel is disabled when both output DMA buffers are empty and there is no additional data available to refill them.
- The MC13783 audio components that were used for playback are disabled to minimize power consumption. This step is done before disabling SSI1 to avoid any extraneous noise or "pop" that may be heard over the headphones.

- Finally, gate SSI1 is disabled and clocked.

3.3.2 Required SoC Peripherals

Table 3-2 lists the SoC hardware components that are required by audio driver for the exclusive use.

Table 3-2. SoC Hardware Components

Component	Use
SSI1 synchronous serial interfaces	Playback
Serial Clock PLL	Provides the master clock signal for SSI1
14.7 MHz clock signal generator	Supplies the CLIA clock input to the MC13783 PMIC (for MC13783 master mode only)
Digital Audio MUX	Connects the SSI1 to the IO MUX in order to access off-chip peripherals
IO MUX pins	Connects the Digital Audio MUX external ports 4 to the MC13783 PMIC
SDMA Controller	Manages the DMA channels that are used for playback

3.3.3 Conflicts with Other SoC Peripherals

There are no known conflicts between the SoC peripherals that are required by the audio driver and any other device driver, and no known issues.

3.3.4 Required MC13783 PMIC Components

The audio driver requires the exclusive use of the MC13783 PMIC hardware components listed in Table 3-3.

Table 3-3. MC13783 PMIC Hardware Components:

Component	Use
Stereo DAC and the audio output section	Playback
Digital audio buses	Transfers data between the SSI and the Stereo DAC
CLIA clock input	Required if the Stereo DAC is to be operated in master mode

NOTE

The audio driver expects that the following MC13783 hardware control registers are accessible by the ARM core: RX0, RX1, Audio Codec, Audio Stereo DAC, Audio Tx, and SSI Network. This means that the ARM core must be connected to the MC13783 control registers using the primary processor interface and not the secondary processor interface. This is the normal configuration for all of the currently supported platforms.

3.4 Software Operation

The audio driver follows the Microsoft-recommended architecture for audio drivers. For information about the architecture and operation, see the Platform Builder Help:

Developing a Device Driver → Windows CE Drivers → Audio Drivers → Audio Driver Development Concepts

3.4.1 Audio Playback

The software operation of the audio driver for playback is similar to that of the hardware configuration. Once the hardware components are configured, then the only thing that the audio driver must do is to handle the output DMA buffer empty interrupts. This is done through the interrupt handler, which refills each of the output DMA buffers with new audio data that has been supplied by the application, and then returns the DMA buffer to the SDMA controller.

3.4.2 Audio Recording

NOTE

The recording function is not supported due to hardware limitations, and is disabled in the driver.

The operation of the audio driver for recording is similar to the hardware configuration. Once the hardware components are configured, then the audio driver handles the input DMA buffer full interrupts. This is done through the interrupt handler, which copies the contents of each input DMA buffer to an application-supplied buffer, and then returns the empty DMA buffer to the SDMA controller. If the application-supplied buffer does not have enough space for all of the new data, any extra data is discarded.

The application is signaled using a callback function when the application-supplied buffer is full.

3.4.3 Audio Driver Compile-time Configuration Options

The audio driver can be configured for a wide variety of operating modes, depending on the hardware and software requirements. See Table 3-4 and Table 3-5 for the compile-time configuration options.

You should not change the audio driver configuration settings without a detailed understanding of the platform's hardware configuration and operating characteristics. Selecting invalid or incorrect configuration settings may result in an audio driver that will not load or work properly. Conversely, you may fine-tune the audio driver performance and resource usage by adjusting these configuration settings. For further information about the configuration options, see the corresponding source files.

Table 3-4. Audio Driver Configuration Options (hwctxt.h)

Configuration Setting Name	Description
INCHANNELS	Defines the number of input/recording channels that are available. Can be set to either 1 or 2. Default is 1.
OUTCHANNELS	Defines the number of output/playback channels that are available. Can be set to either 1 or 2. Default is 2.
BITSPERSAMPLE	The number of data bits per audio sample. This must match with the HWSAMPLE typedef and the AUDIO_SAMPLE_MAX/AUDIO_SAMPLE_MIN values. Default is 16.
INSAMPLERATE	The hardware input/recording sampling rate in Hz. Default is 16000.
OUTSAMPLERATE	The hardware output/playback sampling rate in Hz. Default is 44100.
HWSAMPLE	A typedef that defines the size of each audio data word. This must match the BITSPERSAMPLE and AUDIO_SAMPLE_MAX/AUDIO_SAMPLE_MIN values. Default is INT16.
USE_MIX_SATURATE	Enable a check in the software mixer code to guard against saturation. Default is 1.
AUDIO_SAMPLE_MAX and AUDIO_SAMPLE_MIN	The valid range of each audio data word. Values that are outside of this range will be clipped to the max/min value by the saturation protection code if USE_MIX_SATURATE is set to 1. Default is 32767 and -32768.
AUDIO_DMA_PAGE_SIZE	The size in bytes of each audio DMA buffer. Default is 2048 bytes.
AUDIO_REGKEY_PREFIX	The common prefix to be used for accessing all of the audio driver runtime configuration registry keys. Default is “Drivers\BuiltIn\Audio\PMIC\Config”.
PLAYBACK_DISABLE_DELAY_MSEC and RECORD_DISABLE_DELAY_MSEC	The delay, in milliseconds, that the audio driver will wait following the completion of an I/O operation before actually disabling the audio CODEC hardware. On some devices, such as the MC13783, there is a significant CODEC warm-up delay before an audio playback or recording operation can be performed. Disabling the audio hardware can be delayed for a brief period following each audio operation and thereby skip having to re-enable the hardware if another audio I/O operation is started soon after. The delay interval can be set to zero to disable this feature. The default is 1000 for both playback and recording.

Table 3-5. Audio Driver Configuration Options (hwctxt.cpp)

Configuration Setting Name	Description
BSP_SSI1_MASTER_BOOL and BSP_SSI2_MASTER_BOOL	Selects whether SSI1 and/or SSI2 are to be operated in master mode. Default is FALSE for both settings.
SSI1_MASTER_CLOCK_SOURCE and SSI2_MASTER_CLOCK_SOURCE	Defines the master clock input for each SSI. This is only used when the SSI is operating in master mode. The default settings are DDK_CLOCK_BAUD_SOURCE_SERPLL for both settings.
STEREO_DAC_SSI	The SSI that will be used for playback through the Stereo DAC. Default is m_pSSI1.
VOICE_CODEC_SSI	The SSI that will be used for recording using the Voice CODEC. Default is m_pSSI2.
SSI1_AUDMUX_PORT and SSI2_AUDMUX_PORT	The internal Digital Audio MUX ports that are connected to SSI1 and SSI2. The defaults are PORT1 for SSI1 and PORT2 for SSI2.
VOICE_CODEC_AUDMUX_PORT	The external Digital Audio MUX port that is connected to the Voice CODEC. The default is PORT5.
STEREO_DAC_AUDMUX_PORT	The external Digital Audio MUX port that is connected to the Stereo DAC. The default is PORT4.
VOICE_CODEC_AUDIO_BUS	The digital audio bus that connects the Audio MUX to the Voice CODEC. The default is AUDIO_DATA_BUS_2.
STEREO_DAC_AUDIO_BUS	The digital audio bus that connects the Audio MUX to the Stereo DAC. The default is AUDIO_DATA_BUS_1.
STEREO_DAC_BUS_MODE	The digital audio bus protocol that is to be used. Either timeslotted NETWORK_MODE or I2S_MODE may be selected. The default is NETWORK_MODE.
VOICE_CODEC_BUS_MODE	The digital audio bus protocol that is to be used. Either timeslotted NETWORK_MODE or I2S_MODE may be selected. The default is NETWORK_MODE.
SSI_SFCSR_TX_WATERMARK and SSI_SFCSR_RX_WATERMARK	The transmitter and receiver watermarks that are to be used with SSI1 and SSI2. The default is 4 for both watermark levels.
DEFAULT_OUTPGA_GAIN	Sets the default output amplifier gain level. The default is OUTPGA_GAIN_MINUS_3DB.

3.4.4 DMA Support

As indicated above, the audio driver uses the SDMA controller to transfer the digital audio data between the audio application and the SSI FIFOs. This minimizes the processing required by the ARM core and can also reduce the power consumption during audio playback and recording operations.

Note, however, that the audio driver requires the use of DMA support for proper operation. Unlike some other device drivers, the audio driver does not have any support for an alternative non-DMA or polling-based operating mode. Therefore, the BSP_SDMA_SUPPORT_SSI1 (for audio playback) and BSP_SDMA_SUPPORT_SSI2 (for audio record) macros in the `bsp_cfg.h` header file must always be defined as TRUE even though the audio driver does not explicitly make use of these definitions.

This section describes the audio driver DMA implementation issues and tradeoffs, and the available compile-time DMA-related configuration options.

In order to use DMA transfers, the following items must be properly allocated, managed, and deallocated by the device driver:

- The DMA data buffers where the application data is kept
- The DMA buffer descriptors which are used by the DMA hardware to manage the state of each DMA buffer

The DMA data buffers can be allocated from either "internal memory" (which is provided by on-chip internal RAM) or "external memory" (which is provided by off-chip external DRAM). Table 3-6 lists the issues and considerations for the type of memory to use for the DMA data buffers.

Table 3-6. DMA Memory Allocation Issues and Considerations

Memory Region	Memory Usage Issues and Considerations
Internal	Allows the external memory to be placed in a low power mode while the DMA data buffers are being processed to reduce system power consumption (as long as nothing else on the system requires access to external memory). Also, less power is required to access the internal RAM than to access
	But the total size of the internal memory region is limited (only 16 kB for the i.MX31).
	The limited amount of internal memory may have to be shared by multiple device drivers.
	The entire internal memory region must be manually managed with predefined addressed ranges being reserved for each specific use.
External	The total size of the external memory is typically much greater than the size of the internal memory (128 MB compared to 16 kB for the i.MX31). This provides much greater flexibility in selecting the size of the DMA data buffers.
	There is typically no need to worry about the possible impact and memory requirements of any other device driver.
	Memory allocation is handled using the standard Windows CE system calls.
	The external memory cannot be placed into a low power mode while the DMA is active.

Table 3-7 describes how to configure the build so the audio driver will allocate its DMA data buffers from either internal or external memory.

Table 3-7. Configuring for Internal/External Memory DMA Data Buffer Allocation

Memory Region	Required Configuration Options
Internal	Set the BSP_AUDIO_DMA_BUF_ADDR macro in bsp_cfg.h to an address within the internal memory region. Also set BSP_AUDIO_DMA_BUF_SIZE to the total size (in bytes) for all DMA data buffers that will be allocated.
External	Make sure that the BSP_AUDIO_DMA_BUF_ADDR macro is commented out in bsp_cfg.h

The DMA buffer descriptors can also be allocated from either internal or external memory. However, in this case, the choice is made automatically through the use of the CSPDDK APIs, specifically DDKSdmaAllocChain(). See the CSPDDK documentation for additional information about the DDKSdmaAllocChain() API.

3.4.5 Power Management

The primary method for limiting power consumption in the audio driver is to gate off all clocks to the SSI when those clocks are not needed, and to turn off all audio hardware components at the end of each audio stream. This is accomplished through the **DDKClockSetGatingMode** function call and the various PMIC audio APIs. In the Windows CE 5.0 BSP, the audio module can be disabled, and its clocks turned off, whenever there are no active audio I/O operations. The clock gating and the disabling of related audio hardware components is handled automatically within the audio module and requires no additional configuration or code changes.

The audio driver works correctly after resuming after using the power down mode.

3.4.5.1 PowerUp

This function supports resuming an audio I/O operation that was previously terminated by calling the PowerDown() API. It begin by restoring power and re-enabling all of the required audio hardware components. Next, the audio DMA transfers are restarted to complete the powerup process for the audio driver.

Note that this function is intended to be called only by the Power Manager and must not block or depend on any hardware interrupts. Therefore, all required timed delays must be handled using a polling loop instead of any of the normal “wait for an event to be signaled” functions. This functionality is currently handled by IOCTL_POWER_SET and the function is just a stub.

3.4.5.2 PowerDown

This function supports suspending all currently active audio I/O operations just before the entire system enters the low power state. Note that this function is intended to be called only by the Power Manager and must not block or depend on any hardware interrupts. Therefore, the first thing that this function must do is to signal all of the possible wait events that the normal audio driver thread may be currently waiting on. If not, then the PowerDown thread may be blocked waiting for a critical section that is currently being held by the normal audio driver thread. This is an error and would deadlock the entire system and prevent it from properly entering the low power state.

But when all waiting events are signaled, the normal audio thread is guaranteed (because of priority inversion) to run to the point where it releases the required critical section and allows the PowerDown thread to proceed without the possibility of deadlocking.

When it is ensured that the normal audio thread is not executing inside any critical section, the PowerDown thread can safely proceed to disable all active audio DMA operations and to power down all of the associated audio hardware components. Once this is done, the audio driver remains in a low power state until the PowerUp function is called by the Power Manager. This functionality is currently handled by IOCTL_POWER_SET and the function is just a stub.

3.4.5.3 IOCTL_POWER_SET

This Power Manager IOCTL is implemented for the audio driver. All system suspend and resume handling is handled by the IOCTL which handles the PowerDown and PowerUp functionalities. For all platforms, the following registry entry must be defined:

```
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\Audio]
         "IClass"="{A32942B7-920C-486b-B0E6-92A702A99B35}" ; PMCLASS_GENERIC_DEVICE
```

This registry entry is required for proper power management functionality.

3.4.6 Audio Driver Registry Settings

At least one registry key must be properly defined so that the Device Manager will load the audio driver when the system is booted. Additional registry keys may also be defined, and even changed at runtime, to configure the operation of the audio driver. Both the required and optional registry keys for the audio driver are described in the following sections.

3.4.6.1 Required Audio Driver Registry Settings

The following registry keys are required in order for the Device Manager to properly load the audio device driver during the device's normal boot process. These registry settings should typically not be modified. If they are missing or incorrectly defined, then the audio driver may not be loaded and all audio functions will be disabled.

```
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\Audio]
          "Prefix"="WAV"
          "Dll"="wavedev_MC13783.dll"
          "Index"=dword:1
          "Order"=dword:10
          "Priority256"=dword:95
          "IClass"="{A32942B7-920C-486b-B0E6-92A702A99B35}" ; PMCLASS_GENERIC_DEVICE
```

3.4.6.2 Optional Audio Driver Runtime Configuration Registry Settings

The following optional registry keys can also be defined in order to configure the audio driver's various runtime operating modes. If these registry keys are not defined or if the values are invalid, then the audio driver will use the values shown below as the default settings.

For additional configuration settings that are supported by the audio driver, see the enumerated type definitions in the `pmic_audio.h` header file. The numeric constants used in the following registry key values are the integer values that correspond to the enumerated types defined in `pmic_audio.h` for each specific function or item.

NOTE

The registry setting for the recording does not function for the 3-Stack board.

```
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\Audio\PMIC\Config\Playback]
   "LeftChannel"=dword:40
```

```
    "RightChannel"=dword:80
    "Description"="Stereo headset jack J8 (LeftChannel=0x40, RightChannel=0x80)"

[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\Audio\PMIC\Config\Recording]
    "LeftChannel"=dword:1
    "RightChannel"=dword:2
    "Description"="Stereo input jack J4 (LeftChannel=1, RightChannel=2)"

[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\Audio\PMIC\Config\MicBias1]
     "Enable"=dword:0
     "Description"="Microphone bias circuit 1 disabled (0) or enabled (1)"

[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\Audio\PMIC\Config\MicBias2]
    "Enable"=dword:1
    "Description"="Microphone bias circuit 2 disabled (0) or enabled (1)"

[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\Audio\PMIC\Config\InputAmp]
     "Mode"=dword:1
     "Mode Description"="Voltage-to-Voltage (1) or Current-to-Voltage (2)"
     "Gain"=dword:8
     "Gain Description"="-8 dB (0) to 23 dB (31 or 0x1F) in 1 dB steps"

[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\Audio\PMIC\Config\HeadsetDetect]
     "Enable"=dword:0
     "Description"="Disabled (0) or enabled (1)"
```

NOTE

Changes to the audio driver configuration registry keys can be made at any time. The new settings take effect at the beginning of the next audio I/O operation. To view and modify a device's current registry entries, use the Remote Registry Editor tool provided with Platform Builder.

3.5 Unit Test

The audio driver is tested using the Waveform Audio Driver Test suite included with the Windows CE 5.0 Test Kit (CETK). The test suite includes automated and interactive tests used to test playback and recording functions.

3.5.1 Unit Test Hardware

Table 3-8 lists the hardware needed to run the unit tests.

Table 3-8. Hardware Needed to Run the Unit Tests

Requirements	Description
Stereo headphones or earphones	This is required to confirm that audio playback is working. The headphones or earphones should have a 3.5mm jack for thei.MX31 platform. 
Mono microphone	Not required for 3-Stack board.

3.5.2 Unit Test Software

Table 3-9 lists the software required to run the unit tests.

Table 3-9. Software Needed to Run the Unit Tests

Requirements	Description
Tux.exe	Tux test harness, which is needed for executing the test
Kato.dll	Kato logging engine, which is required for logging test data
Tooltalk.dll	Library required by Tux.exe and Kato.dll. Handles the transport between the target device and the development workstation
wavetest.dll	Test .dll file

3.5.3 Building the Audio Driver CETK Tests

The audio driver tests come pre-built as part of the CETK. No steps are required to build these tests. The wavetest.dll file is included with the CETK files in the following location:

`[Drive]:\Program Files\Windows CE Platform Builder\5.00\cepb\wcetk\ddtk\armv4I`

3.5.4 Running the Audio Driver CETK Tests

Because full-duplex operation is not supported, the command line is:

```
tux -o -d wavetest -c “-e”
```

For detailed information about the audio driver tests, see the Platform Builder Help:

Debugging and Testing → Tools for Debugging and Testing → Windows CE Test Kit → CETK Tests → Waveform Audio Driver Test

Table 3-10 describes the audio driver test cases and indicates whether the tests are expected to pass or fail.

Table 3-10. Audio Driver Test Cases

Test Case	Description
100: Build Verification Test	Searches the registry for active audio driver entries and for each audio device entry that is found, the test then tries to select full-duplex operation (that is, simultaneous play and capture). This test should always pass.
1000: Easy Playback	This test plays wave files (for example, asterisk.wav) from the Windows directory using both the sndPlaySound and PlaySound APIs. You should hear the appropriate sound in the headphones and this test should always pass.
2000: Playback Capabilities	This test reads and displays the audio driver’s playback capabilities. The reported audio driver capabilities must be manually confirmed with the features that are supposed to be supported by the audio driver. This test should always pass.
2001: Playback	This test attempts to play a tone using all of the audio formats that are supposed to be supported by the audio driver. This test should always pass.
2002: Playback Notifications	This test attempts to playback a tone using all possible types of callbacks. This test should always pass.

Table 3-10. Audio Driver Test Cases(Continued)

2003: Playback Using Extended Functions	This test attempts to use the extended playback capabilities that are supported by the audio driver, such as volume control, playback rate control, and pitch control. This test should always pass.
3000: Capture Capabilities	This test reads and displays the audio driver's recording capabilities. The reported audio driver capabilities must be manually confirmed with the features that are supposed to be supported by the audio driver. This test should always pass. This test case is skipped in the 3-Stack system.
3001: Capture	This test attempts to record a tone using all possible types of callbacks. This test should always pass. This test case is skipped in the 3-Stack system.
3002: Capture Notifications	This test attempts to record a tone using all possible types of callbacks. This test should always pass. This test case is skipped in the 3-Stack system.
4000: Test Volume Control	This test attempts to change the output volume. This test should always pass.

3.6 System Level Audio Driver Tests

In addition to running the audio driver tests in the CETK, you can perform various system-level tests that involve the use of the audio driver. The following sections describe ways to test the audio driver without using the CETK.

3.6.1 Checking for a Boot-time Musical Tune

The normal Windows CE boot procedure includes playing a short musical tune just before displaying the touchpanel calibration screen. At this point, the audio driver should already have successfully loaded and you should hear the tune if you attach a headset to the stereo output jack.

3.6.2 Confirming Touchpanel Taps and Keypad Key Presses

The normal Windows CE system configuration includes the ability to playback a short tapping sound when the stylus makes contact with the touchpanel. You should hear these taps when a headset is attached to the stereo output jack.

You should hear a “click” when a key on the keypad is pressed.

3.6.3 Playing Back Sample Audio and Video Files Using the Media Player

The Microsoft-supplied Media Player application can be used to load and play a variety of audio and video media files in a number of different formats. The only requirement is to include the software codecs in the OS image that may be needed to decode the media file. The Media Player includes controls for pausing, resuming, and stopping playback, and advancing playback to a specific point. Additional volume and muting controls are also provided.

3.6.4 Using the SDK Sample Audio Applications for Testing

The Windows CE SDK that is included as part of Platform Builder provides two audio-related sample applications. The **wavrec** sample application, which is not supported on PDK, can be used to test the audio recording function while the TGTPLAT sample application provides a command line-based method of

playing back various media files. For additional information about these sample applications, see the Platform Builder Help: **Windows CE Features → Graphics and Multimedia Technologies → Audio → Waveform Audio → Waveform Audio Samples**.

3.7 Audio Driver API Reference

For detailed reference information for the audio driver, see the Platform Builder Help:

Developing a Device Driver → Windows CE Drivers → Audio Drivers → Audio Driver Reference → Waveform Audio Driver Reference

3.8 Audio Driver Troubleshooting Guide

The following sections describe techniques to identify and fix the most common problems involving the audio driver.

3.8.1 Checking Build-time Configuration Options

Any compile- or link-time errors are probably due to incorrect or invalid configuration settings that were defined in `hwctxt.h` or `hwctxt.cpp`.

See the Audio Driver Compile-time Configuration Options section above for information about the device driver build configuration options.

Perform the build procedure documented in the BSP Users Guide to compile and link the audio driver.

Confirm that the required Platform Builder catalog items are included in the OS design. See the table above for a list of the required and recommended audio driver-related catalog items.

3.8.2 Confirming Audio Driver Loading During Device Boot

First, confirm that the appropriate [HKEY_LOCAL_MACHINE\Drivers\BuiltIn\Audio] registry key has been defined.

Next, confirm that all of the following files exist in the release directory: `wavedev_MC13783.dll`, `waveapi.dll`, `device.exe`. The first DLL is the audio device driver; the second DLL provides the means for applications to access the audio driver. The last file is the Device Manager executable required to load the audio driver.

Finally, if you are booting a release image, you should see all of the following messages in the Platform Builder output window:

```
Loaded symbols for
'D:\WINCE500\PBWORKSPACES\<workspace>\RELDIR\<platform>_ARMV4I_RELEASE\WAVEDEV_MC13783.DLL'
Loaded symbols for
'D:\WINCE500\PBWORKSPACES\<workspace>\RELDIR\<platform>_ARMV4I_RELEASE\WAVEAPI.DLL'
The corresponding messages when booting a debug image are (the timestamp, process ID, and thread
ID numbers may differ from those shown below but the important thing to confirm is that the
modules are being loaded):
4294770428 PID:2bf8a70e TID:2bf9bd56 0x8bf8a4a8: >>> Loading module wavedev_MC13783.dll at
address 0x01A20000-0x01A38000
```

```
Loaded symbols for
'D:\WINCE500\PBWORKSPACES\<workspace>\RELDIR\<platform>_ARMV4I_DEBUG\WAVEDEV_MC13783.DLL'
4294770755 PID:2bf8a70e TID:2bf9bd56 0x8bf8a4a8: >>> Loading module waveapi.dll at address
0x03BF0000-0x03C1B000 (RW data at 0x01FCF000-0x01FCF958)
Loaded symbols for
'D:\WINCE500\PBWORKSPACES\<workspace>\RELDIR\<platform>_ARMV4I_DEBUG\WAVEAPI.DLL'
```

3.8.3 Media Player Application Not Found

Make sure that the Media Player catalog item is included in the OS design (see the table above). The Media Player application will not be included in the final system image if the catalog item is not selected.

3.8.4 Media Player Fails to Load and Play an Audio File

This problem is typically caused by failing to include the appropriate software codec that is required to handle the audio file format. See the list of recommended audio driver catalog items in the table above and make sure that support for the desired audio file format is included.

Chapter 4
Backlight Driver

The backlight driver uses the hardware provided by the display module on the chip to control the backlight on the LCD display. The backlight driver interfaces with the Windows CE Power Manager to provide timed control over the display backlight. A timeout interval controls the length of time that the backlight stays on. The backlight driver implements the required IOCTLs and uses its own defined timer to set the backlight power states.

4.1 Backlight Driver Summary

Table 4-1. Backlight Driver Summary

Driver Attribute	Definition
Target Platform (TGTPLAT)	3DS
Target SOC (TGTSOC)	MX31
CSP Driver Path	On the i.MX31: ..\CSP\ARM\FREESCALE\MXARM11\DRIVERS\ BACKLIGHT\DRIVER
CSP Static Library	mxarm11_backlight.lib
Platform Driver Path	..\PLATFORM\<*tgtplat*> \SRC\DRIVERS\BACKLIGHT\DRIVER
Import Library	N/A
Driver DLL	backlight.dll
Catalog Item	Third Party >BSPs >Freescale <*tgtplat*> >Device Drivers >Backlight >IPU Backlight
SYSGEN Dependency	
BSP Environment Variables	BSP_BACKLIGHT=1 BSP_BACKLIGHT_IPU=1 BSP_BACKLIGHT_MC13783 should not be set.

4.2 Supported Functionality

The backlight driver enables the 3-Stack board to provide the following software and hardware support:

- Conforms to the Device Manager streams interface
- Supports the LCDC hardware method

4.3 Hardware Operation

The hardware consists of independently programmable registers in the LCDC module, the PWM Contrast Control Register, which is used to control the signal output at the contrast pin. By default, the backlight drivers control the pulse width of the built-in pulse-width modulator, which controls the contrast of the LCD screen.

4.4 Software Operation

The backlight driver is a stream interface driver, and is thus accessed through the file system APIs. To use the backlight driver, first create a handle to the device using the **CreateFile** function. Issue subsequent commands to the device using the **DeviceIoControl** function with IOCTL codes to specify the desired operation.

To control the backlight operation, call the **DeviceIoControl** function, which has following options:

- IOCTL_POWER_CAPABILITIES - where you register and inform the Power Manager of capabilities
- IOCTL_POWER_QUERY – where the new power state is returned
- IOCTL_POWER_SET – interface to the hardware that controls the backlight through the PDD layer.
- IOCTL_POWER_GET – the current power state is returned

4.4.1 Backlight Driver Registry Settings

Use the following registry keys to load the backlight driver.

```
[HKEY_CURRENT_USER\ControlPanel\Backlight]
    "BattBacklightLevel"=dword:7F       ; Backlight level settings. 0xFF = Full On
    "ACBacklightLevel"=dword:7F         ; Backlight level settings. 0xFF = Full On
    "BatteryTimeout"=dword:F
    "ACTimeout"=dword:1E
    "UseExt"=dword:1                    ; Enable timeout when on external power
    "UseBattery"=dword:1                ; Enable timeout when on battery
    "AdvancedCPL"="AdvBacklight"        ; Enable Advanced Backlight control panel dialog
```

4.5 Unit Test

Use the application test to test the backlight driver.

4.5.1 Unit Test Hardware

The Epson L4F00242T03 VGA Panel is needed, to display the graphics data for the Backlight application test.

4.5.2 Unit Test Software

Table 4-2 lists the software required to run the Backlight application test.

Table 4-2. Software Requirement

Requirements	Description
backlight.dll	The backlight driver to implement the backlight functions.
Advbacklight.dll	The file implements adding an Advanced button to the Backlight Control Panel application.

4.5.3 Running the Backlight Application Test

Use the steps, listed in Table 4-3, to perform the Backlight application test:

Table 4-3. Steps to Perform Backlight Application Test

Test Cases	Entry Criteria/Procedure/Expected Results
Backlight Level	**Entry Criteria:** N/A **Procedure:** 1. Go to “Setting>Control Panel” 2. Double click on the “Display” icon, then click on the “Backlight” tab 3. Click on the “Advanced…” button 4. Modify the backlight level setting for both battery and external power 5. Observe that the backlight level behaves according to the new setting **Expected Result:** N/A
Backlight Timeout	**Entry Criteria:** N/A **Procedure:** 1. Go to “Setting>Control Panel” 2. Double click on the “Display” icon, then click on the “Backlight” tab 3. Modify the backlight timeout setting for both battery and external power, and then click on “OK” button to apply the changes 4. Observe the time it takes for the backlight to go out, make sure it correspond with the new settings entered in step 3 **Expected Result:** N/A

4.6 Backlight API Reference

The API for the backlight driver conforms to the stream interface and exposes the standard functions. For additional information, see the Platform Builder Help:

Developing a Device Driver → Windows CE Drivers → Streams Interface Drivers

Chapter 5
Battery Driver

The battery driver module provides information to the operating system (OS) about the battery level. The battery driver performs the following tasks:

- Samples the voltage level upon initialization, as well as on power up when coming out of the suspend mode.
- Identifies whether the charger and/or battery are attached, and executes an appropriate charging or discharging operation
- Reports battery capability and power supply state to the OS periodically, by measuring battery voltage. While the battery is being charged, the current and voltage are limited in order to protect the charger and battery, and the OS will not enter suspend mode, because the charging operation would lose control in the suspend mode.

5.1 Battery Driver Summary

Table 5-1 identifies the source code location, library dependencies and other BSP information:

Table 5-1. Battery Driver Summary

Driver Attribute	Definition
Target Platform (TGTPLAT)	3DS
Target SOC (TGTSOC)	MX31
PMIC CSP Driver Path	..\CSP\ARM\FREESCALE\PMIC\MC13783\SDK
CSP Static Library	pmicSdkCsp_MC13783.lib and pmicPdkCsp_MC13783.lib
Platform Driver Path	..\PLATFORM\<*tgtplat*>\SRC\DRIVERS\BATTDRVR\MC13783
Import Library	N/A
Driver DLL	battdrvr_MC13783.dll
Catalog Item	Third Party > BSPs > Freescale <*tgtplat*> > Device Drivers > MC13783 Battery
SYSGEN Dependency	N/A
BSP Environment Variables	BSP_BATTERY=1, BSP_PMIC_MC13783=1

5.2 Supported Functionality

The battery driver enables the 3-Stack board to provide the following software and hardware support:

- Conforms to the Device Manager streams interface.
- Supports the <TGTPLAT> MC13783 PMIC.
- Supports the main battery without the support of the change notification.

5.3 Hardware Operation

The battery driver is implemented with the aid of the MC13783 Power Management Integrated Circuit (PMIC). The PMIC is a multifunctional IC that contains on-chip analog-to-digital converters used to measure the voltage and current levels of the battery. These levels are then used in determining the capacity level of the battery.

5.3.1 Conflicts with other SoC Peripherals

No conflicts.

5.4 Software Operation

When the driver is initialized, the default values for the battery parameters are retrieved from the registry and the battery status information is updated. After initialization, the function BatteryPDDGetStatus() is called periodically to get the status of the Battery and specify whether to charge or discharge the battery. It fills the structure SYSTEM_POWER_STATUS_EX2 and returns it to the system. The Power properties window is updated based on the values in this structure.

5.4.1 Battery Driver Registry Settings

The following registry keys are required to properly load the battery driver.

```
; These registry entries load the battery driver. The IClass value must match
; the BATTERY_DRIVER_CLASS definition in battery.h -- this is how the system
; knows which device is the battery driver.

      [HKEY_LOCAL_MACHINE\Drivers\BuiltIn\Battery]
             "Prefix"="BAT"
             "Flags"=dword:8                      ; DEVFLAGS_NAKEDENTRIES
             "IClass"="{DD176277-CD34-4980-91EE-67DBEF3D8913}"
             "BattFullLiftTime" = dword:8         ;Batt Spec defined: in hr,8hr is assumed
             "BattFullCapacity"=dword:960 ;Batt Spec defined: in mAh,2400mAhr is assumed
             "BattMaxVoltage"=dword:1068          ;Batt Spec defined: in mV,4200mV is assumed
             "BattMinVoltage"=dword:BB8           ;Batt Spec defined: in mV,3000mV is assumed
             "BattPeukertNumber"=dword:73         ;Batt Spec defined, 1.15 is assumed
             "BattChargeEff"=dword:50             ;Batt Spec defined, 0.80 is assumed
          "PollInterval"=dword:000003e8 ;Batt poll interval in milliseconds
             "Dll"="battdrvr_MC13783.dll"


[HKEY_LOCAL_MACHINE\System\Events]
              "SYSTEM/BatteryAPIsReady"="Battery Interface APIs"
```

5.4.2 Power Management

There is no additional power management implementation done specifically for Battery driver other than the implementation described in section 14.5.6 of the Power Management IC (PMIC) reference document.

5.5 Unit Test

To test the battery, switch on the system and watch the Power properties window. When charging, the LED indicator is on and the charge capacity of the battery can be seen increasing until charged to 100%. When the battery is not attached, not charged, or not available, the LED indicator is off.

CAUTION

Do not plug in or remove the battery after booting up the device.

5.5.1 Unit Test Hardware

The 3-Stack board and a battery device are required. Please refer to *i.MX31 3-Stack Platform User's Guide* for battery device specifications.

5.6 Battery API Reference

The API for the battery driver conforms to the stream interface and exposes the standard functions. For additional information, see the Platform Builder Help:

Developing a Device Driver → Windows CE Drivers → Battery Drivers

5.6.1 Battery PDD Functions

5.6.1.1 BSPBattdrvrGetParameters

This function returns the battery and charger voltage levels, the current level, and a flag that indicates if the current is charging or discharging.

Prototype

```
BOOL BSPBattdrvrGetParameters(DWORD *pBatt_V, DWORD *pCharger_V, BOOL
*fCharge, DWORD *I)
```

Parameters

pBatt_V

[out] pointer to the battery voltage value

pCharger_V

[out] pointer to the charger voltage value

fCharge

[out] pointer to the flag TRUE for charging FALSE for discharging

I

[out] pointer to the charging/discharging current

5.6.1.2 BSPBattdrvrGetSample

This function returns the battery voltage sample.

Prototype `BOOL BSPBattdrvrGetSample(UINT16 *psample)`

Parameters *psample*

[out] pointer to the sample value,
psample[0] = battery voltage;
psample[1] = battery current;
psample[2] = charger voltage;
psample[3] = charger current;

5.6.2 Battery Driver Structures

5.6.2.1 Battery Channels Structure

```
typedef enum _BATTDRVR_CHANNELS {
    BattVoltage,
    BattCurrent,
    ChargerVoltage,
    ChargerCurrent,
    TotalChannels,
} BATTDRVR_CHANNELS;
```

5.6.2.2 Battery Information Structure

```
typedef struct _BATT_INFO
{
    DWORD        adc_level;
    DWORD        adc_batt_max_V;
    DWORD        adc_batt_min_V;
    DWORD        adc_batt_max_I;
    DWORD        adc_batt_min_I;
    DWORD        adc_charger_max_V;
    DWORD        adc_charger_min_V;
    DWORD        adc_charger_max_I;
    DWORD        adc_charger_min_I;
    DWORD        charger_V_limit;
} BATT_INFO, *PBATT_INFO;
```

Chapter 6
Bluetooth Driver

The Bluetooth driver is used to drive the APM6628 module to implement Bluetooth functionality that is compatible with Bluetooth v2.0 +EDR.

Bluetooth exchanges data with the i.MX31 through the UART2 port. The APM6628 module adopts the BlueCore4 Bluetooth solution of Cambridge Silicon Radio company.

The Bluetooth driver is provided in binary form rather than source codes.

6.1 Bluetooth Driver Summary

Table 6-1 provides a summary of the source code location, library dependencies, and other BSP information:

Table 6-1. Bluetooth Driver Summary

Driver Attribute	Definition
Target Platform (TGTPLAT)	3DS
Target SOC (TGTSOC)	MX31
MXARM11 CSP Driver Path	N/A
CSP Driver Path	N/A
CSP Static Library	N/A
Platform Driver Path	..\PLATFORM\<*TGTPLAT*>\SRC\DRIVERS\Bluetooth
Import Library	N/A
Driver DLL	bthbcsp.dll btp_avdrv.dll
Required Catalog Items	Third Party -> BSPs -> Freescale<*TGTPLAT*> -> Device Drivers -> Bluetooth Core OS -> Windows CE devices -> Communication Services and Networking -> Networking -> Personal Area Network [PAN] -> Bluetooth -> Bluetooth Protocol Stack with Transport Driver Support -> Bluetooth Stack with Universal Loadable Driver
Optional Catalog Items	Core OS -> Windows CE devices -> Communication Services and Networking -> Networking -> Personal Area Network [PAN] -> Bluetooth -> Bluetooth Profiles Support -> Bluetooth PAN Core OS -> Windows CE devices -> Communication Services and Networking -> Networking -> Personal Area Network [PAN] -> Bluetooth -> Bluetooth Profiles Support -> Bluetooth LAP and Configuration Utility

Table 6-1. Bluetooth Driver Summary(Continued)

SYSGEN Dependency	SYSGEN_BTH = 1
BSP Environment Variables	BSP_Bluetooth = 1 BSP_SERIAL_UART2 = 1

The Recommended Catalog Items listed in the table must be included in the OS design in order to provide Bluetooth profiles.

The Optional Catalog Items are for the Bluetooth PAN profile (optional).

6.2 Requirements

The Bluetooth driver should meet the following requirements:

- Correctly drive Bluetooth module in APM6628 chip.
- Communication between Bluetooth module and UART2 driver with alterable serial baudrate up to 921.6kbps.
- Support A2DP
- Support power management mode: full on / full off.

6.3 Hardware Operation

The Bluetooth HCI driver exchanges data and commands between BCHS stack and Bluetooth hardware through UART2 port.

6.4 Software Operation

Figure 6-1 shows the overall software architecture with an existing MS Bluetooth stack and CSR BCHS stack.

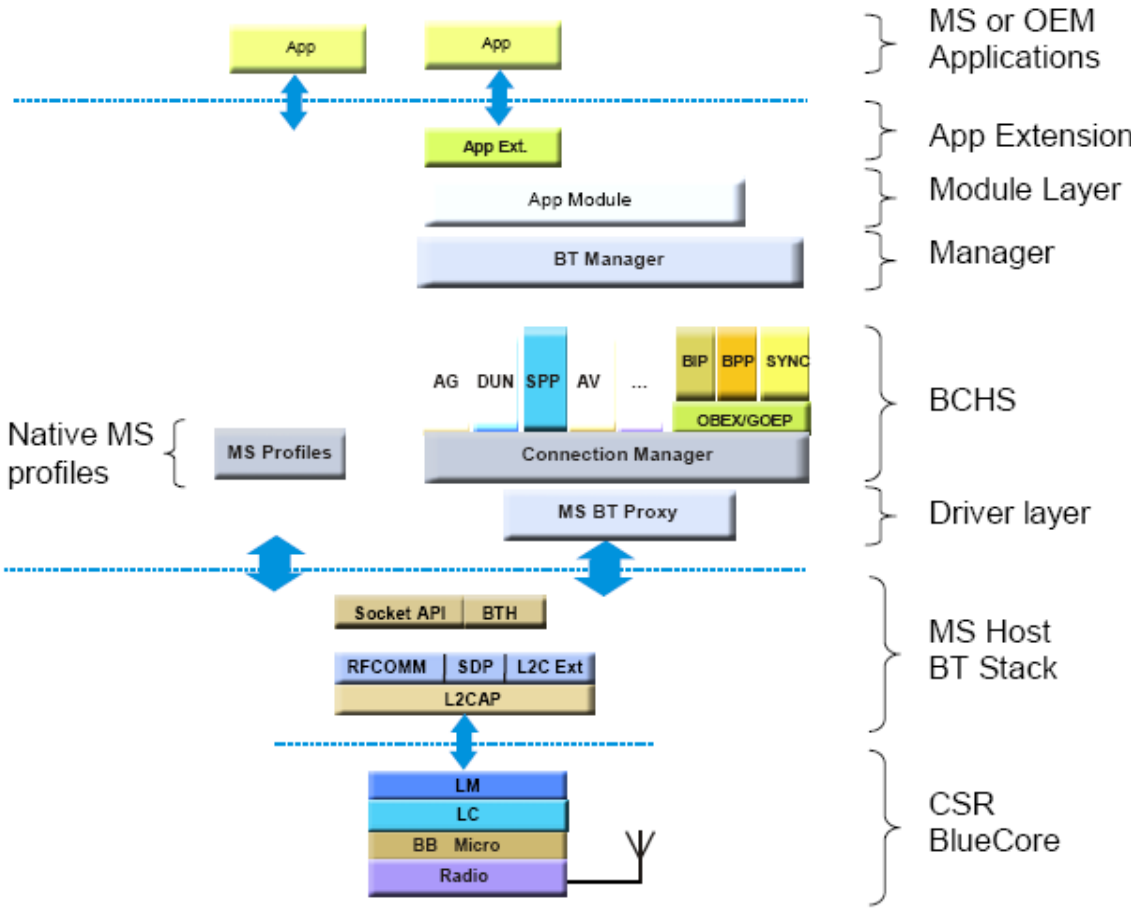

Figure 6-1. Software Architecture

The BCHS is an embedded Bluetooth software package complementing the already existing Microsoft Bluetooth profiles delivered as part of the Microsoft Windows CE OS. BCHS is developed to operate on top of the native Microsoft Bluetooth stack and not as a replacement of the Microsoft Bluetooth stack.

6.4.1 Communicating with the Bluetooth

Communication with the Bluetooth driver and writing Bluetooth UI applications are accomplished through Bluetooth APIs defined by CSR and Microsoft WinSock.

6.4.2 Bluetooth Registry Settings

The following registry keys are required to properly load the Bluetooth driver.

```
IF BPS_Bluetooth
#define    _registry    registry
```

```
#include "$(_TARGETPLATROOT)\SRC\DRIVERS\bluetooth\csr\$(_registry)\bta_mp3player.reg"
#include "$(_TARGETPLATROOT)\SRC\DRIVERS\bluetooth\csr\$(_registry)\btp_av.reg"
#include "$(_TARGETPLATROOT)\SRC\DRIVERS\bluetooth\csr\$(_registry)\btp_bipc.reg"
#include "$(_TARGETPLATROOT)\SRC\DRIVERS\bluetooth\csr\$(_registry)\btp_bips.reg"
#include "$(_TARGETPLATROOT)\SRC\DRIVERS\bluetooth\csr\$(_registry)\btp_driver.reg"
#include "$(_TARGETPLATROOT)\SRC\DRIVERS\bluetooth\csr\$(_registry)\btp_dundrv.reg"
#include "$(_TARGETPLATROOT)\SRC\DRIVERS\bluetooth\csr\$(_registry)\btp_ftsm.reg"
#include "$(_TARGETPLATROOT)\SRC\DRIVERS\bluetooth\csr\$(_registry)\btp_hfm.reg"
#include "$(_TARGETPLATROOT)\SRC\DRIVERS\bluetooth\csr\$(_registry)\btp_pacm.reg"
;  #include "$(_TARGETPLATROOT)\SRC\DRIVERS\bluetooth\csr\$(_registry)\btp_printing.reg"
#include "$(_TARGETPLATROOT)\SRC\DRIVERS\bluetooth\csr\$(_registry)\btp_saps.reg"
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Bluetooth\Transports\BuildIn\1]
             "Driver"="Bthbcsp.dll"
             "name"="COM2:"
             "Baud"=dword:000E1000;921.6Kbps
             "Priority256"=dword:132
             "PacketSize"=dword:192
             "SerialTimeoutConstant"=dword:5
             "SerialIntervalTimeout"=dword:FFFFFFFF
             "ReopenDelay"=dword:100
             "FlashChip"=dword:0
             "SerialBufferSize"=dword:4096
             "DumpKeys"=dword:0
             ;"PowerManagement"=dword:0
             "SniffMethod"=dword:0
ENDIF ;BSP_Bluetooth
```

In addition, UART2 driver should be loaded earlier than Bluetooth driver, so the following modification is necessary:

```
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\COM2]
          "Order"=dword:0
```

Because Bluetooth Audio driver encapsulates the old audio driver(wavedev_MC13783.dll) for A2DP feature, the following registry still need be modified:

```
IF BSP_Bluetooth
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\Wavedev]
    "Dll"="btp_avdrv.dll"
ELSE
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\Audio]
    "Dll"="wavedev_mc13783.dll"
ENDIF ;BSP_Bluetooth
```

6.5 Unit Test

The Bluetooth test includes the CETK test and the manual test for the A2DP feature..

6.5.1 Unit Test Hardware

Table 6-2 lists the required hardware to run the Bluetooth unit tests.

Table 6-2. Hardware Requirements for Bluetooth Unit Tests

Requirements	Description

Table 6-2. Hardware Requirements for Bluetooth Unit Tests(Continued)

Bluetooth Headset	Bluetooth Headset which supports SBC decoder is for testing A2DP feature
Two 3-Stack boards	CETK for Bluetooth need two Bluetooth boards on TCP/IP networking

6.5.2 Unit Test Software

6.5.2.1 CETK Test

Table 6-3 lists the required software on the client board to run the Bluetooth test.

Table 6-3. Software Requirements on Client Board

Requirements	Description
Tux.exe	Tux test harness, which is needed for executing the test
Kato.dll	Kato logging engine, which is required for logging test data
Tooltalk.dll	Library required by Tux.exe and Kato.dll. Handles the transport between the target device and the development workstation
Btw22.dll	test .dll file for the test client

Table 6-4 lists the required software on the server board to run the Bluetooth test.

Table 6-4. Software Requirements on Server Board

Requirements	Description
Tux.exe	Tux test harness, which is needed for executing the test
Kato.dll	Kato logging engine, which is required for logging test data
Tooltalk.dll	Library required by Tux.exe and Kato.dll. Handles the transport between the target device and the development workstation
Btwsvr22.dll	test .dll file for the test server

6.5.2.2 Bluetooth Application

Table 6-5 lists the required software to run the Bluetooth A2DP profile.

Table 6-5. Software Requirements to Run Bluetooth A2DP Profile

Requirements	Description
BT_A2DP.exe	This application is used to apply A2DP feature

6.5.3 Building the Bluetooth Tests

6.5.3.1 Bluetooth CETK Test

The Bluetooth port driver tests come pre-built as part of the CETK. No steps are required to build these tests.

The files are located with the other required CETK files in the following location:

```
[Drive]:\ Program Files\Windows CE Platform Builder\5.00\CEPB\WCETK\DDTK\ARMV4I
```

6.5.4 Running the Bluetooth Tests

6.5.4.1 Running the Bluetooth CETK Test

There are two ways to run the CETK test for Bluetooth:

- **Client/Server method**: Bluetooth boards are tested on the client and server separately.

  Prepare two PCs(recommended) and two 3-Stack board systems.

  a) Boot up two 3-Stack board systems with Ethernet KITL enabled.

  b) Copy `Kato.dll`, `Tooltalk.dll`, `Netall.dll`, and `Btwsvr22.exe` into Windows directory in server board.

  c) In the 3-Stack server board, modify the device name as “server“ through System Tools in the Control Panel.

  d) Open `btwsvr22.exe` from the console window or Windows directory.

  e) In the 3-Stack client board, open the CETK UI from **PB > Tools > Windows CE Test Kit** and make CETK connecting with board.

  f) Modify the Bluetooth command line: **tux -o -d btw22 -c”server”**.

  g) Click **Quick Start**.

- **Standalone method**: One Bluetooth board is used to test.

  If only one 3-Stack board is available, use these steps for testing:

  a) Boot up the 3-Stack boards with KITL enabled.

  b) In the 3-Stack board, select CETK UI from **PB > Tools > Windows CE Test Kit** and make CETK connecting with board.

  c) Click the **Quick Start** command.

6.5.4.2 Running the Bluetooth A2DP Test

A2DP testing purpose: listen to the stereo music played by MediaPlayer from Bluetooth headset

To run the A2DP test:

1. Place the Bluetooth headset in *pairing* mode.
2. Open Bluetooth **Device Properties** tools in the Control Panel and click **Scan Device.**
3. Enter “0000“ in the Authentication Request dialog for your Bluetooth headset.

   Your Bluetooth headset item is displayed in the Bluetooth Manager window.

NOTE

Ensure that the Bluetooth headset icon is correct and not a question mark. If you see a question mark, delete your Bluetooth headset icon from the Bluetooth Manager window, and rescan the Bluetooth device.

4. Execute `BT_A2DP.exe` in the Windows directory, and do not close that application Window.
5. Then double-click one Bluetooth headset icon in Bluetooth Manager window and select **Trusted**, and click **No** in the dialog box, then double-click this icon again and select **Active**, a red check mark should appear on your Bluetooth headset icon ; your another headset icon should be operated alike.
6. The 3-Stack board requires about 15 seconds to set up the connection with the Bluetooth headset. Then, if you play music using a media player, or click the panel, you can hear the sound from your Bluetooth headset.

Chapter 7
Camera Driver

The Camera Driver is based on the Windows CE Video Camera Device Driver Interface, which was introduced with Windows Mobile 5.0. This interface provides basic support for video and still image capture devices. The camera driver conforms to the architecture for Windows CE stream interface drivers, and allows applications to use the middleware layer provided by the DirectShow video capture infrastructure to communicate with and control the camera. This module is compatible with the OV2640 camera sensor modules.

At the lower layer, the Camera Driver performs several tasks; these include communicating with and configuring a camera sensor through the I2C interface, interfacing with the Image Processing Unit (IPU) to perform pre-processing tasks on captured images, and configuring the IPU Synchronous Display Controller (SDC) for direct display of video preview data.

Table 7-1 identifies the source code location, library dependencies and other BSP information:

Table 7-1. Camera Driver Summary

Driver Attribute	Definition
Target Platform (TGTPLAT)	3DS
Target SOC (TGTSOC)	MX31
MXARM11 CSP Driver Path	..\CSP\ARM\FREESCALE\MXARM11\DRIVERS\IPU\CAMERA
CSP Driver Path	N/A
CSP Static Library	mxarm11_camera.lib
Platform Driver Path	..\PLATFORM\<*tgtplat*>\SRC\DRIVERS\IPU\CAMERA
Import Library	N/A
Driver DLL	camera.dll
Catalog Items	N/A
SYSGEN Dependency	SYSGEN_DSHOW_CAPTURE=1
BSP Environment Variables	BSP_CAMERA=1 IMGCAMERAOEM=1

7.1 Supported Functionality

The camera driver enables the 3-Stack board to provide the following software and hardware support:

- Supports the Windows CE Video Camera Device Driver Interface
- Supports Preview, Capture, and Still pins

- Supports a Direct-to-Display preview mode
- Supports the OV2640 camera sensors

7.2 Hardware Operation

Several hardware modules are involved in the operation of the Camera driver. The OV2640 camera sensor captures external image data. All other hardware elements of the Camera driver are within the Image Processing Unit (IPU). The IPU Camera Sensor Interface (CSI) receives data from the sensor and converts the data into a format understood by the IPU. This data subsequently flows through the IPU Image Converter (IC) module, where it undergoes pre-processing. There are two pre-processing paths: one for encoding and one for viewfinding. The pre-processed image data is then transferred by the IPU DMA module to one of two destinations: system memory (encoding or viewfinding data) or the IPU Synchronous Display Controller (SDC) for display (viewfinding data).

For detailed operation and programming information, refer to the chapter on the Image Processing Unit (IPU) in the *MCIMX31 and MCIMX31L Applications Processors Reference Manual*.

7.3 Software Operation

7.3.1 Communicating with the Camera

Communication with the camera driver is accomplished through Camera APIs defined by Microsoft for Windows Mobile 5.0. Applications may access these Camera APIs directly or through the DirectShow video capture support.

7.3.1.1 Using the Windows CE Video Camera Device Driver Interface

The Windows CE Video Camera Device Driver Interface provides basic support for video and still image capture devices. For information about using camera APIs, see the Windows Mobile 5.0 Help: **Developing a Device Driver > Windows Mobile-based Device Drivers > Camera Drivers > Camera Driver Reference**.

7.3.1.2 Using DirectShow for Video Capture

DirectShow provides support in its architecture for the creation of filter graphs for video capture. For information about using DirectShow for video capture, see the Windows Mobile 5.0 Help:

Windows Mobile-based Device Features > Graphics and Multimedia Technologies > Media > DirectShow > DirectShow Application Development > Audio and Video Capture Support > Video Capture

7.3.2 Camera Registry Settings

Two sets of registry settings are important for proper Camera Driver operation. One set is for the camera sensor, and the other is for the Camera Driver. The registry keys required to properly select the camera sensor used on the SoC are described in this section.

7.3.2.1 i.MX31 Camera Registry Settings

In order to include the Windows Mobile Camera Application, an additional set of registry keys is needed.

The following registry keys are required to select the camera sensor used on the SoC.

```
[HKEY_LOCAL_MACHINE\Drivers\CSI]
   "CameraId"=dword:3
```

The CameraId registry key identifies the available camera sensor modules. The valid values are the following:

- 0 to indicate that the camera sensor in use is the iMagic IM8803
- 1 to indicate that the camera sensor in use is the iMagic IM8201.
- 2 to indicate that the camera sensor in use is the Avago ADCC3000.
- 3 to indicate that the camera sensor in use is the OV2640.

The following registry keys are required to properly load the Camera Driver.

```
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\CAMERA]
   "Prefix"="CAM"
   "Dll"="camera.dll"
   "Order"=dword:20
   "Index"=dword:1
   "IClass"="{CB998A05-122C-4166-846A-933E4D7E3C86}"

[HKEY_LOCAL_MACHINE\Software\Microsoft\DirectX\DirectShow\Capture]
   "Prefix"="PIN"
   "Dll"="camera.dll"
   "Order"=dword:20
   "Index"=dword:1
   "IClass"=multi_sz:"{C9D092D6-827A-45E2-8144-DE1982BFC3A8}",
                     "{A32942B7-920C-486b-B0E6-92A702A99B35}"
```

The following registry keys are required to include and configure the Windows Mobile Camera Application.

```
[HKEY_LOCAL_MACHINE\Software\Microsoft\Pictures\Camera\OEM]
    "BackLight"=dword:78
    "EnableZoom"=dword:1
    "CompressionQuality"=dword:0
    "EnableVideo"=dword:1
    "EnableTimeCaption"=dword:1
    "VideoRecordLimitation"="0 15 30"
    "OpenDefault"=dword:0
    "PowerSaveTimeInterval"=dword:1e
    "DisableCamera"=dword:0
    "SupportedVideoFormat"="*.wma;*.wmv"
    "IsPortraitScreen"=dword:1
    "EnableBrightness"=dword:1
    "EnableFlash"=dword:0
    "HandleOOMEvent"=dword:0
    "VideoBufferingDepth"=dword:1312d00
    "OEMMemoryThreshold"=dword:2500000
    "StandBySecs"=dword:20
    "SECSINTIMER"=dword:5
    "PicsInBurst"=dword:3
```

```
[HKEY_LOCAL_MACHINE\System\Pictures\Camera\OEM]
    "SoundFile"="\\windows\\ShutterSound.wav"
    "ForceShutterSound"=dword:1
    "ShutterSoundVolume"=dword:8000

[-HKEY_LOCAL_MACHINE\Software\Microsoft\Pictures\Camera\OEM\PictureResolution\]

[HKEY_LOCAL_MACHINE\Software\Microsoft\Pictures\Camera\OEM\PictureResolution]
    "OptionNum"=dword:7

[HKEY_LOCAL_MACHINE\Software\Microsoft\Pictures\Camera\OEM\PictureResolution\1]
    "ItemString"="88x72"
    "Width"=dword:58
    "Height"=dword:48
    "HighQualityFileSize"=dword:9e6
    "NormalQualityFileSize"=dword:420
    "LowQualityFileSize"=dword:1c4

[HKEY_LOCAL_MACHINE\Software\Microsoft\Pictures\Camera\OEM\PictureResolution\2]
    "ItemString"="160x120"
    "Width"=dword:a0
    "Height"=dword:78
    "HighQualityFileSize"=dword:1a00
    "NormalQualityFileSize"=dword:bb0
    "LowQualityFileSize"=dword:530

[HKEY_LOCAL_MACHINE\Software\Microsoft\Pictures\Camera\OEM\PictureResolution\3]
    "ItemString"="176x144"
    "Width"=dword:b0
    "Height"=dword:90
    "HighQualityFileSize"=dword:2799
    "NormalQualityFileSize"=dword:1080
    "LowQualityFileSize"=dword:712

[HKEY_LOCAL_MACHINE\Software\Microsoft\Pictures\Camera\OEM\PictureResolution\4]
    "ItemString"="320x240"
    "Width"=dword:140
    "Height"=dword:f0
    "HighQualityFileSize"=dword:6800
    "NormalQualityFileSize"=dword:2ec0
    "LowQualityFileSize"=dword:14c0

[HKEY_LOCAL_MACHINE\Software\Microsoft\Pictures\Camera\OEM\PictureResolution\5]
    "ItemString"="352x288"
    "Width"=dword:160
    "Height"=dword:120
    "HighQualityFileSize"=dword:9e66
    "NormalQualityFileSize"=dword:4200
    "LowQualityFileSize"=dword:1c49

[HKEY_LOCAL_MACHINE\Software\Microsoft\Pictures\Camera\OEM\PictureResolution\6]
    "ItemString"="640x480"
    "Width"=dword:280
    "Height"=dword:1e0
    "HighQualityFileSize"=dword:1a000
    "NormalQualityFileSize"=dword:bb00
    "LowQualityFileSize"=dword:5300
```

```
[HKEY_LOCAL_MACHINE\Software\Microsoft\Pictures\Camera\OEM\PictureResolution\7]
    "ItemString"="1280x1024"
    "Width"=dword:500
    "Height"=dword:400
    "HighQualityFileSize"=dword:68000
    "NormalQualityFileSize"=dword:2ec00
    "LowQualityFileSize"=dword:14c00

[HKEY_LOCAL_MACHINE\Software\Microsoft\Pictures\Camera\OEM\VideoProfile]
    "OptionNum"=dword:7

[HKEY_LOCAL_MACHINE\Software\Microsoft\Pictures\Camera\OEM\VideoProfile\1]
    "ItemString"="88x72"
    "Width"=dword:58
    "Height"=dword:48
    "VideoAudioFileSizePerSecond"=dword:50c0
    "VideoOnlyFileSizePerSecond"=dword:1800

[HKEY_LOCAL_MACHINE\Software\Microsoft\Pictures\Camera\OEM\VideoProfile\2]
    "ItemString"="160x120"
    "Width"=dword:a0
    "Height"=dword:78
    "VideoAudioFileSizePerSecond"=dword:11600
    "VideoOnlyFileSizePerSecond"=dword:4400

[HKEY_LOCAL_MACHINE\Software\Microsoft\Pictures\Camera\OEM\VideoProfile\3]
    "ItemString"="176x144"
    "Width"=dword:b0
    "Height"=dword:90
    "VideoAudioFileSizePerSecond"=dword:14300
    "VideoOnlyFileSizePerSecond"=dword:6000

[HKEY_LOCAL_MACHINE\Software\Microsoft\Pictures\Camera\OEM\VideoProfile\4]
    "ItemString"="320x240"
    "Width"=dword:140
    "Height"=dword:f0
    "VideoAudioFileSizePerSecond"=dword:45800
    "VideoOnlyFileSizePerSecond"=dword:11000

[HKEY_LOCAL_MACHINE\Software\Microsoft\Pictures\Camera\OEM\VideoProfile\5]
    "ItemString"="352x288"
    "Width"=dword:160
    "Height"=dword:120
    "VideoAudioFileSizePerSecond"=dword:50c00
    "VideoOnlyFileSizePerSecond"=dword:18000

[HKEY_LOCAL_MACHINE\Software\Microsoft\Pictures\Camera\OEM\VideoProfile\6]
    "ItemString"="640x480"
    "Width"=dword:280
    "Height"=dword:1e0
    "VideoAudioFileSizePerSecond"=dword:116000
    "VideoOnlyFileSizePerSecond"=dword:44000

[HKEY_LOCAL_MACHINE\Software\Microsoft\Pictures\Camera\OEM\VideoProfile\7]
    "ItemString"="1280x1024"
    "Width"=dword:500
```

```
    "Height"=dword:400
    "VideoAudioFileSizePerSecond"=dword:458000
    "VideoOnlyFileSizePerSecond"=dword:110000

[HKEY_LOCAL_MACHINE\Software\Microsoft\Pictures\Camera\OEM\Zoom]
    "OptionNum"=dword:2

[HKEY_LOCAL_MACHINE\Software\Microsoft\Pictures\Camera\OEM\Zoom\1]
    "ItemString"="X1"
    "PhysicalZoom"=dword:1
    "LogicalZoom"=dword:1

[HKEY_LOCAL_MACHINE\Software\Microsoft\Pictures\Camera\OEM\Zoom\2]
    "ItemString"="X2"
    "PhysicalZoom"=dword:2
    "LogicalZoom"=dword:2

[HKEY_LOCAL_MACHINE\Software\Microsoft\Pictures\Camera\OEM\VideoEncoderCLSID]
    "IClass"="{d23b90d0-144f-46bd-841d-59e4eb19dc59}"

[HKEY_LOCAL_MACHINE\Software\Microsoft\Pictures\Camera\OEM\AudioEncoderCLSID]
    "IClass"=""

[HKEY_LOCAL_MACHINE\Software\Microsoft\Pictures\Camera\OEM\MuxCLSID]
    "IClass"="{4f4ba16c-ccb0-4d23-b1e8-672c7dfe4a02}"
    "VideoFileExt"="wmv"

[HKEY_LOCAL_MACHINE\Software\Microsoft\Pictures\Camera\OEM\PreviewColorFormatCLSID]
    "IClass"="{e436eb7b-524f-11ce-9f53-0020af0ba770}"

[HKEY_LOCAL_MACHINE\Software\Microsoft\Pictures\Camera\OEM\CaptureColorFormatCLSID]
    "IClass"="{e436eb7b-524f-11ce-9f53-0020af0ba770}"

[HKEY_LOCAL_MACHINE\Software\Microsoft\Pictures\Camera\OEM\StillColorFormatCLSID]
    "IClass"="{e436eb7b-524f-11ce-9f53-0020af0ba770}"

[HKEY_LOCAL_MACHINE\Software\Microsoft\Pictures\Camera\OEM\VideoEncoderProperty]
    "PropertyNum"=dword:2

[HKEY_LOCAL_MACHINE\Software\Microsoft\Pictures\Camera\OEM\VideoEncoderProperty\1]
    "VideoPropertyName"="_VBRQUALITY"
    "VideoPropertyVT"=dword:3
    "VidoepropertyValue"=dword:4b

[HKEY_LOCAL_MACHINE\Software\Microsoft\Pictures\Camera\OEM\VideoEncoderProperty\2]
    "VideoPropertyName"="_COMPLEXITYEX"
    "VideoPropertyVT"=dword:3
    "VidoepropertyValue"=dword:0

[HKEY_LOCAL_MACHINE\Software\Microsoft\Pictures\Camera\OEM\AudioEncoderProperty]
    "PropertyNum"=dword:0

[HKEY_LOCAL_MACHINE\Software\Microsoft\Pictures\Camera\OEM\AudioCaptureFilter]
    "FormatTag"=dword:1
    "Channels"=dword:1
    "SamplesPerSec"=dword:1f40
    "AvgBytesPerSec"=dword:1f40
```

```
    "BlockAlign"=dword:1
    "BitsPerSample"=dword:8
    "SamplesPerBlock"=dword:0

[HKEY_LOCAL_MACHINE\Software\Microsoft\Pictures\Camera\OEM\VideoCaptureFrameInterval]
    "FrameInterval"=dword:0

[HKEY_LOCAL_MACHINE\Software\Microsoft\Pictures\Camera\OEM\PreviewFrameInterval]
    "FrameInterval"=dword:0

[HKEY_CURRENT_USER\Software\Microsoft\Pictures\Camera\USER]
    "CurrentZoom"=dword:0
    "Flash"=dword:1
    "Brightness"=dword:0
    "FileNumber"=dword:1
    "QualitySetting"=dword:0
    "AudioEnabled"=dword:0
    "Resolution"=dword:3
    "UserVideoTime"=dword:1E
    "DefaultDir"=""
    "FilePrefix"="img"
    "VideoProfile"=dword:2

[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\DirectShow\ImageSink\Quality\0\jpg\Quality]
    "Value"=hex:\
           19,00,00,00
    "Type"=dword:00000004
    "NumberOfValues"=dword:00000001
    "GUID"="{1D5BE4B5-FA4A-452D-9CDD-5DB35105E7EB}"

[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\DirectShow\ImageSink\Quality\1\jpg\Quality]
    "Value"=hex:\
           41,00,00,00
    "Type"=dword:00000004
    "NumberOfValues"=dword:00000001
    "GUID"="{1D5BE4B5-FA4A-452D-9CDD-5DB35105E7EB}"

[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\DirectShow\ImageSink\Quality\2\jpg\Quality]
    "Value"=hex:\
           5a,00,00,00
    "Type"=dword:00000004
    "NumberOfValues"=dword:00000001
    "GUID"="{1D5BE4B5-FA4A-452D-9CDD-5DB35105E7EB}"

[HKEY_LOCAL_MACHINE\Drivers\Capture\Camera]
    "PinCount"=dword:3
```

7.3.3 Power Management

The camera driver consumes power primarily through the operation of various IPU sub-modules, such as the CSI, which synchronizes and receives image data from the camera sensor, and the IC, which performs

pre-processing operations on captured image data. The CSI and IC modules are enabled when the camera is set to a running state. At all times when the camera is not running, the CSI and IC modules are disabled.

Support for transitioning to the Suspend and Resume states if provided through the IOCTL_POWER_SET IOCTL.

7.3.3.1 PowerUp

This function is not implemented for the camera driver.

7.3.3.2 PowerDown

This function is not implemented for the camera driver.

7.3.3.3 IOCTL_POWER_SET

The camera driver implements the IOCTL_POWER_SET IOCTL API with support for the D0 (Full On) and D4 (Off) power states. These states are handled in the following manner:

- D0 – Action is only taken when resuming from the D4 state. If the camera was running when the transition to the D4 state occured, the camera returns to a running state, re-enabling the CSI and IC modules.
- D4 – Action is only taken if the camera is running when the request to transition to the D4 state occurs. In this case, the camera is stopped and the CSI and IC modules are disabled.

7.4 Unit Test

Since the Camera Driver API is introduced with Windows Mobile 5.0, there are no camera tests provided by Microsoft for Windows CE 5.0. Thus, on Windows CE 5.0, a custom test and a custom application are provided to test the Camera Driver.

On Windows Mobile 5.0 platforms, the Camera Driver is subject to several test suites provided with the Windows Mobile CE Test Kit (WMCETK): the Camera Driver Data Structure Verification Test, the Camera Driver I/O Test, the Camera and DirectShow Integration Test, and the Camera Performance Test.

The Camera Driver Data Structure Verification Test queries the driver for the various properties and formats, and verifies that the data structures returned are valid.

The Camera Driver I/O Test verifies the functionality of the preview and capture streams on the camera driver.

The Camera and DirectShow Integration Test verifies the functionality of the camera driver when used under DirectShow.

The Camera Performance Test suite gathers performance data for a number of DirectShow capture scenarios.

Additionally, for Windows Mobile 5.0, a Camera Application may be used to preview and capture images.

7.4.1 Unit Test Hardware

The OV2640 camera sensor is needed to run the Windows CE 5.0 custom Camera CETK test, the Windows CE 5.0 custom Camera application, the Windows Mobile 5.0 Camera WMCETK tests, and the Windows Mobile 5.0 Camera application. The camera sensor is required to capture the camera image data.

7.4.2 Unit Test Software

7.4.2.1 Custom Camera CETK Test

Table 7-2 lists the required software to run the custom Camera Test.

Table 7-2. Software Requirements to Run Camera Test

Requirements	Description
Tux.exe	Tux test harness, which is needed for executing the test
Kato.dll	Kato logging engine, which is required for logging test data
Tooltalk.dll	Library required by Tux.exe and Kato.dll. Handles the transport between the target device and the development workstation
Camtest.dll	Main test .dll file.

7.4.2.2 Custom Camera Application

The `camapp.exe` file is needed to run the custom Camera Application.

7.4.2.3 Camera Application

No additional actions are required to include the Windows Mobile 5.0 Camera Application in an OS image beyond the required registry keys.

7.4.3 Building the Camera Tests

7.4.3.1 Custom Camera CETK Test

In order to build the custom Camera CETK test, you need to build an OS image for the desired configuration.

To build the OS image:

1. In Platform Builder, at the **Build OS** menu, select **Open Release Directory**. This opens a DOS prompt.
2. Change to the Camera Test directory. (for the i.MX31, \WINCE500\SUPPORT\TESTS\Camera)
3. Enter **set WINCEREL=1** on the command prompt and press Enter. This copies the built DLL to the flat release directory.
4. Enter the build command at the prompt and press Enter.

After the build completes, the `camtest.dll` file will be located in the $(_FLATRELEASEDIR) directory.

7.4.3.2 Custom Camera Application

In order to build the custom Camera CETK test, you need to build an OS image for the desired configuration.

To build the OS image:

1. Create an APP folder under the `WINCE500\Platform\3DS\Src` folder.
2. Create an empty `dirs` file under the `WINCE500\Platform\3DS\Src\App` folder.
3. Copy the CAMAPP folder from the `WINCE500\Support\App` folder to the `WINCE500\Platform\3DS\Src\App` folder.
4. Select the File View tab of the Platform Builder Workspace window.
5. Expand **Favorites** > **3DS PLATFORM** > **src** > **App.**
6. Right-click on the CAMAPP folder and enable the **Clean Before Building** option.
7. Right-click on the CAMAPP folder again and select **Build Current Project.**

The CAMAPP execution file (`camapp.exe`) will be created in the workspace release directory

7.4.4 Running the Camera Tests

7.4.4.1 Running the Custom Camera CETK Test

The following command executes the Custom Camera CETK Test:

```
tux –o –d camtest.dll
```

Table 7-3 describes the test cases contained in the test suite:

Table 7-3. Camera Test Cases

Test Case	Description
1	This function initializes the camera for testing. The following tasks are performed: - Retrieving panel dimensions, so that it can be drawn directly to the framebuffer during testing. - Creating message queues for each pin instance - Creating a pin instance for PREVIEW, CAPTURE, and STILL. - Reading the Datarange information for each pin into a global variable.
2	This function tests the functionality of camera capture. Images are captured for the CAPTURE Pin. It contains a loop in which a CAPTURE pin image is read from its message queue, drawn the display, and then a new buffer in enqueued. Intermittently, the camera driver is modified. These modifications include changing the camera zoom and modifying the flipping orientation.
3	This function test the functionality of camera preview. Images are capture for the PREVIEW Pin. It contains a loop in which a PREVIEW pin image is read from its message queue, and then a new buffer is enqueued. Intermittently, the PREVIEW pin or the camera driver is modified. These modifications include changing the camera zoom, modifying the flipping orientation, and changing the display offset in the display panel. Additionally, the STILL pin is tested, capturing a still image and displaying it to the display panel.

7.4.4.2 Running the Custom Camera Application

The following command executes the Custom Camera Application:

```
Windows CE>s camapp.exe
```

7.5 Camera Driver API Reference

For the camera driver API reference, see the Windows Mobile Version 5.0 documentation. There is one additional custom API provided to allow applications to enable direct display of video preview data.

For reference information on basic camera driver functions, methods, and structures, see the Windows Mobile Help:

Developing a Device Driver > Windows Mobile-based Device Drivers > Camera Drivers > Camera Driver Reference

7.5.1 IOCTL_SET_DIRECT_DISPLAY_MODE (For the i.MX31 only)

This Camera **DeviceIoControl** request turns Direct Display mode on or off. With Direct Display mode on, when the PREVIEW pin is set to the RUN state, the video preview data will be sent directly to the display.

Parameters *nInBufferSize*

TRUE (or 1) to enable Direct Display mode, FALSE (or 0) to disable Direct Display mode.

Chapter 8
Chip Support Package Driver Development Kit (ARM11 CSPDDK)

The BSP includes a component called the chip support package driver development kit (CSPDDK), which provides an interface used to access the peripheral features and SoC configuration shared by the system. The CSPDDK executes as a device driver DLL and exports functions for the following SoC components:

- System clocking (CCM)
- GPIO
- DMA (SDMA)
- Pin multiplexing and pad configuration (IOMUX)
- EDIO (not available on the i.MX31)

8.1 CSPDDK Driver Summary

Table 8-1 identifies the source code location, library dependencies, and other BSP information:

Table 8-1. CSPDDK Driver Summary

Driver Attribute	Definition
Target Platform (TGTPLAT)	3DS
Target SOC (TGTSOC)	MX31
MXARM11 CSP Driver Path	..\CSP\ARM\FREESCALE\MXARM11\DRIVERS\CSPDDK
CSP Driver Path	..\CSP\ARM\FREESCALE\<TGTSOC>\DRIVERS\CSPDDK
CSP Static Library	<*TGTSOC*>_ddk.lib
Platform Driver Path	..\PLATFORM\\<*tgtplat*>\SRC\DRIVERS\CSPDDK
Import Library	cspddk.lib
Driver DLL	cspddk.dll
Catalog Item	N/A
SYSGEN Dependency	N/A
BSP Environment Variables	Remove BSP_NOCSPDDK=1

8.2 Supported Functionality

The CSPDDK driver enables the 3-Stack board to provide the following software and hardware support:

- The driver supports an interface that allows synchronized inter-process access to the following set of shared SoC resources:
  - GPIO (DDK_GPIO)
  - SDMA (DDK_SDMA)
  - IOMUX (DDK_IOMUX)
  - CCM (DDK_CLK)
- The driver exposes exported functions that can be invoked without incurring a system call (that is, not a stream driver).

8.3 Hardware Operation

Refer to the *MCIMX31 and MCIMX31L Applications Processors Reference Manual* for detailed operation and programming information.

8.3.1 Conflicts with other SoC peripherals

No conflicts

8.4 Software Operation

8.4.1 Communicating with the CSPDDK

Similar to the CEDDK DLL, the CSPDDK DLL does not require any special initialization. The initialization required by the CSPDDK is performed when the DLL is loaded into the respective process space. Drivers that use the CSPDDK need to link to the CSPDDK export library and invoke the exported functions.

8.4.2 Compile-Time Configuration Options

The CSPDDK exposes compile-time options for configuring the SDMA support. In some cases, these compilation variables are also leveraged by the driver code to expose a central point of controlling SDMA functionality. Table 8-2 describes the available CSPDDK compile options:

Table 8-2. CSPDDK Compile Options

Compilation Variable	Header Location	Description
IMAGE_SHARE_IRAM_SDMA_PA_START	image_cfg.h	Physical starting address in internal RAM (IRAM) where the shared SDMA data structures will be located.
IMAGE_SHARE_IRAM_SDMA_OFFSET	image_cfg.h	Offset in bytes from the base of IRAM for the SDMA data structures.

Table 8-2. CSPDDK Compile Options(Continued)

IMAGE_SHARE_IRAM_SDMA_SIZE	image_cfg.h	Size in bytes of the IRAM reserved for SDMA data structures.
IMAGE_SHARE_SDMA_PA_START	image_cfg.h	Physical starting address in external RAM where the shared SDMA data structures will be located. This address must correspond to the region reserved in config.bib.
IMAGE_SHARE_IRAM_SDMA_SIZE	image_cfg.h	Size in bytes of the external RAM reserved for SDMA data structures. This size must correspond to the region reserved in config.bib.
BSP_SDMA_MC0PTR	bsp_cfg.h	Starting address for the shared SDMA data structures. Set to IMAGE_SHARE_IRAM_SDMA_PA_START to use internal RAM. Set to IMAGE_SHARE_SDMA_PA_START to use external RAM.
BSP_SDMA_CHNPRI_xxx	bsp_cfg.h	Assigns a SDMA channel priority to the respective peripheral. Refer to the individual driver chapters for more information on the specific priorities.
BSP_SDMA_SUPPORT_xxx	bsp_cfg.h	Boolean to specifies if SDMA-based transfers are enabled for each respective peripheral. Refer to the individual driver chapters for more information on the DMA support provided.

The CSPDDK manages the allocation of buffer descriptor chains for drivers and applications. The allocation scheme will first attempt to allocate the buffer descriptor chain from a fixed memory pool within the region specified by BSP_SDMA_MC0PTR. If the CSPDDK is unable to allocate enough storage from this fixed pool, it will dynamically allocate the necessary storage from external memory.

To decrease power consumption in system uses cases, such as audio playback, it is beneficial to configure BSP_SDMA_MC0PTR to point to a reserved internal RAM (IRAM) region and allocate the audio buffers in IRAM. This configuration does not require external memory cycles in the data flow from the audio buffers to the SSI, and it allows the CSPDDK to utilize EMI clock gating to significantly reduce the power consumption. See the audio chapter in this Guide for information on configuring audio DMA support.

8.4.3 Registry Settings

No registry settings need to be modified to use the CSPDDK driver. Because most drivers will use the CSPDDK functionality, the CSPDDK should be one of the first DLLs loaded by Device Manager.

8.4.4 Power Management

The CSPDDK exposes interfaces that allow drivers to self-manage power consumption by controlling clocking and pin configuration. The CSPDDK executes as a shared DLL and does not implement the

Power Manager driver IOCTLs or the PowerUp/PowerDown stream interface. However, the CSPDDK functions is invoked by other drivers during power state transitions.

8.5 Unit Test

Due to the heavy use of the CSPDDK routines by other drivers on the system, the CSPDDK tests are currently limited to testing the interface exposed by the DDK_SDMA.

8.5.1 Unit Test Hardware

No hardware is needed to run the unit tests.

8.5.2 Unit Test Software

Table 8-3 lists the software required to run the unit tests.

Table 8-3. Software Requirements for CSPDDK Driver

Requirements	Description
Tux.exe	Tux test harness, which is needed for executing the test
Kato.dll	Kato logging engine, which is required for logging test data
Tooltalk.dll	Library required by Tux.exe and Kato.dll. Handles the transport between the target device and the development workstation
SDMATEST.dll	Test .dll file

8.5.3 Building the Unit Tests

To build the CSPDDK tests, you need to build an OS image for the desired configuration. To build an OS image:

1. In Platform Builder, from the **Build OS** menu option, select **Open Release Directory**. A DOS prompt is displayed.
2. Change to the SDMA Tests directory. (\WINCE500\SUPPORT\TESTS\SDMA)
3. Enter **set WINCEREL=1** on the command prompt and press **Enter**. This copies the built DLL to the flat release directory.
4. Enter the build command at the prompt and press **Enter**.

After the build completes, the `SDMATEST.dll` file will be located in the $(_FLATRELEASEDIR) directory.

8.5.4 Running the Unit Tests

The command line for running the DDK_SDMA tests is:

```
tux -o -d SDMATEST
```

The CSPDDK_SDMA tests do not contain any test-specific command line options.

Table 8-4 describes the test cases contained in the DDK_SDMA tests.

Table 8-4. DDK_SDMA Test Cases

Test Case	Description
1: SDMA Open/Close Channel	Tests open/close operation of the SDMA virtual channels. Attempts to open all available channels and verify that the correct virtual channel ID is returned. All successfully opened channels are then closed.
2: SDMA Memory-to-Memory	Tests the SDMA's ability to perform a memory-to-memory transfer. A virtual channel is requested and then DMA buffers are used to define a memory transfer. The transfer is done in both directions and the results are verified. This transfer is interrupt-driven and uses the standard OAL interrupt registration procedures normally used by device drivers.

8.6 CSPDDK DLL Reference

8.6.1 CSPDDK DLL System Clocking (DDK_CLK) Reference

The DDK_CLK interface allows device drivers to configure and query system clock settings.

8.6.1.1 DDK_CLK Enumerations

Table 8-5. DDK_CLK Enumerations

Programming Element	Description
DDK_CLOCK_SIGNAL	Clock signal name for querying/setting clock configuration.
DDK_CLOCK_GATE_INDEX	Index for referencing the corresponding clock gating control bits within the CCM.
DDK_CLOCK_GATE_MODE	Clock gating modes supported by CCM clock gating registers.
DDK_CLOCK_BAUD_SOURCE	Input source for baud clock generation.
DDK_CLOCK_CKO_SRC	Clock output source (CKO) signal selections.
DDK_CLOCK_CKO_DIV	Clock output source (CKO) divider selections.

8.6.1.2 DDK_CLK Functions

8.6.1.2.1 DDKClockSetGatingMode

This function sets the clock gating mode of the peripheral.

```
BOOL DDKClockSetGatingMode(
        DDK_CLOCK_GATE_INDEX index,
        DDK_CLOCK_GATE_MODE mode)
```

Parameters

index [in] Index for referencing the peripheral clock gating control bits.

mode *[in] Requested clock gating mode for the peripheral.*

Return Values:

Returns TRUE if successful, otherwise returns FALSE.

8.6.1.2.2 DDKClockGetGatingMode

This function retrieves the clock gating mode of the peripheral.

```
BOOL DDKClockGetGatingMode(
        DDK_CLOCK_GATE_INDEX index,
        DDK_CLOCK_GATE_MODE *pMode)
```

Parameters

index [in] Index for referencing the peripheral clock gating control bits.

pMode *[out] Current clock gating mode for the peripheral.*

Return Values:

Returns TRUE if successful, otherwise returns FALSE.

8.6.1.2.3 DDKClockGetFreq

This function retrieves the clock frequency in Hz for the specified clock signal.

```
BOOL DDKClockGetFreq(
        DDK_CLOCK_SIGNAL sig,
        UINT32 *freq)
```

Parameters

sig [in] Clock signal.

freq [out] Current frequency in Hz.

Return Values:

Returns TRUE if successful, otherwise returns FALSE.

8.6.1.2.4 DDKClockConfigBaud

This function configures the input source clock and dividers for the specified CCM peripheral baud clock output.

```
BOOL DDKClockConfigBaud(
        DDK_CLOCK_SIGNAL sig,
        DDK_CLOCK_BAUD_SOURCE src,
```

```
        UINT32 preDiv,
        UINT32 postDiv)
```

Parameters

sig	[in] Clock signal to configure.
src	[in] Selects the input clock source.
preDiv	[in] Specifies the value programmed into the baud clock predivider.
postDiv	[in] Specifies the value programmed into the baud clock postdivider.

Return Values:

Returns TRUE if successful, otherwise returns FALSE.

8.6.1.2.5 DDKClockSetCKO

This function configures the clock output source (CKO) signal.

```
BOOL DDKClockSetCKO(
        BOOL bEnable,
        DDK_CLOCK_CKO_SRC src,
        DDK_CLOCK_CKO_DIV div)
```

Parameters

bEnable	[in] Set to TRUE to enable CKO output. Set to FALSE to disable CKO output.
src	[in] Selects the CKO source signal.
div	[in] Specifies the CKO divide factor.

Return Values: Returns TRUE if successful, otherwise returns FALSE.

8.6.1.2.6 DDKClockEnablePanicMode

This function informs the DVFC driver that a panic condition exists and forces the system to immediately transition to the highest DVFS setting. This function can be used to prevent the DVFS logic from requesting a lower frequency and voltage setting.

```
BOOL DDKClockEnablePanicMode(void)
```

Parameters None.

Return Values: Returns TRUE if successful, otherwise returns FALSE.

8.6.1.2.7 DDKClockDisablePanicMode

This function informs the DVFC driver that a panic condition no longer exists and allows the system to return to normal DVFS operation.

```
BOOL DDKClockDisablePanicMode(void)
```

Parameters None

Return Values: Returns TRUE if successful, otherwise returns FALSE.

8.6.1.2.8 DDKClockBusScale

This function requests the DVFC driver to scale the AHB bus clock by an integer ratio.

CAUTION

This function cannot be used on platforms using DDR memory.

```
BOOL DDKClockBusScale(UINT32 ratio)
```

Parameters

ratio [in] Specifies the integer ratio used to scale the AHB bus clock.

Return Values: Returns TRUE if successful, otherwise returns FALSE.

8.6.1.3 DDK_CLK Examples

Example 8-1. Example: CSPDDK Clock Gating

```
#include “csp.h”    // Includes CSPDDK definitions

// Enable keypad peripheral clock
DDKClockSetGatingMode(DDK_CLOCK_GATE_INDEX_KPP, DDK_CLOCK_GATE_MODE_ENABLED_ALL);

// Disable keypad peripheral clock
DDKClockSetGatingMode(DDK_CLOCK_GATE_INDEX_KPP, DDK_CLOCK_GATE_MODE_DISABLED);
```

Example 8-2. Example: CSPDDK Clock Rate Query

```
#include “csp.h”    // Includes CSPDDK definitions

UINT32 freq;

// Query the current bus clock
DDKClockGetFreq(DDK_CLOCK_SIGNAL_AHB, &freq);
```

8.6.2 CSPDDK DLL GPIO (DDK_GPIO) Reference

The DDK_GPIO interface allows device drivers to utilize the GPIO ports. Each GPIO port has a single interrupt request line that is shared for all port pins. In addition, configuration, status, and data registers are shared. The DDK_GPIO provides safe access to the shared GPIO resources.

8.6.2.1 DDK_GPIO Enumerations

Table 8-6. DDK_GPIO Enumerations

Programming Element	Description
DDK_GPIO_PORT	Specifies the GPIO module instance.
DDK_GPIO_DIR	Specifies the direction the GPIO pins.
DDK_GPIO_INTR	Specifies the detection logic used for generating GPIO interrupts.

8.6.2.2 DDK_GPIO Functions

8.6.2.2.1 DDKGpioSetConfig

This function sets the GPIO configuration (direction and interrupt) for the specified pin.

```
BOOL DDKGpioSetConfig(
        DDK_GPIO_PORT port,
        UINT32 pin,
        DDK_GPIO_DIR dir,
        DDK_GPIO_INTR intr)
```

Parameters

port [in] GPIO module instance.

pin [in] GPIO pin [0-31].

dir [in] Direction for the pin.

intr [in] Interrupt configuration for the pin.

Return Values: Returns TRUE if successful, otherwise returns FALSE.

8.6.2.2.2 DDKGpioBindIrq

This function binds the specified GPIO line with an IRQ that is registered with the OAL to receive interrupts.

```
BOOL DDKGpioBindIrq(
        DDK_GPIO_PORT port,
        UINT32 pin,
        DWORD irq)
```

Parameters

port [in] GPIO module instance.

pin [in] GPIO pin [0-31].

irq [in] Specifies the hardware IRQ that will be translated into a registered SYSINTR within OEMInterruptHandler when the configured interrupt condition for the GPIO line occurs.

Return Values: Returns TRUE if successful, otherwise returns FALSE.

8.6.2.2.3 DDKGpioUnbindIrq

This function unbinds the specified GPIO line from an IRQ that is registered with the OAL to receive interrupts.

```
BOOL DDKGpioUnbindIrq (
        DDK_GPIO_PORT port,
        UINT32 pin)
```

Parameters

port [in] GPIO module instance.

pin [in] GPIO pin [0-31].

Return Values: Returns TRUE if successful, otherwise returns FALSE.

8.6.2.2.4 DDKGpioWriteData

This function writes the GPIO port data to the specified pins.

```
BOOL DDKGpioWriteData(
        DDK_GPIO_PORT port,
        UINT32 portMask,
        UINT32 data)
```

Parameters

port [in] GPIO module instance.

portMask [in] Bit mask for data port pins to be written.

data [in] Data to be written.

Return Values: Returns TRUE if successful, otherwise returns FALSE.

8.6.2.2.5 DDKGpioWriteDataPin

This function writes the GPIO port data to the specified pin.

```
BOOL DDKGpioWriteDataPin(
        DDK_GPIO_PORT port,
        UINT32 pin,
        UINT32 data)
```

Parameters

port [in] GPIO module instance.

pin [in] GPIO pin [0-31].

data [in] Data to be written [0 or 1].

Return Values: Returns TRUE if successful, otherwise returns FALSE.

8.6.2.2.6 DDKGpioReadData

This function reads the GPIO port data from the specified pins.

```
BOOL DDKGpioReadData(
        DDK_GPIO_PORT port,
        UINT32 portMask,
        UINT32 *pData)
```

Parameters

port [in] GPIO module instance.

portMask [in] Bit mask for data port pins to be read.

pData [out] Points to buffer for data read.

Return Values: Returns TRUE if successful, otherwise returns FALSE.

8.6.2.2.7 DDKGpioReadDataPin

This function reads the GPIO port data from the specified pin.

```
BOOL DDKGpioReadDataPin (
        DDK_GPIO_PORT port,
        UINT32 pin,
        UINT32 *pData)
```

Parameters

port	[in] GPIO module instance.
pin	[in] GPIO pin [0-31].
pData	[out] Points to buffer for data read. Data will be shifted to the LSB.
Return Values:	Returns TRUE if successful, otherwise returns FALSE.

8.6.2.2.8 DDKGpioReadIntr

This function reads the GPIO port interrupt status for the specified pins.

```
BOOL DDKGpioReadIntr(
        DDK_GPIO_PORT port,
        UINT32 portMask,
        UINT32 *pStatus)
```

Parameters

port	[in] GPIO module instance.
portMask	[in] Bit mask for interrupt status bits to be read.
pStatus	[out] Points to buffer for interrupt status.
Return Values:	Returns TRUE if successful, otherwise returns FALSE.

8.6.2.2.9 DDKGpioReadIntrPin

This function reads the GPIO port interrupt status from the specified pin.

```
BOOL DDKGpioReadIntrPin(
        DDK_GPIO_PORT port,
        UINT32 pin,
        UINT32 *pStatus)
```

Parameters

port	[in] GPIO module instance.
pin	[in] GPIO pin [0-31].
pStatus	[out] Points to buffer for interrupt status. Status will be shifted to the LSB.
Return Values:	Returns TRUE if successful, otherwise returns FALSE.

8.6.2.2.10 DDKGpioClearIntrPin

This function clears the GPIO interrupt status for the specified pin.

```
BOOL DDKGpioClearIntrPin(
        DDK_GPIO_PORT port,
        UINT32 pin)
```

Parameters

port [in] GPIO module instance.

pin [in] GPIO pin [0-31].

Return Values: Returns TRUE if successful, otherwise returns FALSE.

8.6.2.3 DDK_GPIO Examples

Example 8-3. Example: CSPDDK GPIO Configuration

```
#include “csp.h”    // Includes CSPDDK definitions

// Configure GPIO1_3 as a level-sensitive interrupt input
DDKGpioSetConfig(DDK_GPIO_PORT1, 3, DDK_GPIO_DIR_IN, DDK_GPIO_INTR_HIGH_LEV);

// Clear interrupt status for GPIO1_3
DDKGpioClearIntrPin(DDK_GPIO_PORT1, 3);

// Bind the GPIO interrupt request to the keypad IRQ registered with the OAL.
// An assertion of the GPIO1_3 interrupt will cause keypad IST to be signaled just
// as if the keypad IRQ was asserting.
DDKGpioBindIrq(DDK_GPIO_PORT1, 3, IRQ_KPP);
```

8.6.3 CSPDDK DLL IOMUX (DDK_IOMUX) Reference

The DDK_IOMUX interface allows device drivers to configure signal multiplexing and pad configuration. This control resides inside the IOMUX registers and is shared for the entire system. The DDK_IOMUX support allows drivers to dynamically update and query their signal multiplexing and pad configuration.

8.6.3.1 DDK_IOMUX Enumerations

Table 8-7. DDK_IOMUX Enumerations

Programming Element	Description
DDK_IOMUX_PIN	Specifies the functional pin name used to configure the IOMUX. The enum value corresponds to the bit offset within the SW_MUX_CTL registers.
DDK_IOMUX_OUT	Specifies the muxing on the output path for a signal.
DDK_IOMUX_IN	Specifies the muxing on the input path for a signal.
DDK_IOMUX_GPR	Specifies the general purpose register (GPR) bits within the IOMUX used to control various muxing features within the SoC.
DDK_IOMUX_PAD	Specifies the functional pad name used to configure the IOMUX. The enum value corresponds to the bit offset within the SW_PAD_CTL registers.
DDK_IOMUX_PAD_SLEW	Specifies the slew rate for a pad.
DDK_IOMUX_PAD_DRIVE	Specifies the drive strength for a pad.
DDK_IOMUX_PAD_MODE	Specifies the CMOS/open drain mode for a pad.
DDK_IOMUX_PAD_TRIG	Specifies the trigger for a pad.
DDK_IOMUX_PAD_PULL	Specifies the pull-up/pull-down/keeper configuration for a pad.

8.6.3.2 DDK_IOMUX Functions

8.6.3.2.1 DDKIomuxSetPinMux

This function sets the IOMUX configuration for the specified IOMUX pin.

```
BOOL DDKIomuxSetPinMux(
         DDK_IOMUX_PIN pin,
         DDK_IOMUX_OUT outMux,
         DDK_IOMUX_IN inMux)
```

Parameters

pin [in] Functional pin name used to select the IOMUX output/input path that will be configured.

outMux [in] Output path configuration.

inMux [in] Input path configuration.

Return Values: Returns TRUE if successful, otherwise returns FALSE.

8.6.3.2.2 DDKIomuxGetPinMux

This function gets the IOMUX configuration for the specified IOMUX pin.

```
BOOL DDKIomuxGetPinMux(
         DDK_IOMUX_PIN pin,
         DDK_IOMUX_OUT *pOutMux,
         DDK_IOMUX_IN *pInMux)
```

Parameters

pin	[in] Functional pin name used to select the IOMUX output/input path that will be configured.
pOutMux	[out] Output path configuration.
pInMux	[out] Input path configuration.
Return Values:	Returns TRUE if successful, otherwise returns FALSE.

8.6.3.2.3 DDKIomuxSetPadConfig

This function sets the IOMUX pad configuration for the specified IOMUX pin.

```
BOOL DDKIomuxSetPadConfig(
         DDK_IOMUX_PAD pad,
         DDK_IOMUX_PAD_SLEW slew,
         DDK_IOMUX_PAD_DRIVE drive,
         DDK_IOMUX_PAD_MODE mode,
         DDK_IOMUX_PAD_TRIG trig,
         DDK_IOMUX_PAD_PULL pull)
```

Parameters

pad	[in] Functional pad name used to select the pad that will be configured.
slew	[in] Slew rate configuration.
drive	[in] Drive strength configuration.
mode	[in] CMOS/open-drain output mode configuration.
trig	[in] Trigger configuration.
pull	[in] Pull-up/pull-down/keeper configuration.
Return Values:	Returns TRUE if successful, otherwise returns FALSE.

8.6.3.2.4 DDKIomuxGetPadConfig

This function gets the IOMUX pad configuration for the specified IOMUX pad.

```
BOOL DDKIomuxSetPadConfig(
         DDK_IOMUX_PAD pad,
         DDK_IOMUX_PAD_SLEW *pSlew,
         DDK_IOMUX_PAD_DRIVE *pDrive,
         DDK_IOMUX_PAD_MODE *pMode,
         DDK_IOMUX_PAD_TRIG *pTrig,
         DDK_IOMUX_PAD_PULL *pPull)
```

Parameters

pad	[in] Functional pad name used to select the pad that will be configured.
pSlew	[in] Slew rate configuration.
pDrive	[in] Drive strength configuration.
pMode	[in] CMOS/open-drain output mode configuration.
pTrig	[in] Trigger configuration.
pPull	[in] Pull-up/pull-down/keeper configuration.
Return Values:	Returns TRUE if successful, otherwise returns FALSE.

8.6.3.2.5 DDKIomuxSetGpr

This function writes a value into the IOMUX GPR register. The GPR controls the muxing of signals within the SoC.

```
BOOL DDKIomuxSetGpr(
         UINT32 mask,
         UINT32 data)
```

Parameters

mask	Bit mask for GPR bits to be written
data	Data to be written
Return Values:	Returns TRUE if successful, otherwise returns FALSE.

8.6.3.2.6 DDKIomuxSetGprBit

This function writes a value into the specified IOMUX GPR bit. These GPR bits are used to control the muxing of signals within the SoC.

```
BOOL DDKIomuxSetGprBit(
         DDK_IOMUX_GPR bit,
         UINT32 data)
```

Parameters

bit	[in] GPR bit to be configured.
data	[in] Value for the GPR bit [0 or 1].
Return Values:	Returns TRUE if successful, otherwise returns FALSE.

8.6.3.3 DDK_IOMUX Examples

Example 8-4. Example: CSPDDK IOMUX Signal Multiplexing

```
#include “csp.h”    // Includes CSPDDK definitions

// Configure the signal multiplexing for GPIO1_3. Route the internal input and
// output path of the GPIO1_3 pin to the GPIO module
DDKIomuxSetPinMux(DDK_IOMUX_PIN_GPIO1_3, DDK_IOMUX_OUT_GPIO, DDK_IOMUX_IN_GPIO);
```

Example 8-5. Example: CSPDDK IOMUX Pad Configuration

```
#include “csp.h”    // Includes CSPDDK definitions

// Configure the GPIO1_3 pad for the following configuration: slow slew rate,
// normal drive strength, CMOS, Schmitt trigger, 100K pull-up.
DDKIomuxSetPadConfig(DDK_IOMUX_PIN_GPIO1_3, DDK_IOMUX_PAD_SLEW_SLOW,
    DDK_IOMUX_PAD_DRIVE_NORMAL, DDK_IOMUX_PAD_MODE_CMOS, DDK_IOMUX_PAD_TRIG_SCHMITT,
    DDK_IOMUX_PAD_PULL_UP_100K);
```

8.6.4 CSPDDK DLL SDMA (DDK_SDMA) Reference

The DDK_SDMA interface allows device drivers to allocate, configure, and control shared SDMA resources.

8.6.4.1 DDK_SDMA Enumerations

Table 8-8. DDK_SDMA Enumerations

Programming Element	Description
DDK_DMA_ACCESS	Specifies width of the data for a peripheral DMA transfer.
DDK_DMA_FLAGS	Specifies mode flags within the DMA buffer descriptor.
DDK_DMA_REQ	Specifies DMA request used to trigger SDMA channel execution.

8.6.4.2 DDK_SDMA Functions

8.6.4.2.1 DDKSdmaOpenChan

This function attempts to find an available virtual SDMA channel that can be used to support a memory-to-memory, peripheral-to-memory, or memory-to-peripheral transfers.

```
UINT8 DDKSdmaOpenChan(
        DDK_DMA_REQ dmaReq,
        UINT8 priority,
        LPTSTR lpName,
        DWORD irq)
```

Parameters

dmaReq [in] Specifies the DMA request that will be bound to a virtual channel.

priority [in] Priority assigned to the opened channel.

lpName [in] Not currently used. Set to NULL.

irq [in] Only used if lpName is set to NULL. Specifies the hardware IRQ that will be translated into a registered SYSINTR within OEMInterruptHandler when a transfer interrupt occurs. Set to IRQ_NONE if no interrupt should be generated by the channel.

Return Values: Returns a non-zero virtual channel index if successful, otherwise returns 0.

8.6.4.2.2 DDKSdmaUpdateSharedChan

This function allows a channel that has multiple DMA requests combined into a shared DMA event to be reconfigured for one of the alternate DMA requests.

```
BOOL DDKSdmaUpdateSharedChan(
        UINT8 chan,
        DDK_DMA_REQ dmaReq)
```

Parameters

chan [in] Virtual channel returned by DDKSdmaOpenChan.

dmaReq [in] Specifies the DMA request that will be bound to a virtual channel.

Return Values: Returns TRUE if successful, otherwise returns FALSE.

8.6.4.2.3 DDKSdmaCloseChan

This function closes a virtual DMA channel previously opened by DDKSdmaOpenChan.

```
BOOL DDKSdmaCloseChan(
        UINT8 chan)
```

Parameters

chan [in] Virtual channel returned by DDKSdmaOpenChan function.

Return Values: Returns TRUE if successful, otherwise returns FALSE.

8.6.4.2.4 DDKSdmaAllocChain

This function allocates a chain of buffer descriptors for a virtual DMA channel.

```
BOOL DDKSdmaAllocChain(
        UINT8 chan,
        UINT32 numBufDesc)
```

Parameters

chan	[in] Virtual channel returned by DDKSdmaOpenChan.
Return Values:	Returns TRUE if successful, otherwise returns FALSE.

8.6.4.2.5 DDKSdmaFreeChain

This function frees a chain of buffer descriptors previously allocated with DDKSdmaAllocChain.

```
BOOL DDKSdmaFreeChain(
        UINT8 chan)
```

Parameters

chan	[in] Virtual channel returned by DDKSdmaOpenChan.
Return Values:	Returns TRUE if successful, otherwise returns FALSE.

8.6.4.2.6 DDKSdmaSetBufDesc

This function configures a buffer descriptor for a DMA transfer.

```
BOOL DDKSdmaSetBufDesc(
        UINT8 chan,
        UINT32 index,
        UINT32 modeFlags,
        UINT32 memAddr1PA,
        UINT32 memAddr2PA,
        DDK_DMA_ACCESS dataWidth,
        UINT16 numBytes)
```

Parameters

chan	[in] Virtual channel returned by DDKSdmaOpenChan.
index	[in] Index of buffer descriptor within the chain to be configured.
modeFlags	[in] Specifies the buffer descriptor mode word flags that control the “continue”, “wrap”, and “interrupt” settings.
memAddr1PA	[in] For memory-to-memory transfers, this parameter specifies the physical memory source address for the transfer. For memory-to-peripheral transfers, this parameter specifies the physical memory source address for the transfer. For peripheral-to-memory transfers, this parameter specifies the physical memory destination address for the transfer.
memAddr2PA	[in] Used only for memory-to-memory transfers to specify the physical memory destination address for the transfer. Ignored for memory-to-peripheral and peripheral-to-memory transfers.
dataWidth	[in] Used only for memory-to-peripheral and peripheral-to-memory transfers to specify the width of the data for the peripheral transfer. Ignored for memory-to-memory transfers.
numBytes	[in] Virtual channel returned by DDKSdmaOpenChan.
Return Values:	Returns TRUE if successful, otherwise returns FALSE.

8.6.4.2.7 DDKSdmaGetBufDescStatus

This function retrieves the status of the “done” and “error” bits from a single buffer descriptor within of a chain.

```
BOOL DDKSdmaGetBufDescStatus(
        UINT8 chan,
        UINT32 index,
        UINT32 *pStatus)
```

Parameters

chan	[in] Virtual channel returned by DDKSdmaOpenChan.
index	[in] Index of buffer descriptor within the chain.
pStatus	[in] Points to a buffer that will be filled with the status of the buffer descriptor.
Return Values:	Returns TRUE if successful, otherwise returns FALSE.

8.6.4.2.8 DDKSdmaGetChainStatus

This function retrieves the status of the “done” and “error” bits from all of the buffer descriptors of a chain.

```
BOOL DDKSdmaGetChainStatus(
        UINT8 chan,
        UINT32 *pStatus)
```

Parameters

chan	[in] Virtual channel returned by DDKSdmaOpenChan.
pStatus	[in] Points to an array that will be filled with the status of each buffer descriptor in the chain.
Return Values:	Returns TRUE if successful, otherwise returns FALSE.

8.6.4.2.9 DDKSdmaClearBufDescStatus

This function clears the status of the “done” and “error” bits within the specified buffer descriptor.

```
BOOL DDKSdmaClearBufDescStatus(
        UINT8 chan,
        UINT32 index)
```

Parameters

chan	[in] Virtual channel returned by DDKSdmaOpenChan.
index	[in] Index of buffer descriptor within the chain.
Return Values:	Returns TRUE if successful, otherwise returns FALSE.

8.6.4.2.10 DDKSdmaClearChainStatus

This function clears the status of the “done” and “error” bits within all of the buffer descriptors of a chain.

```
BOOL DDKSdmaClearChainStatus(
        UINT8 chan)
```

Parameters

chan	[in] Virtual channel returned by DDKSdmaOpenChan.
Return Values:	Returns TRUE if successful, otherwise returns FALSE.

8.6.4.2.11 DDKSdmaInitChain

This function initializes a buffer descriptor chain and the context for a channel. It should be invoked when before a virtual DMA channel is initially started, and when the DMA channel is stopped and restarted.

```
BOOL DDKSdmaInitChain(
        UINT8 chan,
        UINT32 waterMark)
```

Parameters

chan	[in] Virtual channel returned by DDKSdmaOpenChan.
waterMark	[in] Specifies the watermark level used by the peripheral to generate a DMA request. This parameter tells the DMA how many transfers to complete for each assertion of the DMA request. Ignored for memory-to-memory transfers.
Return Values:	Returns TRUE if successful, otherwise returns FALSE.

8.6.4.2.12 DDKSdmaClearChainStatus

This function clears the status of the “done” and “error” bits within all of the buffer descriptors of a chain.

```
BOOL DDKSdmaClearChainStatus(
        UINT8 chan)
```

Parameters

chan	[in] Virtual channel returned by DDKSdmaOpenChan.
Return Values:	Returns TRUE if successful, otherwise returns FALSE.

8.6.4.2.13 DDKSdmaStartChan

This function starts the specified channel.

```
BOOL DDKSdmaStartChan(
        UINT8 chan)
```

Parameters

chan	[in] Virtual channel returned by DDKSdmaOpenChan.
Return Values:	Returns TRUE if successful, otherwise returns FALSE.

8.6.4.2.14 DDKSdmaStopChan

This function stops the specified channel.

```
BOOL DDKSdmaStopChan(
        UINT8 chan,
        BOOL bKill)
```

Parameters

chan	[in] Virtual channel returned by DDKSdmaOpenChan.
bKill	[in] Set TRUE to terminate the channel if it is actively running. Set FALSE to allow the channel to continue running until it yields.
Return Values:	Returns TRUE if successful, otherwise returns FALSE.

Chapter 9
Display Driver

The Windows CE 5.0 BSP display driver is based on the Microsoft DirectDraw Graphics Primitive Engine (DDGPE) classes and supports the Microsoft DirectDraw interface. This driver combines the functionality of a standard LCD display with DirectDraw support. The display driver interfaces with the Image Processing Unit (IPU).

For dumb displays, the IPU Synchronous Display Controller (SDC) combines graphics and video planes and generates display controls with programmable timing. For smart displays, the IPU Post-Processor combines graphics and video planes, and the IPU Asynchronous Display Controller (ADC) generates display controls and programmable timing.

The display driver supports the following display types:

- EPSON L4F00242T03 VGA LCD panel
- PAL and NTSC TV through the Chrontel CH7024 TV encoder chip

9.1 Display Driver Summary

Table 9-1 identifies the source code location, library dependencies, and other BSP information

Table 9-1. Display Driver Summary

Driver Attribute	Definition
Target Platform (TGTPLAT)	3DS
Target SOC (TGTSOC)	MX31
MXARM11 CSP Driver Path	N/A
CSP Driver Path	N/A
CSP Static Library	N/A
Platform Driver Path	..\PLATFORM\<*tgtplat*>\SRC\DRIVERS\IPU\DISPLAY\DDIPU_SDC
Import Library	ddgpe.lib, gpe.lib
Driver DLL	ddraw_ipu_sdc.dll
Catalog Items	Third Party > BSPs > Freescale <*tgtplat*> > Device Drivers > EPSON L4F00242T03 (VGA) Third Party > BSPs > Freescale <*tgtplat*> > Device Drivers > TV Output CH7023/CH7024
SYSGEN Dependency	SYSGEN_DDRAW=1
BSP Environment Variables	BSP_PP=1 BSP_DISPLAY_EPSON_L4F00242T03 = 1 for Epson LCD Panel BSP_TVOUT_CHRONTEL_CH702X = 1 for TV Output

9.2 Supported Functionality

The display driver enables the 3-Stack board to provide the following software and hardware support:

- EPSON 2.8" VGA Display With Touch Screen (L4F00242T03)
- VGA portrait display resolution(480x640)
- RGB565 interface
- DirectDraw Hardware Abstraction Layer (DDHAL)
- Overlay surface
- Video overlays containing image data in the FOURCC UYVY pixel format
- Hardware-accelerated color space conversion in video overlays
- Hardware-accelerated image resizing in video overlays
- Overlay surface color key feature
- Overlay surface alpha blending feature
- Two power management modes, full on and full off
- System suspend
- Screen rotation
- Dynamic switch between TV and LCD display

9.3 Hardware Operation

For operation and programming information, see the chapter on the Image Processing Unit (IPU) in the *MCIMX31 and MCIMX31L Applications Processors Reference Manual*.

9.3.1 Rotation Control

The application `rotate.exe` provides the user a way to change the screen orientation while the Windows CE image is running. Clicking rotate application toggles the orientation of the screen between a 0 and 270 degree rotation angle. The default path of rotate.exe is `\windows`.

NOTE

Due to lack of support for the co-existence of GDI screen rotation and DirectDraw), a DirectDraw display driver with rotation support enabled may yield more failures in the GDI/DIRECTDRAW CETK test suite. It is recommended to run these CETK tests with rotation support disabled or under 0 rotation degree. See the Windows CE 5.0 Help, stating that “GDI screen rotation cannot be used with DirectDraw”.

9.3.2 TV Output Mode

Application `tvout.exe` provides the user a way to change between LCD and TV Output mode (PAL standard). Clicking the TV-Out application toggles between these two modes. The default path of `tvout.exe` is `\windows`.

The TV-Out application sets the TV output mode to PAL or NTSC standards, depending on the received parameter. “tvout.exe 0” sets the TV output mode to PAL, “tvout.exe 1” sets it to NTSC.

The display driver ensures that the output is LCD mode, when responding to the power management to enter the power states D0 (Full On) and D4 (Off).

The TV output mode requires a 270 degree rotation, to enable the primary surface to fit the physical resolution 640x480 that TV can support.

NOTE

Due to lack of support for the co-existence of GDI screen rotation and DirectDraw (see the Windows CE 5.0 Help, stating that “GDI screen rotation cannot be used with DirectDraw”), a DirectDraw display driver with rotation support enabled may yield more failures in the GDI/DIRECTDRAW CETK test suite. It is recommended to run these CETK tests with rotation support disabled or under 0 rotation degree.

9.4 Software Operation

9.4.1 Communicating with the Display

Communication with the display driver is accomplished through Microsoft-defined APIs. A framework for accessing the display driver is provided through the Graphics Device Interface (GDI) and DirectDraw.

9.4.1.1 Using the GDI

The Graphics Device Interface (GDI) provides basic controls for the display of text and graphics. For information, see the Platform Builder Help:

Windows CE Features > Shell and User Interface > Graphics, Windowing and Events > GWES Application Development > Graphics Device Interface

9.4.1.2 Using DirectDraw

The DirectDraw API provides support for hardware-accelerated 2-D graphics, offering fast access to display hardware while retaining compatibility with the GDI.

For information about the DirectDraw API, see the DirectDraw Help or the MSDN documentation library topic:

Windows CE Features > Graphics and Multimedia Technologies > Graphics > DirectDraw

The following DirectDraw features are supported in the display driver by the IPU hardware:

- Page flipping with one backbuffer.

- Overlay surfaces using RGB or the FOURCC UYVY pixel format.
- Overlaying using a color key for the overlay surface for RGB colors.
- Overlaying using a color key for the non-overlay graphics surface for RGB colors.
- Stretching of overlay surfaces.

The IPU contains Post-Processing hardware, which is used within the display driver to accelerate the following operations:

- Color space conversion of UYVY overlay data to RGB. This conversion is required in order to combine the overlay data with RGB graphics plane data in the IPU SDC.
- Resizing of the overlay surface.
- Rotation of the overlay surface (used when the screen orientation is rotated).
- Resizing and rotation of the primary graphics surface when TV Output mode is enabled and active. This is required to obtain a 640x480 resolution image for output to a TV.

NOTE

Setting environment variable "BSP_DISPLAY_DELAYFLIP" to “1” enables the feature to support delay flip (which is synchronous) for overlay surfaces.

9.4.1.3 Using Display Driver Escape Codes

In some cases, applications might need to communicate directly with a display driver. To make this possible, an escape code mechanism is provided as part of the display driver. For a detailed description of standard display driver escape codes, see the Platform Builder Help:

Developing a Device Driver > **Windows CE Drivers** > **Display Drivers** > **Display Driver Development Concepts** > **Display Driver Escape Codes**

9.4.2 Configuring the Display

The display configuration is based on the **PanelType** registry key, which is described in the **Display Registry Settings** section below. The **PanelType** registry key indicates the display panel that is being used. There is only one supported display panel: The EPSON L4F00242T03 VGA LCD panel.

9.4.2.1 Rotation Support

The DirectDraw display driver may be configured to allow screen rotation, through a parameter in the `bsp_cfg.h` file. If the BSP_DIRECTDRAW_SUPPORT_ROTATION parameter is set to TRUE, the DirectDraw display driver will support rotation. If it is set to FALSE, it will not.

NOTE

Due to lack of support for the co-existence of GDI screen rotation and DirectDraw , a DirectDraw display driver with rotation support enabled may yield more failures in the GDI/DIRECTDRAW CETK test suite. It is recommended to run these CETK tests with rotation support disabled or under 0 rotation degree. See the Windows CE 5.0 Help, stating that “GDI screen rotation cannot be used with DirectDraw”.

9.4.2.2 Display Registry Settings

The following registry keys are optionally included, depending on the display panel catalog item included in the OS design.

If the Epson VGA panel is selected, the following registry keys are included:

```
[HKEY_LOCAL_MACHINE\Drivers\Display\DDIPU_SDC]
   "Bpp"=dword:10               ; 16bpp
   "PanelType"=dword:1          ; Epson VGA Panel
   "VideoMemSize"=dword:450000 ; 4.3MB
```

If TV Output is included in the OS design, the following registry keys are also included:

```
[HKEY_LOCAL_MACHINE\Drivers\Display\DDIPU_SDC]
   "TVSupported"=dword:1        ; Flag indicates TV out mode is supported
   "VideoMemSize"=dword:500000   ; 5.0MB
```

The **VideoMemSize** is set to a value sufficiently high to permit the allocation of memory for several surfaces: The primary graphics surface, two video overlay surfaces, a surface used for TV output (if TV output is enabled) and additional surfaces created by applications, such as the Windows CE Media Player.

Bpp refers to bits per pixel: 0x10, or 16.

9.4.3 Power Management

The display driver consumes power primarily through the operation of various IPU sub-modules, such as the SDC, which combines and displays video and graphics data, and through the operation of the display panel. To facilitate management of these modules, the display driver implements the power management I/O Control (IOCTL) codes like IOCTL_POWER_CAPABILITIES, IOCTL_POWER_QUERY, IOCTL_POWER_GET and IOCTL_POWER_SET.

9.4.3.1 PowerUp

This function is not implemented for the display driver.

9.4.3.2 PowerDown

This function is not implemented for the display driver.

9.4.3.3 IOCTL_POWER_SET

The display driver implements the IOCTL_POWER_SET IOCTL API with support for the D0 (Full On) and D4 (Off) power states. These states are handled in the following manner:

- D0 – The display panel is enabled. The IPU's Display Interface (DI) and SDC modules are enabled.
- D4 – The DI and SDC modules of the IPU are disabled. The display panel is disabled.

9.5 Unit Test

The display driver is subject to two test suites provided with the Windows CE Test Kit (CETK): the Graphics Device Interface (GDI) Test and the DirectDraw Test. Additionally, video playback may be verified using the Windows Media Player application.

The GDI Test is designed to test a graphics device interface. This test verifies that basic shapes, including rectangles, triangles, circles, and ellipses, are drawn correctly. The test also examines the color palette of the display, verifies that the display is correctly divided into multiple regions, and tests whether a device context can be properly created, stored, retrieved, and destroyed.

The DirectDraw Test analyzes basic DirectDraw functionality including block image transfers (blits), scaling, color keying, color filling, flipping, and overlaying.

Windows Media Player may be used to play back WMV video files and visually verify correct operation of video overlays, accelerated color space conversion, and accelerated image resizing.

9.5.1 Unit Test Hardware

The EPSON L4F00242T03 VGA Panel is needed to run the GDI and DirectDraw tests. The panel displays the graphics data.

9.5.2 Unit Test Software

9.5.2.1 GDI Tests

Table 9-2 lists the software required to run the GDI tests.

Table 9-2. Software Requirements to Run GDI Tests

Requirements	Description
Tux.exe	Tux test harness, which is needed for executing the test
Kato.dll	Kato logging engine, which is required for logging test data
Tooltalk.dll	Library required by Tux.exe and Kato.dll. Handles the transport between the target device and the development workstation
Gdiapi.dll	Main test .dll file.
Ddi_test.dll	Graphics Primitive Engine (GPE)–based display driver that the GDI API uses to verify the success of each test case. If Ddi_test.dll is unavailable, run the test with manual verification.

9.5.2.2 DirectDraw Tests

Table 9-3 lists the software required to run the DirectDraw tests:

Table 9-3. Software Requirements to Run DirectDraw Tests

Requirements	Description
Tux.exe	Tux test harness, which is needed for executing the test
Kato.dll	Kato logging engine, which is required for logging test data
Tooltalk.dll	Library required by Tux.exe and Kato.dll. Handles the transport between the target device and the development workstation
DDrawTK.dll	Test .dll file

9.5.2.3 Windows Media Player Tests

Table 9-4 lists the software required to perform WMV playback with Windows Media Player:

Table 9-4. Software Requirements to Perform WMV Playback

Requirements	Description
Ceplayer.exe	Windows Media Player sample application.
*.wmv sample video files	Sample windows media files.

9.5.3 Building the Display Tests

The GDI and DirectDraw tests come pre-built as part of the CETK. Ensure that you have the latest CETK suite. No steps are required to build these tests. The DDrawTK.dll, Gdiapi.dll, and Ddi_test.dll files can be found with the other required CETK files in the following location:

```
[Drive]:\Program Files\Windows CE Platform Builder\5.00\cepb\wcetk\ddtk\armv4I
```

For Windows Media Player testing, there are no build steps required. The Windows Media Player catalog item must be added to the OS image to ensure that `ceplayer.exe` is included in the image. Additionally, sample WMV files must be included in the image to demonstrate playback.

9.5.4 Running the Display Tests

9.5.4.1 Running the GDI Tests

The command line for running the GDI tests is:

```
tux -o -d gdiapi.dll
```

For information about the GDI tests and command line options, see the Platform Builder Help:

Debugging and Testing > **Tools for Debugging and Testing** > **Windows CE Test Kit** > **CETK Tests** > **DirectDraw Test** > **Modifying the Graphics Device Interface Test**

Table 9-5 describes the test cases contained in the GDI test suite.

Table 9-5. GDI Test Cases

Test case	Description
100-104: Clip	Tests the functionality of clipping using different shapes and verifies the functionality of complex clip regions.
200-231: Draw	Calls functions that draw and functions that apply complex effects to drawing. These test cases perform blitting, line drawing, filling, color table manipulation, bitmap type creation, and device attribute modification. **NOTE**: Test case 231, AlphaBlend, is skipped, as it is not supported in the image.
300-307: Palette	Verifies color matching and color conversion for palettes, and modifies associated palettes.
500-512: Region	Tests region management by calling functions that modify region rectangles.
600-608: Brush and pen	Assesses the functionality of brushes and brush alignments.
700-710: Device attribute	Verifies device attributes and exercises functions that modify device attributes.
800-808: Device context	Creates, retrieves, saves, and restores a device context.
900-905: Device object	Calls functions that retrieve, modify, and delete GDI objects.
1000-1011: Font	Verifies font enumeration, selection, and attributes.
1100-1108: Text	Writes text to various locations on the display. If the font required by the test is not available, some test cases do not run.
1200-1205: Print	Passes bad parameters to printing functions.
1300-1303: Verify	Assesses the functionality of test verification functions, such as **CheckScreenHalves** and **CheckAllWhite**.
1400, 1401: Manual	Manually tests font drawing. You can use these test cases to exercise code paths. To step through these test cases, press the left SHIFT key.

9.5.4.2 Running the DirectDraw Tests

The command line for running the DirectDraw tests is:

```
tux –o –d ddrawtk
```

For information about the DirectDraw tests and command line options, see the Platform Builder Help:

Testing > **Tools for Debugging and Testing** > **Windows CE Test Kit** > **CETK Tests** > **DirectDraw Test** > **Modifying the DirectDraw Test**

Table 9-6 describes the test cases contained in the DirectDraw test suite.

Table 9-6. DirectDraw Test Cases

Test case	Description
100: Get Caps	Retrieves the capabilities of the hardware abstraction layer (HAL), verifies that the operation is successful, and then displays all capabilities retrieved. This test case fails if it cannot retrieve the capabilities of the HAL.
101: Enumerate Display Modes	Enumerates the DirectDraw display modes, verifies that the enumeration completes, and then displays the results of the enumeration.
200: Blt (Windowed Mode)	Executes blits to and from various surfaces. The test case verifies that the actual destination matches the expected destination for each blit. This test case fails if any blits are unsuccessful.
210: ColorKey Blt (Windowed Mode)	Executes a variety of color key blits to and from assorted surfaces. The test case verifies that the actual destination matches the expected destination for each test. This test case fails if any color key blits are unsuccessful.
220: Color Filling Blts (Windowed Mode)	Executes a variety of color fill blits to assorted surfaces. The test case verifies that the surface is filled with the specified color. This test case fails if any color fill blits are unsuccessful.
300: Blt (Exclusive Mode)	Executes a variety of blits to and from assorted surfaces. The test case verifies that the actual destination matches the expected destination for test. This test case fails if any blits are unsuccessful.
310: ColorKey Blt (Exclusive Mode)	Executes a variety of color key blits to and from assorted surfaces. The test case verifies that the actual destination matches the expected destination for each test. This test case fails if any color key blits are unsuccessful.
320: Color Filling Blts (Exclusive Mode)	Executes a variety of color fill blits to assorted surfaces. The test case verifies that the surface is filled with the specified color. This test case fails if any color fill blits are unsuccessful.
330: Flip (Exclusive Mode)	Executes a variety of blits to a flipping chain and verifies that the flips are successful and that all surfaces display correctly. This test case fails if any flips or surface verifications fail.
400: CreateVideoPort (Video Port Container Test)	Enumerates the available video ports and connections for the video ports. This test case then verifies that each enumerated connection can be created.
410: EnumVideoPorts (Video Port Container Test)	Enumerates the available video ports and then enumerates the video ports based on specific capabilities.
420: GetVideoPortConnectInfo (Video Port Container Test)	Verifies that the **GetVideoPortConnectInfo** function appropriately handles a variety of input conditions.
430: QueryVideoPortStatus (Video Port Container Test)	Verifies that the **QueryVideoPortStatus** function appropriately handles each video port.
500: GetBandwidthInfo (Video Port Test)	Verifies that the **GetBandwidthInfo** function returns consistent information about each type of video port available.
502: GetSetColorControls (Video Port Test)	Verifies that the **GetColorControls** and **SetColorControls** functions set and return consistent color controls if color control is supported.
504: GetInputOutputFormats (Video Port Test)	Verifies that the **GetInputFormats** and **GetOutputFormats** functions return consistent information under a variety of calling conditions.

Table 9-6. DirectDraw Test Cases(Continued)

Test case	Description
506: GetFieldPolarity (Video Port Test)	Verifies that the **GetFieldPolarity** function returns consistent information about the field polarity of a video port.
508: GetVideoLine (Video Port Test)	Verifies that the **GetVideoLine** function returns consistent information.
510: GetVideoSignalStatus (Video Port Test)	Verifies that the **GetVideoSignalStatus** function returns consistent information.
512: SetTargetSurface (Video Port Test)	Verifies that the **SetTargetSurface** function appropriately handles improper input.
514: StartVideo (Video Port Test)	Verifies that calls to the **StartVideo** function succeed with a variety of flags set.
516: StopVideo (Video Port Test)	Verifies that a call to the **StopVideo** function succeeds.
518: UpdateVideo (Video Port Test)	Verifies that calls to the **UpdateVideo** function succeed with a variety of flags set.
520: WaitForSync (Video Port Test)	Verifies that the **WaitForSync** function behaves consistently with each possible flag.
1200: Blt (Interactive Windowed Mode)	Executes blits that include color fills and scaling. You must manually verify that each blit is successful.
1240: Overlay Blt (Interactive Windowed Mode)	Executes blits to an overlay surface. The test cannot verify that the contents of the overlay surface display correctly, so you should manually verify that the output is correct. If the test succeeds, an overlay appears in the middle of the screen with a blue and yellow checkerboard pattern. This test may run as many as three times with the same output, testing RGB, YUYV, and VYUY pixel formats on the overlay surface.
1250: ColorKeyOverlay Blt (Interactive Windowed Mode)	Executes color key blits to an overlay surface. The test cannot verify that the contents of the overlay surface display correctly, so you should manually verify that the output is correct. If the test succeeds, a blue checkerboard pattern appears in a variety of locations across the display of the target device. If the driver enables stretching, the test also stretches the checkerboard pattern. This test may run as many as three times with the same output, testing RGB, YUYV, and VYUY pixel formats on the overlay surface.
1260: ColorFill Overlay Blt (Interactive Windowed Mode)	Executes color fill blits to an overlay surface. The test cannot verify that the contents of the overlay surface display correctly, so you should manually verify that the output is correct. This test case cycles through red, green and blue on the primary overlay surface. This test may run as many as three times with the same output, testing RGB, YUYV, and VYUY pixel formats on the overlay surface.
1300: Blt (Interactive Exclusive Mode)	Executes blits that you must verify when prompted. The test does not contain an automated mechanism for verifying scaling.
1340: Overlay Blt (Interactive Exclusive Mode)	Executes blits to an overlay surface. The test cannot verify that the contents of the overlay surface display correctly, so you should manually verify that the output is correct. If the test succeeds, an overlay is located in the middle of the screen on the target device with blue and yellow horizontal lines. The primary surface behind the overlay surface should fill the entire display. This test may run as many as three times with the same output, testing RGB, YUYV, and VYUY pixel formats on the overlay surface.

Table 9-6. DirectDraw Test Cases(Continued)

Test case	Description
1350: ColorKeyOverlay Blt (Interactive Exclusive Mode)	Executes color key blits to an overlay surface. The test cannot verify that the contents of the overlay surface display correctly, so you should manually verify that the output is correct. If the test succeeds, blue horizontal bars appear across the display of the target device. The primary surface behind the overlay surface should fill the entire display. This test may run as many as three times with the same output, testing RGB, YUYV, and VYUY pixel formats on the overlay surface.
1360: ColorFill Overlay Blt (Interactive Exclusive Mode)	Executes color fill blits to an overlay surface. The test cannot verify that the contents of the overlay surface display correctly, so you should manually verify that the output is correct. This test case cycles through red, green and blue on the primary surface and the overlay surface. The color fills on the primary surface should fill the entire display. This test may run as many as three times with the same output, testing RGB, YUYV, and VYUY pixel formats on the overlay surface.

NOTE

The following DirectDraw test cases are not supported by the DirectDraw driver, and are therefore skipped: 410, 420, 430, 500, 502, 504, 506, 508, 510, 512, 514, 516, 518, 520.

9.5.4.3 Running the Windows Media Player tests

The command line for starting playback of a WMV test video clip in Windows Media Player is `ceplayer [wmv test file]` (for example, “ceplayer motocross_208x160_30fps.wmv”). If audio support is not included in the current BSP, the message **Audio hardware is missing or disabled** will pop up when the WMV file is being loaded. Click **OK** to continue to WMV playback.

To confirm the correct operation of this test, observe the application and verify that the video clip is playing at a smooth rate (it should not drop frames or otherwise appear jerky). It should have a clear image, normal coloring, and correct image sizing.

9.6 Display Driver API Reference

For information about the display driver APIs, see Platform Builder Help. No additional custom API information is required for the features currently supported in the display driver.

For reference information on basic display driver functions, methods, and structures, see the Platform Builder Help:

Developing a Device Driver > **Windows CE Drivers** > **Display Drivers** > **Display Driver Reference**

For reference information on DirectDraw functions, callbacks, and structures, see the Platform Builder Help:

Windows CE Features > **Graphics and Multimedia Technologies** > **Graphics** > **DirectDraw**

Chapter 10
Dynamic Voltage and Frequency Control (DVFC)

The BSP includes the Dynamic Voltage and Frequency Control (DVFC) driver, which provides combined support for DVFS (Dynamic Voltage Frequency Scaling) and DPTC (Dynamic Process Temperature Compensation). The DVFC driver plays an important role in the reduction of active power consumption by dynamically adjusting the voltage and frequency settings of the system. The DVFC driver responds to the DVFS and DPTC hardware logic monitors the CPU loading and process/temperature performance of the silicon.

10.1 DVFC Driver Summary

Table 10-1 the source code location, library dependencies and other BSP information.

Table 10-1. DVFC Driver Summary

Driver Attribute	Definition
Target Platform (TGTPLAT)	3DS
Target SOC (TGTSOC)	MX31
MXARM11 CSP Driver Path	N/A
CSP Driver Path	...\CSP\ARM\FREESCALE\<*TGTSOC*>\DRIVERS\DVFC
CSP Static Library	<*TGTSOC*>_dvfc.lib
Platform Driver Path	...\PLATFORM\<*tgtplat*>\SRC\DRIVERS\DVFC\MC13783
Import Library	pmicSdk_mc13783.lib
Driver DLL	dvfc_mc13783.dll
Catalog Item	Third Party > BSPs > Freescale <*tgtplat*>>Device Drivers> MC13783 DVFC
SYSGEN Dependency	N/A
BSP Environment Variables	BSP_PMIC_MC13783 = 1 BSP_DVFC_ MC13783 = 1

10.2 Supported Functionality

The DVFC driver enables the 3-Stack board to provide the following software and hardware support:

- Executes as a device driver and provides synchronized support of the DPTC and DVFS power management features
- MX31 supports DPTC and DVFS
- Supports integer frequency scaling
- Provides integrated voltage control supplied from MC13783.
- Exposes stream interface for initialization and power management
- Supports D0 and D4 driver power states
- DPTC/DVFS uses FPM as reference clock in SDK

10.3 Hardware Operation

The DVFC driver is dependent upon the MC13783 interface for dynamic voltage control. The MC13783 board must be present on the i.MX31 3-Stack board. The i.MX31 CPU board must be configured to source power from the MC13783.

For further information, see the *MCIMX31 and MCIMX31L Applications Processors Reference Manual* and *MC13783 Power Management and Audio Circuit User's Guide.*

10.3.1 Conflicts with other SoC peripherals

10.3.1.1 i.MX31 Peripheral Conflicts

The signals used for the dynamic voltage control interface between the i.MX31 and MC13783 cannot be used for other purposes. In particular, the i.MX31 GPIO1_5 pin is used by the DVFS hardware to determine when the new voltage setting has been reached. This pin should not be configured for GPIO purposes.

10.4 Software Operation

10.4.1 Loading and Initialization

The DVFC driver is automatically loaded by device manager as a stream driver. As part of the loading procedure of stream drivers, the device manager invokes the corresponding stream initialization function exported by the DVFC driver. The initialization sequence includes a call to platform-specific code (`BSPDvfcInit`) to allow the OEM to configure and tune the DVFC driver operation.

10.4.1.1 Tuning Hardware Load Tracking

The DVFC driver utilizes the hardware load tracking available within the i.MX31 DVFS logic. The load tracking hardware monitors the CPU activity and notifies the system to adjust the DVFS setting to meet the required CPU performance. By adjusting parameters used to configure the load tracking hardware,

users can control the CPU loading characteristics that trigger DVFS transitions. These parameters are configured using DVFC driver registry keys. Refer to the Registry Settings section in this chapter for more information.

10.4.2 External Interface

The DVFC driver(for the i.MX31) allows other drivers/applications to control some aspects of the DVFS operation. Due to the tight coupling with the system clock configuration, this interface is exposed within CSPDDK clocking support. Refer to the CSPDDK documentation for the following functions:

1. DDKClockEnablePanicMode
2. DDKClockDisablePanicMode
3. DDKClockBusScale

10.4.3 Registry Settings

Table 10-2 describes the registry keys supported by the DVFC driver.

Table 10-2. Registry Keys Supported by the DVFC Driver

Key Name	Key Type	Description
`ThreadPriority`	`REG_DWORD`	This key specifies the thread priority assigned to the DVFC interrupt service thread. **Note:** If this key is missing, a default thread priority of 250 will be used.
`TrackingLoadDown`	`REG_DWORD`	This key specifies the CPU loading threshold (expressed as a percentage) used by the load tracking hardware to determine when a lower frequency/voltage setting is requested. If the average CPU load is less than `TrackingLoadDown` over a continuous sample window specified by the `TrackingWindowDown` registry key, the DVFS logic will request a lower frequency/voltage setting. **Note:** If this key is missing, a default CPU load of 50% will be used as the threshold.
`TrackingLoadUp`	`REG_DWORD`	This key specifies the CPU loading threshold (expressed as a percentage) used by the load tracking hardware to determine when a higher frequency/voltage setting is requested. If the average CPU load is greater than `TrackingLoadUp` over a continuous sample window specified by `TrackingWindowUp` registry key, the DVFS logic will request a higher frequency/voltage setting. **Note:** If this key is missing, a default CPU load of 90% will be used as the threshold.
`TrackingWindowDown`	`REG_DWORD`	This key specifies the continuous sample window (expressed in msec) used by the load tracking hardware to determine when a lower frequency/voltage setting is requested. If the average CPU load is less than the `TrackingLoadDown` registry key over a continuous sample window specified by `TrackingWindowDown`, the DVFS logic will request a lower frequency/voltage setting. **Note:** If this key is missing, a default tracking window of 5 msec will be used.
`TrackingWindowUp`	`REG_DWORD`	This key specifies the continuous sample window (expressed in msec) used by the load tracking hardware to determine when a higher frequency/voltage setting is requested. If the average CPU load is greater than the `TrackingLoadUp` registry key over a continuous sample window specified by `TrackingWindowUp`, the DVFS logic will request a higher frequency/voltage setting. **Note:** If this key is missing, a default tracking window of 5 msec will be used.

Table 10-2. Registry Keys Supported by the DVFC Driver(Continued)

Key Name	Key Type	Description
TrackingEMAC	REG_DWORD	The load tracking hardware computes an exponential moving average (EMA) of the tracked CPU load. This key specifies the number of samples that are included in the EMA calculation. A lower number of EMA samples will decrease the hysteresis in the EMA load tracking calculation. Conversely, a higher number of EMA samples will increase the hysteresis in the EMA load tracking calculation. Refer to the i.MX31 hardware manual for more information and available settings. This parameter will be used to update the EMAC bits of the LTR2 register in the i.MX31 CCM. **Note:** If this key is missing, a default setting of 31 samples (EMA code = 0x10) will be used.
DvfsForceOff	REG_DWORD	No effect **Note:** DVFC can be enable/disable by EBOOT
DptcForceOff	REG_DWORD	No effect **Note:**
DvsSpeed	REG_DWORD	This key specifies the transition rate for the output voltage supplied from the MC13783 PMIC. Refer to the MC13783 hardware manual for more information and available settings. This parameter will be used to update the DVSSPEED selection bits of the MC13783. **Note:** If this key is missing, a default setting of 16 us / 25 mV step (DVSSPEED = 0x3) will be used.
LowSpeedVolt	REG_DWORD	This key specifies the fixed voltage supplied to the CPU from the MC13783 PMIC for the low-speed CPU DVFS setpoint. The voltage is expressed as a coded value that reflects the output voltage selections available from the MC13783 buck switchers. Refer to the MC13783 hardware manual for more information and available settings. This parameter will be used to update the SW1ADVS selection bits of the MC13783. **Note:** If this key is missing, a default of 1.350V (SW1ADVS = 0x12) will be used.
LowBusVolt	REG_DWORD	This key specifies the fixed voltage supplied to the CPU from the MC13783 PMIC for the low-speed CPU and low-speed bus (activated by DDKClockScaleBus) DVFS setpoint. The voltage is expressed as a coded value that reflects the output voltage selections available from the MC13783 buck switchers. Refer to the MC13783 hardware manual for more information and available settings. This parameter will be used to update the SW1ADVS selection bits of the MC13783. **Note:** If this key is missing, a default of 1.275V (SW1ADVS = 0x0F) will be used.
Verbose	REG_DWORD	Setting this key to a non-zero value will enable retail messages within the DVFC driver. The retail messages will provide information regarding DVFS and DPTC configuration and setpoint transitions. **Note:** If this key is missing, retail messages will be disabled by default.

10.4.4 Power Management

The DVFC is an integral part of the power management supported by the BSP. However, since the DVFC runs as a driver on the system, it too must support the Power Manager device driver interface. This allows the DVFC driver to be notified of when the system is suspending/resuming and configure the DVFS and DPTC hardware blocks appropriately.

10.4.4.1 PowerUp

This stream interface function is not implemented for the DVFC driver.

10.4.4.2 PowerDown

This stream interface function is not implemented for the DVFC driver.

10.4.4.3 IOCTL_POWER_CAPABILITIES

The DVFC driver advertises that D0 and D4 device power states are supported.

10.4.4.4 IOCTL_POWER_SET

The DVFC driver supports requests to enter D0-D4 device power state. Power states D1-D3 are treated as D4.

10.4.4.5 IOCTL_POWER_GET

The DVFC driver reports the current device power state (D0 or D4).

10.5 Unit Test

No unit test cases provided.

Chapter 11
Ethernet Boot Loader

The boot loader utility is an integral part of the OEM device development process. In some cases, it is also included in the final OEM product. The general purpose of the boot loader is to place the run-time image into memory, and then jump to the OS startup routine.

The boot loader can obtain the run-time image in several ways, including loading it over a cabled connection, such as Ethernet, a universal serial bus (USB), or serial connection. The boot loader also loads the OS from a local storage device, such as Compact Flash, or a hard disk. The boot loader might store the run-time image in RAM or in nonvolatile storage, such as flash memory, electrically erasable programmable read only memory (EEPROM), or some other storage device for later use. This chapter focuses on the Ethernet bootloader.

Using the Ethernet boot loader during the board support package (BSP) development process saves time. You can quickly download a new run-time image to a target device using a boot loader rather than manually transferring the run-time image to a target device using a flash memory programmer or IEEE 1149.1-compliant test access port and boundary-scanning technology.

The Ethernet boot loaders provided by Microsoft are typically made up of BLCOMMON, EBOOT, and network drivers. The code implemented by OEMs also makes up part of the boot loaders. The BLCOMMON, EBOOT, and network drivers are code libraries that are designed to be reusable and portable. They implement common boot loader functionality and make development of new boot loaders easier.

The Ethernet boot loader can be downloaded and programmed into NAND flash onto the Freescale 3-Stack board, where it communicates with the K9F2G08R0A model using a NAND read and write interface implementation. Furthermore, the Ethernet boot loader can only be downloaded through an Ethernet path, so the driver of the SMSC LAN9217 Ethernet chip must be implemented.

11.1 Ethernet Boot Loader Summary

Table 11-1 identifies the source code location, library dependencies, and other BSP information.

Table 11-1. Ethernet Boot Loader Summary

Driver Attribute	Definition
Target Platform (TGTPLAT)	3DS
Target SOC (TGTSOC)	MX31
MXARM11 CSP Driver Path	N/A
CSP Driver Path	N/A
CSP Static Library	N/A

Table 11-1. Ethernet Boot Loader Summary

Driver Attribute	Definition
Platform Driver Path	..\PLATFORM\<*tgtplat*>\SRC\Bootloader\EBOOT
Import Library	oal_blcommon.lib, oal_blnk.lib, oal_kitl.lib, oal_other.lib, oal_log.lib, oal_memory_mxarm11.lib, oal_cache_mxarm11.lib, lan9217_lib.lib, nandfmd_lib.lib, oal_flash.lib, bspcmn.lib, eboot.lib, fulllibc.lib, fsl_usbfn_rndiskitl.lib, rne_mdd.lib
Target Program	EBOOT.nb0, EBOOT.bin
Catalog Item	Third Party BSPs > Freescale <*tgtplat*> > Ethernet Bootloader(eboot)
SYSGEN Dependency	N/A
BSP Environment Variables	N/A

11.2 Supported Functionality

The Ethernet boot loader enables the 3-Stack board to provide the following software and hardware support:

- Supports image downloading, programming ,and verification in K9F2G08R0A NAND flash or image downloading and execution in RAM
- Empowers the Debug port to provide error message, load status, and the progress of the output and user input
- Empowers the SMSC LAN9217 Ethernet chip to establish TFTP transport between device and workstation
- Supports self-update to program eboot.bin and xldr.bin to NAND flash
- Will boot up image in both image downloading and image flashing cases
- Supports initialization of necessary hardware components including clock module
- Performs checksum checking on the downloaded data and should avoid changing the flash memory contents until the checksum is confirmed to be correct
- Shares the hardware platform initialization code with OAL to be able to initialize the hardware platform
- Maintains a menu to reflect user options

11.3 Hardware Operation

For detailed operation and programming information, see the K9F2G08R0A NAND flash datasheet, SMSC LAN9217 Ethernet chip datasheet, and the chapter on IOMAX in the *MCIMX31 and MCIMX31L Applications Processors Reference Manual*.

11.3.1 Conflicts with Other SoC Peripherals

11.3.1.1 MX31 Peripheral Conflicts

The Ethernet bootloader does not have conflicts with any other module.

11.4 Software Operation

The Ethernet boot loader follows the Microsoft recommended architecture and process for Ethernet boot loader development.

For details of this architecture and its operation, see the Platform Builder Help:

Bringing Up a Hardware Platform > Developing a Board Support Package > Developing a Boot Loader

The Ethernet boot loader commits all of the necessary interface implementation of BLCOMMON.

11.4.1 Communicating with the BLCOMMON

Platform Builder includes the BLCOMMON library, which is a common infrastructure that can be reused by all boot loaders. For a description of the functions in the new BLCOMMON infrastructure, see the source and header files in:

```
%_WINCEROOT%\Public\Common\Oak\Drivers\Ethdbg\Blcommon\Blcommon.c, and
%_WINCEROOT%\Public\Common\Oak\Inc\Blcommon.h
```

Microsoft suggests committing certain processes to implement the flow of control with BLCOMMON infrastructure.

1. During the hardware platform boot process, the startup code in the OEM adaptation layer (OAL) calls the hardware platform-independent BootloaderMain function.
2. BootloaderMain initializes the hardware platform, chooses the download transport, and then initializes the download transport by calling a set of control flow functions.
3. The boot loader starts downloading the image to the target device using a set of download and flash functions. During the download process, the boot loader might implement functions to show download progress.
4. When the download is complete, the boot loader boots the OS by calling the OEMLaunch function, which jumps to the OS entry point.

11.4.2 Implement Elements

Certain elements must be implemented to empower Ethernet boot loader to bootup device and download images.

1. OAL startup code - The OEM-specific startup routine that initializes the kernel. It jumps to the main entry point implemented in:
   `%_WINCEROOT%\Platform\<Hardware Platform Name>\Src\BootLoader\EBOOT\Main.c`
2. Kernel startup code - A minimal version of the normal kernel startup code for the boot loader used primarily to initialize the cache.
3. OEM hardware platform initialization code - the OEM-specific code that can include functionality, such as displaying a splash screen or checking switches to allow alternate entry to diagnostic code or to the firmware monitor. A splash screen is an initial screen that is displayed by interactive software, which usually contains a logo, version information, author credits, or a copyright notice.
4. Image download code - Code that downloads a run-time image using the download transport.
5. Ethernet I/O code - Code that downloads a run-time image using an Ethernet connection.
6. Debug serial I/O code - Routines that read and write data to the debug serial port. To simplify development of the debug serial port support I/O code, use a standard universal asynchronous receiver transmitter (UART) on the target device. The standard UART should be 16550-compatible. The boot loader uses the same debug serial port routines implemented in the OAL.
7. Flash write code - Routines that write data to flash memory. This code is necessary for your boot loader to support downloading and flashing the image into nonvolatile storage for later use.

11.4.3 Register Settings

There are no related register settings for the Ethernet boot loader.

11.4.4 Power Management

Power management is not yet implemented in the Ethernet boot loader.

11.5 Unit Test

Microsoft does not supply a test case in CETK; the Ethernet boot loader needs to pass only the functional test.

11.5.1 Building an Ethernet Boot Loader Image

Because the Ethernet boot loader is a necessary component in the workspace, it has been added to OS design by default. The Ethernet boot loader is built to binary files, including eboot.nb0 and eboot.bin:

- Platform Builder uses `eboot.bin` file to download and flash
- Realview ICE and other binary flashing tools, such as the ATK tool use `eboot.nb0`

11.5.2 Testing Ethernet Boot Loader Image

All functions implemented by Ethernet boot loader are tested.

To update XLDR:

1. Click **Platform**>**Settings**.
2. In the General section, select XLDR.bin as the file name for the run-time image.
3. In Platform Builder, download into the 3-Stack board using EBOOT.

NOTE

Do not use the **File** > **Open** ... method to open XLDR.bin. Doing so produces an error message stating that the file cannot be opened as a workspace because “it is not a valid Windows CE BIN file”.

Test Item	Test Method
Eboot installation by ICE compatible tools	Use ATK tool to flash xldr.nb0 and eboot.nb0. Refer to user guide for detailed manual of ATK tool.
Eboot self installtion	First make sure boot loader works well on device and then download xldr.bin and eboot.bin by Platform Builder to burn them into the board.
Eboot downloading kernel images	Build a RAM version kernel image and open nk.bin by Platform Builder. Then use Eboot to download nk.bin into device to run the kernel. Eboot can establish Ethernet connections between device and workstation.
Eboot flashing kernel images	Build a NAND version kernel image and open nk.bin by Platform Builder. Then use Eboot to download and flash nk.bin into device to run the kernel. Eboot and perform NAND writing, reading and verification operations.
Eboot support debug console	Error and debug message could show in console windows enpowered by Eboot. End user can input options through debug console to change Eboot configuration.
Eboot bootup hardware	When the board bootup, Eboot could lead the booting process of hardware components.

11.6 Ethernet Boot Loader Reference

For the Ethernet boot loader development documentation, see the Platform Builder Help. For Ethernet boot loader menu descriptions, see the *i.MX31 IC Reference Guide*.

11.6.1 Creating and Implementing Boot Loader Stubs

The OEM will use boot loader stubs to commit basic functions for BLCOMMON. You will need to create stubs for certain generic hardware platform routines and flash memory operations, because BLCOMMON expects to call these functions, but they have not yet been implemented. You can revisit these functions and then fully implement them later.

The stub versions added to Main.c are for the following generic hardware platform routines: OEMDebugInit, OEMPlatformInit, OEMPreDownload, OEMLaunch, OEMReadData, OEMShowProgress, OEMWriteDebugByte.

For more information, see the Platform Builder Help:

Bringing Up a Hardware Platform > Developing a Board Support Package > Developing a Boot Loader > How to Develop a Boot Loader

11.6.2 Ethernet Chip Driver Implementation

The BLCOMMON library and the EBOOT.lib support library, which assists with DHCP, UDP, and TFTP services, calls Ethernet controller-related functions. These functions are typically wrappers for the access primitive routines exposed in the Ethernet debug libraries.

Ethernet boot loader must empower the SMSC LAN9217 Ethernet chip to establish TFTP transport between the device and workstation. The functions OEMReadData, OEMEthGetFrame, OEMEthSendFrame and OEMEthGetSecs must be implemented.

For more information, see the Platform Builder Help:

Bringing Up a Hardware Platform > Developing a Board Support Package > Developing a Boot Loader > How to Develop a Boot Loader

11.6.3 Ethernet Boot Loader Menu

When Ethernet boot loader begins to run, a boot menu is displayed, showing the options listed in Table 11-2.

Table 11-2. Ethernet Boot Loader Menu Options

Option	Description
0) IP address: 0.0.0.0	IPv4 address of current Ethernet network. 0.0.0.0 if not using DHCP.
1) Subnet Mask: 0.0.0.0	IPv4 mask code of current Ethernet adapter. 0.0.0.0 if not using DHCP.
2) Boot delay: 3 seconds	Waiting time for use to press [ENTER] to enter boot menu.
3) DHCP: Enabled	Specifies whether to enable the dynamic host configuration protocol to get a dynamic IP address from DHCP server. If not, then the system uses the static IP address, and you must configure the IP address and subnet mask.
4) Reset to factory default configuration	Reconfigures the menu options to default settings.
5) Autoboot: NK from NAND	Selects the device boot up location: NAND flash, RAM, or disabled. If disabled is selected, booting is not performed; BOOTME packets are sent.
6) MAC address: 0-5-66-12-34-56	MAC access control address of Ethernet adapter, also known as the hardware address.
7) Format OS NAND region	Formats the NAND flash area, which stores the operating system.
8) Format ALL NAND regions	Formats the entire NAND flash area.

Table 11-2. Ethernet Boot Loader Menu Options(Continued)

I) KITL interrupt mode: Enable	Specifies whether to enable KITL working on interrupt mode; otherwise KITL work on pollup mode.
P) KITL passive mode: Enable	Specifies whether to enable KITL working in the passive mode which means device does not connect to the workstation actively. Otherwise, the workstation must connect to the device actively.
C) L2 cache enable: Disable	Specifies whether to enable level two cache of CPU.
W) L2 cache mode: WriteThrough	Specifies whether to set cache working on write-through or write-back mode.Write through means upon write and update data in cache, update data in memory at the same time. Write-back means upon write and update data in cache, flush the data into memory when the cache line is to be replaced or flush cache instructions are executed explicitly.
B) AHB Clock freq: 66.5M	Advanced high-performance bus frequency. AHB is used to connect high performance modules, such as CPU, DMA and DSP, and so on.
A) ARM Clock freq: 532M	ARM core clock frequency.
T) DPTC mode: Disable	Specifies whether to enable dynamic process and temperature compensation mode. DPTC module is a power management module to detect the minimum operation voltage for the IC, taking into account the extreme of processing activity and ambient temperature for a given frequency.
F) DVFS mode: Disable	Specifies whether to enable dynamic voltage and frequency scaling mode. DVFS is a power management module to reduce active power consumption by scaling voltage and frequency acccording to required MIPs.
V) SW2 Voltage: 1.8V	Configures the voltage of soft switch 2.
X) PowerPolicy: Disable	Specifies whether to apply power management policy.
R) USB KITL: Disable	Specifies whether to enable USB KITL.
S) Save configuration	Saves all the settings.
D) Download image from Ethernet now	Downloads image bin file from workstation through Ethernet transport.
U) Download image from USB now	Downloads the image bin file from workstation through USB transport.
L) Launch existing flash resident image now	Launches the image from NAND flash.

For more information, see the Platform Builder Help:

Bringing Up a Hardware Platform > Developing a Board Support Package > Developing a Boot Loader > How to Develop a Boot Loader

Chapter 12
FM Radio Driver

This FM radio driver controls the Si4702 chip. The driver is compatible with the Stream Interface driver framework. This chapter describes the Si4702 chip, and explains how to develop the FM radio application that interfaces directly to the Si4702 chip.

12.1 Radio Driver Summary

Table 12-1 identifies the source code location, library dependencies and other BSP information.

Table 12-1. FM Radio Driver Summary

Driver Attribute	Definition
Target Platform (TGTPLAT)	3DS
Target SOC (TGTSOC)	MX31
CSP Driver Path	N/A
CSP Static Library	N/A
Platform Driver Path	..\PLATFORM\<*tgtplat*>\SRC\DRIVERS\RADIO
Import Library	N/A
Driver DLL	fm_radio.dll
Catalog Item	Third Party > BSPs > Freescale <*tgtplat*> > Device Drivers > Si4702 FM Radio Driver
SYSGEN Dependency	N/A
BSP Environment Variables	BSP_RADIO=1

12.2 Supported Functionality

The Radio driver enables the 3-Stack board to provide the following software and hardware support:

- Conforms to the Device Manager streams interface
- Supports the Si4702 chip
- Supports the main functions of FM radio: Power on/off, Set Frequency, Set Volume, Muted, and Auto Scan

12.3 Hardware Operation

The driver uses I2C to interact with Si4702 hardware. For information, see the *Silicon Laboratories Si4702 Guide.*

12.4 Software Operation

The interface needed to control the Radio driver is provided by the IOCTLs.

12.4.1 Radio Driver Registry Settings

The following registry keys are required to properly load the Radio driver.

```
; These registry entries load the FM Radio driver. The IClass value be GUID for generic ;
power-managed devices.

          [HKEY_LOCAL_MACHINE\Drivers\BuiltIn\RADIO]
             "Prefix"="RDO"
             "Dll"="fm_radio.dll"
             "Index"=dword:1
             "Order"=dword:30
             "IClass"="{A32942B7-920C-486b-B0E6-92A702A99B35}"
```

12.4.2 Power Management

The primary method for limiting power consumption in the Radio module is to power down the chip when the FM Radio driver is not in use. The application calls IOCTL_SET_POWER with the parameter POWER_OFF to power down the chip.

12.4.2.1 PowerUp

This function is not implemented for the Radio driver.

12.4.2.2 PowerDown

This function is not implemented for the Radio driver.

12.4.2.3 IOCTL_POWER_CAPABILITIES

This IOCTL describes the power management capabilities using Power Manager. The radio module supports only two power states: D0 and D4.

12.4.2.4 IOCTL_POWER_SET

This IOCTL requests a change from one device power state to another. D0 and D4 are the only two supported **CEDEVICE_POWER_STATE** in Radio driver. Any request that is not D0 is changed to a D4 request, which puts the system in a suspend state; for a value of D0 the system is resumed.

12.4.2.5 IOCTL_POWER_GET

This IOCTL returns the current device power state. By design, the Power Manager knows the device power state of all power-manageable devices. It does not issue an **IOCTL_POWER_GET** call to the device unless an application calls **GetDevicePower** with the POWER_FORCE flag set.

12.5 Unit Test

The Radio CETK test cases verify the functionality of the radio driver with the Si4702 chip. The FM Radio Application can also be used to verify the radio driver. For information, see the FM Radio Application section of the user's guide.

12.5.1 Unit Test Hardware

The 3-Stack board is required.

12.5.2 Building the Radio Tests

In order to build the radio tests, you need to build an OS image for the desired configuration.

To build an OS image:

1. In Platform Builder, at the **Build OS** menu, select **Open Release Directory**. This opens a DOS prompt.
2. Change to the Radio Tests directory. (\WINCE500\SUPPORT\TESTS\RADIO)
3. Enter **set WINCEREL=1** on the command prompt and hit return. This copies the built DLL to the flat release directory.
4. Enter the build command (*build -c*) at the prompt and press Enter.

After the build completes, the `radio_test.dll` file will be located in the $(_FLATRELEASEDIR) directory.

12.5.3 Running the Radio Tests

The command line for running the radio tests is:

```
tux –o –d radio_test
```

To redirect the test results to a file, add the option **–f**.

Radio tests do not contain any test-specific command line options.

12.6 Radio IOCTL Reference

This section describes the RADIO I/O control codes (IOCTLs). These IOCTLs are used in calls to DeviceIoControl to issue commands to the radio device modules. Only parameters for this IOCTL are described. Most of the IOCTLs are explained in other, relevant sections.

12.6.1 Radio Driver IOCTLS

12.6.1.1 RADIO_IOCTL_GET_CAPS

This **DeviceIoControl** request determines the capability of the hardware.

Parameters *hOpenContext*

[in] Handle to the device that is to perform the operation. To obtain a device handle, call the CreateFile function.

pBufIn

NULL

pBufOut

pointer to RADIO_CAPS type data return to caller.

12.6.1.2 RADIO_IOCTL_GET_TUNER

This **DeviceIoControl** request used to retrieve the tuner information of the hardware.

Parameters *hOpenContext*

[in] Handle to the device that is to perform the operation. To obtain a device handle, call the CreateFile function.

pBufIn

NULL

pBufOut

pointer to RADIO_TUNER type data return to caller.

12.6.1.3 RADIO_IOCTL_SET_TUNER

This **DeviceIoControl** request sets tuner information of the hardware.

Parameters *hOpenContext*

[in] Handle to the device that is to perform the operation. To obtain a device handle, call the CreateFile function.

pBufIn

pointer to RADIO_TUNER type data filled by caller.

pBufOut

NULL

12.6.1.4 RADIO_IOCTL_GET_AUDIO

This **DeviceIoControl** request used to retrieve the audio information of the hardware.

Parameters *hOpenContext*

[in] Handle to the device that is to perform the operation. To obtain a device handle, call the CreateFile function.

pBufIn

NULL

pBufOut

pointer to RADIO_AUDIO type data return to caller.

12.6.1.5 RADIO_IOCTL_SET_AUDIO

This **DeviceIoControl** request sets audio information of the hardware, such as volume and whether muted.

Parameters *hOpenContext*

[in] Handle to the device that is to perform the operation. To obtain a device handle, call the CreateFile function.

pBufIn

pointer to RADIO_AUDIO type data filled by caller.

pBufOut

NULL

12.6.1.6 RADIO_IOCTL_GET_FREQ

This **DeviceIoControl** request used to retrieve the current frequency of the hardware.

Parameters *hOpenContext*

[in] Handle to the device that is to perform the operation. To obtain a device handle, call the CreateFile function.

pBufIn

NULL

pBufOut

pointer to the current frequency return to caller.

12.6.1.7 RADIO_IOCTL_SET_FREQ

This **DeviceIoControl** request tunes to the frequency.

Parameters *hOpenContext*

[in] Handle to the device that is to perform the operation. To obtain a device handle, call the CreateFile function.

pBufIn

pointer to the frequency filled by caller.

pBufOut

NULL

12.6.1.8 RADIO_IOCTL_GET_POWER

This **DeviceIoControl** request is used to retrieve the power state of the hardware.

Parameters *hOpenContext*

[in] Handle to the device that is to perform the operation. To obtain a device handle, call the CreateFile function.

pBufIn

NULL

pBufOut

pointer to RADIO_POWER type data return to caller.

12.6.1.9 RADIO_IOCTL_SET_POWER

This **DeviceIoControl** request sets the power state of the hardware.

Parameters *hOpenContext*

[in] Handle to the device that is to perform the operation. To obtain a device handle, call the CreateFile function.

pBufIn

pointer to RADIO_POWER type data filled by caller.

pBufOut

NULL

12.6.1.10 RADIO_IOCTL_AUTO_TUNE

This **DeviceIoControl** request auto scans all available channels.

Parameters *hOpenContext*

[in] Handle to the device that is to perform the operation. To obtain a device handle, call the CreateFile function.

pBufIn

pointer to RADIO_AUTOTUNE type data filled by caller.

pBufOut

NULL

12.6.1.11 RADIO_IOCTL_GET_LAST_ERROR

This **DeviceIoControl** returns the last return code.

Parameters *hOpenContext*

[in] Handle to the device that is to perform the operation. To obtain a device handle, call the CreateFile function.

pBufIn

NULL .

pBufOut

pointer to the current return code returned to the caller

12.6.2 Radio Driver Structures

12.6.2.1 Radio Tuner Structure

```
typedef struct
{
    TCHAR   name[32]; // i.e "FM"
    INT32   band_id;
    UINT32  range_low;   //KHZ
    UINT32  range_high; //KHZ
    UINT32  signal;
    UINT32  normal_signal; //acceptable signal
    UINT32  mode; //MONO, STEREO
    UINT32  reserved;
} RADIO_TUNER;
```

12.6.2.2 Radio Caps Structure

```
typedef struct
{
    TCHAR  driver[32];// that is, "Radio"
    TCHAR  chip[32]; // that is, "Silicon Laboratories Si4702"
    UINT32 version;   //
    UINT32 caps; // Device capabilities
    UINT32 bands;
    UINT32 reserved;
} RADIO_CAPS;
```

12.6.2.3 Radio Audio Structure

```
typedef struct
{
    UINT32  volume;
    UINT32  muted;
} RADIO_AUDIO;
```

12.6.2.4 Radio Power State Structure

```
typedef enum
{
    RADIO_POWER_OFF = 0,
    RADIO_POWER_ON
} RADIO_POWER;
```

12.6.2.5 Radio Auto Tune Structure

```
typedef enum
{
    RADIO_AUTOTUNE_FROM_BEGIN = 0,
    RADIO_AUTOTUNE_FROM_CUR,
    RADIO_AUTOTUNE_FROM_END
} RADIO_AUTOTUNE_POS;

typedef enum
{
    RADIO_AUTOTUNE_SEEKUP = 0,
    RADIO_AUTOTUNE_SEEKDOWN
} RADIO_AUTOTUNE_DIR;

typedef struct
{
    RADIO_AUTOTUNE_POS pos;
    RADIO_AUTOTUNE_DIR  dir;
} RADIO_AUTOTUNE;
```

Chapter 13
General Purpose Timer Driver

The general purpose timer (GPT) is a multipurpose module used to measure intervals or generate periodic output. The timer counter value can be captured in a register using an event on an external pin. The GPT can also generate an event on a chip boundary signal and an interrupt when the timer reaches a programmed value. The i.MX31 board supports only one general purpose timer.

13.1 GPT Driver Summary

Table 13-1 identifies the source code location, library dependencies, and other BSP information for the i.MX31 platform.

Table 13-1. GPT Driver Summary

Driver Attribute	Definition
Target Platform (TGTPLAT)	3DS
Target SOC (TGTSOC)	MX31
MXARM11 CSP Driver Path	..\CSP\ARM\FREESCALE\MXARM11\DRIVERS\GPT
CSP Driver Path	N/A
CSP Static Library	mxarm11_gpt.lib
Platform Driver Path	..\PLATFORM\<tgtplat>\SRC\DRIVERS\GPT
Import Library	N/A
Driver DLL	gpt.dll
Catalog Item	Third Party > BSPs > Freescale <tgtplat>> Device Drivers >GPT
SYSGEN Dependency	N/A
BSP Environment Variables	BSP_GPT=1

13.2 Supported Functionality

The GPT driver enables the 3-Stack board to provide the following software and hardware support:

- Configured as a loadable .dll module so that other drivers can use the interface of the GPT driver
- Supports clock source selection, including external clock source
- Supports both reset and free-run mode count operation
- Supports two power management modes: power on/off
- Supports the SDK interface
- Unit test cases must be created for testing the GPT driver

13.3 Hardware Operation

For detailed hardware operation and programming information, see the General Purpose Timer chapter in the *MCIMX31 and MCIMX31L Applications Processors Reference Manual*.

13.3.1 Conflicts with other SoC Peripherals

No conflicts.

13.4 Software Operation

NOTE

For the i.MX31, if Platform Builder profiling support is to be used, the GPT driver cannot be included in the workspace.

13.4.1 Communicating with the GPT

The GPT driver controls the GPT, which provides high resolution (microsecond) timing functionality to other platform modules. The GPT is a stream interface driver, and is thus accessed through the file system APIs. To communicate using the GPT, first obtain a handle to the device using the **GptOpenHandle** function. Issue subsequent commands to the device using various APIs supported by this driver. To use this API, you must include the `mxarm11_gptsdk.lib` library.

13.4.2 Creating a Handle to the GPT

To communicate with the GPT, first create a handle to the device using the **GptOpenHandle** API. The default GPT port is 1.

To open a Handle to the GPT

```
// Global data
// Handle to the GPT device
HANDLE g_hGpt = NULL;

// opening the default GPT port.
g_hGpt = GptOpenHandle();
```

For more information, see the **GptOpenHandle** section of the GPT API reference.

13.4.3 Configuring the GPT

To configure the GPT for communications, you must select the timer source by calling **GptSetTimerSrc** API; start the timer and enable the timer event trigger by calling **GptStart** API; and show the current timer source by calling **GptShowTimerSrc** API.

Before this action can be taken, a handle to the GPT port must already be opened.

Call the **GptSetTimerSrc** API to select timer source.

```
// selecting the GPT source
GptSetTimerSrc(g_hGpt, pGptTimerSrcPkt) ;
```

Call the **GptStart** API to enable and start the timer.

```
// configuring and starting the GPT
GptStart(g_hGpt) ;
```

Call the **GptShowTimerSrc** API to show current timer source.

```
// showing current GPT timer source
GptShowTimerSrc(g_hGpt) ;
```

For more information, see the **GptStart** section of the GPT API reference.

13.4.4 Write Operations

The Write operations for the GPT involve setting the time through the **GptSetTimer** API. Before this action can be taken, a handle to the GPT must already be opened.

The timer mode can be either timerModeFreeRunning or timerModePeriodic. The period has the unit of microsecond.

```
// Name to create the named event for Timer
#define GPT_EVENT_NAME  L"GptTest1"
// GPT Timer packet
GPT_TIMER_SET_PKT gptTimerDelayPkt;

// create an event for the timer interrupt
hGptIntr = GptCreateTimerEvent(hGpt, GPT_EVENT_NAME);

gptTimerDelayPkt.timerMode = timerModePeriodic;
gptTimerDelayPkt.period = 10000000;

// Setting the GPT timer
GptSetTimer(g_hGpt, &gptTimerDelayPkt);
```

For more information, see the **GptSetTimer** section of the GPT API reference.

13.4.5 Closing the Handle to the GPT

To close the GPT handle, call the **GptCloseHandle** API. But before performing the close operation, stop the timer using **GptStop** API. It is always advised to call **GptReleaseTimerEvent** to release any pending timer events before closing the handle.

Before these actions can be taken, a handle to the GPT must already be opened.

To close the GPT Handle,

```
// Name to create the named event for Timer
#define GPT_EVENT_NAME  L"GptTest1"

// releasing the Timer Event.
GptReleaseTimerEvent(g_hGpt, eventString);
GptStop(g_hGpt)
GptCloseHandle(g_hGpt);
```

For more information, see the **GptReleaseTimerEvent**, **GptStop** and **GptCloseHandle** sections in the GPT API reference.

13.4.6 Power Management

The primary method for limiting power consumption in the GPT module is to gate off all clocks to the module when the GPT is not being used. The clock is enabled when an application calls **GPT_Open( )**. and remains enabled as long as the device is open. The GPT clock is turned off when the application closes the device using **GPT_Close()**.

13.4.6.1 PowerUp

This function restores the state of the GPT clocks to the state held prior to entering suspend. If the GPT was counting before suspend, GPT will continue to count from the place where it was stopped.

13.4.6.2 PowerDown

This function disables the clock to the GPT module. If the GPT was counting, then the count value freezes at the point when the clock is removed.

13.4.6.3 IOCTL_POWER_CAPABILITIES

N/A

13.4.6.4 IOCTL_POWER_SET

N/A

13.4.6.5 IOCTL_POWER_GET

N/A

13.4.7 GPT Registry Settings

13.4.7.1 GPT Registry Settings for i.MX31

```
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\GPT]
  "Prefix"="GPT"
  "Dll"="gpt.dll"
  "Index"=dword:1
```

13.5 Unit Test

The GPT tests verify that the GPT driver properly initializes and controls the GPT.

13.5.1 Unit Test Hardware

No additional hardware is needed to run the unit tests.

13.5.2 Unit Test Software

Table 13-2 lists the required software to run the unit tests.

Table 13-2. Software Requirements for Unit Tests for GPT Driver

Requirements	Description
Tux.exe	Tux test harness, which is needed for executing the test
Kato.dll	Kato logging engine, which is required for logging test data
Tooltalk.dll	Library required by Tux.exe and Kato.dll. Handles the transport between the target device and the development workstation
GPTTEST.dll	Test .dll file

13.5.3 Building the GPT Tests

In order to build the GPT tests, you must build an OS image for the desired configuration.

To build the OS image:

1. In Platform Builder, at the **Build OS** menu, select **Open Release Directory**. This opens a DOS prompt.
2. Change to the GPT Tests directory. the directory is \WINCE500\SUPPORT\TESTS\GPT.
3. Enter **set WINCEREL=1** on the command prompt and press Enter. This copies the built DLL to the flat release directory.
4. Enter the build command at the prompt and press Enter.

After the build completes, the `GPTTEST.dll` file will be located in the $(_FLATRELEASEDIR) directory.

13.5.4 Running the GPT Tests

The command line for running the GPT tests is ***tux –o –d gpttest***. The GPT tests do not contain any test-specific command line options.

To add the GPT test to CETK, perform the following steps:

1. Go to the **Tests** menu and select **User Defined**.
2. Follow the wizard and add the `GPTTEST.dll` located in the release folder as the test module.
3. Follow the wizard until it finishes.

Table 13-3 describes the test cases in the GPT tests.

Table 13-3. GPT Test Cases

Test Case	Description
30001: TST_StartBeforeCfg	Attempt to start the GPT timer without setting the timer period (expected failure)
30002: TST_OpenMultipleHandle	Attempt to open multiple GPT Handles (expected failure)
30003: TST_ComparewithSysTick	Check timer accuracy with system clock

Table 13-3. GPT Test Cases(Continued)

30004:TST_ChangeClockSrc	Run the timer with different timer source
30005:TST_PeriodicMode	Periodic mode test
30006: TST_FreerunMode	Free run mode test. It may take about 15 hours to finish if you run it after Test Case 4. Suggest to run it separately after system power on.

You could run a single test case using the following command:

tux -o -d gpttest -*x* 3000*x*

, where *x* is any number between 1 and 6.

13.6 GPT Driver API Reference

13.6.1 GPT Driver Functions

13.6.1.1 GptOpenHandle

This API creates a handle to the GPT stream driver:

```
HANDLE GptOpenHandle(
void
);
```

Parameters

This API accepts no parameters.

Return Values

An open handle to the specified file indicates success.
INVALID_HANDLE_VALUE indicates failure.

Remarks Use the GptCloseHandle function to close the handle returned by GptOpenHandle().

13.6.1.2 GptCreateTimerEvent

This API is used to create the GPT Timer event.

```
HANDLE GptCreateTimerEvent(
            HANDLE hGpt,
            LPTSTR eventName
);
```

Parameters

hGpt

[in] Handle to the GPT driver returned by **GptOpenHandle** API.

eventName

[in] Pointer to a null-terminated string that specifies the name of the object.

Return Values

A non-null handle to the specified event indicates success. NULL indicates failure.

Remarks

Use the **GptReleaseTimerEvent** function to close the event. The system closes the handle automatically when the process terminates. The event object is destroyed when its last handle has been closed.

13.6.1.3 GptShowTimerSrc

This API show the current timer source for the GPT

```
BOOL GptShowTimerSrc(
    HANDLE hGpt
);
```

Parameters

hGpt

[in] Handle to the GPT driver returned by **GptOpenHandle** API.

Return Values

TRUE on success and FALSE indicates a failure.

Remarks

Prints a message indicating which clock source is selected.

13.6.1.4 GptSetTimerSrc

This API show the current timer source for the GPT

```
BOOL GptSetTimerSrc(
    HANDLE hGpt,
    PGPT_TIMER_SRC_PKT pGptTimerSrcPkt

);
```

Parameters

hGpt

[in] Handle to the GPT driver returned by **GptOpenHandle** API.

pGptTimerSrcPkt

[in] An object of the **PGPT_TIMER_SRC_PKT** structure

Return Values

TRUE on success and FALSE indicates a failure.

Remarks

Select clock source between CLK_HIFRQ, CLK_32K, CLK_IPG, CLK_EXT

13.6.1.5 GptSetTimer

This API sets the timer period for the GPT.

```
BOOL GptSetTimer(
    HANDLE hGpt,
    PGPT_TIMER_SET_PKT pGptTimerDelayPkt
);
```

Parameters

hGpt

[in] Handle to the GPT driver returned by **GptOpenHandle** API.

pGptTimerDelayPkt

[in] An object of the **GPT_TIMER_SET_PKT** structure.

Return Values

TRUE on success and FALSE indicates a failure.

Remarks

Mode member in *pGptTimerDelayPkt* structure can be set to one of the timer modes *timerModeFreeRunning* or *timerModePeriodic*.

13.6.1.6 GptStart

This API enables the GPT interrupt and starts the GPT timer.

```
BOOL GptStart(
        HANDLE hGpt
);
```

Parameters

hGpt

[in] Handle to the GPT driver returned by **GptOpenHandle** API.

Return Values

TRUE on success and FALSE indicates a failure.

Remarks

None.

13.6.1.7 GptStop

This API disables the GPT interrupt and stops the GPT timer.

```
BOOL GptStop(
        HANDLE hGpt
);
```

Parameters

hGpt

[in] Handle to the GPT driver returned by **GptOpenHandle** API.

Return Values

TRUE on success and FALSE indicates a failure.

Remarks

None.

13.6.1.8 GptReleaseTimerEvent

This API closes the currently open GPT Timer Event.

```
BOOL GptReleaseTimerEvent(
        HANDLE hGpt,
        LPTSTR eventName
);
```

Parameters

hGpt

[in] Handle to the GPT driver returned by **GptOpenHandle** API.

eventName

[in] Pointer to a null-terminated string that specifies the name of the object

Return Values

Nonzero indicates success.
Zero indicates failure.
To get extended error information, call **GetLastError()**.

Remarks

None.

13.6.1.9 GptCloseHandle

This API closes a handle to the GPT driver.

```
BOOL GptCloseHandle(
        HANDLE hGpt
);
```

Parameters

hGpt

[in] Handle to the GPT driver returned by **GptOpenHandle** API.

Return Values

Nonzero indicates success.
Zero indicates failure.
To get extended error information, call **GetLastError().**

Remarks

None.

13.6.2 GPT Driver Structures

13.6.2.1 GPT_TIMER_SRC_PKT

```
typedef struct
{
timerMode_c timerMode;
UINT32 period;
} GPT_TIMER_SET_PKT, *PGPT_TIMER_SET_PKT;
```

Members

timerMode

*S*elects between two supported modes: reset or periodic mode (*timerModePeriodic*) and free-running mode (*timerModeFreeRunning*).

period

Counter period (in microsecond)

13.6.2.2 GPT_TIMER_SET_PKT

```
typedef struct
{
timerSrc_c timerSrc;
} GPT_TIMER_SRC_PKT, *PGPT_TIMER_SRC_PKT;
```

Members

timerSrc

*S*elects between 4 supported timer source, GPT_IPGCLK, GPT_HIGHCLK, GPT_EXTCLK and GPT_32KCLK

Chapter 14
Global Positioning System Driver

The Global Positioning System (GPS) enables a GPS receiver to determine its location, speed/direction, and time.

This 3-Stack platform supports the BroadCom Barracuda Single Chip A-GPS Solution. Barracuda™ is an A-GPS solution that integrates a high performance Assisted-GPS baseband signal processor with a low-noise GPS RF Tuner into a single CMOS die. Barracuda delivers exceptional sensitivity (-160 dBm), low power consumption and fast time-to-first-fix (TTFF) in a small, inexpensive package.

The external GPS module is supported using the serial port and GPIO resources. Because the chipset features a host-based architecture, you must load certain software components on the platform in order to enable full operation.

14.1 GPS Driver Summary

Most GPS software modules are provided in binary form only. This application also provides source code format for the driver that supports access to the hardware.

To enable the GPS module, select the corresponding elements from the Platform Builder Catalog for the current OS Design. The binary files and the registry settings that correspond to the elements you selected will be included in the OS run-time image.

The GPS module uses UART on the 3-Stack platform. Reset and power on/ power off to the GPS module are controlled by the GPIO pins of the i.MX31.

The GPS module functionality is segmented into subsystems. You do not need to select all of the subsystems in order to enable GPS on the platform.

Figure 14-1 shows the architecture of GPS driver. There are three layers in whole GPS software system, as follows:

- Application layer
- GPS core driver layer
- GPS HAL driver layer

14.1.1 Application layer

Handset applications, TCP/IP stack and GSM layer3 in Figure 14-1 below to application layer.

Handset applications, such as VisualGpsce.exe or any other mapping software, can receive standard NMEA data to show position with a friendly user interface.

TCP/IP stack and GSM layer3 can provide assisted GPS navigation sevice when satellite signal is not good enough to get fix.

14.1.2 GPS core driver layer

The deep color part (GLL) belongs to GPS core driver layer.

The GPS core driver runs at host and communicates with GPS chip by calling GPS HAL driver. The driver is used for position computation and assistance data management.

14.1.3 GPS HAL driver layer

GPS HAL drivers provide hardware related resource, such as serial port driver, non-volatile storage and GPIO functions, and so on.

The only things that needs to be provided is GPIO functions, which are used to control GPS power state and reset functionalities. The driver is called gpscontroldriver.dll, and source code can be found at ..\PLATFORM\<*tgtplat*>\SRC\DRIVERS\GPSCTRL.

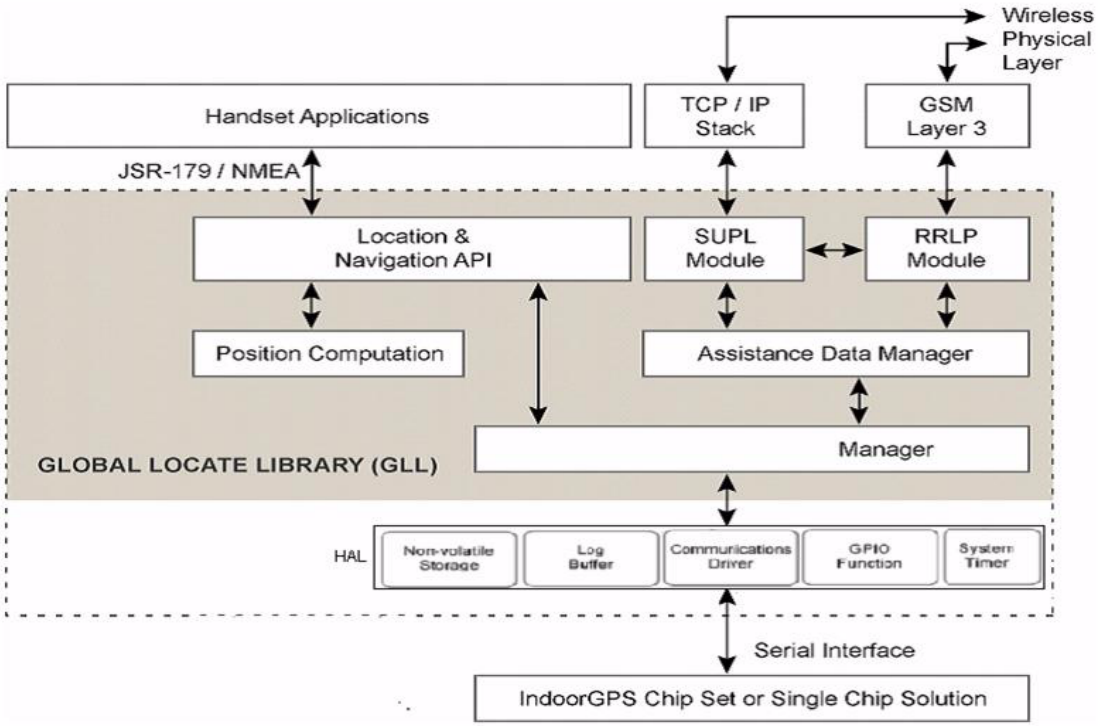

Figure 14-1. Software architecture of GPS driver.

Table 14-1. GPS Driver Summary

Driver Attribute	Definition
Target Platform (TGTPLAT)	3DS

Table 14-1. GPS Driver Summary

Target SOC (TGTSOC)	MX31
MXARM11 CSP Driver Path	N/A
CSP Driver Path	N/A
CSP Static Library	N/A
Platform Driver Path	..\PLATFORM\<*tgtplat*>\SRC\DRIVERS\GPS ..\PLATFORM\<*tgtplat*>\SRC\DRIVERS\GPSCTRL
Import Library	N/A
Driver DLL	Gpsct.exe;GpsNavigationLauncher.exe;GpsctServiceLauncher.exe;GpsctService.dll;GlvcDriver.dll;gpscontroldriver.dll LtoManager.exe, LtoManager.exe.config, LtoManagerLauncher.exe, log4net.dll, OpenNetCF_GL.dll, OpenNetCF.Net_GL.dll, OpenNetCF.Windows.Form_GL.dll
Catalog Items	Third Party ->BSPs ->Freescale <*tgtplat*> -> Device Drivers ->GPS-> GPS core drivers, Third Party ->BSPs ->Freescale <*tgtplat*> -> Device Drivers ->GPS-> GPS control driver,
SYSGEN Dependency	N/A
BSP Environment Variables	BSP_GPS_COREDRIVER = 1 BSP_GPS_CONTROL_DRIVER= 1 BSP_SERIAL_UART3 = 1

14.2 Supported Functionality

The GPS driver enables the 3-Stack board to provide the following software and hardware support:

- Integrates the Barracuda GPS module from BroadCom company
- Supports power management mode full on/full off

14.3 Hardware Operation

14.3.1 UART port

The GPS module uses UART3 to communicate with host.

14.3.2 GPIO control

Two GPIO pins of the 3-Stack platform are used to control the GPS module (Table 14-2).

Table 14-2. GPIO Pins

GPIO Name	PIN	Value Description
BSP_GPIO_GPS_RESET	MCU2_15	0: Reset of GPS module is asserted 1: Reset of GPS module is de-asserted
BSP_GPIO_PWR_EN_GPS	MCU3_2	1: GPS module is powered on 0: GPS module is powered off

14.3.3 Conflicts with other SoC Peripherals

No conflicts.

14.4 Software Operation

14.4.1 Communicating with the GPS Module

Software applications communicate with the GPS module through a virtual COM port (that is, COM8). The virtual COM port is a standard stream interface driver, and is thus accessed through the file system APIs. For example, the Win32 API CreateFile() call can be used to obtain a handle and ReadFile() can be used to read the NMEA data stream output by the GPS module.

14.4.2 Power Management

The 3-Stack platform functions GPS_PowerUp and GPS_PowerDown are used to bring the GPS module into and out of standby mode. The code is designed to keep the power consumption of the GPS module at a minimal level when the standby power state is invoked.

14.4.3 GPS Driver Registry Settings

14.4.3.1 Configuration Registry Keys

Contact BroadCom for details.

14.5 Unit Test

If available, you may want to use a third-party GPS application to perform unit tests. For assistance and test information, contact BroadCom.

Chapter 15
Hantro Codecs Drivers

Hantro codecs drivers provide a feature-rich video experience with high-performance capabilities and low power consumption.

The Hantro MPEG4 Video VGA Encoder is an IP wrapper for the 5250 Hardware VGA MPEG4/H.263 Encoder. The Hantro MPEG4/H.263 Decoder is a software implementation with IPU post-processing hardware acceleration. The Hantro codecs are a Microsoft DirectShow-based implementation.

15.1 Codecs Drivers Summary

Table 15-1 identifies the binary code location, library dependencies, and other BSP information.

Table 15-1. Hantro Codecs Drivers Summary

Driver Attribute	Definition
Target Platform (TGTPLAT)	3DS
Target SOC (TGTSOC)	MX31
MXARM11 CSP Driver Path	N/A
CSP Driver Path	N/A
CSP Static Library	N/A
Platform Driver Path	..\PLATFORM\<*tgtplat*>\SRC\DRIVERS\HANTRO_CODECS
Import Library	strmbase.lib, msdmo.lib, strmiids.lib, dmoguids.lib, coredll.lib, quartz.lib, ole32.lib, oleaut32.lib, uuid.lib, mmtimer.lib
Driver DLL/LIB	htr_pp.lib, htrdecdmo.dll, htrdecdmo.lib, htrencftr.dll, htrencftr.lib, mp4enc.dll, mp4enc.lib, sw_decoder.lib, htrdecfltr.dll, htrdecfltr.lib
Catalog Item	Third Party > BSPs > Freescale <*tgtplat*> > Device Drivers > Hantro_Codecs
SYSGEN Dependency	SYSGEN_DSHOW_DMO=1 SYSGEN_DSHOW=1 SYSGEN_DSHOW_VIDREND=1
BSP Environment Variables	BSP_HANTRO_CODECS=1

15.2 Supported Functionality

The Hantro codecs enable the 3-Stack board to provide the following software and hardware support:

- The encoder supports the MPEG-4 Simple Profile and H.263 Profile 0.
- The encoder supports video resolutions up to VGA (640x480) at up to 30 frames per second.
- The encoder supports Microsoft DirectShow filter.
- The decoder supports MPEG4 Simple Profile levels from 0 to 5 and H.263 Profile 0 levels from 10 to 45.
- The decoder supports resolutions up to VGA (640x480) at up to 30 frames per second.
- The decoder supports Microsoft DirectShow DMO (DirectX Media Objects) and DirectShow filter.

15.3 Hardware Operation

15.3.1 Conflicts with other SoC peripherals

Hantro codecs do not have conflicts with any other module.

15.4 Software Operation

The codecs drivers follow the Microsoft-recommended architecture for DirectShow. For details of this architecture and its operation, see the Platform Builder Help or www.microsoft.com.

15.4.1 Power Management

Power management is not implemented in the Hantro codecs drivers.

15.4.2 Codecs Registry Settings

The following registry keys are required to properly load the decoder drivers:

```
[HKEY_CLASSES_ROOT\CLSID\{6AD7CB74-75B7-4887-86AE-90099A10C5E2}]
@="Hantro Decoder DMO"
[HKEY_CLASSES_ROOT\CLSID\{6AD7CB74-75B7-4887-86AE-90099A10C5E2}\InprocServer32]
@="htrdecdmo.dll"
"ThreadingModel"="Both"

[HKEY_CLASSES_ROOT\DirectShow\MediaObjects\6ad7cb74-75b7-4887-86ae-90099a10c5e2]
@="Hantro Decoder DMO"
"InputTypes"=hex:76,69,64,73,00,00,10,00,80,00,00,aa,00,38,9b,71,4d,50,34,56,\
  00,00,10,00,80,00,00,aa,00,38,9b,71,76,69,64,73,00,00,10,00,80,00,00,aa,00,\
  38,9b,71,6d,70,34,76,00,00,10,00,80,00,00,aa,00,38,9b,71,76,69,64,73,00,00,\
  10,00,80,00,00,aa,00,38,9b,71,48,32,36,33,00,00,10,00,80,00,00,aa,00,38,9b,\
  71,76,69,64,73,00,00,10,00,80,00,00,aa,00,38,9b,71,68,32,36,33,00,00,10,00,\
  80,00,00,aa,00,38,9b,71
"OutputTypes"=hex:76,69,64,73,00,00,10,00,80,00,00,aa,00,38,9b,71,49,59,55,56,\
  00,00,10,00,80,00,00,aa,00,38,9b,71,76,69,64,73,00,00,10,00,80,00,00,aa,00,\
  38,9b,71,49,59,55,56,00,00,10,00,80,00,00,aa,00,38,9b,71

[HKEY_CLASSES_ROOT\DirectShow\MediaObjects\Categories\4a69b442-28be-4991-969c-b500adf5d
8a8\6ad7cb74-75b7-4887-86ae-90099a10c5e2]

[HKEY_LOCAL_MACHINE\SOFTWARE\Classes\CLSID\{6AD7CB74-75B7-4887-86AE-90099A10C5E2}]
@="Hantro Decoder DMO"
[HKEY_LOCAL_MACHINE\SOFTWARE\Classes\CLSID\{6AD7CB74-75B7-4887-86AE-90099A10C5E2}\Inpro
cServer32]
@="htrdecdmo.dll"
"ThreadingModel"="Both"

[HKEY_LOCAL_MACHINE\SOFTWARE\Classes\DirectShow\MediaObjects\6ad7cb74-75b7-4887-86ae-90
099a10c5e2]
@="Hantro Decoder DMO"
"InputTypes"=hex:76,69,64,73,00,00,10,00,80,00,00,aa,00,38,9b,71,4d,50,34,56,\
  00,00,10,00,80,00,00,aa,00,38,9b,71,76,69,64,73,00,00,10,00,80,00,00,aa,00,\
  38,9b,71,6d,70,34,76,00,00,10,00,80,00,00,aa,00,38,9b,71,76,69,64,73,00,00,\
  10,00,80,00,00,aa,00,38,9b,71,48,32,36,33,00,00,10,00,80,00,00,aa,00,38,9b,\
  71,76,69,64,73,00,00,10,00,80,00,00,aa,00,38,9b,71,68,32,36,33,00,00,10,00,\
  80,00,00,aa,00,38,9b,71
"OutputTypes"=hex:76,69,64,73,00,00,10,00,80,00,00,aa,00,38,9b,71,49,59,55,56,\
  00,00,10,00,80,00,00,aa,00,38,9b,71,76,69,64,73,00,00,10,00,80,00,00,aa,00,\
  38,9b,71,49,59,55,56,00,00,10,00,80,00,00,aa,00,38,9b,71

[HKEY_LOCAL_MACHINE\SOFTWARE\Classes\DirectShow\MediaObjects\Categories\4a69b442-28be-4
991-969c-b500adf5d8a8\6ad7cb74-75b7-4887-86ae-90099a10c5e2]
```

The following registry keys are required to properly load the encoder drivers:

```
[HKEY_CLASSES_ROOT\CLSID\{9FD1B4C9-BF1E-4E7D-ACE6-586BBB3280C6}]
@="Hantro MPEG-4/H.263 Video Encoder Filter"
"Merit"=dword:00600000
[HKEY_CLASSES_ROOT\CLSID\{9FD1B4C9-BF1E-4E7D-ACE6-586BBB3280C6}\InprocServer32]
@="htrencftr.dll"
"ThreadingModel"="Both"

[HKEY_CLASSES_ROOT\Filter\{9FD1B4C9-BF1E-4E7D-ACE6-586BBB3280C6}]
@="Hantro MPEG-4/H.263 Video Encoder Filter"

[HKEY_CLASSES_ROOT\CLSID\{9FD1B4C9-BF1E-4E7D-ACE6-586BBB3280C6}\Pins]

[HKEY_CLASSES_ROOT\CLSID\{9FD1B4C9-BF1E-4E7D-ACE6-586BBB3280C6}\Pins\Input]
"Direction"=dword:00000000
"IsRendered"=dword:00000000
"AllowedZero"=dword:00000000
"AllowedMany"=dword:00000000
"ConnectsToPin"="Output"
[HKEY_CLASSES_ROOT\CLSID\{9FD1B4C9-BF1E-4E7D-ACE6-586BBB3280C6}\Pins\Input\Types]
[HKEY_CLASSES_ROOT\CLSID\{9FD1B4C9-BF1E-4E7D-ACE6-586BBB3280C6}\Pins\Input\Types\{73646
976-0000-0010-8000-00AA00389B71}]
[HKEY_CLASSES_ROOT\CLSID\{9FD1B4C9-BF1E-4E7D-ACE6-586BBB3280C6}\Pins\Input\Types\{73646
976-0000-0010-8000-00AA00389B71}\]
[HKEY_CLASSES_ROOT\CLSID\{9FD1B4C9-BF1E-4E7D-ACE6-586BBB3280C6}\Pins\Input\Types\{73646
976-0000-0010-8000-00AA00389B71}\{56555949-0000-0010-8000-00AA00389B71}]
[HKEY_CLASSES_ROOT\CLSID\{9FD1B4C9-BF1E-4E7D-ACE6-586BBB3280C6}\Pins\Input\Types\{73646
976-0000-0010-8000-00AA00389B71}\{56555949-0000-0010-8000-00AA00389B71}\]

[HKEY_CLASSES_ROOT\CLSID\{9FD1B4C9-BF1E-4E7D-ACE6-586BBB3280C6}\Pins\Output]
"Direction"=dword:00000001
"IsRendered"=dword:00000000
"AllowedZero"=dword:00000000
"AllowedMany"=dword:00000000
"ConnectsToPin"="Input"
[HKEY_CLASSES_ROOT\CLSID\{9FD1B4C9-BF1E-4E7D-ACE6-586BBB3280C6}\Pins\Output\Types]

[HKEY_CLASSES_ROOT\CLSID\{9FD1B4C9-BF1E-4E7D-ACE6-586BBB3280C6}\Pins\Output\Types\{7364
6976-0000-0010-8000-00AA00389B71}]

[HKEY_CLASSES_ROOT\CLSID\{9FD1B4C9-BF1E-4E7D-ACE6-586BBB3280C6}\Pins\Output\Types\{7364
6976-0000-0010-8000-00AA00389B71}\]

[HKEY_CLASSES_ROOT\CLSID\{9FD1B4C9-BF1E-4E7D-ACE6-586BBB3280C6}\Pins\Output\Types\{7364
6976-0000-0010-8000-00AA00389B71}\{5634504D-0000-0010-8000-00AA00389B71}]

[HKEY_CLASSES_ROOT\CLSID\{9FD1B4C9-BF1E-4E7D-ACE6-586BBB3280C6}\Pins\Output\Types\{7364
6976-0000-0010-8000-00AA00389B71}\{5634504D-0000-0010-8000-00AA00389B71}\]

[HKEY_CLASSES_ROOT\CLSID\{9FD1B4C9-BF1E-4E7D-ACE6-586BBB3280C6}\Pins\Output\Types\{7364
6976-0000-0010-8000-00AA00389B71}\{7634706D-0000-0010-8000-00AA00389B71}]

[HKEY_CLASSES_ROOT\CLSID\{9FD1B4C9-BF1E-4E7D-ACE6-586BBB3280C6}\Pins\Output\Types\{7364
6976-0000-0010-8000-00AA00389B71}\{7634706D-0000-0010-8000-00AA00389B71}\]
```

```
[HKEY_CLASSES_ROOT\CLSID\{9FD1B4C9-BF1E-4E7D-ACE6-586BBB3280C6}\Pins\Output\Types\{7364
6976-0000-0010-8000-00AA00389B71}\{33363248-0000-0010-8000-00AA00389B71}]

[HKEY_CLASSES_ROOT\CLSID\{9FD1B4C9-BF1E-4E7D-ACE6-586BBB3280C6}\Pins\Output\Types\{7364
6976-0000-0010-8000-00AA00389B71}\{33363248-0000-0010-8000-00AA00389B71}\]

[HKEY_CLASSES_ROOT\CLSID\{9FD1B4C9-BF1E-4E7D-ACE6-586BBB3280C6}\Pins\Output\Types\{7364
6976-0000-0010-8000-00AA00389B71}\{33363268-0000-0010-8000-00AA00389B71}]

[HKEY_CLASSES_ROOT\CLSID\{9FD1B4C9-BF1E-4E7D-ACE6-586BBB3280C6}\Pins\Output\Types\{7364
6976-0000-0010-8000-00AA00389B71}\{33363268-0000-0010-8000-00AA00389B71}\]

[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\Mp4Enc]
   "Prefix"="MPE"
   "Dll"="mp4enc.dll"
   "IClass"="{A32942B7-920C-486b-B0E6-92A702A99B35}"

; Hantro MPEG-4/H.263 Video Decoder Filter

; DLL registration
[HKEY_CLASSES_ROOT\CLSID\{9A81E196-3BBA-4821-B18B-21BB496F80F8}]
@="Hantro MPEG-4/H.263 Video Decoder Filter"
"Merit"=dword:00600000
[HKEY_CLASSES_ROOT\CLSID\{9A81E196-3BBA-4821-B18B-21BB496F80F8}\InprocServer32]
@="htrdecfltr.dll"
"ThreadingModel"="Both"

; Registration for DirectShow filtergraph
[HKEY_CLASSES_ROOT\Filter\{9A81E196-3BBA-4821-B18B-21BB496F80F8}]
@="Hantro MPEG-4/H.263 Video Decoder Filter"

; Input pin
; Direction= =0 [Input]
; Rendered= 0 [Not rendered]
; AllowedZero= 0 [Required to be connected always when running]
; AllowedMany= 0 [Many instances NOT allowed]
; Output types  = MEDIATYPE_Video; MP4V, mp4v, H263, h263
[HKEY_CLASSES_ROOT\CLSID\{9FD1B4C9-BF1E-4E7D-ACE6-586BBB3280C6}\Pins\Output]
"Direction"=dword:00000000
"IsRendered"=dword:00000000
"AllowedZero"=dword:00000000
"AllowedMany"=dword:00000000
"ConnectsToPin"="Input"
[HKEY_CLASSES_ROOT\CLSID\{9FD1B4C9-BF1E-4E7D-ACE6-586BBB3280C6}\Pins\Output\Types]
[HKEY_CLASSES_ROOT\CLSID\{9FD1B4C9-BF1E-4E7D-ACE6-586BBB3280C6}\Pins\Output\Types\{7364
6976-0000-0010-8000-00AA00389B71}]
[HKEY_CLASSES_ROOT\CLSID\{9FD1B4C9-BF1E-4E7D-ACE6-586BBB3280C6}\Pins\Output\Types\{7364
6976-0000-0010-8000-00AA00389B71}\]
[HKEY_CLASSES_ROOT\CLSID\{9FD1B4C9-BF1E-4E7D-ACE6-586BBB3280C6}\Pins\Output\Types\{7364
6976-0000-0010-8000-00AA00389B71}\{5634504D-0000-0010-8000-00AA00389B71}]
[HKEY_CLASSES_ROOT\CLSID\{9FD1B4C9-BF1E-4E7D-ACE6-586BBB3280C6}\Pins\Output\Types\{7364
6976-0000-0010-8000-00AA00389B71}\{5634504D-0000-0010-8000-00AA00389B71}\]
[HKEY_CLASSES_ROOT\CLSID\{9FD1B4C9-BF1E-4E7D-ACE6-586BBB3280C6}\Pins\Output\Types\{7364
6976-0000-0010-8000-00AA00389B71}\{7634706D-0000-0010-8000-00AA00389B71}]
[HKEY_CLASSES_ROOT\CLSID\{9FD1B4C9-BF1E-4E7D-ACE6-586BBB3280C6}\Pins\Output\Types\{7364
6976-0000-0010-8000-00AA00389B71}\{7634706D-0000-0010-8000-00AA00389B71}\]
```

```
[HKEY_CLASSES_ROOT\CLSID\{9FD1B4C9-BF1E-4E7D-ACE6-586BBB3280C6}\Pins\Output\Types\{7364
6976-0000-0010-8000-00AA00389B71}\{33363248-0000-0010-8000-00AA00389B71}]
[HKEY_CLASSES_ROOT\CLSID\{9FD1B4C9-BF1E-4E7D-ACE6-586BBB3280C6}\Pins\Output\Types\{7364
6976-0000-0010-8000-00AA00389B71}\{33363248-0000-0010-8000-00AA00389B71}\]
[HKEY_CLASSES_ROOT\CLSID\{9FD1B4C9-BF1E-4E7D-ACE6-586BBB3280C6}\Pins\Output\Types\{7364
6976-0000-0010-8000-00AA00389B71}\{33363268-0000-0010-8000-00AA00389B71}]
[HKEY_CLASSES_ROOT\CLSID\{9FD1B4C9-BF1E-4E7D-ACE6-586BBB3280C6}\Pins\Output\Types\{7364
6976-0000-0010-8000-00AA00389B71}\{33363268-0000-0010-8000-00AA00389B71}\]

; Pin descriptions
[HKEY_CLASSES_ROOT\CLSID\{9A81E196-3BBA-4821-B18B-21BB496F80F8}\Pins]

; Output pin
; Direction= 1 [Output]
; Rendered= 0 [Not rendered]
; AllowedZero= 0 [Required to be connected always when running]
; AllowedMany= 0 [Many instances NOT allowed]
; Input types   = MEDIATYPE_Video; MEDIASUBTYPE_IYUV, MEDIASUBTYPE_YV12
[HKEY_CLASSES_ROOT\CLSID\{9FD1B4C9-BF1E-4E7D-ACE6-586BBB3280C6}\Pins\Input]
"Direction"=dword:00000000
"IsRendered"=dword:00000000
"AllowedZero"=dword:00000000
"AllowedMany"=dword:00000000
"ConnectsToPin"="Output"
[HKEY_CLASSES_ROOT\CLSID\{9FD1B4C9-BF1E-4E7D-ACE6-586BBB3280C6}\Pins\Input\Types]
[HKEY_CLASSES_ROOT\CLSID\{9FD1B4C9-BF1E-4E7D-ACE6-586BBB3280C6}\Pins\Input\Types\{73646
976-0000-0010-8000-00AA00389B71}]
[HKEY_CLASSES_ROOT\CLSID\{9FD1B4C9-BF1E-4E7D-ACE6-586BBB3280C6}\Pins\Input\Types\{73646
976-0000-0010-8000-00AA00389B71}\]
[HKEY_CLASSES_ROOT\CLSID\{9FD1B4C9-BF1E-4E7D-ACE6-586BBB3280C6}\Pins\Input\Types\{73646
976-0000-0010-8000-00AA00389B71}\{56555949-0000-0010-8000-00AA00389B71}]
[HKEY_CLASSES_ROOT\CLSID\{9FD1B4C9-BF1E-4E7D-ACE6-586BBB3280C6}\Pins\Input\Types\{73646
976-0000-0010-8000-00AA00389B71}\{56555949-0000-0010-8000-00AA00389B71}\]
[HKEY_CLASSES_ROOT\CLSID\{9FD1B4C9-BF1E-4E7D-ACE6-586BBB3280C6}\Pins\Input\Types\{73646
976-0000-0010-8000-00AA00389B71}\{32315659-0000-0010-8000-00AA00389B71}]
[HKEY_CLASSES_ROOT\CLSID\{9FD1B4C9-BF1E-4E7D-ACE6-586BBB3280C6}\Pins\Input\Types\{73646
976-0000-0010-8000-00AA00389B71}\{32315659-0000-0010-8000-00AA00389B71}\]
[HKEY_CLASSES_ROOT\CLSID\{9FD1B4C9-BF1E-4E7D-ACE6-586BBB3280C6}\Pins\Input\Types\{73646
976-0000-0010-8000-00AA00389B71}\{E436EB7B-0000-0010-8000-00AA00389B71}]
[HKEY_CLASSES_ROOT\CLSID\{9FD1B4C9-BF1E-4E7D-ACE6-586BBB3280C6}\Pins\Input\Types\{73646
976-0000-0010-8000-00AA00389B71}\{E436EB7B-0000-0010-8000-00AA00389B71}\]
```

15.5 Unit Test

The Hantro codecs unit test uses simple test bench source code and test data. The test bench is an example for developing DirectShow applications. If you need this test application, contact FAE/AE for assistance.

In order to reduce the BSP package size, a limited amount of test data has been included. You may develop additional test data. For instructions see “Section 15.5.3, “Customizing the Test Bench/Data”.

15.5.1 Unit Test Software

Table 15-2 lists the software required to run the unit tests.

Table 15-2. Software Requirements for Hantro Codecs Driver

Requirements	Description
htrenctb.exe	Test bench for the encoder testing for the i.MX31 with the default test data.
htrdecdmotb.exe	Test bench for the decoder dmo testing for the i.MX31 with the default test data.
htrdecfltrtb.exe	Test bench for the decoder filter testing for the i.MX31 with the default test data.

15.5.2 Building the Codecs Test Bench

To build the codecs test bench in Windows CE, you must build an OS image for the desired configuration.

To build an OS image:

1. In Platform Builder, from the **Build OS** menu, select **Open Release Directory**. This opens a DOS prompt.
2. Change to the HANTRO_CODECS Tests directory (`\WINCE500\SUPPORT\TESTS\HANTRO_CODECS`)
3. Enter **set WINCEREL=1** on the command prompt and hit <return>. This copies the generated ".exe" to the flat release directory.
4. To build the encoder test bench for the i.MX31, change to `\WINCE500\SUPPORT\TESTS\HANTRO_CODECS\ENCODE\htrenctb`, enter **build –c** at the prompt, and then press Enter. The `htrenctb.exe` file will be located in the $(_FLATRELEASEDIR) directory.
5. Copy `\WINCE500\SUPPORT\TESTS\HANTRO_CODECS\ENCODE\testdata` to the $(_FLATRELEASEDIR) directory.
6. To build the decoder DMO test bench, change to `\WINCE500\SUPPORT\TESTS\HANTRO_CODECS\DECODE\htrdectb`, enter **build –c** at the prompt, and then press Enter. The `htrdecdmotb.exe` file will be located in the $(_FLATRELEASEDIR) directory.
7. And/or to build the decoder filter test bench, change to `\WINCE500\SUPPORT\TESTS\HANTRO_CODECS\DECODE\htrdecfltrtb`, enter **build –c** at the prompt, and then press Enter. The `htrdecfltrtb.exe` file will be located in the $(_FLATRELEASEDIR) directory.
8. Copy `\WINCE500\SUPPORT\TESTS\HANTRO_CODECS\DECODE\test_sequences` to the $(_FLATRELEASEDIR) directory.

15.5.3 Customizing the Test Bench/Data

To include your customized test files for encoding:

1. Place the data files and reference stream files at:
   `\WINCE500\SUPPORT\TESTS\HANTRO_CODECS\ENCODE\testdata`

2. Modify the `TestCases.h` file for the i.MX31
   `\WINCE500\SUPPORT\TESTS\HANTRO_CODECS\ENCODE\htrenctb\TestCases.h`
3. Follow the steps in the previous section for building the encoder test bench.

NOTE

In `TestSettings.h`, the defaults are “true” to “keepOutputFiles” and “false” to “runComparation”. You can change them.

To include your customized test files for decoding:

1. Place the data files and reference stream files in:
   `\WINCE500\SUPPORT\TESTS\HANTRO_CODECS\DECODE\test_sequences`
2. Modify the file[] and referenceData[] variables in:
   `\WINCE500\SUPPORT\TESTS\HANTRO_CODECS\DECODE\htrdectb\ DecoderFilterTestBench.cpp`
3. And/or modify the
4. `\WINCE500\SUPPORT\TESTS\HANTRO_CODECS\DECODE\htrdecfltrtb\ HtrParameters_mpeg4.h`
5. Follow the steps in the previous section for building the decoder test bench.

NOTE

For the decoder DMO test bench, the #define VIDEO_RENDERER in DecoderFilterTestBench.cpp is uncommented. This enables DirectShow to be the default provided Video Renderer. To use file-based testing, comment #define VIDEO_RENDERER in `DecoderFilterTestBench.cpp`.

15.5.4 Running the Codecs Tests

After downloading an OS image to the board, execute the test bench from the Windows CE target shell with one of the following commands:

- Encoder filter test for the i.MX31:
  `s htrenctb.exe`
- Decoder DMO test for the i.MX31:
  `s htrdecdmotb.exe`
- Decoder filter test for the i.MX31:
  `s htrdecfltrtb.exe`

15.6 Codecs Drivers API Reference

For detailed information about DirectShow, see the Platform Builder Help or www.microsoft.com.

For the Hantro codecs engine APIs information, see the *5250 API User Manual* and the *MPEG4 Decoder API User Manual*, in `\WINCE500\SUPPORT\TESTS\HANTRO _CODECS\DOCS` of this BSP release package.

Chapter 16
Inter-Integrated Circuit (I^2C) Driver

The Inter-Integrated Circuit (I^2C) module provides the functionality of a standard I^2C slave and master. The I^2C module is compatible with the standard Phillips I^2C bus protocol.

16.1 I^2C Driver Summary

Table 16-1 identifies the source code location, library dependencies, and other BSP information.

Table 16-1. I^2C Driver Summary

Driver Attribute	Definition
Target Platform (TGTPLAT)	3DS
Target SOC (TGTSOC)	MX31
MXARM11 CSP Driver Path	..\CSP\ARM\FREESCALE\MXARM11\DRIVERS\I2C
CSP Driver Path	..\CSP\ARM\FREESCALE\<TGTSOC>\DRIVERS\I2C
CSP Static Libraries	mxarm11_i2c.lib, <*TGTSOC*>_i2c.lib
Platform Driver Path	..\PLATFORM\<*tgtplat*>\SRC\DRIVERS\I2C
Import Library	N/A
Driver DLL	i2c.dll
Catalog Item	Third Party > BSPs > Freescale <*tgtplat*> > Device Drivers > I2C Bus
SYSGEN Dependency	N/A
BSP Environment Variables	BSP_I2CBUS=1

16.2 Supported Functionality

The I^2C driver enables the 3-Stack board to provide the following software and hardware support:

- Supports the I^2C communication protocol
- Supports multiple I^2C controllers
- Supports the I^2C master mode of operation
- Does not support the I^2C slave mode of operation
- Is a stream interface driver implementing the programming interface defined in this document
- Supports two power management modes, full on and full off

16.3 Hardware Operation

Refer to the chapter on I^2C in the *MCIMX31 and MCIMX31L Applications Processors Reference Manual* for detailed operation and programming information.

16.3.1 Conflicts with other SoC Peripherals

The i.MX31 platform contains three I^2C modules, but only one of these modules may be used on the i.MX31 SDK board, the I^2C1 module. I^2C2 does not have any allocated pins, and I2C3 shares pins with the CSPI2 module. The CSPI2 signals are selected in the IOMUX, as they are required for proper communication with the MC13783 PMIC.

16.4 Software Operation

16.4.1 Communicating with the I2C

The I2C is a stream interface driver, and is thus accessed through the file system APIs. To communicate using the I2C, create a handle to the device using the **CreateFile** function. Issue subsequent commands to the device using the **DeviceIoControl** function with IOCTL codes specifying the desired operation. If preferred, the **DeviceIoControl** function calls can be replaced with macros that hide the **DeviceIoControl** call details. The basic steps are detailed below.

16.4.2 Creating a Handle to the I2C

Call the **CreateFile** function to open a connection to the I2C device. Specify an I2C port in this call. The format is “I2CX”, with X being the number indicating the I2C port. This number should not exceed the number of I2C instances on the platform. If an I2C port does not exist, **CreateFile** returns ERROR_FILE_NOT_FOUND.

To open a handle to the I2C:

1. Insert a colon after the I2C port for the first parameter, *lpFileName*.
   — For example, specify I2C1: as the I2C port.
2. Specify FILE_SHARE_READ | FILE_SHARE_WRITE in the *dwShareMode* parameter. Multiple handles to an I2C port are supported by the driver.
3. Specify OPEN_EXISTING in the *dwCreationDisposition* parameter.
   — This flag is required.
4. Specify FILE_FLAG_RANDOM_ACCESS in the *dwFlagsAndAttributes* parameter.

The following code example shows how to open an I2C port.

```
// Open the serial port.
hI2C = CreateFile (CAM_I2C_PORT,  // name of device
                   GENERIC_READ | GENERIC_WRITE,       // access (read-write) mode
                   FILE_SHARE_READ | FILE_SHARE_WRITE, // sharing mode
                   NULL,           // security attributes (ignored)
                   OPEN_EXISTING,  // creation disposition
                   FILE_FLAG_RANDOM_ACCESS,           // flags/attributes
```

```
                    NULL);          // template file (ignored)
```

Before writing to or reading from an I2C port, configure the port.

When an application opens an I2C port, it uses the default configuration settings, which might not be suitable for the device at the other end of the connection.

16.4.3 Configuring the I2C Port

Configuring the I2C port for communications involves two main operations:

- setting the I2C frequency, and
- setting the Self Address (the address for the I2C port on the platform)

Before these actions can be taken, a handle to the I2C port must already be opened. Each of these steps requires a call to the **DeviceIoControl** function. The following parameters are required: the I2C port handle; appropriate IOCTL code; and other input and output parameters.

To configure an I2C port:

1. Set the *hDevice* parameter to the previously acquired I2C port handle.
2. Set the *dwIoControlCode* to one of the following IOCTL codes:
   — I2C_IOCTL_SET_FREQUENCY
   — I2C_IOCTL_SET_SELF_ADDR
3. Set the *lpInBuffer* to point to the variable that you want to use for the I2C port setting. Set nInBufferSize to the size of that variable.
4. Set *lpOutBuffer*, *lpBytesReturned*, and *lpOverlapped* to NULL.
5. Set *nOutBufferSize* to 0.

The following code example shows how to configure the I2C port.

```
// Clock frequency set at 1MHz
DWORD dwFrequency = 1000000;         // I2C frequency
BYTE bySelf = 0x20;                  // Self address value

// Set I2C frequency
DeviceIoControl(hI2C,                       // file handle to the driver
                I2C_IOCTL_SET_FREQUENCY, // I/O control code
                (PBYTE) &dwFrequency,    // in buffer
                sizeof(dwFrequency),     // in buffer size
                NULL,                    // out buffer
                0,                       // out buffer size
                NULL,                    // number of bytes returned
                NULL);                   // ignored (=NULL)

// Set I2C self address
DeviceIoControl(hI2C,                       // file handle to the driver
                I2C_IOCTL_SET_SELF_ADDR, // I/O control code
                (PBYTE) &bySelf,         // in buffer
                sizeof(bySelf),          // in buffer size
                NULL,                    // out buffer
                0,                       // out buffer size
                NULL,                    // number of bytes returned
```

```
                    NULL);                  // ignored (=NULL)
```

As a substitute for the **DeviceIoControl** calls above, macros may be used to simplify the code. The following are examples:

```
I2C_MACRO_SET_FREQUENCY(hI2C, dwFrequency);
I2C_MACRO_SET_SELF_ADDR(hI2C, bySelf);
```

16.4.4 Data Transfer Operations

The I2C driver provides one command, **Transfer**, that facilitates performing both reads and writes through the I2C. The basic unit of data transfer in the I2C driver is the I2C_PACKET, which contains a buffer for reading or writing data and a flag that specifies whether the desired operation is a Read or a Write. An array of these packets makes up an I2C_TRANSFER_BLOCK object, which is needed to perform a Transfer operation. The steps below detail the process of performing write and read operations through the I2C.

Before these actions can be taken, a handle to the I2C port must already be opened. Each of these steps requires a call to the **DeviceIoControl** function. As parameters, the I2C port handle, appropriate IOCTL code, and other input and output parameters are required.

To perform an I2C transfer:

1. Create an array of I2C_PACKET objects and initialize the fields of each packet as follows:
   a) Set the *byRW* field to I2C_RW_WRITE to specify that the I2C operation is a Write, or I2C_RW_READ to specify that the I2C operation is a Read.
   b) Set the *byAddr* field to the 7-bit I2C slave address of the device to which the data will be written.

      NOTE

      The byAddr field requires the 7-bit I2C slave address, aligned to the least significant 7 bits. This address will be shifted left one bit and ORed with the read/write bit to compose the 8-bit value sent out during the I2C slave address cycle. In older versions of this driver, the slave address was entered as the most significant 7 bits of the 8-bit value.

   c) If *byRW* is set to I2C_RW_WRITE, create a buffer of bytes and fill it with the data to write to the slave device. Set the *pbyBuf* field to point to this buffer. If is set to I2C_RW_READ, create a buffer of bytes to hold the data which will be read from the slave device.
   d) Set the *wLen* field to size, in bytes, of the read or write buffer. This will indicate the number of bytes to write or read.
   e) Set the *lpiResult* field to point to an integer that will hold the return value from the write operation.
2. Set the *hDevice* parameter to the previously acquired I2C port handle.
3. Set the *dwIoControlCode* to the I2C_IOCTL_TRANSFER IOCTL code.
4. Set the *lpInBuffer* to point to the I2C_TRANSFER_BLOCK object created in step 1. Set *nInBufferSize* to the size of that packet object.
5. Set *lpOutBuffer*, *lpBytesReturned*, and *lpOverlapped* to NULL. Set *nOutBufferSize* to 0.

6. After calling the **DeviceIoControl** function, check the *lpiResult* field to ensure that the operation was successful. If *lpiResult* points to the I2C_NO_ERROR value, the operation was successful. Otherwise, there was an error.

The following code example demonstrates how to perform a transfer that contains one write and one read packet. The write is performed before the read operation.

```
    I2C_TRANSFER_BLOCK I2CXferBlock;
    I2C_PACKET I2CPacket[2];
    BYTE byAddr = 0x2D;    // Slave Address
    BYTE byOutData = 0x39; // Data to write
    BYTE byInData;         // Read buffer

    // Packet 0 contains write operation
    I2CPacket[0].pbyBuf = (PBYTE) &byOutData;
    I2CPacket[0].wLen = sizeof(byOutData);

    I2CPacket[0].byRW = I2C_RW_WRITE;
    I2CPacket[0].byAddr = byAddr;
    I2CPacket[0].lpiResult = lpiResult;

// Packet 1 contains read operation
    I2CPacket[1].pbyBuf = (PBYTE) &byInData;
    I2CPacket[1].wLen = sizeof(byInData);

    I2CPacket[1].byRW = I2C_RW_READ;
    I2CPacket[1].byAddr = byAddr;
    I2CPacket[1].lpiResult = lpiResult;

    I2CXferBlock.pI2CPackets = I2CPacket;
    I2CXferBlock.iNumPackets = 2;

    // Transfer data through I2C
    DeviceIoControl(hI2C,                  // file handle to the driver
                    I2C_IOCTL_WRITEREG,  // I/O control code
                    (PBYTE) &I2CXferBlock,  // in buffer
                    sizeof(I2CXferBlock),   // in buffer size
                    NULL,                // out buffer
                    0,                   // out buffer size
                    NULL,                // number of bytes returned
                    NULL);               // ignored (=NULL)
```

As a substitute for the **DeviceIoControl** call above, macros may be used to simplify the code. The following is an example:

```
    I2C_MACRO_TRANSFER(hI2C, &I2CXferBlock);
```

Repeated Start

The array of I2C_PACKET objects passed to the Transfer command is guaranteed to be performed sequentially, without interruption or preemption by another driver that is attempting to access the I2C module. An I2C START command initiates the transmission of the first packet in the I2C_TRANSFER_BLOCK array. For subsequent packets, a change in the direction of communication

(from Read to Write or Write to Read) or a change in the target slave address triggers a REPEATED START command before the transmission of the packet. Thus, if a REPEATED START is required between data transfers with a target I2C device, all of those data transfers should be contained within a single I2C_TRANSFER_BLOCK. The final packet in the I2C_TRANSFER_BLOCK is succeeded by an I2C STOP command.

16.4.5 Closing the Handle to the I2C

Call the **CloseHandle** function to close a handle to the I2C when an application is done using it.

CloseHandle has one parameter, which is the handle returned by the **CreateFile** function call that opened the I2C port.

There is a two-second delay after **CloseHandle** is called before the port is closed and resources are freed. This delay allows pending operations to complete.

16.4.6 Power Management

The primary method for limiting power consumption in the I2C module is to gate off all clocks to the module when those clocks are not needed. This is accomplished through the **DDKClockSetGatingMode** function call. In the BSP, the I2C module always operates in master mode and never in slave mode. As a result, the I2C module can be disabled, and its clocks turned off, whenever the module is not processing I2C packets. By contrast, were the I2C module to operate in slave mode, the module would have to be enabled, and have its clocks turned on, at all times in order to properly receive the interrupt that signals the start of a data transfer from another I2C master device.

As described in the **Data Transfer Operations** section, I2C data transfer operations are handled in I2C_TRANSFER_BLOCK objects, which contain one or more packets of I2C data. The I2C driver turns on the I2C clocks and enables the I2C module before processing an I2C_TRANSFER_BLOCK, and then disables and turns off clocks to the I2C module after the block of packets has been processed. This limits the time during which the I2C module is consuming power to the time during which the I2C is actively performing data transfers.

16.4.6.1 PowerUp

This function is not implemented for the I2C driver. Power to the I2C module is managed as I2C transfer operations are processed. There are no additional power management steps needed for the I2C.

16.4.6.2 PowerDown

This function is not implemented for the I2C driver.

16.4.6.3 IOCTL_POWER_SET

This function is not implemented for the I2C driver.

16.4.7 I2C Registry Settings

The following registry keys are required to properly load the I2C1 module.

```
IF BSP_I2CBUS
; @XIPREGION IF PACKAGE_OEMDRIVERS
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\I2C1]
   "Prefix"="I2C"
   "Dll"="i2c.dll"
   "Index"=dword:1
   "Order"=dword:4

ENDIF
```

16.5 Unit Test

There is no CETK Test for I2C.

NOTE

The camera module uses the I2C interface to control its setting, so the I2C function can be verified by the camera module.

16.6 I2C Driver API Reference

16.6.1 I2C Driver IOCTLS

This section consists of descriptions for the I2C I/O control codes (IOCTLs). These IOCTLs are used in calls to **DeviceIoControl** to issue commands to the I2C device. Only relevant parameters for the IOCTL have a description provided.

16.6.1.1 I2C_IOCTL_GET_CLOCK_RATE

This **DeviceIoControl** request retrieves the clock rate divisor. Note that the value is not the absolute peripheral clock frequency. The value retrieved should be compared against the I2C information in the *MCIMX31 and MCIMX31L Applications Processors Reference Manual* to obtain the true frequency.

Parameters

lpOutBuffer	Pointer to the divisor index. The true clock frequency is platform-dependent.
nOutBufferSize	Size in bytes of the divisor index.

16.6.1.2 I2C_IOCTL_GET_SELF_ADDR

This **DeviceIoControl** request retrieves the address of the I2C device. Note that this macro is only meaningful if it is currently in Slave mode.

Parameters

lpOutBuffer Pointer to the current I2C device address. The valid range of the address is [0x00, 0x7F].

nOutBufferSize Size in bytes of the I2C device address.

16.6.1.3 I2C_IOCTL_IS_MASTER

This **DeviceIoControl** request determines whether the I2C is currently in Master mode.

Parameters

lpOutBuffer Pointer to a BYTE that will contain the return value from the Master mode inquiry. TRUE if currently in Master mode; FALSE if currently in Slave mode.

nOutBufferSize Size in bytes of the return value. This should be one byte.

16.6.1.4 I2C_IOCTL_IS_SLAVE

This **DeviceIoControl** request determines whether the I2C is currently in Slave mode.

Parameters

lpOutBuffer Pointer to a BYTE that will contain the return value from the Slave mode inquiry. TRUE if currently in Slave mode; FALSE if currently in Master mode.

nOutBufferSize Size in bytes of the return value. This should be one byte.

16.6.1.5 I2C_IOCTL_RESET

This **DeviceIoControl** request performs a hardware reset. Note that the I2C driver will still maintain all of the current information of the device, including all of the initialized addresses.

16.6.1.6 I2C_IOCTL_SET_CLOCK_RATE

This **DeviceIoControl** request initializes the I2C device with the given clock rate. Note that this IOCTL does not expect to receive the absolute peripheral clock frequency. Rather, it will be expecting the clock rate divisor index stated in the I2C section of the *MCIMX31 and MCIMX31L Applications Processors Reference Manual*. If absolute clock frequency is required, use I2C_MACRO_SET_FREQUENCY.

Parameters

lpInBuffer Pointer to the divisor index. Refer to I2C specification to obtain the true clock frequency.

nInBufferSize Size in bytes of the divisor index.

16.6.1.7 I2C_IOCTL_SET_FREQUENCY

This **DeviceIoControl** request estimates the nearest clock rate acceptable for I2C device and initialize the I2C device to use the estimated clock rate divisor. If the estimated clock rate divisor index is required, see the macro I2C_MACRO_GET_CLOCK_RATE to determine the estimated index.

Parameters

lpInBuffer	Pointer to the desired I2C frequency.
nInBufferSize	Size in bytes of the I2C frequency requested.

16.6.1.8 I2C_IOCTL_SET_MASTER_MODE

This **DeviceIoControl** request sets the I2C device to Master mode.

16.6.1.9 I2C_IOCTL_SET_SELF_ADDR

This **DeviceIoControl** request initializes the I2C device with the given address.

Parameters

lpInBuffer	Pointer to the expected I2C device address. The valid range of addresses is [0x00, 0x7F].
nInBufferSize	Size in bytes of the I2C device address.
Remarks	The device will be expected to respond when any master on the I2C bus wishes to proceed with any transfer. Note that this IOCTL will have no effect if the I2C device is in Master mode.

16.6.1.10 I2C_IOCTL_SET_SLAVE_MODE

This **DeviceIoControl** request sets the I2C device to Slave mode.

16.6.1.11 I2C_IOCTL_TRANSFER

This **DeviceIoControl** request performs the transfer (read or write) of one or more packets of data to a target device. An I2C_TRANSFER_BLOCK object is expected, which contains an array of I2C_PACKET objects to be executed sequentially. All of the required information should be stored in the I2C_TRANSFER_BLOCK passed in the *lpInBuffer* field.

Parameters

lpInBuffer	Pointer to an I2C_TRANSFER_BLOCK structure containing a pointer to an array of I2C_PACKET objects specifying all of the information required to perform the requested Read and Write operations.
nInBufferSize	Size in bytes of the I2C_TRANSFER_BLOCK.

16.6.2 I2C Driver Macros

16.6.2.1 I2C_MACRO_GET_CLOCK_RATE

This macro will retrieve the clock rate divisor.

```
I2C_MACRO_GET_CLOCK_RATE(
  HANDLE hDev,
  WORD wClkRate
);
```

Parameters

hDev	The I2C device handle retrieved from CreateFile().
wClkRate	Contains the divisor index. Refer to I2C information in *MCIMX31 and MCIMX31L Applications Processors Reference Manual* to obtain the true clock frequency.
Return Values	Returns TRUE or FALSE. If the result is TRUE, the operation is successful.
Remarks	Note that the value is not the absolute peripheral clock frequency. The value retrieved should be compared against the I2C specification to obtain the true frequency.

16.6.2.2 I2C_MACRO_GET_SELF_ADDR

This macro will retrieve the current I2C device address. Note that this macro is only meaningful if it is currently in Slave mode.

```
I2C_MACRO_GET_SELF_ADDR(
  HANDLE hDev,
  WORD bySelfAddr
);
```

Parameters

hDev	The I2C device handle retrieved from CreateFile().
dwSelfAddr	The current I2C device address. The valid range of address is [0x00, 0x7F].
Return Values	Returns TRUE or FALSE. If the result is TRUE, the operation is successful.

16.6.2.3 I2C_MACRO_IS_MASTER

This macro determines whether the I2C is currently in Master mode.

```
I2C_MACRO_IS_MASTER(
  HANDLE hDev,
  BOOL bIsMaster
);
```

Parameters

hDev	The I2C device handle retrieved from CreateFile().
bIsMaster	TRUE if the I2C device is in Master mode.
Return Values	Returns TRUE or FALSE. If the result is TRUE, the operation is successful.

16.6.2.4 I2C_MACRO_IS_SLAVE

This macro determines whether the I2C is currently in Slave mode.

```
I2C_MACRO_IS_SLAVE(
  HANDLE hDev,
  BOOL bIsSlave
);
```

Parameters

hDev	The I2C device handle retrieved from CreateFile().
bIsSlave	TRUE if the I2C device is in Slave mode.
Return Values	Returns TRUE or FALSE. If the result is TRUE, the operation is successful.

16.6.2.5 I2C_MACRO_RESET

This macro perform a hardware reset. Note that the I2C driver will still maintain all of the current information of the device, including the initialized addresses.

```
I2C_MACRO_RESET(
  HANDLE hDev,
);
```

Parameters

hDev	The I2C device handle retrieved from CreateFile().
Return Values	Returns TRUE or FALSE. If the result is TRUE, the operation is successful.

16.6.2.6 I2C_MACRO_SET_CLOCK_RATE

This macro will initialize the I2C device with the given clock rate.

```
I2C_MACRO_SET_CLOCK_RATE(
  HANDLE hDev,
  WORD wClkRate
);
```

Parameters

hDev	The I2C device handle retrieved from CreateFile().
wClkRate	Contains the divisor index. Refer to I2C specification to obtain the true clock frequency.
Return Values	Returns TRUE or FALSE. If the result is TRUE, the operation is successful.
Remarks	Note that this macro does not expect to receive the absolute peripheral clock frequency. Rather, it will be expecting the clock rate divisor index stated in the I2C specification. If absolute clock frequency must be used, please use the macro I2C_MACRO_SET_FREQUENCY.

16.6.2.7 I2C_MACRO_SET_FREQUENCY

This macro will estimate the nearest clock rate acceptable for I2C device and initialize the I2C device to use the estimated clock rate divisor. If the estimated clock rate divisor index is required, please refer to the macro I2C_MACRO_GET_CLOCK_RATE to determine the estimated index.

```
I2C_MACRO_SET_FREQUENCY (
  HANDLE hDev,
  DWORD dwFreq
);
```

Parameters

hDev	The I2C device handle retrieved from CreateFile().
dwFreq	The desired frequency.
Return Values	Returns TRUE or FALSE. If the result is TRUE, the operation is successful.

16.6.2.8 I2C_MACRO_SET_MASTER_MODE

This macro set the I2C device to Master mode.

```
I2C_MACRO_SET_MASTER_MODE(
  HANDLE hDev
);
```

Parameters

hDev	The I2C device handle retrieved from CreateFile().
Return Values	Returns TRUE or FALSE. If the result is TRUE, the operation is successful.

16.6.2.9 I2C_MACRO_SET_SELF_ADDR

This macro initializes the I2C device with the given address.

```
I2C_MACRO_SET_SELF_ADDR(
  HANDLE hDev,
  BYTE bySelfAddr
);
```

Parameters

hDev	The I2C device handle retrieved from CreateFile().
bySelfAddr	The expected I2C device address. The valid range for the address is [0x00, 0x7F].
Return Values	Returns TRUE or FALSE. If the result is TRUE, the operation is successful.
Remarks	The device will be expected to respond when any master on the I2C bus wishes to proceed with any transfer. Note that this macro will have no effect if the I2C device is in Master mode.

16.6.2.10 I2C_MACRO_SET_SLAVE_MODE

This macro set the I2C device to Slave mode.

```
I2C_MACRO_SET_SLAVE_MODE(
  HANDLE hDev
);
```

Parameters

hDev	The I2C device handle retrieved from CreateFile().
Return Values	Returns TRUE or FALSE. If the result is TRUE, the operation is successful.

16.6.2.11 I2C_MACRO_TRANSFER

This macro performs a sequence of data transfers to a target device. All of the required information should be stored in the I2C_TRANSFER_BLOCK object passed in the *pI2CTransferBlock* field.

```
I2C_MACRO_TRANSMIT(
  HANDLE hDev,
  PI2C_TRANSFER_BLOCK pI2CTransferBlock
);
```

Parameters

hDev	The I2C device handle retrieved from CreateFile().
pI2CTransferBlock	
pI2CPackets	[in] Pointer to an array of packets to be transferred sequentially.
iNumPackets	[in] The number of packets pointed to by *pI2CPackets* (the number of packets to be transferred).
Return Values	Returns TRUE or FALSE. If the result is TRUE, the operation is successful.

16.6.3 I2C Driver Structures

16.6.3.1 I2C_PACKET

This structure contains the information needed to write or read data using an I2C port.

```
typedef struct {
  BYTE byAddr;
  BYTE byRW;
  PBYTE pbyBuf;
  WORD wLen;
  LPINT lpiResult;
} I2C_PACKET, *PI2C_PACKET;
```

Members

byAddr This 7-bit slave address specifies the target I2C device to or from which data will be read or written.

byRW Determines whether the packet is a read or a write packet. Set to I2C_RW_READ for reading and I2C_RW_WRITE for writing.

pbyBuf A pointer to a buffer of bytes. For a Read operation, this is the buffer into which data will be read. For a Write operation, this buffer contains the data to write to the target device.

wLen If the operation is a Read, *wLen* specifies the number of bytes to read into *pbyBuf*. If the operation is a Write, *wLen* specifies the number of bytes to write from *pbyBuf*.

lpiResult Pointer to an int that contains the return code from the transfer operation.

16.6.3.2 I2C_TRANSFER_BLOCK

This structure contains an array of packets to be transferred using an I2C port.

```
typedef struct {
  I2C_PACKET *pI2CPackets;
  INT32 iNumPackets;
} I2C_TRANSFER_BLOCK, *PI2C_TRANSFER_BLOCK;
```

Members

pI2CPackets A pointer to an array of I2C_PACKET objects.

iNumPackets The number of I2C_PACKET objects pointed to by *pI2CPackets*.

Chapter 17
Keypad Driver

The Keypad Port (KPP) module is used to scan a keypad matrix. This module can detect, debounce, and decode one key on the keypad, or two keys pressed simultaneously. The keypad driver converts input from the KPP into keyboard events that the driver enters into the input system.

17.1 Keypad Driver Summary

Table 17-1 identifies the source code location, library dependencies, and other BSP information.

Table 17-1. Keypad Driver Summary

Driver Attribute	Definition
Target Platform (TGTPLAT)	3DS
Target SOC (TGTSOC)	MX31
MXARM11 CSP Driver Path	..\CSP\ARM\FREESCALE\MXARM11\DRIVERS\KEYBD
CSP Driver Path	N/A
CSP Static Library	mxarm11_Keypad.lib mxarm11_PddList.lib
Platform Driver Path	..\PLATFORM\\<*tgtplat*>\SRC\DRIVERS\KEYBD
Import Library	N/A
Driver DLL	Kbdmouse.dll
Catalog Item	Third Party > BSPs > Freescale <*tgtplat*> > Device Drivers > Input Devices > Keypad
SYSGEN Dependency	N/A
BSP Environment Variables	BSP_KBDMOUSE_EVBKPD=1

17.2 Supported Functionality

The keypad driver enables the 3-Stack board to provide the following software and hardware support:

- Conforms to the Microsoft Layout Manager Interface
- Supports multiple simultaneous key presses
- Supports two power management modes, full on and full off

17.3 Hardware Operation

Refer to the chapter on the KPP in the *MCIMX31 and MCIMX31L Applications Processors Reference Manual* for detailed operation and programming information.

17.3.1 Keypad Driver

The keypad driver interfaces with the Windows CE Keyboard Driver Architecture to provide key input support. The nine-key keypad is located on the Personality board.

Table 17-2 identifies the keypad mapping.

Table 17-2. Keypad Mapping

LABEL	Key value
S7(up)	UP
S8(down)	DOWN
S10(left)	LEFT
S9(right)	RIGHT
S11(enter)	ENTER
S12(menu 1)	ALT
S15(menu 2)	TAB
S16(menu 3)	SPACE
S17(menu 4)	ESC

The ALT key provides the user with greater ability to navigate Windows CE. The following key combinations use the ALT key to provide shortcuts (Table 17-3):

Table 17-3. Shortcut Keys

Press	To
ALT + TAB	Switch between open items.
ALT + underlined letter in a menu name	Display the corresponding menu.
ALT + Enter	Open the properties for the selected object.

17.3.2 Conflicts with other SoC Peripherals

No conflicts.

17.4 Software Operation

The keypad driver follows the Microsoft-recommended architecture for keyboard drivers. For details of this architecture and its operation, see the Platform Builder Help:

17.4.1 Keypad Scan Codes and Virtual Keys

Each key on the keypad has a unique scan code, which is added to a buffer whenever that key is pressed or released. These scan codes, which are hardware specific, are first converted to intermediate PS/2 keyboard scan code values and then converted into virtual keys, which are hardware independent numbers that identify the key. On the other hand, if a key is pressed from the keyboard, the generated scan code is directly converted into virtual keys. For alphabetic keys, the ASCII code for the capitalized letter is the virtual key. For other keys, the virtual key is defined by Microsoft and starts with “VK_”.

Table 17-4 identifies the mapping for scan code to virtual key.

Table 17-4. Mapping for Scan Code to Virtual Key

Key	Keypad Scan Code	Virtual Key
UP	0	VK_UP
DOWN	3	VK_DOWN
LEFT	4	VK_LEFT
RIGHT	1	VK_RIGHT
ENTER	7	VK_RETURN
ALT	2	VK_MENU
ESC	11	VK_ESCAPE
TAB	5	VK_TAB
SPACE	8	VK_SPACE

17.4.2 Power Management

The primary method for limiting power consumption in the Keypad module is to gate off all clocks to the module when those clocks are not needed. This is accomplished through the **DDKClockSetGatingMode** function call. In this module, the clocks are enabled only when it is required to access any Keypad register. Once done using the registers, the clocks are brought back to their previous state.

17.4.2.1 BSPKppPowerOn

This function is used to power up the keypad. This function will do the necessary configuration settings in the registers to bring up the keypad and then the clocks are brought back to their original state, as it was just before the module was powered down.

17.4.2.2 BSPKppPowerOff

This function powers down the keypad. Before turning off the module, the current state of the clock settings to this module are saved, and then the system waits for a period until the keypad does not report any further key-down/key-up event. Then, the clocks to this module are turned off.

17.4.2.3 IOCTL_POWER_CAPABILITIES

N/A

17.4.2.4 IOCTL_POWER_SET

N/A

17.4.2.5 IOCTL_POWER_GET

N/A

17.4.3 Keypad Registry Settings

The following registry keys are required to properly load the keypad device layout and input language.

```
[HKEY_LOCAL_MACHINE\HARDWARE\DEVICEMAP\KEYBD]
    "CalVKey"=dword:0
    "ContLessVKey"=dword:0
    "ContMoreVKey"=dword:0
    "TaskManVKey"=dword:2E
    "Keyboard Type"=dword:4
    "Keyboard SubType"=dword:0
    "Keyboard Function Keys"=dword:0
    "Keyboard Layout"="00000409"
    "DriverName"="kbdmouse.dll"

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Layouts\00000409]
    "Layout File"="kbdmouse.dll"
    "Layout Text"="US-Keypad"
    "KPPLayout"="kbdmouse.dll"

[HKEY_CURRENT_USER\Keyboard Layout\Preload\4]
    @="00000409"
```

17.5 Unit Test

With only nine keys, the keypad is not a full function keypad, and it will not pass the Keyboard Test included as part of the Windows CE 5.0 Test Kit (CETK). Therefore, a specific procedure is used to test all keys.

17.5.1 Unit Test Hardware

N/A

17.5.2 Unit Test Software

N/A

17.5.3 Building the Keyboard Tests

N/A

17.5.4 Running the Keyboard Tests

To test the keypad:

1. Run the application "Microsoft WordPad".
2. Input "Tab".
3. Input "Space".
4. Input "Alt" to open the menu bar.
5. Run the application "Internet Explorer".
6. Click the question mark in Internet Explorer to display Help.
7. Input the "ESC" to close Help.
8. Input the "Alt + Tab" to call the "Task Manager".
9. Quit the application "Microsoft WordPad". A popup dialog is displayed.
10. Click **Yes**.

17.6 Keypad Driver API Reference

For detailed reference information for the Keypad driver, see Platform Builder Help:

Developing a Device Driver > **Windows CE Drivers** > **Keyboard Drivers** > **Keyboard Driver Reference**

17.6.1 Keypad Mapping

Table 17-5 shows a mapping of Keyboard PDD functions to the functions used in the Keypad driver.

Table 17-5. Mapping between PDD Functions and Keypad Driver Functions

PDD Function Pointer	Keypad Driver Function
PFN_KEYBD_PDD_ENTRY	KPP_Entry
PFN_KEYBD_PDD_GET_KEYBD_EVENT	KeybdPdd_GetEventEx2
PFN_KEYBD_PDD_POWER_HANDLER	KPP_PowerHandler

Chapter 18
LAN9217 Product Ethernet Driver

The LAN9217 Product Ethernet driver provides connectivity with an IEEE 802.3 Ethernet using the SMSC LAN9217 Ethernet controller. The driver can communicate with the Ethernet at 10/100 megabits per second, as the LAN9217 Ethernet Controller is a 10Base-T/100 Base-TX Ethernet controller. The driver uses the LAN9217 internal MII compatible transceiver.

The LAN9217 Product Ethernet driver is a NDIS 5.0 compliant miniport driver.

18.1 LAN9217 Product Ethernet Driver Summary

Table 18-1 identifies the source code location, library dependencies, and other BSP information.

Table 18-1. LAN9217 Product Ethernet Driver Summary

Driver Attribute	Definition
Target Platform (TGTPLAT)	3DS
Target SOC (TGTSOC)	MX31
MXARM11 CSP Driver Path	N/A
<*tgtplat*> CSP Driver Path	N/A
CSP Static Library	N/A
Platform Driver Path	..\PLATFORM\<*tgtplat*>\SRC\DRIVERS\LAN9217
Import Library	ndis.lib
Driver DLL	lan9217.dll
Catalog Item	Catalog > Third Party > BSPs > Freescale <*tgtplat*>: ARMV4I > Device Drivers > LAN9217
SYSGEN Dependency	SYSGEN_NDIS=1 SYSGEN_TCPIP=1 SYSGEN_WINSOCK=1
BSP Environment Variables	BSP_ETHER_LAN9217=1

18.2 Supported Functionality

The LAN9217 Product Ethernet driver enables the 3-Stack board to provide the following software and hardware support:

- Conforms to the Microsoft Network Driver Interface Specification (NDIS) architecture in Windows CE
- Supports IEEE 802.3 Ethernet protocols for communication
- Supports two power management modes, full on and full off.

18.3 Hardware Operation

The LAN9217 chip is an on-board peripheral that is connected to the processor through the PBC (Peripheral Bus Controller). Refer to the Peripheral Bus Controller CPLD document and LAN9217 data sheet for detailed operation and programming information.

18.3.1 Conflicts with other SoC Peripherals

No conflicts. (Refer to Peripheral Bus Controller CPLD document for details).

18.4 Software Operation

The Product Ethernet driver follows the Microsoft-recommended architecture for NDIS miniport drivers. For details of this architecture and its operation, see the Platform Builder Help:

Developing a Device Driver > **Windows CE Drivers** > **Network Drivers** > **Network Driver Development Concepts** > **Miniports, Intermediate Drivers, and Protocol Drivers**

18.4.1 Power Management

The power management is currently not implemented for the LAN9217 Product Ethernet driver.

18.4.2 Product Ethernet Registry Settings

The following registry keys are required to properly load the LAN9217 Product Ethernet driver and to configure the TCP/IP for Ethernet interface. In the following example, the dynamic IP address is assigned using DHCP; the variable `EnableDHCP` should be set to 1.

```
[HKEY_LOCAL_MACHINE\Comm\lan9217]
    "DisplayName"="lan9217 Ethernet Driver"
    "Group"="NDIS"
    "ImagePath"="lan9217.dll"

[HKEY_LOCAL_MACHINE\Comm\lan9217\Linkage]
    "Route"=multi_sz:"lan9217"

[HKEY_LOCAL_MACHINE\Comm\lan9217]
    "DisplayName"="lan9217 Ethernet Driver"
    "Group"="NDIS"
    "ImagePath"="lan9217.dll"
```

```
[HKEY_LOCAL_MACHINE\Comm\lan9217\Parms]
    "BusNumber"=dword:0
    "BusType"=dword:0
    "InterruptNumber"=dword:1; pio interrupt
    "IoBaseAddress"=dword:B6000000 ; ETHERNET_BASE (Physical Addr)
    "PhyAddress"=dword:FF         ; PHY address (0x20:Auto, 0xFF:Internal)
    "RxDMAMode"=dword:0 ; 1-DMA, 0-PIO
    "TxDMAMode"=dword:0 ; 1-DMA, 0-PIO
    "FlowControl"=dword:1     ; 1-Enabled, 0-Disabled

    ; LinkMode will replace Duplex, Speed and FlowControl
    ; bit7: RESERVED, bit6: ANEG, bit5: ASymmetric Pause, bit4: Symmetric Pause
    ; bit3: 100FD, bit2: 100HD, bit1: 10FD, bit0: 10HD
    "LinkMode"=dword:7F

[HKEY_LOCAL_MACHINE\Comm\lan9217\Parms\TcpIp]
    "EnableDHCP"=dword:1
    "IpAddress"="0.0.0.0"
    "Subnetmask"="0.0.0.0"
    "DefaultGateway"="0.0.0.0"
    "UseZeroBroadcast"=dword:0
```

18.5 Unit Test

Test the LAN9217 Product Ethernet driver using the following:

1. Network utilities/operations: Ping to and from the <*TGTSOC*> device, perform FTP transfers (file put and get) with the <*TGTSOC*> device as FTP server, and perform Internet browsing with Pocket Internet Explorer on the <*TGTSOC*> device.
2. Winsock CETK test cases: Winsock 2.0 Test (v4/v6), and Winsock Performance Test with <*TGTSOC*> device as client.

18.5.1 Unit Test Hardware

Table 18-2 lists the required hardware to run the unit tests.

Table 18-2. Hardware Requirements for LAN9217 Product Ethernet Driver

Requirements	Description
<*TGTSOC*> board with LAN9217	Board that hosts the LAN9217 Product Ethernet driver
PC/machine	To act as counterpart for network operation
An Ethernet or a cross Ethernet cable	To form an Ethernet

18.5.2 Unit Test Software

Table 18-3 lists the required software to run the unit tests.

Table 18-3. Software Requirements to Run LAN9217 Product Ethernet Driver Unit Tests

Requirements	Description
Tux.exe	Tux test harness, which is needed for executing the test
Kato.dll	Kato logging engine, which is required for logging test data
Tooltalk.dll	Library required by Tux.exe and Kato.dll. Handles the transport between the target device and the development workstation
Ws2bvt.dll	Test .dll file for Winsock 2.0 Test (v4/v6).
Perflog.dll	Module that contains functions that monitor and log performance for Winsock Performance Test.
Perf_winsock2.dll	Test .dll file for Winsock Performance Test.
Perf_winsockd2.exe	Test .exe file (server program) for Winsock Performance Test.
Ndt.dll	Protocol driver for One-card network card miniport driver test
Ndt_1c.dll	Test .dll for One-card network card miniport driver test
Ndp.dll	MS_NDP protocol driver for NDIS performance test
Perf_ndis.dll	Test .dll file NDIS performance test

18.5.3 Building the LAN9217 Product Ethernet Tests

18.5.3.1 Network Utilities Related Tests

Enable the following registry entries to allow get/put of files using the anonymous FTP upload, and to be able to access all the files and folders under the root directory on the target:

```
[HKEY_LOCAL_MACHINE\COMM\FTPD]
        "AllowAnonymousUpload" = dword:1
        "DefaultDir" = "\\"
```

For minimum network support and ping, enable the following components in the OS design:

Under Mobile Handheld[Windows CE devices] > Communication Services and Networking > Networking Features:
Network Driver Architecture (NDIS)
TCP/IP
TCP/IP > IP Helper API
Winsock Support
Network Utilities (IpConfig, Ping, Route)

For FTP, enable the following components in the OS design:
Mobile Handheld[Windows CE devices] > Communication Services and Networking > Servers:
FTP Server

For Video streaming, enable the following components in the OS design:
Mobile Handheld[Windows CE devices] > Graphics and Multimedia Technologies > Media:
Media Formats > AVI Filter
Streaming Media Playback
Video Codecs and Renderers > Video/Image Compression Manager
Video Codecs and Renderers > Overlay Mixer
Video Codecs and Renderers > WMV/MPEG-4 Video Codec
Windows Media Player > Windows Media Player
Windows Media Player > Windows Media Player OCX
Windows Media Player > Windows Media Technologies
Windows Media Player > Windows Media Technologies > Windows Media Multicast and Multi-Bit Rate
Windows Media Player > Windows Media Technologies > Windows Media Streaming from Local Storage
Windows Media Player > Windows Media Technologies > Windows Media Streaming over HTTP

It will be helpful to add the command line shell and console support:
Shell and User Interface > Shell > Command Shell:
Command Processor
Command Window

18.5.3.2 Winsock 2.0 Test (v4/v6)

The Winsock 2.0 Test (v4/v6) comes pre-built as part of the CETK. No steps are required to build these tests. The `Ws2bvt.dll` is with the other required CETK files in the following location:

```
[Drive]:\Program Files\Windows CE Platform Builder\5.00\cepb\wcetk\ddtk\armv4I
```

18.5.3.3 Winsock Performance Test

The Winsock Performance Test comes pre-built as part of the CETK. No steps are required to build these tests. The `Perf_winsock2.dll` and `Perf_winsockd2.exe` is with the other required CETK files in the following locations:

`Perf_winsock2.dll` is located in:

```
[Drive]:\Program Files\Windows CE Platform Builder\5.00\cepb\wcetk\ddtk\armv4I
```

`Perf_winsockd2.exe` is located in:

```
[Drive]:\Program Files\Windows CE Platform Builder\5.00\cepb\wcetk\ddtk\desktop
```

18.5.3.4 One-Card Network Card Miniport Driver Test

The one-card network card miniport driver test comes pre-built as part of the CETK. No steps are required to build these tests. The `ndt.dll` and `ndt_1c.dll` are with the other required CETK files in the following location:

```
[Drive]:\Program Files\Windows CE Platform Builder\5.00\cepb\wcetk\ddtk\armv4I
```

18.5.3.5 NDIS Performance Test

The NDIS performance test comes pre-built as part of the CETK. No steps are required to build these tests. The `ndp.dll` and `perf_ndis.dll` are with the other required CETK files in the following location:

```
[Drive]:\Program Files\Windows CE Platform Builder\5.00\cepb\wcetk\ddtk\armv4I
```

18.5.4 Running the LAN9217 Product Ethernet Tests

18.5.4.1 Network Utilities Related Tests

18.5.4.1.1 Ping Tests

Run the ping tests from the <*TGTSOC*> device and from the PC.

18.5.4.1.2 Browsing

Perform the network browsing tests after setting the following on the device front panel:

- DNS servers in the TCP/IP properties of LAN9217 network interface (Control Panel > Network and Dial-up Connections)
- Proxy server, if used in the network used for test, on the Pocket Internet explorer.

18.5.4.1.3 FTP Tests

To run the FTP tests, start the FTP service on the <*TGTSOC*> device using the following command at the DOS prompt:

```
services start FTP0:
```

18.5.4.2 Video Streaming Tests

Access the web sites that provide video clips, such as: http://www.smartvideo.com. The set-up for Internet browsing (as mentioned above) is mandatory.

18.5.4.3 Winsock 2.0 Test (v4/v6)

Execute the test on the <*TGTSOC*> device using the following command in the command line on the <*TGTSOC*>.

```
tux –o –d Ws2bvt.dl
```

For detailed information on the Winsock 2.0 Test (v4/v6) tests, see the Platform Builder Help:

Debugging and Testing > **Tools for Debugging and Testing** > **Windows CE Test Kit** > **CETK Tests** > **Winsock 2.0 Test (v4/v6)**

18.5.4.4 Winsock Performance Test

Start the server on the PC by entering "`Perf_winsockd2 - install`" at the command line. `Perf_winsockd2` is located in the `<Platform Builder installation path>\Cepb\Wcetk\Ddtk\Desktop` folder.

Then execute the client side test on the second device using `tux –o –d Perf_winsock2.dll –c "-s 10.185.60.56"` in the command line on the <*TGTSOC*>, where `10.185.60.56` represents an IP address for the PC. Enter an appropriate IP address.

For detailed information on the Winsock Performance tests, see the Platform Builder Help:

Debugging and Testing > **Tools for Debugging and Testing** > **Windows CE Test Kit** > **CETK Tests** > **Winsock Performance Test**

18.5.4.5 One-Card Network Card Miniport Driver Test

To perform this test, include `ndt.dll` and `ndt_1c.dll` in the image, and starting the test by entering `tux -o -d ndt_1c.dll -c "-t LAN9217"` on the command line on the <*TGTSOC*>.

For detailed information on the Winsock Performance tests, see the Platform Builder Help:

Debugging and Testing > **Tools for Debugging and Testing** > **Windows CE Test Kit** > **CETK Tests** > **One-card Network Card Miniport Driver Test**

18.5.4.6 NDIS Performance Test

To perform this test, include `ndp.dll` and `perf_ndis.dll` in the image, and enter `tux -o -d perf_ndis.dll -c "LAN9217"` on the command line on the <*TGTSOC*>.

For detailed information on the Winsock Performance tests, see the Platform Builder Help:

Debugging and Testing > **Tools for Debugging and Testing** > **Windows CE Test Kit** > **CETK Tests** > **NDIS Performance Test**

18.6 LAN9217 Product Ethernet Driver API Reference

The LAN9217 Product Ethernet driver conforms to NDIS 5.0 specification by Microsoft for the miniport network drivers.

Chapter 19
MBX Direct3D Mobile/OpenGL ES Drivers

The MBX Lite graphics processor is an IP wrapper for 2D/3D hardware acceleration, and is designed for ultra-low-power cost-sensitive system-on-chip (SoC) applications, such as mainstream mobile phones, PDAs, and handheld gaming devices.

The MBX D3DM and OpenGL ES drivers interface with the i.MX31 Image Processing Unit (IPU) Synchronous Display Controller (SDC) to combine graphics and video planes, and to generate display controls with programmable timing. This module is compatible with the Epson L4F00242T03 panel.

The MBX Direct3D Mobile (D3DM) driver provides the actual drawing services that Microsoft Direct3D Mobile middleware uses. The middleware is a thin layer of software that handles call transport, synchronization, and OS integration issues; the driver manages the memory for display surfaces.

OpenGL ES is a royalty-free, cross-platform API for full-function 2D and 3D graphics on embedded systems. OpenGL ES 1.X is for fixed function hardware and offers acceleration, image quality, and performance.

The i.MX31 MBX supports the D3DM and OpenGL ES 1.1 in Windows CE 5.0.

19.1 Direct3D Mobile/OpenGL ES Drivers Summary

Table 19-1 identifies the source code location, library dependencies, and other BSP information.

Table 19-1. Direct3D Mobile/OpenGL ES Drivers Summary

Driver Attribute	Definition
Target Platform (TGTPLAT)	3DS
Target SOC (TGTSOC)	MX31
MXARM11 CSP Driver Path	N/A
CSP Driver Path	N/A
CSP Static Library	N/A
Platform Driver Path	..\PLATFORM\<*tgtplat*>\SRC\DRIVERS\MBX
Import Library	ddgpe.lib, gpe.lib
Driver DLL	Clcdckmif.dll, sdc_display.dll, ddi_powervr.dll, gxdma.dll, libGLES_CM.dll, libpvrWCEWSEGL.dll, IMGEGL.dll, pvr_d3dm.dll, pvr_kernel.dll, um3partyif.dll

Table 19-1. Direct3D Mobile/OpenGL ES Drivers Summary

Driver Attribute	Definition
Catalog Item for 3-Stack	Third Party > BSPs Freescale <tgtplat> > Device Drivers > Epson L4F00242T03 Third Party > BSPs Freescale <tgtplat> > Device Drivers > MBX MBX i.MX31 Base Driver Third Party > BSPs Freescale <tgtplat> > Device Drivers > MBX MBX i.MX31 D3DM Core Third Party > BSPs Freescale <tgtplat> > Device Drivers > MBX MBX i.MX31 Ogles Core
SYSGEN Dependency	SYSGEN_DDRAW=1 SYSGEN_D3DM=1
BSP Environment Variables for 3-Stack	BSP_MBX BSP_DISPLAY_EPSON_L4F00242T03 = 1 for epson LCD Panel

19.2 Supported Functionality

The MBX D3DM and OpenGL ES drivers enable the 3-Stack board to provide the following software and hardware support:

- Drivers derive from the DirectDraw Graphics Primitive Engine (DDGPE) class
- Drivers support the DirectDraw Hardware Abstraction Layer (DDHAL),support overlay surface for pixel format 565,UYVY,YV12,Overlay surface color key feature
- Driver support TVout
- Drivers support the Epson L4F00242T03 panels
- Direct3D Mobile supports Microsoft Direct3D Mobile Specification
- OpenGL ES supports OpenGL ES 1.1 Specification
- MBX driver use VFP for hardware accelerate

19.3 Hardware Operation

Refer to the chapter on the MBX in the *MCIMX31 and MCIMX31L Applications Processors Reference Manual* for detailed operation and programming information.

19.3.1 Conflicts with Other SoC Peripherals

MBX does not have conflicts with any other module.

19.4 Software Operation

The MBX D3DM driver follows the Microsoft-recommended architecture for Direct3D Mobile drivers. For details of this architecture and its operation, see the Platform Builder Help:

Developing a Device Driver > **Windows CE Drivers** > **Direct3D Mobile Display Drivers**

The MBX driver uses a standalone DirectDraw driver. If MBX is included in the OS image, the IPU DirectDraw driver will be replaced by the MBX DirectDraw driver. If VFP accelerate is used for MBX, a mathematical operation may fail.

19.4.1 Communicating with the MBX

Communications with the MBX drivers are accomplished through Microsoft-defined APIs or OpenGL ES APIs. The MBX Direct3D Mobile driver uses the local hooking model. The application's process space includes both the Microsoft® Direct3D® Mobile middleware (`d3dm.dll`) and the MBX Direct3D Mobile driver.

Because the locally hooked drivers are not loaded into the Graphics, Windowing, and Event Subsystem (GWES), they do not have direct access to the hardware. As a result, these drivers must rely on some other graphics technology, such as DirectDraw, to present rendered output to the user.

19.4.2 Configuring the LCD Display Panels

The display is configured based on the **PanelType** registry key, which is described in the **DisplayRegistry Settings** section below. The **PanelType** registry key indicates the display panel that is being used. The supported display panel is the Epson L4F00242T03 LCD panel.

19.4.2.1 LCD Display Registry Settings

The following registry keys are optionally included, depending on the display panel catalog item included in the OS design. If the Epson L4F00242T03 VGA panel is selected, the following registry keys are included:

```
[HKEY_LOCAL_MACHINE\Drivers\Display\DDIPU_SDC]
"Bpp"=dword:10 ; 16bpp
"PanelType"=dword:1 ; Epson VGA Panel
"VideoMemSize"=dword:350000 ; 3.5MB "Order"=dword:9
```

19.4.3 TV Output Mode

Application `tvout.exe` provides the user a way to change between LCD and TV Output mode (PAL standard). Clicking the TV-Out application toggles between these two modes. The default path of `tvout.exe` is `\windows`.

The TV-Out application sets the TV output mode to PAL or NTSC standards, depending on the received parameter. “tvout.exe 0” sets the TV output mode to PAL, “tvout.exe 1” sets it to NTSC.

The display driver ensures that the output is LCD mode, when responding to the power management to enter the power states D0 (Full On) and D4 (Off).

The TV output mode requires a 270 degree rotation, to enable the primary surface to fit the physical resolution 640x480 that TV can support.

NOTE

Due to lack of support for the co-existence of GDI screen rotation and DirectDraw (see the Windows CE 5.0 Help, stating that “GDI screen rotation cannot be used with DirectDraw”), a DirectDraw display driver with rotation support enabled may yield more failures in the GDI/DIRECTDRAW CETK test suite. It is recommended to run these CETK tests with rotation support disabled or under 0 rotation degree.

19.4.4 Power Management

The D3DM and OpenGL ES drivers consumes power primarily through the operation of various MBX and IPU sub-modules, such as the SDC, which combines and displays video and graphics data, and through the operation of the display panel. To facilitate management of these modules, the 3D driver implements the power management I/O Control (IOCTL) codes like IOCTL_POWER_CAPABILITIES, IOCTL_POWER_QUERY, IOCTL_POWER_GET and IOCTL_POWER_SET.

19.4.4.1 PowerUp

This function is not implemented for the display driver.

19.4.4.2 PowerDown

This function is not implemented for the display driver.

19.4.4.3 IOCTL_POWER_SET

The display driver implements the IOCTL_POWER_SET IOCTL API with support for the D0 (Full On) and D4 (Off) power states. These states are handled in the following manner:

- D0 – The display panel is enabled. The IPU’s Display Interface (DI) and SDC modules are enabled.
- D4 – The DI and SDC modules of the IPU are disabled. The display panel is disabled.

19.4.4.4 Direct3D Mobile and OpenGL ES Registry Settings

The following registry keys are required to properly load the MBX D3DM and OpenGL ES drivers.

```
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\PVRKernel]
  "Prefix"="PKM"
  "Dll"="pvr_kernel.dll"
  "Order"=dword:10
  ;"Order"=dword:1
  "Keep"=dword:1
  "BM_POOL0_PHY_BASE"=dword:86800000
  ; Indicate PKM is a generic power manageable interface
  "IClass"="{A32942B7-920C-486b-B0E6-92A702A99B35}"
```

```
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\PVR3rdPartyKernel]
  "Prefix"="P3P"
  "Dll"="clcdckmif.dll"
  "Order"=dword:10
  ;"Order"=dword:2
  "Keep"=dword:1
  "CLCDC_MEM_BASE"=dword:87300000
  ; Indicate P3P is a generic power manageable interface
  "IClass"="{A32942B7-920C-486b-B0E6-92A702A99B35}"

[HKEY_CURRENT_USER\ControlPanel\Keybd]
"Contrast"=dword:80

[HKEY_LOCAL_MACHINE\SYSTEM\GWE]
   ; "PORepaint"=dword:0 - the display driver handles everything
   ; "PORepaint"=dword:1 - gwe should save and restore the bits
   ; "PORepaint"=dword:2 - gwe should invalidate and repaint
   ; "PORepaint"=dword:3 - gwe and driver need to save video memory
   ;
   "PORepaint"=dword:3

[HKEY_LOCAL_MACHINE\System\GDI\Drivers]
   "Display"="ddi_powervr.dll"

[HKEY_LOCAL_MACHINE\Drivers\Display\PowerVR]
  "IsrDll"="GIISR.DLL"
  "IsrHandler"="ISRHandler"

; mode list is truncated to just one entry with the following settings
  "OutputMask"=dword:003f3f3f

; Screen rotation control
"DisableDynamicScreenRotation"=dword:0

; Framebuffer allocation stratagy based on the allocation flags
; GPE_REQUIRE_VIDEO_MEMORY and GPE_PREFER_VIDEO_MEMORY
;
;"FBAllocStrategy"=dword:21

[HKEY_LOCAL_MACHINE\Drivers\Display\PowerVR]
```

```
"HWRecoveryTimeout"="350"

[HKEY_LOCAL_MACHINE\Drivers\Display\PowerVR\MBX1\Game Settings\OpenGLES]
[HKEY_LOCAL_MACHINE\Drivers\Display\PowerVR\MBX1\Game Settings\D3DM]
[HKEY_LOCAL_MACHINE\PowerVR\MBX1\Game Settings\OpenGLES]
[HKEY_LOCAL_MACHINE\PowerVR\MBX1\Game Settings\D3DM]

[HKEY_LOCAL_MACHINE\system\gdi\rotation]
  "Angle"=dword:0

[HKEY_LOCAL_MACHINE\System\D3DM\Drivers]
  "LocalHook"="pvr_d3dm.dll"
```

19.5 Unit Test

To add all MBX-related drivers on Windows CE to the image, including D3DM, OpenGL ES, DirectDraw, LCD, and IPU SDC, add the following three MBX modules from the catalog:

- MBX MX31 Base driver
- MBX MX31 D3DM Core
- MBX MX31 Ogles Core

The sections below focus on the D3DM Windows CETK test and the D3DM/OpenGL ES demos test. For further information about the DirectDraw/GDI CETK and Windows Media Player tests, include for overlay surface and primary surface test, see the documentation section for the Display Driver.

19.5.1 Unit Test Hardware

The Epson L4F00242T03 VGA Panel is needed to run the unit tests. The panel displays the graphics data.

19.5.2 Unit Test Software

19.5.2.1 Direct3D Mobile Interface Tests

Table 19-2 lists the required software to run the D3DM Interface tests.

Table 19-2. Software Requirements to Run the D3DM Interface Tests

Requirements	Description
Tux.exe	Tux test harness, which is needed for executing the test
Kato.dll	Kato logging engine, which is required for logging test data
Tooltalk.dll	Library required by Tux.exe and Kato.dll. Handles the transport between the target device and the development workstation
D3DM_Interface.dll	Dynamic-link library for the test.

To run the Direct3D Mobile Interface Test:

1. In your OS design, set the SYSGEN_D3DM variable from: **Catalog Core OS** > **Windows CE** > **Devices**> **Graphics**> **Direct3D Mobile**.
2. Include a Direct3D Mobile driver in your OS design.

19.5.2.2 Direct3D Mobile Driver Verification Tests

Table 19-3 lists the required software to run the D3DM Verification tests.

Table 19-3. Software Requirements to Run the D3DM Verification Tests

Requirements	Description
Tux.exe	Tux test harness, which is needed for executing the test
Kato.dll	Kato logging engine, which is required for logging test data
Tooltalk.dll	Library required by Tux.exe and Kato.dll. Handles the transport between the target device and the development workstation
D3DM_DriverVerif.dll	Dynamic-link library for the test.
QAD3DMX.dll	Library that supplies functions to compute three-dimensional mathematical operations required by the test.

To run the Direct3D Mobile Driver Verification Test:

1. In your OS design, set the SYSGEN_D3DM variable from **Catalog Core OS** > **Windows CE** > **Devices**> **Graphics**> **Direct3D Mobile**, and the SYSGEN_D3DMREF variable from **Catalog Device Drivers** > **Direct3D Mobile**
2. Include a Direct3D Mobile driver in your OS design.

19.5.2.3 Direct3D Mobile Driver Comparison Tests

Table 19-4 lists the required software to run the D3DM Comparison tests.

Table 19-4. Software Requirements to Run the D3DM Comparison Tests

Requirements	Description
Tux.exe	Tux test harness, which is needed for executing the test
Kato.dll	Kato logging engine, which is required for logging test data
Tooltalk.dll	Library required by Tux.exe and Kato.dll. Handles the transport between the target device and the development workstation
D3DM_DriverComp.dll	Dynamic-link library for the test.
D3DMImageManagement.dll	Library that provides utilities for capturing, comparing, and storing rendered output.
QAD3DMX.dll	Library that supplies functions to compute three-dimensional mathematical operations required by the test

To run the Direct3D Mobile Driver Comparison Test:

1. In your OS design, set the SYSGEN_D3DM variable from **Catalog Core OS** > **Windows CE** > **Devices**> **Graphics**> **Direct3D Mobile**, and the SYSGEN_D3DMREF variable from **Catalog Device Drivers** > **Direct3D Mobile**> **Direct3D Mobile** > **Mobile Reference Driver**.
2. Include a Direct3D Mobile driver in your OS design.

19.5.3 Building the Direct3D Mobile Tests

The D3DM Interface Tests, D3DM Driver Verification Tests and D3DM Driver Comparison Tests come with the CETK. No steps are required to build these tests.

The following files are located with the other required CETK files:

`D3DM_Interface.dll`, `D3DM_DriverVerif.dll`, `D3DM_DriverComp.dll`, `3DMImageManagement.dll`, `QAD3DMX.dll`

These are located on the following location:

```
[Drive]:\Program Files\Windows CE Platform Builder\5.00\cepb\wcetk\ddtk\armv4I
```

19.5.4 Running the Direct3D Mobile Tests

19.5.4.1 Running the Direct3D Mobile Interface Tests

The command line for running the D3DM Interface tests is:

```
tux –o –d d3dm_interface.dll
```

For detailed information on the D3DM Interface tests and command line options, see the Platform Builder Help:

Tests > **Direct3D Mobile** > **Interface Test**> **Debugging and Testing Tools for Debugging and Testing Windows CE Test Kit CETK**

Table 19-5 lists the test cases in the D3DM Interface test suite.

Table 19-5. D3DM Interface Test Cases

Test Case	Description
1-99	Tests the methods for the IDirect3DMobile interface.
101-199	Tests the methods for the IDirect3DMobileDevice interface.
201-299	Tests the methods for the IDirect3DMobileIndexBuffer interface
301-399	Tests the methods for the IDirect3DMobileSurface interface
401-499	Tests the methods for the IDirect3DMobileTexture interface
501-599	Tests the methods for the IDirect3DMobileVertexBuffer interface.
2001-2099	Tests the security of a Direct3D Mobile driver

The test result for Interface test is 123 pass, 23 skipped. The skipped cases are: 104,123,143,144,169,205,206,207,405,406,407,409,410,418,419,420,422,423,424,505,506,507,2001.

19.5.4.2 Running the Direct3D Mobile Driver Verification Tests

The command line for running the D3DM Driver Verification tests is:

```
tux –o –d d3dm_DriverVerif.dll
```

For detailed information on the D3DM Interface tests and command line options, see the Platform Builder Help:

Tests > **Direct3D Mobile** > **Interface Test**> **Debugging and Testing** > **Tools for Debugging and Testing** > **Windows CE Test Kit CETK**

Table 19-6 describes the test cases contained in the D3DM Driver Verification test suite.

Table 19-6. D3DM Driver Verification Test Cases

Test Case	Description
1-199	Tests the IDirect3DMobileDevice::ProcessVertices method. These test cases verify transformation and lighting, primarily covering permutations of the following factors: D3DMRS_AMBIENT with various settings D3DMRS_AMBIENTMATERIALSOURCE: D3DMMCS_COLOR1, D3DMMCS_COLOR2, D3DMMCS_MATERIAL D3DMRS_COLORVERTEX: FALSE, TRUE D3DMRS_DIFFUSEMATERIALSOURCE: D3DMMCS_COLOR1, D3DMMCS_COLOR2, D3DMMCS_MATERIAL D3DMRS_LIGHTING: FALSE, TRUE D3DMRS_LOCALVIEWER: FALSE, TRUE D3DMRS_SPECULARENABLE: FALSE, TRUE D3DMRS_SPECULARMATERIALSOURCE: D3DMMCS_COLOR1, D3DMMCS_COLOR2, D3DMMCS_MATERIAL D3DMTRANSFORMSTATETYPE settings, including various D3DMMATRIX values for D3DMTS_VIEW, D3DMTS_WORLD, and D3DMTS_PROJECTION Various D3DMVIEWPORT settings for the IDirect3DMobileDevice::SetTransform method Various D3DMLIGHT settings for the IDirect3DMobileDevice::SetLight method
201-899	Tests blending. These test cases render scenes and verify output, primarily covering permutations of the following factors: D3DMRS_SRCBLEND and D3DMRS_DESTBLEND: D3DMBLEND_ZERO, 3DMBLEND_ONE D3DMBLEND_SRCCOLOR D3DMBLEND_INVSRCCOLOR D3DMBLEND_SRCALPHA D3DMBLEND_INVSRCALPHA D3DMBLEND_DESTALPHA D3DMBLEND_INVDESTALPHA D3DMBLEND_DESTCOLOR D3DMBLEND_INVDESTCOLOR D3DMBLEND_SRCALPHASAT D3DMRS_BLENDOP D3DMBLENDOP_ADD D3DMBLENDOP_SUBTRACT D3DMBLENDOP_REVSUBTRACT D3DMBLENDOP_MIN D3DMBLENDOP_MAX Flexible vertex format (FVF) component values, including various color values

Table 19-6. D3DM Driver Verification Test Cases(Continued)

Test Case	Description
900-1099	Tests stencils. These test cases render scenes and verify output, primarily covering permutations of the following factors: Various values of D3DMRS_STENCILMASK D3DMRS_STENCILFAIL and D3DMRS_STENCILPASS D3DMSTENCILOP_KEEP D3DMSTENCILOP_ZERO D3DMSTENCILOP_REPLACE D3DMSTENCILOP_INCRSAT D3DMSTENCILOP_DECRSAT D3DMSTENCILOP_INVERT D3DMSTENCILOP_INCR D3DMSTENCILOP_DECR D3DMRS_STENCILFUNC D3DMCMP_GREATER D3DMCMP_LESS Various values of D3DMRS_STENCILREF
1100-1199	Tests resource management. These test cases exercise the managed resource functionality of a driver. These test cases create resources until video memory constraints require that one or more resources be evicted. These test cases assess resources of the following types: D3DMRTYPE_INDEXBUFFER D3DMRTYPE_VERTEXBUFFER D3DMRTYPE_TEXTURE

The test result for verification test is 22 pass, 749 skipped. The passed cases are:
279,280,281,282,283,284,288,289,290,291,292,293,294,295,299,300,301,302,303,304,305,306,

19.5.4.3 Running the Direct3D Mobile Comparison Tests

The command line for running the D3DM Comparison tests is:

```
tux –o –d d3dm_DriverComp.dll
```

For detailed information on the D3DM Interface tests and command line options, see the Platform Builder Help:

Tests > **Direct3D Mobile** > **Interface Test**> **Debugging and Testing** > **Tools for Debugging and Testing** > **Windows CE Test Kit CETK**

Table 19-7 describes the test cases contained in the D3DM Driver Comparison test suite.

Table 19-7. D3DM Driver Comparison Test Cases

Test Case	Description
0-99	Tests culling. These test cases render scenes that primarily cover permutations of the following factors: D3DMRS_CULLMODE: D3DMCULL_CW, D3DMCULL_CCW, D3DMCULL_NONE Primitive orientations: Back facing, Front facing, Edge on
100-199	Tests lighting. These test cases render scenes that primarily cover permutations of the following factors: D3DMRS_AMBIENT, with a variety of settings D3DMRS_AMBIENTMATERIALSOURCE: D3DMMCS_COLOR1, D3DMMCS_COLOR2, D3DMMCS_MATERIAL D3DMRS_COLORVERTEX: FALSE, TRUE D3DMRS_DIFFUSEMATERIALSOURCE: D3DMMCS_COLOR1, D3DMMCS_COLOR2, D3DMMCS_MATERIAL D3DMRS_LIGHTING: FALSE, TRUE D3DMRS_LOCALVIEWER: FALSE, TRUE D3DMRS_SPECULARENABLE: FALSE, TRUE D3DMRS_SPECULARMATERIALSOURCE: D3DMMCS_COLOR1, D3DMMCS_COLOR2, D3DMMCS_MATERIAL Flexible vertex format (FVF) values including various positions, normals, and colors Various D3DMMATERIAL values for the IDirect3DMobileDevice::SetMaterial method Various D3DMLIGHT values for the IDirect3DMobileDevice::SetLight method Various D3DMMATRIX values for the IDirect3DMobileDevice::SetTransform method
200-299	Tests depth buffers. These test cases render scenes that primarily cover permutations of the following factors: Various D3DMCLEAR_ZBUFFER depth values for the IDirect3DMobileDevice::Clear method D3DMPRIMITIVETYPE: D3DMPT_POINTLIST, D3DMPT_LINELIST, D3DMPT_TRIANGLELIST D3DMRS_ZENABLE: D3DMZB_FALSE, D3DMZB_TRUE D3DMRS_ZFUNC: D3DMCMP_ALWAYS, D3DMCMP_NEVER, D3DMCMP_LESS, D3DMCMP_EQUAL, D3DMCMP_LESSEQUAL, D3DMCMP_GREATER, D3DMCMP_NOTEQUAL, D3DMCMP_GREATEREQUAL D3DMRS_ZWRITEENABLE: FALSE, TRUE Vertex positions to exercise various depths

Table 19-7. D3DM Driver Comparison Test Cases(Continued)

Test Case	Description
300-399	Tests primitive rasterization. These test cases render scenes that primarily cover permutations of the following factors: D3DMPRIMITIVETYPE: D3DMPT_POINTLIST, D3DMPT_LINELIST, D3DMPT_LINESTRIP, D3DMPT_TRIANGLELIST, D3DMPT_TRIANGLESTRIP, D3DMPT_TRIANGLEFAN D3DMRS_COLORWRITEENABLE: With and without D3DMCOLORWRITEENABLE_RED, With and without D3DMCOLORWRITEENABLE_GREEN, With and without D3DMCOLORWRITEENABLE_BLUE D3DMRS_FILLMODE: D3DMFILL_SOLID, D3DMFILL_POINT, D3DMFILL_WIREFRAME FVF number format: D3DMFMT_D3DMVALUE_FIXED, D3DMFMT_D3DMVALUE_FLOAT Primitive drawing functions: DrawPrimitive, DrawIndexedPrimitive Primitive drawing range: Various StartVertex and PrimitiveCount values for the DrawPrimitive function, both including cases that use all vertices within the active IDirect3DMobileVertexBuffer, and including cases that use only a subset of the vertices within the active IDirect3DMobileVertexBuffer Various BaseVertexIndex, minIndex, NumVertices, startIndex, and PrimCount values for the DrawIndexedPrimitive function, including cases that use all vertices and indices within the active IDirect3DMobileVertexBuffer or IDirect3DMobileIndexBuffer, and cases that use only a subset of the vertices within the active IDirect3DMobileVertexBuffer or IDirect3DMobileIndexBuffer
400-599	Tests clipping. These test cases render scenes that primarily cover permutations of the following factors: D3DMPRIMITIVETYPE: D3DMPT_LINELIST, D3DMPT_TRIANGLELIST D3DMRS_FILLMODE: D3DMFILL_POINT, D3DMFILL_SOLID, D3DMFILL_WIREFRAME D3DMRS_SHADEMODE: D3DMSHADE_GOURAUD, D3DMSHADE_FLAT FVF components with D3DMFVF_DIFFUSE, and with or without D3DMFVF_SPECULAR Primitive position relative to the view frustum: Primitives that are wholly outside of the view frustum Primitives that are wholly inside of the view frustum Primitives that are partially inside and partially outside of the view frustum
600-699	Tests FVF. These test cases render scenes that primarily cover permutations of the following factors: D3DMRS_SHADEMODE: D3DMSHADE_GOURAUD, D3DMSHADE_FLAT D3DMTSS_TEXTURETRANSFORMFLAGS: With and without D3DMTTFF_PROJECTED With and without D3DMTTFF_COUNT2 With and without D3DMTTFF_COUNT3 FVF components: With and without D3DMFVF_DIFFUSE With and without D3DMFVF_SPECULAR With and without the texture coordinate counts D3DMFVF_TEX1, D3DMFVF_TEX2, D3DMFVF_TEX3, and D3DMFVF_TEX4 With and without the texture coordinate sizes D3DMFVF_TEXCOORDSIZE1(0), D3DMFVF_TEXCOORDSIZE2(0), and D3DMFVF_TEXCOORDSIZE3(0)

Table 19-7. D3DM Driver Comparison Test Cases(Continued)

Test Case	Description
700-899	Tests fogging. These test cases render scenes that primarily cover permutations of the following factors: D3DMRS_FOGCOLOR: Various D3DMCOLOR settings D3DMRS_FOGDENSITY: Various values in the [0.0f, 1.0f] range D3DMRS_FOGENABLE: TRUE , FALSE D3DMRS_FOGSTART, D3DMRS_FOGEND: For D3DMRS_FOGTABLEMODE, various values in the [0.0f, 1.0f] range For D3DMRS_FOGVERTEXMODE, various values in the [fNear, fFar] range, where fNear and fFar are view frustum planes D3DMRS_FOGTABLEMODE: D3DMFOG_NONE, D3DMFOG_LINEAR, D3DMFOG_EXP, D3DMFOG_EXP2 D3DMRS_FOGVERTEXMODE: D3DMFOG_NONE, D3DMFOG_LINEAR, D3DMFOG_EXP, D3DMFOG_EXP2 D3DMTRANSFORMSTATETYPE: D3DMTS_PROJECTION settings with various near and far planes
900-999	Tests mipmaps. These test cases render scenes that primarily cover permutations of the following factors: D3DMTSS_MAGFILTER: D3DMTEXF_POINT, D3DMTEXF_LINEAR, D3DMTEXF_ANISOTROPIC D3DMTSS_MINFILTER: D3DMTEXF_POINT, D3DMTEXF_LINEAR, D3DMTEXF_ANISOTROPIC D3DMTSS_MIPFILTER: D3DMTEXF_NONE, D3DMTEXF_POINT, D3DMTEXF_LINEAR D3DMTSS_MIPMAPLODBIAS: Various values in the [-2.0f, 2.0f] range Primitive vertices, which vary to produce primitive extents that can isolate specific mipmap levels
1000-2099	Tests transformations. These test cases render scenes that primarily cover permutations of the following factors: D3DMTRANSFORMSTATETYPE settings: Various rotation, shear, scale, and translation matrices for D3DMTS_VIEW Various rotation, shear, scale, and translation matrices for D3DMTS_WORLD
2100-2299	Tests StretchRect operations. These test cases render scenes that primarily cover permutations of the following factors: D3DMPOOL, for source and destination surfaces: D3DMPOOL_VIDEOMEM, D3DMPOOL_SYSTEMMEM D3DMTEXTUREFILTERTYPE: D3DMTEXF_NONE, D3DMTEXF_LINEAR, D3DMTEXF_POINT IDirect3DMobileSurface, originating from GetBackBuffer, CreateImageSurface, and IDirect3DMobileTexture::GetSurfaceLevel Relative source and destination RECT extents: Identical extents, Extents requiring shrinking, Extents requiring stretching
2500-2599	Tests CopyRect operations. These test cases render scenes that primarily cover permutations of the following factors: D3DMPOOL, for source and destination surfaces: D3DMPOOL_VIDEOMEM, D3DMPOOL_SYSTEMMEM IDirect3DMobileSurface, originating from GetBackBuffer, CreateImageSurface, and IDirect3DMobileTexture::GetSurfaceLevel

Table 19-7. D3DM Driver Comparison Test Cases(Continued)

Test Case	Description
2700-2799	Tests ColorFill operations. These test cases render scenes that primarily cover permutations of the following factors: D3DMPOOL, for source and destination surfaces: D3DMPOOL_VIDEOMEM, D3DMPOOL_SYSTEMMEM IDirect3DMobileSurface, originating from GetBackBuffer, CreateImageSurface, and IDirect3DMobileTexture::GetSurfaceLevel
2800-2899	Tests last pixels. These test cases render scenes that primarily cover permutations of the following factors: D3DMRS_LASTPIXEL: FALSE, TRUE D3DMPRIMITIVETYPE: D3DMPT_LINELIST, D3DMPT_LINESTRIP
3000-3099	Tests texture wrapping. These test cases render scenes that primarily cover permutations of the following factors: D3DMRS_WRAP0 settings: D3DMWRAPCOORD_0, D3DMWRAPCOORD_1 D3DMWRAPCOORD_0 | D3DMWRAPCOORD_1 FVF components: D3DMFVF_TEX1 with D3DMFVF_TEXCOORDSIZE1(0) D3DMFVF_TEX1 with D3DMFVF_TEXCOORDSIZE2(0) FVF component values, including various position and texture coordinate values
3100-3299	Tests alpha test operations. These test cases render scenes that primarily cover permutations of the following factors: D3DMTSS_ALPHAARG1: D3DMTA_DIFFUSE, D3DMTA_TEXTURE D3DMTSS_ALPHAARG2: D3DMTA_CURRENT, D3DMTA_DIFFUSE D3DMRS_ALPHAFUNC: D3DMCMP_ALWAYS, D3DMCMP_NEVER , D3DMCMP_LESS, D3DMCMP_EQUAL, D3DMCMP_LESSEQUAL, D3DMCMP_GREATER, D3DMCMP_NOTEQUAL, D3DMCMP_GREATEREQUAL D3DMTSS_ALPHAOP: D3DMTOP_MODULATE, D3DMTOP_SELECTARG1 D3DMRS_ALPHAREF: Various values in the [0,255] range, Contents of various surfaces
3500-3599	Tests depth bias. These test cases render scenes that primarily cover permutations of the following factors: D3DMRS_DEPTHBIAS D3DMRS_SLOPESCALEDEPTHBIAS FVF component values, including various position and diffuse color values
3600-3699	Tests swap chains. These test cases render scenes that primarily cover permutations of the following factors: Various source and destination RECT structures for IDirect3DMobileDevice::Present

Table 19-7. D3DM Driver Comparison Test Cases(Continued)

Test Case	Description
3700-3799	Tests drawing. These test cases render scenes that primarily cover permutations of the following factors: D3DMPRIMITIVETYPE: D3DMPT_TRIANGLELIST, D3DMPT_TRIANGLESTRIP D3DMRS_CULLMODE: D3DMCULL_CW , D3DMCULL_CCW , D3DMCULL_NONE Various drawing orders among primitives with identical depth values
4000-4799	Tests texture stages. These test cases render scenes that primarily cover permutations of the following factors: D3DMTEXTUREOP: D3DMTOP_SELECTARG1 D3DMTOP_SELECTARG2 D3DMTOP_MODULATE D3DMTOP_MODULATE2X D3DMTOP_MODULATE4X D3DMTOP_ADD D3DMTOP_ADDSIGNED D3DMTOP_ADDSIGNED2X D3DMTOP_SUBTRACT D3DMTOP_ADDSMOOTH D3DMTOP_BLENDDIFFUSEALPHA D3DMTOP_BLENDTEXTUREALPHA D3DMTOP_BLENDFACTORALPHA D3DMTOP_BLENDTEXTUREALPHAPM D3DMTOP_MULTIPLYADD D3DMTOP_LERP D3DMTSS_COLORARG0, D3DMTSS_ALPHAARG0, D3DMTSS_COLORARG1, D3DMTSS_ALPHAARG1, D3DMTSS_COLORARG2, D3DMTSS_ALPHAARG2: D3DMTA_TFACTOR D3DMTA_SPECULAR D3DMTA_TEXTURE D3DMTA_DIFFUSE D3DMTA_COMPLEMENT D3DMTA_ALPHAREPLICATE Various D3DMCOLOR values input as texture parameters

The test result for comparison test is 232 passed, 14 fail, and 2072 skipped. The fail 14 test case and skipped case are known issues for the MBX driver. They are 3504,4061,4236,4238,4239,4241,4242,4244,4260,4262,4263,4265,4269,4271.

19.5.5 Direct3D Mobile/OpenGL ES Application Samples/Demos

In order to reduce the OS image size, a limited number of pre-built demos have been included in the BSP package.

19.5.5.1 Direct3D Mobile Application Samples

There are seven D3DM application samples located in `\WINCE500\PUBLIC\DIRECTX\SDK\SAMPLES\D3DM`, which will be built when BSP_MBX is enabled. Table 19-8 identifies these tests.

Table 19-8. D3DM Mobile Application Samples

Tests	Pass
D3dm_createdevice.exe	Y
D3dm_fixedpoint.exe	Y
D3dm_lights.exe	Y
d3dm_matrices.exe	Y
d3dm_textures.exe	Y
d3dm_twotri.exe	Y
d3dm_vertices.exe	Y

19.5.5.2 Direct3D Mobile/OpenGL ES Demos

Table 19-9. D3DM Mobile/OpenGL ES Demos

Tests	Pass
D3DMEvilSkull.exe	Y
OGLESVase.exe	Y

See the MX31 MBX SDK package for more demo applications in source code and binary code.

19.6 Drivers API Reference

19.6.1 Direct3D Mobile

For documentation for the Direct3D driver APIs, see the Platform Builder Help. No additional custom API information is required for the features currently supported in the Direct3D Mobile driver. For reference information on basic Direct3D Mobile driver functions, methods, and structures, see the Platform Builder Help:

Developing a Device Driver > **Windows CE Drivers** > **Direct3D Mobile Display Drivers** > **Direct3D Mobile Driver Reference**

For reference information on Direct3D Mobile functions, callbacks, and structures, see the Platform Builder Help:

Windows CE Features > **Graphics and Multimedia Technologies** > **Graphics Direct3D Mobile**

19.6.2 OpenGL ES

Documentation for the OpenGL driver APIs can be found at http://www.khronos.org/opengles.

Chapter 20
NAND Flash Media Driver (FMD)

Windows CE provides driver support for flash media devices using the FMD (Flash Media Driver) and FAL (Flash Abstraction Layer) software architecture. The FMD and FAL allow NAND flash storage to be exposed as a block driver that is accessed using the file system.

The FMD software layers that are ported to the i.MX31 NAND flash controller are responsible for the actual I/O with the corresponding NAND flash devices respectively.

The NAND FMD driver included in the i.MX31 BSP is targeted for the NAND flash device shipped with the i.MX31 SDK, but these drivers can be easily ported to other NAND flash devices.

There are two categories of NAND flash devices:

- small page size, where page size is 512 bytes
- large page size, where page size is 2048 bytes

20.1 NAND FMD Summary

Table 20-1 identifies the source code location, library dependencies and other BSP information.

Table 20-1. NAND Flash Media Driver Summary

Driver Attribute	Definition
Target Platform (TGTPLAT)	3DS
Target SOC (TGTSOC)	MX31
MXARM11 CSP Driver Path	N/A
CSP Driver Path	N/A
CSP Static Library	N/A
Platform Driver Path	\WINCE500\PLATFORM\<*tgtplat*>\SRC\COMMON\NANDFMD \WINCE500\PLATFORM\<*tgtplat*>\SRC\DRIVERS\BLOCK\NANDFMD
Import Library	N/A
Driver DLL	nandfmd.dll
Catalog Item	Third Party > BSPs > Freescale > MX31> Storage Drivers > Samsung K9K2G08U0A NAND Flash.
SYSGEN Dependency	N/A
BSP Environment Variables	BSP_NAND_FMD=1 & BSP_NAND_K9K2G08U0A=1 (Set in platform>setting>Environment)

20.2 Supported Functionality

The NAND FMD driver enables the 3-Stack board to provide the following software and hardware support:

- Configured as a loadable .dll module
- Supports small page size (512 bytes) and large page size (2048 bytes)
- Supports hardware ECC
- Configured as an extensible driver for supporting other types of NAND chip required by customers
- Supports bad block management
- Supports reserved block (block reserved for Xloader, EBOOT, OS Image) management
- Supports interface for both EBOOT and OS
- Supports the Microsoft FMD(Flash Media Driver) interface
- The data transfer rate is at least 400 Kbytes / second
- Supports DMA operation if possible by hardware
- Supports bad block replacement to ensure the data reliability
- Supports EMI clock gating for power management

20.2.1 Conflicts with other SoC Peripherals

The NAND flash controller interface consists of shared EMI signals and NAND-specific signals. The NAND-specific signals (NFWE_B, NFRE_B, NFALE, NFCLE, NFWP_B, NFCE_B, NFRB) can be configured for alternate functionality (ATA, USB H2, GPIO) using the i.MX31 IOMUX. The configuration supported by the BSP does not use this alternate functionality and dedicates these signals for NAND flash controller use. Changing this configuration would result in a conflict and prevent proper operation of the NAND FMD.

20.3 Software Operation

For development concepts for flash media drivers, see the Platform Builder Help:

Developing a Device Driver > Windows CE Drivers > Flash Media Drivers

The NAND FMD supported in the MX31 PDK BSPs implements the required FMD functions for interfacing to NAND flash devices.

20.3.1 Compile-Time Configuration Options

The NAND FMD driver abstracts the details of the NAND flash memory device to a single header file. This header file is found in the `\WINCE500\PLATFORM\<tgtplat>\SRC\COMMON\NANDFMD` directory and named according to the NAND device.

To support a different NAND device, create a new header using one of the existing NAND device headers as a template, and then update the device-specific information. Then update the reference to the device-specific header in `\WINCE500\PLATFORM\<tgtplat>\SRC\COMMON\NANDFMD\nandfmd.h` and recompile the NAND FMD driver for the new device.

20.3.2 Loading and Initialization

The loading and initialization of the NAND FMD is primarily controlled using registry keys. See the Platform Builder Help:

Windows CE Features > File Systems and Data Store > File Systems and Data Store Registry Settings

20.3.3 Registry Settings

The registry keys implemented for the NAND FMD provides basic support for loading and configuring the NAND as a file system mount.

For information about the additional configuration options, see the Platform Builder Help:

Windows CE Features > File Systems and Data Store > File Systems and Data Store Registry Settings

20.3.4 DMA Support

The NAND FMD currently does not provide DMA support. The conditional compilation variable BSP_SDMA_SUPPORT_NANDFC found in the BSP header `bsp_cfg.h` is ignored.

20.3.5 Power Management

The power management support provided by the NAND FMD leverages clock gating features available within the hardware. The NAND flash controller is a component of the EMI (External Memory Interface). The clock gating for the EMI is managed globally for all EMI components. This implies that while the NAND flash controller does not have individual clock gating capability, the clocks to the NAND flash controller will be disabled when the EMI is clock gated.

The CSPDDK manages the configuration of the EMI clock gating. The NAND FMD driver notifies the CSPDDK when the NAND flash interface is active. This is achieved using the DDKClockSetGatingMode within the FMD functions to signal when the EMI clocks must remain active for accessing the NAND memory device.

20.4 Unit Test

The NAND FMD was testing using Windows CE 5.0 Test Kit and additional system use cases. This section describes the test scenarios that were used to verify the operation of the NAND FMD.

20.4.1 CETK Testing

The CETK includes Storage Device tests that can be used to exercise the NAND FMD. The following table lists the CETK tests that were performed and provides the test configuration necessary to target the NAND FMD.

NOTE

Depending on the state of the NAND flash memory, it may be necessary to format and partition the NAND device using Storage Manager prior to running the CETK tests that do not reformat the device automatically.

Table 20-2. CETK Testing

CETK Test	Command Line
Other Tests > Partition Driver Test	tux -o -d msparttest -c “-z”
Storage Device > Storage Device Block Diver Read/Write Test	tux -o -d rwtest -c “-z” Note: This test does not recognize the storage device profile parameter. You may need to remove other storage devices from the image so that the device targeted for the test appears as DSK1 on the system.
Storage Device > Storage Device Block Diver Benchmark Test	tux -o -d rw_all -c “-p FlashDisk -z”
Storage Device > Storage Device Block Diver API Test	tux -o -d disktest -c “-p FlashDisk -z”
Storage Device > Flash Memory Read/Write and Performance Test	For Windows CE CETK: tux -o -d flshwear -c “-z” For Mobile CETK: tux -o -d flshwear -c “-z /profile FlashDisk”
Storage Device > File System Driver Test	tux -o -d fsdtst -c “-p FlashDisk -z”

20.4.2 System Testing

The following system tests were performed to verify the operation of the NAND FMD:

Use the **Start** > **Settings** > **Control Panel** > **Storage Manager** to format and create partitions on the mounted NAND device.

Establish ActiveSync connection over USB and transfer files to/from the NAND storage.

Write media files to NAND storage. Use Windows Media Player to playback media files from NAND storage.

Chapter 21
Postfilter Driver

The Postfilter Driver provides an API to access to hardware acceleration for H.264 deblocking and MPEG4 deblocking and deringing. The Postfilter driver interfaces with the Image Processing Unit (IPU) Postfilter (PF) submodule. The Postfilter driver conforms to the architecture for Windows CE stream interface drivers.

21.1 Postfilter Driver Summary

Table 21-1 identifies the source code location, library dependencies and other BSP information:

Table 21-1. Postfilter Driver Summary

Driver Attribute	Definition
Target Platform (TGTPLAT)	3DS
Target SOC (TGTSOC)	MX31
CSP Driver Path	..\CSP\ARM\FREESCALE\MXARM11\DRIVERS\IPU\PF
CSP Static Library	mxarm11_pf.lib
Platform Driver Path	..\PLATFORM\<*tgtplat*>\SRC\DRIVERS\IPU\PF
Import Library	N/A
Driver DLL	pf.dll
Catalog Items	N/A
SYSGEN Dependency	N/A
BSP Environment Variables	BSP_PF=1

21.2 Supported Functionality

The Postfilter driver enables the 3-Stack board to provide the following software and hardware support:

- Supports three postfiltering modes: H.264 deblocking, MPEG4 deblocking, and MPEG4 deblocking and deringing.
- Supports pause functionality for H.264 deblocking, allowing the programmer to specify a pause row on which the operation will be paused. The paused operation may then be resumed at any time.
- Is a stream interface driver implementing the programming interface defined in this document
- Supports two power management modes, full on and full off.

21.3 Hardware Operation

Refer to the chapter on IPU in the *MCIMX31 and MCIMX31L Applications Processors Reference Manual* for detailed operation and programming information.

21.3.1 Conflicts with other SoC Peripherals

There are no peripheral conflicts on this SoC.

21.4 Software Operation

21.4.1 Communicating with the Postfilter Driver

The Postfilter is a stream interface driver, and is thus accessed through the file system APIs. To communicate using the Postfilter, first obtain a handle to the device using the **PFOpenHandle** function. Subsequent commands to the device are issued using various APIs supported by this driver.

21.4.2 Creating a Handle to the Postfilter Driver

To communicate with the PF driver, first create a handle to the device using the **PFOpenHandle** API. The default PF port is 1.

To open a Handle to the PF:

```
// Handle to the PF device
HANDLE g_hPF = NULL;

// opening the default PF port.
g_hPF = PFOpenHandle();
```

For more information on this API, see the **PFOpenHandle** section under the PF API reference.

21.4.3 Configuring the Postfilter Driver

The **PFConfigure** API must be called to configure several important settings for the Postfilter driver. The **pfConfigData** data structure must be filled out and passed as a parameter to **PFConfigure**.

You will need three types of information needed to configure a Postfilter operation:

- The postfiltering mode (for example, H.264 deblocking or MPEG4 deblocking)
- The input frame parameters, including width, height, and stride
- Parameters for the input buffer containing quantization parameter and boundary strength data, including the physical address and size of the buffer

  Note: The Postfiltering hardware requires the physical address of a physically contiguous buffer. The **AllocPhysMem()** Windows CE API is recommended for allocating this physically contiguous buffer.

The following code example shows how to configure the Postfilter driver.

```
// Configure Postfilter for MPEG4 deblocking
pfConfigData configData;
UINT32 iWidth, iHeight;

iWidth = 176;
iHeight = 144;
iQPBytesPerFrame = iHeight / 16 * iWidth / 16;

// Set up configuration data
configData.mode = pfMode_MPEG4Deblock;
configData.frameSize.width = iWidth;
configData.frameSize.height = iHeight;
configData.frameStride = iWidth;

configData.qpBuf = pQPPhysAddr; // Start address of a physically contiguous buffer
configData.qpSize = iQPBytesPerFrame;

PFConfigure(hPF, &configData);
```

21.4.4 Executing Postfilter Operations

Once the Postfilter driver is configured, a postfiltering task can be commenced. A call to the **PFStart** function starts the configured operation. **PFStart** takes as parameters information about the input and output buffers for the current postfiltering task. This information includes the size of the buffer, a pointer the physical address of the start of the Y data buffer, and offsets to the U and Y data buffers (**Note**: For the planar YUV data provided as input for postfiltering, the Y, U, and V data buffers must be physically contiguous in memory). Additionally, for the case of H.264 deblocking operations, a pause row may be specified. When the pause row is reached during the deblocking operation, the task is paused, and does not resume until the **PFResume** API is called. To disable pausing, set the pause row to 0.

```
//-------------------------------------------------------------
// Set up Start Data Structure for MPEG4 deblocking operation
//-------------------------------------------------------------
pfStartParams startData;
pfBuffer inBuf, outBuf;
DWORD iYUVBytesPerFrame = iWidth * iHeight * 3/2;

// Set up input and output buffers.
inBuf.size = iYUVBytesPerFrame;
inBuf.yBufPtr = pInputFramePhysAddr; // Physical address of input buffer
inBuf.uOffset = iWidth * iHeight;
inBuf.vOffset = inBuf.uOffset * 5/4;

outBuf.size = iYUVBytesPerFrame;
outBuf.yBufPtr = pOutputFramePhysAddr; // Physical address of output buffer
outBuf.uOffset = inBuf.uOffset;
outBuf.vOffset = inBuf.vOffset;

startData.in = &inBuf;
startData.out = &outBuf;
startData.h264_pause_row = 0;

// Start PF
PFStart(hPF, &startData);
```

Three events are signaled, which represent the completion of three phases of the Postfilter task:

- the completion of the Y component,
- the completion of the Cr componment, and
- the completion of the Cb component, which corresponds to the End-Of-Frame (EOF) of the Postfilter task

These events are signaled through named Windows CE Event objects. You must create a Windows CE Handle, using either PF_Y_EVENT_NAME, PF_CR_EVENT_NAME or PF_EOF_EVENT_NAME strings defined in the `pf.h` header file, to signal the application when the postfilter task has completed. This handle will be used in a call to the **WaitForSingleObject** function.

The following sample code creates a handle to the Postfilter EOF event, and waits for that event to be signaled.

```
HANDLE g_hPFEOFEvent;

// Create event for Postfilter EOF
g_hPFEOFEvent = CreateEvent(NULL, FALSE, FALSE, PF_EOF_EVENT_NAME);

// Wait for End of Frame
WaitForSingleObject(g_hPFEOFEvent, INFINITE);
```

21.4.5 Closing the Handle to the Postfilter Driver

Call the **PFCloseHandle** function to close a handle to the Postfilter driver when an application is done using it.

21.4.6 Postfilter Registry Settings

The following registry keys are required to properly load the Postfilter driver module.

```
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\PF]
   "Prefix"="POF"
   "Dll"="pf.dll"
   "Order"=dword:20
   "Index"=dword:1
```

21.4.7 Power Management

The Postfilter driver consumes power primarily through the operation of the Postfilter IPU sub-module. If the Postfilter driver is included in the OS image, the Postfilter submodule will be enabled during boot-up, and will remain enabled until the system is shut down. No additional power management is currently supported.

21.4.7.1 PowerUp

This function is not implemented for the Postfilter driver.

21.4.7.2 PowerDown

This function is not implemented for the Postfilter driver.

21.4.7.3 IOCTL_POWER_SET

This function is not implemented for the Postfilter driver.

21.5 Unit Test

The Postfilter driver unit tests verify the proper operation of the Postfilter driver modes of operation.

For H.264 deblocking mode, a full set of input and output reference data are provided to perform a bitmatch verification of the Postfilter operation. For MPEG4 operations, input data is provided, but no reference output data is provided. Thus, for MPEG4 postfiltering modes, an output YUV file is generated (and may be viewed using a YUV viewer too), but no bitmatching is performed.

21.5.1 Unit Test Software

Table 21-2 lists the required software to run the Postfilter driver tests.

Table 21-2. Software Requirements to Run Postfilter Driver Files

Requirements	Description
Tux.exe	Tux test harness, which is needed for executing the test
Kato.dll	Kato logging engine, which is required for logging test data
Tooltalk.dll	Library required by Tux.exe and Kato.dll. Handles the transport between the target device and the development workstation
PFTest.dll	Main test .dll file.

21.5.2 Building the Postfilter Tests

21.5.2.1 Unit Test in Windows CE

To build the Postfilter unit test, you need to build an OS image for the desired configuration (make sure the Postfilter driver is enabled).

To build an OS image:

1. In Platform Builder, from the **Build OS** menu, select **Open Release Directory**. This opens a DOS prompt.
2. Change to the Postfilter test directory (`\WINCE500\SUPPORT\TESTS\PF`).
3. Enter **set WINCEREL=1** on the command prompt and press Enter. This copies the generated ".dll" to the flat release directory.
4. Enter **build –c** at the prompt, and then press Enter. The `pftest.dll` file will be located in the `$(_FLATRELEASEDIR)` directory.

5. Copy all test data files (`CIF_six_frames_h264.bs`, `CIF_six_frames_h264.qp`, `CIF_six_frames_h264.yuv`, `H264RefOutput.yuv`, `QCIF_twenty_frames_mpeg4.qp`, `QCIF_twenty_frames_mpeg4.yuv`) from `\WINCE500\SUPPORT\TESTS\PF` to the `$(_FLATRELEASEDIR)` directory.
6. Copy `Tux.exe`, `Kato.dll`, `Tooltalk.dll` from Windows CE CETK to the `$(_FLATRELEASEDIR)` directory.

21.5.3 Running the Postfilter Tests

After downloading an OS image to the board, execute the unit test from the target shell with the following command:

```
s tux –o –d \release\pftest.dll
```

21.6 Postfilter Driver API Reference

21.6.1 Postfilter Driver Functions

21.6.1.1 PFOpenHandle

This API creates a handle to the Postfilter stream driver:

```
HANDLE PFOpenHandle(
        void
);
```

Parameters

This API accepts no parameters.

Return Values

An open handle to the specified file indicates success. INVALID_HANDLE_VALUE indicates failure.

Remarks

A handle returned successfully from this function call is required in all subsequent calls to other PF API functions. Use the **PFCloseHandle** function to close the handle returned by **PFOpenHandle**.

21.6.1.2 PFCloseHandle

This API function closes a handle to the PF driver:

```
BOOL PFCloseHandle(

        HANDLE hPF
);
```

Parameters

hPF [in] Handle to the PF driver returned by PFOpenHandle API.

Return Values

TRUE indicates success.

FALSE indicates failure.

To get extended error information, call GetLastError.

An open handle to the specified file indicates success.

Remarks

None.

21.6.1.3 PFConfigure

This API configures the Postfilter driver:

```
void PFConfigure(
        HANDLE hPF,
        pPfConfigData pConfigData
);
```

Parameters

hPF	[in] Handle to the PF driver returned by **PFOpenHandle** API.
pConfigData	[in] An object of the ***pfConfigData*** structure.

Return Values

None.

Remarks

This function performs configuration steps that are required before starting a Post-Filtering operation. Calling **PFStart** without previously calling **PFConfigure** will result in an error.

21.6.1.4 PFStart

This API function starts a Postfilter operation.

```
void PFStart(
        HANDLE hPF,
        pPfStartParams pStartParms
);
```

Parameters

hPF	[in] Handle to the PF driver returned by **PFOpenHandle** API.
pStartParms	[in] An object of the **pfStartParams** structure. For H.264 Postfilter mode, no output buffer is required, as the input buffer is used for input and output.

Return Values

None.

Remarks

Calling **PFStart** without previously calling **PFConfigure** will result in an error.

Completion of the Postfilter operation is signaled through a named event using the name PF_EOF_EVENT_NAME. A user can call **CreateEvent** and **WaitForSingleObject** to create and wait on the PostFilter end-of-frame event.

21.6.1.5 PFStart2

This API function starts a Postfilter operation.

```
void PFStart2(
        HANDLE hPP,
        pPfStartParams pStartParms,
        unsigned int VirtualFlag,
        unsigned int yOffset
);
```

Parameters

hPF	[in] Handle to the PF driver returned by **PFOpenHandle** API.
pStartParms	[in] An object of the **pfStartParams** structure. For H.264 Postfilter mode, no output buffer is required, as the input buffer is used for input and output.
VirtualFlag[in]	Flag to indicate if virtual memory pointer.
yOffset[in]	offset to the pointer, in byte.

Return Values

None.

Remarks

PFStart2 extends **PFStart** by passing two more parameters.

21.6.1.6 PFResume

This API function resumes an H.264 deblocking operation that was previously started with a pause row specified. A new pause row may be specified, or the operation may be allowed to run to completion.

```
DWORD PFResume(
        HANDLE hPF,
        UINT32 h264_pause_row
);
```

Parameters

hPF	[in] Handle to the PF driver returned by **PFOpenHandle** API.
h264_pause_row	[in] Integer indicating the Y row at which to pause the operation. Should be set to 0 to disable an additional pause.

Return Values

If successful, PF_SUCCESS.

If failure, one of the following:

PF_ERR_NOT_RUNNING – The Postfilter operation is not running.

PF_ERR_PAUSE_NOT_ENABLED – The H.264 pause is not enabled.

PF_ERR_INVALID_PARAMETER – The pause row parameter is not in a valid range.

Remarks

Calling **PFResume** without previously calling **PFStart** with the pause row enabled will result in an error.

21.6.2 PF Driver Enumerations

21.6.2.1 pfMode

Enumeration of Postfilter operation modes.

```
typedef enum pfModeEnum
{
    pfType_Disabled,           // No post-filtering
    pfType_MPEG4Deblock,       // MPEG4 Deblock only
    pfType_MPEG4Dering,        // MPEG4 Dering only
    pfType_MPEG4DeblockDering, // MPEG4 Deblock and Dering
    pfType_H264Deblock,        // H.264 Deblock
} pfMode;
```

Elements

pfType_Disabled Postfiltering disabled.

pfType_MPEG4Deblock

Postfiltering operation is MPEG4 Deblock only.

pfType_MPEG4Dering Postfiltering operation is MPEG4 Dering only.

pfType_MPEG4DeblockDering

Postfiltering operation is MPEG4 Deblock and Dering.

pfType_H264Deblock Postfiltering operation is H.264 Deblock.

Remarks

None.

21.6.3 PF Driver Structures

21.6.3.1 pfBuffer

Structure to describe the YUV buffers used in Postfilter operations.

```
typedef struct pfBufferStruct
{
    int      size;
    UINT32  *yBufPtr;
    UINT32   uOffset;
    UINT32   vOffset;
} pfBuffer, *pPfBuffer;
```

Members

size	Size of the allocated buffer, in bytes.
yBufPtr	Pointer to the start of the Y buffer.
uBufOffset	Offset, in bytes, of the U buffer, relative to the start of the Y buffer. If set to 0, a default calculation will be made based on the height and stride of the frame.
vBufOffset	Offset, in bytes, of the V buffer, relative to the start of the Y buffer. If set to 0, a default calculation will be made based on the height and stride of the frame.

21.6.3.2 pfConfigData

Structure used to configure the Postfilter driver for an operation.

```
typedef struct pfConfigDataStruct
{
    pfMode mode;
    pfFrameSize frameSize;
    UINT32 frameStride;
    UINT32 *qpBuf;
    UINT32 qpSize;
} pfConfigData, *pPfConfigData;
```

Members

mode	The Post-filter operation desired.
frameSize	The dimensions of the frame for Postfiltering.
frameStride	The stride of the frame, in bytes.
qpBuf	A pointer to a buffer containing, sequentially, the quantization parameter (QP) and boundary strength (BS) data for the Postfilter operation.
qpSize	The size of the buffer containing the QP and BS data.

Remarks

For H.264 deblocking, there is one 32-bit quantization parameter word for each 16 x16 pixel macroblock. Additionally, there is one 8-bit boundary strength word for each 4 x 4 pixel block.

For MPEG4 deblocking and deringing, there is one 8-bit quantization parameter word for each 16 x16 pixel macroblock. No boundary strength data is needed for MPEG4 deblocking and deringing.

21.6.3.3 pfFrameSize

Structure for the Postfiltering frame size.

```
typedef struct pfFrameSizeStruct {
    UINT16 width;
    UINT16 height;
} pfFrameSize, *pPfFrameSize;
```

Members

width	Frame width, in pixels.
height	Frame height, in pixels.

21.6.3.4 pfStartParams

This structure is used in calls to **PFStart** to provide information needed to start the Postfilter operation.

```
typedef struct PfStartParamsStruct
{
        pPfBuffer in;
        pPfBuffer out;
        UINT32 h264_pause_row;
        void* qp_buf;
        void* bs_buf;
        int qp_size;
} pfStartParams, *pPfStartParams;
```

Members

in	Pointer to the input buffer.
out	Pointer to the output buffer.
h264_pause_row	Row to pause at for H.264 mode. Set to 0 to disable pause.
qp_buf	pointer to current qb buffer
bs_buf	pointer to current bs buffer
qp_size	qb and bs buffer size

For more information, refer to the “Postfilter Flow Control” section of *MCIMX31 and MCIMX31L Applications Processors Reference Manual*.

Chapter 22
Power Manager

The Power Manager module is used to help control the power efficiency of the system. Power Manager manages device power, improves overall OS power efficiency, and coexists with applications and drivers that do not support Power Manager. It also controls the power settings in peripheral devices.

22.1 Power Manager Summary

Table 22-1 identifies the source code location, library dependencies, and other BSP information.

Table 22-1. Power Manager Summary

Driver Attribute	Definition
Target Platform (TGTPLAT)	3DS
Target SOC (TGTSOC)	MX31
MXARM11 CSP Driver Path	N/A
CSP Driver Path	N/A
CSP Static Library	N/A
Platform Driver Path	N/A
Import Library	N/A
Driver DLL	Pm.dll
Catalog Item	Core OS >Windows CE Devices >Core OS Services >Power Management > Power Management (Full)
SYSGEN Dependency	SYSGEN_PM
BSP Environment Variables	N/A

22.2 Supported Functionality

The Power Manager module test is provided in the Platform Builder public directory.

22.3 Hardware Operation

Power Manager does not interface directly to peripheral devices.

22.4 3-Stack Software Operation

The Platform Builder Help documents the Power Manager framework and sample Power Manager. See the following:

Developing a Device Driver > **Power Management**

For information about the system power states implemented in the sample Power Manager, see Platform Builder Help:

Developing a Device Driver > **Power States** > **System Power States** >**Example System Power States**

22.4.1 Power Management

Power Manager exports a stream interface and registers as a generic power management device driver. Power Manager can receive IO_POWER_XX notifications in order to configure PMIC and wakeup devices in thread context.

Table 22-2 identifies the power consumption goals and the steps that can be performed to achieve the power consumption goals:

Table 22-2. Power Management

Mode	Goal	Action
suspend	20mw	(1) Keep power supply to memory, and set SW2A to work in low power mode (2) Lower the voltage to ARM and QPER, and set SW1 to work in low power mode (3) Set all PIMC regulators and GPO control by standby pin (4) Close PMIC SW3 supply

Table 22-2. Power Management(Continued)

Mode	Goal	Action
audio playback	210mw	(1) keep power supply to memory, set SW2A, SW2B work in low power mode (2) Lower the voltage to ARM and QPER, and set SW1 to work in lower power mode (3) Close PIMC regulators that are not being used (4) Close GPO 1, 2, and 3 (5) Close PMIC SW3 supply (6)Replace MSFT WMA codec with FSL codec (7) Set AHB Clock freq to 66.5M (8) Set DVFS mode to Enable (9) Set SW2 Voltage to 1.7V.
video playback	1000mw	(1) same as audio playback

22.4.2 Image Configuration

To build an audio playback power consumption image:

1. Remove ATA and USB High speed Host 2 driver module from the workspace.
2. Copy the FSL WMA codec library `wmaplsv_rawdec_wince_arm_iw.lib` to the `WINCE500\PUBLIC\DIRECTX\OAK\LIB\ARMV4I\RETAIL` folder
3. Replace MSFT WMA codec.
4. Add the DVFS module.
5. Perform a sysgen.

To enable optimization audio playback power consumption function, you will need to build an image for it.

To build an image:

1. Boot up the device and enter the EBOOT menu.
2. Set the AHB Clock freq to 66.5 megahertz.
3. Set DVFS mode to **Enable**.
4. Set SW2 Voltage to 1.7 volts.
5. Set the Power Policy to **Enable**.
6. Save the changes and reboot the device.

NOTE

Because the 3-Stack Power Policy is only aware of audio/video playback functions when they are loaded from NAND, other modules may not work properly when Power Policy is enabled.

22.4.3 Registry Settings

In this system power state, the user is interacting actively with the system.

```
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Power\State\On]
    "Default"=dword:0           ; D0
    "Flags"=dword:10000         ; POWER_STATE_ON
```

In this system power state, the user may be interacting with the system, but not actively. For instance, they might be looking at the screen or they might not. In this power state the system is "idle" but still in use by the user, so all devices would still be operational (but possibly with some latency).

```
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Power\State\UserIdle]
    "Default"=dword:1           ; D1
    "Flags"=dword:0
```

In this system power state, the user is not considered to be using the system, even passively. However, the system is not suspended and system programs may be doing work on the user's behalf. In this power state the system is "idle" but might still be used by system programs. Devices that are not actively doing work might be powered down.

```
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Power\State\SystemIdle]
    "Default"=dword:2           ; D2
    "Flags"=dword:0
```

In this system power state, the system is suspended. Devices are turned off, interrupts are not being serviced, and the CPU is stopped.

```
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Power\State\Suspend]
    "Default"=dword:3           ; D3
    "Flags"=dword:200000        ; POWER_STATE_SUSPEND
```

Entering this system power state reboots the system with a clean object store. If an OEM includes this state in their platform, they must support KernelIoControl() with IOCTL_HAL_REBOOT.

```
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Power\State\ColdReboot]
    "Default"=dword:4           ; D4
    "Flags"=dword:800000        ; POWER_STATE_RESET
```

Entering this system power state reboots the system. If an OEM includes this state in their platform, they must support KernelIoControl() with IOCTL_HAL_REBOOT.

```
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Power\State\Reboot]
    "Default"=dword:4           ; D4
    "Flags"=dword:800000        ; POWER_STATE_RESET
```

Entering this system power state shuts down the system. All devices are powered off, resuming may require user intervention. The system will cold boot on resume. Supporting this power state requires that the OEM customize the Power Manager to recognize POWER_STATE_OFF and take platform-specific action to remove power.

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Power\State\ShutDown]
    "Default"=dword:4           ; D4
```

```
    "Flags"=dword:20000         ; POWER_STATE_OFF
```

Default Activity Timers

These registry values set up activity timers inside the Power Manager. GWES and/or other system components need to reset them periodically to keep the associated inactivity event from being set.

Defining timers causes the PM to create a set of named events for resetting the timer and for obtaining its activity status. See the PM documentation for more information.

```
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Power\ActivityTimers\UserActivity]
    "Timeout"=dword:1                       ; in seconds
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Power\ActivityTimers\SystemActivity]
    "Timeout"=dword:1                       ; in seconds

; PM stream interface
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\PMX]
    "Prefix"="PMX"
    "Dll"="pm.dll"
    "Index"=dword:1
    "Order"=dword:99
    "IClass"="{A32942B7-920C-486b-B0E6-92A702A99B35}"    ; PMCLASS_GENERIC_DEVICE

; Shorten timeout time Power Management
[HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Power\Timeouts]
    "ACUserIdle"=dword:00000005         ; in seconds
    "ACSystemIdle"=dword:00000005       ; in seconds
    "BattUserIdle"=dword:00000005       ; in seconds
    "BattSystemIdle"=dword:00000005     ; in seconds
```

22.5 Unit Test

The power applet in the control panel is used to test the Power Manager. The timer settings to transition between system power states can be adjusted and proper behavior can be observed.

22.6 Power Manager API Reference

The Power Manager interfaces with applications and device drivers. See Platform Builder Help for information about defining the program interfaces.

22.6.1 Application Interface

For information about the Power Manager APIs for applications, see the following Platform Builder Help topics:

Developing a Device Driver > **Power Management** > **Power Manager Interfaces** > **Application Interface**
Developing a Device Driver > **Power Management** > **Power Manager Interfaces** > **Notification Interface**
Developing a Device Driver > **Power Management** > **Power Management Reference**

22.6.2 Device Driver Interface

For information about the interface for device drivers, see the following Platform Builder Help topics:

Developing a Device Driver > **Power Management** > **Power Manager Interfaces** > **Device Driver Interface**
Developing a Device Driver > **Power Management** > **Power Management Reference**

Chapter 23
Power Management IC (PMIC)

This chapter provides the information that you need to:

- Develop device drivers that interface directly to the hardware components provided by Freescale Semiconductor's power management ICs (PMICs). The PMIC that is specifically referenced in this document is the MC13783.
- Develop applications that make use of the special hardware capabilities that are provided by the PMIC (for example, audio I/O, USB On-The-Go connectivity, and so on).

This chapter fully describes the API provided by Freescale which allows complete access to the full functionality of the PMICs.

This document is intended for device driver and application developers who need to understand and gain access to the functionality provided by the PMICs.

23.1 PMIC Driver Summary

Table 23-1 identifies the source code location, library dependencies, and other BSP information.

Table 23-1. PMIC Driver Summary

Driver Attribute	Definition
Target Platform (TGTPLAT)	3DS
Target SOC (TGTSOC)	MX31
CSP Driver Path	..\CSP\ARM\FREESCALE\PMIC\MC13783\PDK
CSP Driver Path	N/A
CSP Static Library	pmicPdkCsp_MC13783.lib
Platform Driver Path	..\PLATFORM\<*tgtplat*>\SRC\DRIVERS\PMIC\MC13783\PDK
Import Library	N/A
Driver DLL	pmicPdk_MC13783.dll
Catalog Item	N/A
SYSGEN Dependency	N/A
BSP Environment Variables	BSP_PMIC_MC13783=1

23.2 Supported Functionality

The PMIC device driver framework for Windows CE is a stream interface driver and a SDK DLL.

For a description of the stream interface driver, see the Windows CE Platform Builder Help:

The PMIC Stream Interface driver controls the PMIC hardware directly through the SPI bus. The Stream Interface driver provides an IOCTL interface for SDK DLLs. The SDK DLL provide APIs for Windows CE drivers and applications.

The API describes the PMIC functionality of the following areas:

- Register Access
- Audio
- Battery
- Regulators
- Keys (Power, PTT)
- ADC /Touch
- End of Life comparator
- Power Fail
- Battery Charger
- Vibrator
- GPIO
- CE Bus

23.2.1 PMIC API Framework

The API framework and the APIs defined in this document are intended to be reused for all Freescale power management ICs. The current implementation of the APIs supports the MC13783 power management IC.

The APIs presented in this document were developed to provide a unified interface to all of the functions and features provided by the power management ICs. When a specific function exists on all of the power management ICs, then the API will behave identically (for example, selecting the USB connectivity operating mode).

A device driver and API framework for Windows CE has already been implemented for the MC13783 PMIC. The existing MC13783 framework will be reused as-is, but the APIs will be redefined so that a single unified set of APIs can be used to access the features and functions provided by the MC13783 PMICs.

These are the key objectives and benefits of this new generic PMIC API:

- Provide the ability to easily accommodate additional PMICs in the future (perhaps with additional features or configuration options) without breaking existing software.
- Provide a uniform API for accessing all Freescale PMICs so that device drivers and applications can be ported to different hardware platforms with little or no changes.
- Provide the ability to access all of the underlying hardware capabilities provided by a specific PMIC. Of course, any software that uses PMIC-specific functions or configurations will work only if the appropriate PMIC hardware is available. However, the software should still provide

appropriate error codes and “fail gracefully” if it is used on a platform for which the requested function or configuration is not supported.

23.3 Hardware Operation

Refer to the MC13783 (MC13783) document for details on the MC13783 PMIC.

23.3.1 MX31 Peripheral Conflicts

There are no i.MX31 pin conflicts on the 3-Stack board. The CSPI2 is used to communicate with the MC13783 PMIC on the i.MX31 platform, and the CSPI2 signals are selected in the IOMUX.

23.4 Software Operation

23.4.1 Configuring the PMIC

The PMIC modules can be used by applications or device drivers. For example, the Battery APIs of the PMIC will be used by the Battery driver.

Configuring the PMIC port for communications requires that a handle to the desired PMIC port is opened prior to accessing the module registers. This handle is needed to call the **DeviceIoControl** function. The function parameters include the PMIC port handle, appropriate IOCTL code, and other input and output parameters.

23.4.2 Creating a Handle to the PMIC

Before calling any of the PMIC API’s make sure that PMIC device is attached by calling the **CreateFile** function which opens a file and it returns a handle that can be used to access the MC13783 hardware. If MC13783 hardware does not exist, **CreateFile** returns ERROR_FILE_NOT_FOUND.

To open a handle to the PMIC:

1. Insert a colon after the PMI1 port for the first parameter, *lpFileName*. For example, specify PMI1: as the PMIC port.
2. Specify FILE_SHARE_READ | FILE_SHARE_WRITE in the *dwShareMode* parameter. Multiple handles to a PMIC port are supported by the driver.
3. Specify OPEN_EXISTING in the *dwCreationDisposition* parameter. This flag is required.
4. Specify FILE_FLAG_RANDOM_ACCESS in the *dwFlagsAndAttributes* parameter.

The following code example shows how to open a PMIC port.

```
hPMI = CreateFile(TEXT("PMI1:"), GENERIC_READ | GENERIC_WRITE,  access (read-write) mode
            FILE_SHARE_READ | FILE_SHARE_WRITE, NULL, OPEN_EXISTING,  sharing mode
            FILE_FLAG_RANDOM_ACCESS, NULL);  security attributes
        if ((hPMI == NULL) || (hPMI == INVALID_HANDLE_VALUE))
        {
            ERRORMSG(1, (_T("Failed in create File()\r\n")));
        }
```

NOTE

All the above specified steps are performed when PMIC devices attach is called. If hPMI handle is null then perform the above steps.

23.4.3 Write Operations

The PMIC driver does not provide an interface to write through the PMIC_Write (stream write) function in PMIC driver. PMIC_Write is a stub function and returns success always.

23.4.4 Read Operations

Like the write operation, the PMIC driver does not provides an interface to read through the PMIC_Read function in PMIC driver. This is a stub function and returns success always.

23.4.5 Closing the Handle to the PMIC

Call the **CloseHandle** function to close a handle to the PMIC when an application is done using it.

CloseHandle has one parameter, which the handle is returned by the CreateFile function call that opened the PMIC port.

23.4.6 Power Management

The primary method for limiting power consumption in the PMIC module is to gate off all clocks to the module when those clocks are not needed. This is accomplished through the **DDKClockSetGatingMode** function call. PMIC module clock is enabled whenever any of the PMIC registers needs to be accessed and then disabled once it is done.

23.4.6.1 PowerUp

This function is not implemented for the PMIC driver.

23.4.6.2 PowerDown

This function is not implemented for the PMIC driver.

23.4.6.3 IOCTL_POWER_CAPABILITIES

The power management capabilities are advertised with power manager through this IOCTL. The PMIC module supports only two power states: D0 and D4.

23.4.6.4 IOCTL_POWER_SET

This IOCTL requests a change from one device power state to another. D0 and D4 are the only two supported **CEDEVICE_POWER_STATE** in PMIC driver. Any request that is not D0 is changed to a D4 request. This results in the system entering into suspend state. For a value of D0 the system will again be resumed.

23.4.6.5 IOCTL_POWER_GET

This IOCTL returns the current device power state. By design, the Power Manager knows the device power state of all power-manageable devices. It will not generally issue an **IOCTL_POWER_GET** call to the device unless an application calls **GetDevicePower** with the POWER_FORCE flag set.

23.4.7 PMIC Registry Settings

There are no registry settings that need to be modified to use the PMIC API's.

23.4.8 A/D Converter and Touch

The ADC is a 16 channel, 10-bit converter with a state machine to control the various models of operation. Read and write access to the A/D converter is accomplished through the SPI bus.

Table 23-2. A/D Converter and Touch

MC13783 A/D Channel Definition and Scanning Table		
AD_SEL	ADA[2:0]	Signal Read
0	000	BATT
0	001	BATTISNS
0	010	BPSNS
0	011	CHRGRAW
0	100	CHRGISNS
0	101	ADIN5/PTHEN
0	110	ADIN6/LICELL
0	111	ADIN7/DTHEN
1	000	ADIN8
1	001	ADIN9
1	010	ADIN10
1	011	ADIN11
1	100	TSX1
1	101	TSX2
1	110	TSY1
1	111	TSY2

MC13783 has a touch screen interrupt. This interrupt occurs when a pen-down event is detected. The Windows CE touch driver should handle these interrupt events. Please refer to Section 23.6.2, "Interrupt Handling for a description of interrupt handling.

23.4.8.1 Data Types

```
typedef enum _PMIC_ADC_CONVERTOR_MODE
{
      ADC_8CHAN_1X = 0,   // RAND = 0, 8 channels
      ADC_1CHAN_8X        // RAND = 1, reads 8 sequential values
} PMIC_ADC_CONVERTOR_MODE;
Touch Modes
typedef enum _MC13783_TOUCH_MODE {
      TM_INACTIVE = 0,
      TM_INTRUPT,
      TM_RESISTIVE,
      TM_POSITION,
} MC13783_TOUCH_MODE;
```

23.4.8.2 Functions

23.4.8.2.1 PmicADCGetSingleChannelOneSample

This function gets one channel one sample.

Prototype *PMIC_STATUS PmicADCGetSingleChannelOneSample(UINT16 channel, UINT16* pResult);*

Parameters: *channel [in]*

A selected channel.

pResult [out]

Pointer to the sampled value.

Returns: PMIC_STATUS

23.4.8.2.2 PmicADCGetSingleChannelEightSamples

This function gets one channel eight samples.

Prototype *PMIC_STATUS PmicADCGetSingleChannelEightSamples(UINT16 channel, UINT16* pResult);*

Parameters: *channel [in]*

A selected channel.

pResult [out]

Pointer to the sampled values (up to 8 sampled values).

Returns: PMIC_STATUS

23.4.8.2.3 PmicADCGetMultipleChannelsSamples

This function gets sample for multiple channels.

Prototype `PMIC_STATUS PmicADCGetMultipleChannelsSamples(UINT16 channels, UINT16* pResult);`

Parameters: `channels [in]`

Selected channels (up to 16 channels).

`pResult [out]`

Pointer to the sampled values (up to 16 sampled values).

Returns: PMIC_STATUS

23.4.8.2.4 PmicADCTouchRead

This function reads a touch screen sample.

Prototype `PMIC_STATUS PmicADCTouchRead(UINT16* x, UINT16* y);`

Parameters: `x [out]`

X-coordinate of the point.

`y [out]`

Y-coordinate of the point.

Returns: PMIC_STATUS

Remarks This function reads 3 pairs of samples for MC13783.

23.4.8.2.5 PmicADCTouchStandby

This function causes the PMIC touch screen controller to enter standby mode and wait for the next pen down condition.

Prototype `PMIC_STATUS PmicADCTouchStandby(bool intEna);`

Parameters: `intEna [in]`

interrupt enable.

Returns: PMIC_STATUS

23.4.8.2.6 PmicADCSetComparatorThresholds

This function sets WHIGH and WLOW for automatic ADC result comparators.

Prototype `PMIC_STATUS PmicADCSetComparatorThresholds(UINT16 whigh, UINT16 wlow);`

Parameters: `whigh [in]`

a high comparator threshold.

`wlow [in]`

a low comparator threshold.

Returns: PMIC_STATUS

23.4.8.2.7 PmicADCGetHandsetCurrent

This function gets handset battery current measurement values.

Prototype `PMIC_STATUS PmicADCGetHandsetCurrent(PMIC_ADC_CONVERTOR_MODE mode,UINT16 *pResult);`

Parameters: `mode [in]`

An ADC converter mode: ADC_8CHAN_1X or ADC_1CHAN_8X

`pResult [out]`

Pointer to the handset battery current measurement value(s)

Returns: PMIC_STATUS

Remarks ADC_8CHAN_1X Mode:

This function returns one sample for battery current channel (BATTISNS).

ADC_1CHAN_8X Mode:

For MC13783, this function returns the 8 samples for battery current channel order like : ADA0=BATT; ADA1= BATT-BATTISNS; ADA2=BATT; ADA3= BATT-BATTISNS; ADA4=BATT; ADA5= BATT-BATTISNS; ADA6=BATT; ADA7= BATT-BATTISNS.

23.4.8.2.8 PmicADCInit

This function initializes PMIC ADC's resources.

Prototype `PMIC_STATUS PmicADCInit(void);`

Parameters: None.

Returns: PMIC_STATUS

23.4.8.2.9 PmicADCDeinit

This function deinitializes PMIC ADC's resources.

Prototype `void PmicADCDeinit(void);`

Parameters: None

Returns: PMIC_STATUS

23.4.8.3 Power Management

There is no additional power management implementation done specifically for Atlas ADC other than the implementation described in section Power Management of this document.

23.5 Unit Test

The PMIC CETK test cases verify the functionality of the various PMIC components.

23.5.1 Unit Test Hardware

The 3-Stack board is required.

23.5.2 Unit Test Software

Table 23-3 lists the software required to run the unit tests.

Table 23-3. Software Requirements to Run Unit Tests

Requirements	Description
Tux.exe	Tux test harness, which is needed for executing the test
Kato.dll	Kato logging engine, which is required for logging test data
Tooltalk.dll	Library required by Tux.exe and Kato.dll. Handles the transport between the target device and the development workstation
PMICtest.dll	Test .dll file

23.5.3 Building the PMIC Tests

In order to build the PMIC tests, you need to build an OS image for the desired configuration.

To build the OS image:

1. In Platform Builder, from the **Build OS** menu, select **Open Release Directory**. This opens a DOS prompt.
2. Change to the PMIC Tests directory. (`\WINCE500\SUPPORT\TESTS\PMIC`)
3. Enter **set WINCEREL=1** on the command prompt and hit return. This copies the built DLL to the flat release directory.
4. Enter the build command **build -c** at the prompt and press Enter.

After the build completes, the `pmictest.dll` file will be located in the `$(_FLATRELEASEDIR)` directory.

23.5.4 Running the PMIC Tests

The command line for running the PMIC tests is:

```
tux –o –d pmictest
```

To redirect the test results to a file, add the **–f** option. The PMIC tests do not contain any test specific command line options.

Table 23-4 describes the test cases contained in the PMIC tests.

Table 23-4. PMIC Test Cases

#	Test name	Description
1	PMIC Register Access	This test does read/write verification of IMR register on the PMIC.
5	Power Control	This test verifies the power control functionality on the PMIC.
6	AdcGetOneSample	This test gets one sample from each of 16 channels by requesting the driver to sample one channel at a time.
7	AdcGet8Samples	This test gets 8 samples from each of 16 channels.
8	AdcGetMultiChannelSamples	This test gets one sample from each of 16 channels by requesting the driver to sample all 16 channels at once.
9	AdcGetHandsetCurrent	This test gets samples of handset current.
10	AdcTouchRead	This test gets 3 (x,y) coordinates from the touch screen.

23.6 PMIC Reference API

23.6.1 PMIC Driver IOCTLS

This section describes the PMIC I/O control codes (IOCTLs). These IOCTLs are used in calls to DeviceIoControl to issue commands to the PMIC device modules. Only relevant parameters for the IOCTL are described. These IOCTLs are used within the APIs developed for specific modules of the PMIC device. Most of the IOCTLs are described in other, relevant sections.

23.6.1.1 PMIC_IOCTL_LLA_READ_REG

This **DeviceIoControl** request reads the register content.

Parameters

hPMI

[in] Handle to the device that is to perform the operation. To obtain a device handle, call the CreateFile function.

lpInBuffer

index of the register.

lpOutBuffer

[out] Long pointer to a buffer that receives the output data for the operation. Set to NULL if the dwIoControlCode parameter specifies an operation that does not produce output data.

23.6.1.2 PMIC_IOCTL_LLA_WRITE_REG

This **DeviceIoControl** request writes the data to the said register of the PMIC device.

Parameters *hPMI*

[in] Handle to the device that is to perform the operation. To obtain a device handle, call the CreateFile function.

lpInBuffer

index of the register.

lpOutBuffer

pointer to data which needs to be written to the said register.

23.6.1.3 PMIC_IOCTL_LLA_INT_REGISTER

This **DeviceIoControl** is used to register interrupt.

Parameters *hPMI*

[in] Handle to the device that is to perform the operation. To obtain a device handle, call the CreateFile function.

lpInBuffer

index of the register.

lpOutBuffer

pointer to event name and interrupt id.

Code example:

```
param.int_id = int_id;
param.event_name = event_name;
ret = DeviceIoControl(hPMI, PMIC_IOCTL_LLA_INT_REGISTER, &param,
      sizeof(param), NULL, 0, NULL, NULL);
```

23.6.1.4 PMIC_IOCTL_LLA_INT_DEREGISTER

This **DeviceIoControl** is used to deregister PMIC interrupt.

Parameters *hPMI*

[in] Handle to the device that is to perform the operation. To obtain a device handle, call the CreateFile function.

lpInBuffer

index of the register.

lpOutBuffer

null.

Code example:

```
param.int_id = int_id;
ret = DeviceIoControl(hPMI, PMIC_IOCTL_LLA_INT_DEREGISTER, &param,
      sizeof(param), NULL, 0, NULL, NULL);
```

23.6.1.5 PMIC_IOCTL_LLA_INT_COMPLETE

Parameters *hPMI*

[in] Handle to the device that is to perform the operation. To obtain a device handle, call the CreateFile function.

lpInBuffer

index of the register.

lpOutBuffer

pointer to interrupt id.

Code example:

```
param.int_id = int_id;
ret = DeviceIoControl(hPMI, PMIC_IOCTL_LLA_INT_COMPLETE, &param,
      sizeof(param), NULL, 0, NULL, NULL); PMIC_IOCTL_LLA_INT_ENABLE
```

This I/O control is used to enable the interrupt.

Parameters *hPMI*

[in] Handle to the device that is to perform the operation. To obtain a device handle, call the CreateFile function.

lpInBuffer

index of the register.

lpOutBuffer

pointer to interrupt id.

Code example :

```
param.int_id = int_id;
ret = DeviceIoControl(hPMI, PMIC_IOCTL_LLA_INT_COMPLETE, &param,
      sizeof(param), NULL, 0, NULL, NULL);
```

23.6.1.6 PMIC_IOCTL_LLA_INT_DISABLE

This I/O control is used to disable the interrupt.

Parameters *hPMI*

[in] Handle to the device that is to perform the operation. To obtain a device handle, call the CreateFile function.

lpInBuffer

index of the register.

lpOutBuffer

pointer to interrupt id.

Code example :

```
param.int_id = int_id;
ret = DeviceIoControl(hPMI, PMIC_IOCTL_LLA_INT_COMPLETE, &param,
      sizeof(param), NULL, 0, NULL, NULL);
```

23.6.2 Interrupt Handling

23.6.2.1 About Interrupt Handling

The PMIC has interrupt generation capability to inform the CPU when events occur. This is signaled to the processors driving the primary SPI and secondary SPI busses through the PRIINT and SECINT lines, respectively. There is only one interrupt line connected to each processor, so the kernel can only know that there is an interrupt from the PMIC, but without knowing exactly which module generated the interrupt.

There is one PMIC Interrupt Service Thread (IST) to handle all interrupts from the PMIC. The PMIC IST will be invoked by the kernel once the kernel receives an interrupt from the PMIC.

This IST will first query the PMIC to determine the source of the interrupt. The IST maintains a table to track if an interrupt has been registered by a driver or application. If the interrupt is registered, the IST will then set a predefined event.

For any drivers and applications that need notification of an interrupt, they must register the interrupt and wait for the event. They also need to reset the event after handling the event.

23.6.2.2 Interrupt Events

Drivers or applications that want to monitor an interrupt should create a named event for each interrupt. The event name is passed to PMIC driver when registering the interrupt.

The PMIC IST will trigger the event when the corresponding interrupt occurs.

23.6.2.2.1 PMIC Interrupt Events

Table 23-5 shows the events and corresponding MC13783 interrupts.

Table 23-5. PMIC Interrupt Events

PMIC Interrupt	Description
ADCDONEI	ADC has finished requested conversions
ADCBISDONEI	ADCBIS has finished requested conversions
TSI	Touch screen wakeup
WHI	A/D word read in ADC digital comparison mode exceeding the high limit
WLI	A/D word read in ADC digital comparison mode reading below the low limit
CHGDETI	Charger attach and removal
CHGOVI	Charger over-voltage detection
CHGREVI	Charger path reverse current
CHGSHORTI	Charger path short circuit
CCCVI	BP regulator current or voltage regulation. Indicates that the charger has switched its mode from CC to CV or from CV to CC. Charger removal does not trigger this interrupt.
CHGCURRI	Charge current has dropped below threshold
BPONI	BP turn on threshold detection
LOBATLI	End of lift/low battery detection
LOBATHI	Low battery warning
USBI	USB VBUS detection
IDI	USB ID line detection
SE1I	Single ended 1 detection
CKDETI	Carkit detection
1HZI	1HZ timetick
TODAI	Time of day alarm. Triggered when TOD counter is equal to the value is TODA and the DAY counter is equal to the value in DAYA.
ONOFD1I	ON1B event. Connection for a power on/off button.
ONOFD2I	ON2B event. Connection for an accessory power on/off button
ONOFD3I	ON3B event. Connection for a third power on/off button.
SYSRSTI	Indicates system reset has occurred
RTCRSTI	Indicates RTC reset has occurred
PCI	Indicates power cut has occurred
WARMI	Warm start event. Indicates the application powered up from user off mode.
MEMHLDI	Memory hold event. Indicates the application powered up from memory hold mode.
PWRRDYI	Power gate and DVS power ready

Table 23-5. PMIC Interrupt Events(Continued)

PMIC Interrupt	Description
THWARNLI	Thermal warning lower threshold
THWARNHI	Thermal warning higher threshold
CLKI	Clock source change
SEMAFI	Semaphore
MC2BI	Microphone bias 2 detect
HSDETI	Headset attach
HSLI	Stereo headset detect
ALSPTHI	Thermal shutdown Alsp. Maximum allowable junction temperature within Alsp is reached.
AHSSHORTI	Short circuit on Ahs outputs

23.6.2.3 Interrupt Data Structures

```
typedef enum _PMIC_MC13783_INT_ID {
    PMIC_MC13783_INT_ADCDONEI = 0,
    PMIC_MC13783_INT_ADCBISDONEI = 1,
    PMIC_MC13783_INT_TSI = 2,
    PMIC_MC13783_INT_WHI = 3,
    PMIC_MC13783_INT_WLI = 4,
    PMIC_MC13783_INT_CHGDETI = 6,
    PMIC_MC13783_INT_CHGOVI = 7,
    PMIC_MC13783_INT_CHGREVI = 8,
    PMIC_MC13783_INT_CHGSHORTI = 9,
    PMIC_MC13783_INT_CCCVI = 10,
    PMIC_MC13783_INT_CHGCURRI = 11,
    PMIC_MC13783_INT_BPONI = 12,
    PMIC_MC13783_INT_LOBATLI = 13,
    PMIC_MC13783_INT_LOBATHI = 14,
    PMIC_MC13783_INT_USBI = 16,
    PMIC_MC13783_INT_IDI = 19,
    PMIC_MC13783_INT_SE1I = 21,
    PMIC_MC13783_INT_CKDETI = 22,
    PMIC_MC13783_INT_1HZI = 32,
    PMIC_MC13783_INT_TODAI = 33,
    PMIC_MC13783_INT_ONOFD1I = 35,
    PMIC_MC13783_INT_ONOFD2I = 36,
    PMIC_MC13783_INT_ONOFD3I = 37,
    PMIC_MC13783_INT_SYSRSTI = 38,
    PMIC_MC13783_INT_RTCRSTI = 39,
    PMIC_MC13783_INT_PCI = 40,
    PMIC_MC13783_INT_WARMI = 41,
    PMIC_MC13783_INT_MEMHLDI = 42,
    PMIC_MC13783_INT_PWRRDYI = 43,
    PMIC_MC13783_INT_THWARNLI = 44,
    PMIC_MC13783_INT_THWARNHI = 45,
    PMIC_MC13783_INT_CLKI = 46,
    PMIC_MC13783_INT_SEMAFI = 47,
    PMIC_MC13783_INT_MC2BI = 49,
    PMIC_MC13783_INT_HSDETI = 50,
```

```
    PMIC_MC13783_INT_HSLI = 51,
    PMIC_MC13783_INT_ALSPTHI = 52,
    PMIC_MC13783_INT_AHSSHORTI = 53,
    PMIC_INT_MAX_ID
} PMIC_MC13783_INT_ID;
```

23.6.2.4 Functions

23.6.2.4.1 PmicInterruptRegister

PmicInterruptRegister function registers an interrupt so that the interrupt event will be signaled when the interrupt occurs.

All PMIC interrupts are masked at the initialization. A driver or an application must register the interrupt if the interrupt is to be enabled.

Prototype *PMIC_STATUS PmicInterruptRegister(PMIC_INT_ID int_id, LPTSTR name);*

Parameters: *int_id*

[in] The interrupt to be registered

name

[in] The event name

Return Value Status code

Remarks:

In this function the PMIC_IOCTL_LLA_INT_REGISTER IOCTL code is used and below is the code example.

```
 param.int_id = int_id;
param.event_name = event_name;
ret = DeviceIoControl(hPMI, PMIC_IOCTL_LLA_INT_REGISTER, &param,
          sizeof(param), NULL, 0, NULL, NULL);
if (ret)
{
    return PMIC_SUCCESS;
}
else
{
    return PMIC_ERROR;
}
```

23.6.2.4.2 PmicInterruptDeregister

PmicInterruptDeregister function deregisters an interrupt. If an interrupt is not registered by any driver or application, it will be masked.

Prototype *PMIC_STATUS PmicInterruptDeregister(PMIC_INT_ID int_id);*

Parameters *int_id*

[in] The interrupt to be deregistered

Return Value Status code

Remarks

In this function the PMIC_IOCTL_LLA_INT_DEREGISTE call is used and below is the code example

```
param = int_id;

ret = DeviceIoControl(hPMI, PMIC_IOCTL_LLA_INT_DEREGISTER, &param,
          sizeof(param), NULL, 0, NULL, NULL);
if (ret)
{
    return PMIC_SUCCESS;
}
else
{
    return PMIC_ERROR;
}
```

23.6.2.4.3 PmicInterruptHandlingComplete

PmicInterruptHandlingComplete function notifies the PMIC stream interface driver completion of an interrupt handling, so that the stream interface driver can enable that interrupt again.

Prototype *PMIC_STATUS PmicInterruptHandlingComplete(PMIC_INT_ID int_id);*

Parameters int_id

[in] The interrupt index.

Return Value Status code

Remarks

In this function the PMIC_IOCTL_LLA_INT_COMPLETE call is used and below is the code example

```
param = int_id;
ret = DeviceIoControl(hPMI, PMIC_IOCTL_LLA_INT_COMPLETE, &param,
          sizeof(param), NULL, 0, NULL, NULL);
if (ret)
  {
     return PMIC_SUCCESS;
  }
  else
  {
     return PMIC_ERROR;
   }
```

23.6.2.4.4 PmicInterruptDisable

PmicInterruptDisable function temporarily disables an interrupt. The interrupt is still registered. The driver or application can enable the interrupt again by calling PmicInterruptEnable().

Prototype *PMIC_STATUS PmicInterruptDisable(PMIC_INT_ID int_id);*

Parameters *int_id*

[in] The interrupt index.

Return Value Status code

Remarks

In this function the PMIC_IOCTL_LLA_INT_DISABLE call is used and below is the code example

```
param = int_id;
ret = DeviceIoControl(hPMI, PMIC_IOCTL_LLA_INT_DISABLE, &param,
          sizeof(param), NULL, 0, NULL, NULL);
if (ret)
{
    return PMIC_SUCCESS;
}
else
{
    return PMIC_ERROR;
}
```

23.6.2.4.5 PmicInterruptEnable

PmicInterruptEnable function re-enable an interrupt.

Prototype *PMIC_STATUS PmicInterruptEnable(PMIC_INT_ID int_id);*

Parameters *int_id*

[in] The interrupt index.

Return Value Status code

Remarks

In this function the PMIC_IOCTL_LLA_INT_ENABLE call is used and below is the code example

```
 param = int_id;
ret = DeviceIoControl(hPMI, PMIC_IOCTL_LLA_INT_ENABLE, &param,
          sizeof(param), NULL, 0, NULL, NULL);
if (ret)
{
    return PMIC_SUCCESS;
}
else
{
    return PMIC_ERROR;
}
```

Code example of registering PMIC pen down interrupts.

```
if (PmicInterruptRegister(PMIC_MC13783_INT_TSI, _T("EVENT_TS"))
        != PMIC_SUCCESS)
    {
        ERRORMSG(1, (_T("PmicInterruptRegister failed\r\n")));
        goto cleanUp;
    }
```

Deregister for PMIC pen down interrupts.

```
PmicInterruptDeregister(PMIC_MC13783_INT_TSI);
```

23.6.3 Register Access API

The PMIC Low Level Access API allows drivers and/or applications to read and write PMIC registers. There are some restrictions to prohibit drivers/application from accessing some registers. Interrupt registers is one example. The interrupt library functions will be in this Low Level Access DLL.

23.6.3.1 Functions

23.6.3.1.1 Read Register

This function reads a PMIC register.

Prototype *PMIC_STATUS*
PmicRegisterRead(unsigned char index, UINT32 reg);*

Parameters *index*

[in] register index.

reg

[out] The contents of the register.

Return Value Status code

23.6.3.1.2 Write Register

This function writes a PMIC register.

Prototype *PMIC_STATUS*
PmicRegisterWrite(unsigned char index, UINT32 reg, UINT32 mask);

Parameters *index*

[in] register index.

reg

[in] data to be written.

mask

[in] bitmap mask to indicate which bits in parameter reg should be written to PMIC register.

Return Value Status code

The following code example shows how to use PmicRegisterWrite / PmicRegisterRead function to write/read to/from the PMIC module registers.

```
TESTPROCAPI PMICTest1(UINT uMsg, TPPARAM tpParam, LPFUNCTION_TABLE_ENTRY lpFTE)
{
    UINT32 reg;

     Validate that the shell wants the test to run
    if (uMsg != TPM_EXECUTE)
    {
        return TPR_NOT_HANDLED;
    }
    g_pKato>Log(LOG_COMMENT, TEXT("PMICTest1() +\r\n"));
     Read IMR
    PmicRegisterRead(1, &reg);
    g_pKato>Log(LOG_COMMENT, TEXT("Register IMR is 0X%X\r\n"), (reg & 0xFFFFFF));
    g_pKato>Log(LOG_COMMENT, TEXT("Now, try to change IMR to 0x0FF\r\n"));
    PmicRegisterWrite(1, 0x0FF, 0xFFFFFF);
    PmicRegisterRead(1, &reg);
    g_pKato>Log(LOG_COMMENT, TEXT("Register IMR is 0X%X\r\n"), (reg & 0xFFFFFF));
     Enter ISR loop
    if ((reg&0xFFFFFF) == 0xFF)
    {
        GPT_TEST_FUNCTION_EXIT();
        return TPR_PASS;
    }
    else
    {
        GPT_TEST_FUNCTION_EXIT();
        return TPR_FAIL;
    }
}
```

23.6.4 Power Control Reference

23.6.4.1 PwCtrl API

This section provides information about API provided by PwCtrl API DLL. Using the APIs listed in Table 23-6, MC13783 Power control module can be accessed.

Table 23-6. MC13783 Power Control Module

Module	Usage
PmicPwrctrlSetPowerCutTimer	used to set the power cut timer duration
PmicPwrctrlGetPowerCutTimer	used to get the power cut timer duration
PmicPwrctrlEnablePowerCut	used to enable the power cut
PmicPwrctrlDisablePowerCut	used to disable the power cut
PmicPwrctrlSetPowerCutCounter	used to set the power cut counter
PmicPwrctrlGetPowerCutCounter	used to get the power cut counter
PmicPwrctrlSetPowerCutMaxCounter	used to set the maximum number of power cut counter

Table 23-6. MC13783 Power Control Module(Continued)

Module	Usage
PmicPwrctrlGetPowerCutMaxCounter	used to get the setting of maximum power cut counter
PmicPwrctrlEnableCounter	function will set PC_COUNT_EN=1
PmicPwrctrlDisableCounter	function will set PC_COUNT_EN=0
PmicPwrctrlSetMemHoldTimer	used to set the duration of memory hold timer
PmicPwrctrlGetMemHoldTimer	used to get the setting of memory hold timer
PmicPwrctrlSetMemHoldTimerAllOn	used to set the duration of the memory hold timer to infinity
PmicPwrctrlClearMemHoldTimerAllOn	used to clear the infinity duration of the memory hold timer
PmicPwrctrlEnableClk32kMCU	used to enable the CLK32KMCU
PmicPwrctrlDisableClk32kMCU	used to disable the CLK32KMCU
PmicPwrctrlEnableUserOffModeWhenDelay	used to place the phone in User Off Mode after a delay
PmicPwrctrlDisableUserOffModeWhenDelay	used to set not to place the phone in User Off Mode after a delay
PmicPwrctrlSetVBKUPRegulator	used to set the VBKUP regulator
PmicPwrctrlSetVBKUPRegulatorVoltage	used to set the VBKUP regulator voltage
PmicPwrctrlEnableWarmStart	used to set the phone to transit from the ON state to the User Off state when either the USER_OFF pin is pulled high or the USER_OFF_SPI bit is set
PmicPwrctrlDisableWarmStart	This function is used to disable the warm start and set the phone to transit from the ON state to the MEMHOLD ONLY state when either the USER_OFF pin is pulled high or the USER_OFF_SPI bit is set
PmicPwrctrlEnableRegenAssig	Used enables the REGEN pin of selected voltage regulator
PmicPwrctrlDisableRegenAssig	Used disable the REGEN pin of selected voltage regulator
PmicPwrctrlGetRegenAssig	reads the REGEN pin value for said voltage regulator

23.6.4.2 Functions and Data Structures

```
PMIC_STATUS PmicPwrctrlSetPowerCutTimer (UINT8 duration);
PMIC_STATUS PmicPwrctrlGetPowerCutTimer (UINT8* duration);
PMIC_STATUS PmicPwrctrlEnablePowerCut (void);
PMIC_STATUS PmicPwrctrlDisablePowerCut (void);
PMIC_STATUS PmicPwrctrlSetPowerCutCounter (UINT8 counter);
PMIC_STATUS PmicPwrctrlGetPowerCutCounter (UINT8* counter);
PMIC_STATUS PmicPwrctrlSetPowerCutMaxCounter (UINT8 counter);
PMIC_STATUS PmicPwrctrlGetPowerCutMaxCounter (UINT8* counter);
PMIC_STATUS PmicPwrctrlEnableCounter(void);
PMIC_STATUS PmicPwrctrlDisableCounter (void);
PMIC_STATUS PmicPwrctrlSetMemHoldTimer (UINT8 duration);
PMIC_STATUS PmicPwrctrlGetMemHoldTimer (UINT8* duration);
PMIC_STATUS PmicPwrctrlSetMemHoldTimerAllOn (void);
PMIC_STATUS PmicPwrctrlClearMemHoldTimerAllOn (void);
PMIC_STATUS PmicPwrctrlEnableClk32kMCU (void);
```

```
PMIC_STATUS PmicPwrctrlDisableClk32kMCU (void);
PMIC_STATUS PmicPwrctrlEnableUserOffModeWhenDelay (void);
PMIC_STATUS PmicPwrctrlDisableUserOffModeWhenDelay (void);
PMIC_STATUS              PmicPwrctrlSetVBKUPRegulator              (MC13783_PWRCTRL_REG_VBKUP,
MC13783_PWRCTRL_VBKUP_MODE);
PMIC_STATUS PmicPwrctrlSetVBKUPRegulatorVoltage (MC13783_PWRCTRL_REG_VBKUP, UINT8);
PMIC_STATUS PmicPwrctrlEnableWarmStart (void);
PMIC_STATUS PmicPwrctrlDisableWarmStart (void);
PMIC_STATUS PmicPwrctrlEnableRegenAssig (t_regulator regu);
PMIC_STATUS PmicPwrctrlDisableRegenAssig (t_regulator regu);
PMIC_STATUS PmicPwrctrlGetRegenAssig (t_regulator regu, UINT8* value);

 The backup regulators VBKUP1 and VBKUP2 provide two independent low power
 supplies during memory hold, user off and power cut operation.
typedef enum _MC13783_PWRCTRL_REG_VBKUP{
   VBKUP1,
   VBKUP2,
} MC13783_PWRCTRL_REG_VBKUP;

typedef enum _MC13783_PWRCTRL_VBKUP_MODE{
   VBKUP_MODE1,    Backup Regulator Off in Non Power Cut Modes and Off in Power Cut Modes
   VBKUP_MODE2,    Backup Regulator Off in Non Power Cut Modes and On in Power Cut Modes
   VBKUP_MODE3,    Backup Regulator On in Non Power Cut Modes and Off in Power Cut Modes
   VBKUP_MODE4,    Backup Regulator On in Non Power Cut Modes and On in Power Cut Modes
} MC13783_PWRCTRL_VBKUP_MODE;

/*!
 * This enumeration define all regulator enabled by regen
 */
typedef enum {
        /*!
         * VAudio
         */
        REGU_VAUDIO=0,
        /*!
         * VIOHI
         */
        REGU_VIOHI,
        /*!
         * VIOLO
         */
        REGU_VIOLO,
        /*!
         * VDIG
         */
        REGU_VDIG,
        /*!
         * VGEN
         */
        REGU_VGEN,
        /*!
         * VRFDIG
```

```
 */
REGU_VRFDIG, /*5*/
/*!
 * VRFREF
 */
REGU_VRFREF,
/*!
 * VRFCP
 */
REGU_VRFCP,
/*!
 * VSIM
 */
REGU_VSIM,
/*!
 * VESIM
 */
REGU_VESIM,
/*!
 * VCAM
 */
REGU_VCAM, /*10*/
/*!
 * VRFBG
 */
REGU_VRFBG,
/*!
 * VVIB
 */
REGU_VVIB,
/*!
 * VRF1
 */
REGU_VRF1,
/*!
 * VRF2
 */
REGU_VRF2,
/*!
 * VMMC1
 */
REGU_VMMC1,
/*!
 * VMMC2
 */
REGU_VMMC2,
/*!
 * GPO1
 */
REGU_GPO1,
/*!
 * GPP2
```

```
         */
        REGU_GPO2,
        /*!
         * GPO3
         */
        REGU_GPO3,
        /*!
         * GPO4
         */
        REGU_GPO4,
        /*!
         * REGU_NUMBER
         */
        REGU_NUMBER,
} t_regulator;
/*!
 * This tab define bit for regen of all regulator
 */
int   REGULATOR_REGEN_BIT[REGU_NUMBER]={
        0, /* VAUDIO */
        1, /* VIOHI  */
        2, /* VIOLO  */
        3, /* VDIG   */
        4, /* VGEN   */
        5, /* VRFDIG */
        6, /* VRFREF */
        7, /* VRFCP  */
        -1,  /* VSIM   */
        -1,  /* VESIM  */
        8, /* VCAM   */
        9, /* VRFBG  */
        -1,  /* VVIB   */
        10, /* VRF1   */
        11, /* VRF2   */
        12, /* VMMC1  */
        13, /* VMMC2  */
        16, /* VGPO1  */
        17, /* VGPO2  */
        18, /* VGPO3  */
        19, /* VGPO4  */
};
```

23.6.4.2.1 PmicPwrctrlSetPowerCutTimer

Prototype `PMIC_STATUS PmicPwrctrlSetPowerCutTimer (UINT8 duration);`

This function is used to set the power cut timer duration.

Parameters: `duration [in]`

The value to set to power cut timer register, it's from 0 to 255.
The timer will be set to a duration of 0 to 31.875 seconds, in 125 ms increments.

Returns: *status*

PMIC_SUCCESS for success and PMIC_ERROR for failure

Remarks

23.6.4.2.2 PmicPwrctrlGetPowerCutTimer

Prototype `PMIC_STATUS PmicPwrctrlGetPowerCutTimer (UINT8* duration);`

Parameters: `duration [out]`

The duration to set to power cut timer

Returns: *status*

PMIC_SUCCESS for success and PMIC_ERROR for failure.

23.6.4.2.3 PmicPwrctrlEnablePowerCut

Prototype `PMIC_STATUS PmicPwrctrlEnablePowerCut (void);`

This function is used to enable the power cut.

Parameters: None

Returns: *status*

PMIC_SUCCESS for success and PMIC_ERROR for failure.

23.6.4.2.4 PmicPwrctrlDisablePowerCut

Prototype `PMIC_STATUS PmicPwrctrlDisablePowerCut (void)`

This function is used to disable the power cut.

Parameters: None

Returns: *status*

PMIC_SUCCESS for success and PMIC_ERROR for failure

23.6.4.2.5 PmicPwrctrlSetPowerCutCounter

Prototype `PMIC_STATUS PmicPwrctrlSetPowerCutCounter (UINT8 counter);`

This function is used to set the power cut counter.

Parameters: `counter [in]`

The counter number value to be set to the register. It's value from 0 to 15. The power cut counter is a 4 bit counter that keeps track of the number of rising edges of the UV_TIMER (power cut events) that have occurred since the counter was last initialized.

Returns: *status*

PMIC_SUCCESS for success and PMIC_ERROR for failure

23.6.4.2.6 PmicPwrctrlGetPowerCutCounter

Prototype `PMIC_STATUS PmicPwrctrlGetPowerCutCounter (UINT8* counter);`

This function is used to get the power cut counter.

Parameters: `counter [out]`

to get the counter number

Returns: *status*

PMIC_SUCCESS for success and PMIC_ERROR for failure

23.6.4.2.7 PmicPwrctrlSetPowerCutMaxCounter

Prototype `PMIC_STATUS PmicPwrctrlSetPowerCutMaxCounter (UINT8 counter);`

This function is used to set the maximum number of power cut counter.

Parameters: `counter [in]`

Maximum counter number to set. It's value from 0 to 15. The power cut register provides a method for disabling power cuts if this situation manifests itself.

If PC_COUNT >= PC_MAX_COUNT, then the number of resets that have occurred since the power cut counter was last initialized exceeds the established limit, and power cuts will be disabled.

Returns: *status*

PMIC_SUCCESS for success and PMIC_ERROR for failure

23.6.4.2.8 PmicPwrctrlGetPowerCutMaxCounter

Prototype `PMIC_STATUS PmicPwrctrlGetPowerCutMaxCounter (UINT8* counter);`

This function is used to get the setting of maximum power cut counter.

Parameters: `counter [out]`

To get the maximum counter number

Returns: *status*

PMIC_SUCCESS for success and PMIC_ERROR for failure

23.6.4.2.9 PmicPwrctrlEnableCounter

Prototype `PMIC_STATUS PmicPwrctrlEnableCounter(void);`

The power cut register provides a method for disabling power cuts if this situation manifests itself. If PC_COUNT >= PC_MAX_COUNT, then the number of resets that have occurred since the power cut counter was last initialized exceeds the established limit, and power cuts will be disabled.

This function can be disabled by setting PC_COUNT_EN=0. In this case, each power cut event will increment the power cut counter, but power cut coverage will not be disabled, even if PC_COUNT exceeds PC_MAX_COUNT.

This PmicPwrctrlEnableCounter function will set PC_COUNT_EN=1.

Parameters: None

Returns: *status*

PMIC_SUCCESS for success and PMIC_ERROR for failure

23.6.4.2.10 PmicPwrctrlDisableCounter

Prototype `PMIC_STATUS PmicPwrctrlDisableCounter (void);`

The power cut register provides a method for disabling power cuts if this situation manifests itself. If PC_COUNT >= PC_MAX_COUNT, then the number of resets that have occurred since the power cut counter was last initialized exceeds the established limit, and power cuts will be disabled.

This function can be disabled by setting PC_COUNT_EN=0. In this case, each power cut event will increment the power cut counter, but power cut coverage will not be disabled, even if PC_COUNT exceeds PC_MAX_COUNT. This PmicPwrctrlEnableCounter function will set PC_COUNT_EN=0.

Parameters: None

Returns: *status*

PMIC_SUCCESS for success and PMIC_ERROR for failure

23.6.4.2.11 PmicPwrctrlSetMemHoldTimer

Prototype `PMIC_STATUS PmicPwrctrlSetMemHoldTimer (UINT8 duration);`

This function is used to set the duration of memory hold timer.

Parameters: `duration [in]`

The value to set to memory hold timer register. It's from 0 to 15. The resolution of the memory hold timer is 32 seconds for a maximum duration of 512 seconds.

Returns: *status*

PMIC_SUCCESS for success and PMIC_ERROR for failure

23.6.4.2.12 PmicPwrctrlGetMemHoldTimer

Prototype `PMIC_STATUS PmicPwrctrlGetMemHoldTimer (UINT8* duration);`

This function is used to get the setting of memory hold timer

Parameters: `duration [out]`

To get the duration of the timer

Returns: *status*

PMIC_SUCCESS for success and PMIC_ERROR for failure

23.6.4.2.13 PmicPwrctrlSetMemHoldTimerAllOn

Prototype `PMIC_STATUS PmicPwrctrlSetMemHoldTimerAllOn (void);`

This function is used to set the duration of the memory hold timer to infinity

Parameters: None

Returns: *status*

PMIC_SUCCESS for success and PMIC_ERROR for failure

23.6.4.2.14 PmicPwrctrlClearMemHoldTimerAllOn

Prototype `PMIC_STATUS PmicPwrctrlClearMemHoldTimerAllOn (void);`

This function is used to clear the infinity duration of the memory hold timer

Parameters: None

Returns: *status*

PMIC_SUCCESS for success and PMIC_ERROR for failure

23.6.4.2.15 PmicPwrctrlEnableClk32kMCU

Prototype `PMIC_STATUS PmicPwrctrlEnableClk32kMCU (void);`

This function is used to enable the CLK32KMCU

Parameters: None

Returns: *status*

PMIC_SUCCESS for success and PMIC_ERROR for failure

23.6.4.3 PmicPwrctrlDisableClk32kMCU

Prototype `PMIC_STATUS PmicPwrctrlDisableClk32kMCU (void);`

This function is used to disable the CLK32KMCU

Parameters: None

Returns: *status*

PMIC_SUCCESS for success and PMIC_ERROR for failure

23.6.4.4 PmicPwrctrlEnableUserOffModeWhenDelay

Prototype `PMIC_STATUS PmicPwrctrlEnableUserOffModeWhenDelay (void);`

This function is used to place the phone in User Off Mode after a delay.

Parameters: None

Returns: *status*

PMIC_SUCCESS for success and PMIC_ERROR for failure

23.6.4.5 PmicPwrctrlDisableUserOffModeWhenDelay

Prototype `PMIC_STATUS PmicPwrctrlDisableUserOffModeWhenDelay (void);`

This function is used to set not to place the phone in User Off Mode after a delay.

Parameters: None

Returns: *status*

PMIC_SUCCESS for success and PMIC_ERROR for failure

23.6.4.6 PmicPwrctrlSetVBKUPRegulator

Prototype `PMIC_STATUS PmicPwrctrlSetVBKUPRegulator (MC13783_PWRCTRL_REG_VBKUP reg, MC13783_PWRCTRL_VBKUP_MODE mode);`

This function is used to set the VBKUP regulator

Parameters: `reg [in]`

the backup regulator to set

`mode [in]`

the mode to set to backup regulator

VBKUP_MODE1 - VBKUPxEN = 0, VBKUPxAUTO = 0

Backup Regulator Off in Non Power Cut Modes and Off in Power Cut Modes

VBKUP_MODE2 - VBKUPxEN = 0, VBKUPxAUTO = 1

Backup Regulator Off in Non Power Cut Modes and On in Power Cut Modes

VBKUP_MODE3 - VBKUPxEN = 1, VBKUPxAUTO = 0

Backup Regulator On in Non Power Cut Modes and Off in Power Cut Modes

VBKUP_MODE4 - VBKUPxEN = 1, VBKUPxAUTO = 1

Backup Regulator On in Non Power Cut Modes and On in Power Cut Modes

Returns: *status*

PMIC_SUCCESS for success and PMIC_ERROR for failure

23.6.4.7 PmicPwrctrlSetVBKUPRegulatorVoltage

Prototype `PMIC_STATUS PmicPwrctrlSetVBKUPRegulatorVoltage (MC13783_PWRCTRL_REG_VBKUP reg, UINT8 volt);`

This function is used to set the VBKUP regulator voltage

Parameters: `reg [in]`

the backup regulator to set

`volt [in]`

the voltage to set to backup regulator

Returns: *status*

PMIC_SUCCESS for success and PMIC_ERROR for failure

23.6.4.8 PmicPwrctrlEnableWarmStart

Prototype `PMIC_STATUS PmicPwrctrlEnableWarmStart (void);`

This function is used to set the phone to transit from the ON state to the User Off state when either the USER_OFF pin is pulled high or the USER_OFF_SPI bit is set (after an 8ms delay in the Memwait state).

Parameters: None

Returns: *status*

PMIC_SUCCESS for success and PMIC_ERROR for failure

23.6.4.9 PmicPwrctrlDisableWarmStart

Prototype `PMIC_STATUS PmicPwrctrlDisableWarmStart (void);`

This function is used to disable the warm start and set the phone to transit from the ON state to the MEMHOLD ONLY state when either the USER_OFF pin is pulled high or the USER_OFF_SPI bit is set (after an 8ms delay in the Memwait state).

Parameters: None

Returns: *status*

PMIC_SUCCESS for success and PMIC_ERROR for failure

23.6.4.10 PmicPwrctrlEnableRegenAssig

Prototype `PMIC_STATUS PmicPwrctrlEnableRegenAssig (t_regulator regu);`

This function enables the REGEN pin of selected voltage regulator. The REGEN function can be used in two ways. It can be used as a regulator enable pin like with SIMEN where the SPI programming is static and the REGEN pin is dynamic. It can also be used in a static fashion where REGEN is maintained high while the regulators get enabled and disabled dynamically through SPI. In that case REGEN functions as a master enable.

Parameters: `t_regulator regu`

Returns: *status*

PMIC_SUCCESS for success and PMIC_ERROR for failure

23.6.4.11 PmicPwrctrlDisableRegenAssig

Prototype `PMIC_STATUS PmicPwrctrlDisableRegenAssig (t_regulator regu);`

This function Disable the REGEN pin of selected voltage regulator.

Parameters: `t_regulator regu`

Returns: *status*

PMIC_SUCCESS for success and PMIC_ERROR for failure

23.6.4.12 PmicPwrctrlGetRegenAssig

Prototype `PMIC_STATUS PmicPwrctrlGetRegenAssig (t_regulator regu , UINT8* value);`

This function reads the REGEN pin value for said voltage regulator.

Parameters: `t_regulator regu` , value

Returns: *status*

PMIC_SUCCESS for success and PMIC_ERROR for failure

23.6.5 PowerCutTimer Functions

The maximum duration of a power cut is determined by the power cut timer PCT[7:0]. By SPI this timer is set to a preset value. When a power cut occurs the timer will internally be decremented till it expires, meaning counted down to zero. The contents of PCT[7:0] does not reflect the actual counted down value but will keep the programmed value and therefore does not have to be reprogrammed after each power cut.

Using the following functions enable/disable/maximum duration of a power cut is determined

```
PMIC_STATUS PmicPwrctrlSetPowerCutTimer (UINT8 duration);
PMIC_STATUS PmicPwrctrlGetPowerCutTimer (UINT8* duration);
PMIC_STATUS PmicPwrctrlEnablePowerCut (void);
PMIC_STATUS PmicPwrctrlDisablePowerCut (void);
```

The following code example shows how to use the power control functions of PMIC module.

```
int MC13783_power_cut_conf(struct t_power_cut_conf *pc)
{
        if(!pc>pc_counter_en)
        {
        PmicPwrctrlDisablePowerCut();
        }
        else
        PmicPwrctrlEnablePowerCut();

        if(!pc>pc_auto_user_off)
        {
        PmicPwrctrlDisableUserOffModeWhenDelay();
        }
        else
        PmicPwrctrlEnableUserOffModeWhenDelay();

        if(!pc>pc_auto_user_off)
        {
        PmicPwrctrlDisableClk32kMCU();
        }
        else
        PmicPwrctrlEnableClk32kMCU();


        if(pc>pc_timer)

        PmicPwrctrlSetPowerCutTimer (pc>pc_timer);

        if(pc>pc_counter)

        PmicPwrctrlSetPowerCutCounter(pc>pc_counter);

        if(pc>pc_max_nb_pc)
        PmicPwrctrlSetPowerCutMaxCounter(pc>pc_max_nb_pc);
        if(pc>pc_ext_timer)
        PmicPwrctrlSetMemHoldTimer (pc>pc_ext_timer);
        if(pc>pc_ext_timer_inf)
        PmicPwrctrlSetMemHoldTimerAllOn();
        else
        PmicPwrctrlClearMemHoldTimerAllOn();
        return 0;
}
```

23.6.6 Memory Hold Operation functions

The Memory Hold circuit provides power to the memory during a power cut through VBKUP1. To avoid leakage from the VBKUP1 into circuitry connected to BP during a power cut, an external PMOS is to be placed between the memory supplies.

The following functions set/get the duration of memory hold timer.

```
PMIC_STATUS PmicPwrctrlSetMemHoldTimer (UINT8 duration)
PMIC_STATUS PmicPwrctrlGetMemHoldTimer (UINT8* duration)
```

The following functions set/clear the duration of the memory hold timer to infinity

```
PMIC_STATUS PmicPwrctrlSetMemHoldTimerAllOn (void)
PMIC_STATUS PmicPwrctrlClearMemHoldTimerAllOn (void)
```

The following code example uses the power controller memory hold operation functions of PMIC module.

```
int MC13783_power_cut_get_conf(struct t_power_cut_conf *pc)
{
        UINT8 duration;
        UINT8 counter;
        UINT32 reg;
        unsigned char index;
        PmicPwrctrlGetPowerCutTimer (&duration);
        pc>pc_timer = duration;

        PmicPwrctrlGetPowerCutCounter (&counter);
        pc>pc_counter = counter;

        PmicPwrctrlGetPowerCutMaxCounter(&duration);
        pc>pc_max_nb_pc = duration;
        PmicPwrctrlGetMemHoldTimer (&duration);
        pc>pc_ext_timer = duration;
        index = 0x0E;MC13783_PWR_CTL1_ADDR
        PmicRegisterRead(index, &reg);
        if((reg &0x00100000))
        pc>pc_ext_timer_inf = 1;((reg &0x00100000) >> 20);
        else
        pc>pc_ext_timer_inf = 0;

        pc>pc_max_nb_pc = ((reg &0x0000F800) >> 11);

        PmicPwrctrlGetMemHoldTimer (&duration);
        pc>pc_ext_timer=duration;

        index = 0x0D;MC13783_PWR_CTL0_ADDR
        PmicRegisterRead(index, &reg);
        if((reg &0x00000002))
        pc>pc_counter_en = 1;
        else
        pc>pc_counter_en = 0;

        if((reg &0x00000008))
        pc>pc_auto_user_off =1;
        else
        pc>pc_auto_user_off =0;

        if((reg &0x00000020))
        pc>pc_user_off_32k_en=1;
        else
        pc>pc_user_off_32k_en=0;
return 0;
}
```

23.6.7 Power Cut Counter Functions

PwCtrl provides a method for disabling power cuts if this situation manifests itself. If PC_COUNT >= PC_MAX_COUNT, then the number of resets that have occurred since the power cut counter was last initialized exceeds the established limit, and power cuts will be disabled. PwCtrl counters can be disabled by setting PC_COUNT_EN=0. In this case, each power cut event will increment the power cut counter, but power cut coverage will not be disabled, even if PC_COUNT exceeds PC_MAX_COUNT.

Following functions are used to set/get the power cut counter values.

```
PMIC_STATUS PmicPwrctrlSetPowerCutCounter (UINT8 counter)
PMIC_STATUS PmicPwrctrlGetPowerCutCounter (UINT8* counter)
PMIC_STATUS PmicPwrctrlSetPowerCutMaxCounter (UINT8 counter)
PMIC_STATUS PmicPwrctrlGetPowerCutMaxCounter (UINT8* counter)
```

Following functions are used to enable/disable the duration of the power cut counters.

```
PMIC_STATUS PmicPwrctrlEnablePowerCut (void)
PMIC_STATUS PmicPwrctrlDisablePowerCut (void)
```

The following code example shows how to use the power controller power cut counter functions of PMIC module. Some the functions are used in the above examples.

```
int MC13783_power_cut_get_conf(struct t_power_cut_conf *pc)
{
        UINT8 duration;
        UINT8 counter;
        UINT32 reg;
        unsigned char index;
        PmicPwrctrlGetPowerCutTimer (&duration);
        pc>pc_timer = duration;

        PmicPwrctrlGetPowerCutCounter (&counter);
        pc>pc_counter = counter;

        PmicPwrctrlGetPowerCutMaxCounter(&duration);
        pc>pc_max_nb_pc = duration;
        PmicPwrctrlGetMemHoldTimer (&duration);
        pc>pc_ext_timer = duration;
        index = 0x0E;MC13783_PWR_CTL1_ADDR
        PmicRegisterRead(index, &reg);
        if((reg &0x00100000))
        pc>pc_ext_timer_inf = 1;((reg &0x00100000) >> 20);
        else
        pc>pc_ext_timer_inf = 0;

        pc>pc_max_nb_pc = ((reg &0x0000F800) >> 11);

        PmicPwrctrlGetMemHoldTimer (&duration);
        pc>pc_ext_timer=duration;

        index = 0x0D;MC13783_PWR_CTL0_ADDR
        PmicRegisterRead(index, &reg);
        if((reg &0x00000002))
```

```
        pc>pc_counter_en = 1;
        else
        pc>pc_counter_en = 0;

        if((reg &0x00000008))
        pc>pc_auto_user_off =1;
        else
        pc>pc_auto_user_off =0;

        if((reg &0x00000020))
        pc>pc_user_off_32k_en=1;
        else
        pc>pc_user_off_32k_en=0;
        return 0;
}
```

23.6.8 Power Management

There is no additional power management implementation done specifically for Atlas Power Control other than the implementation described in section Power Management of this document.

23.6.9 Voltage Regulator

The ARM11/ARM9 processor cores and memories are supposed to be supplied by the switchers. All other building blocks are supplied either directly from the battery or through a linear regulator.

For convenience these regulators are labeled to indicate their intended purpose. This concerns VRF1 and VRF2 for the transceiver transmit and receive supplies, VRFREF, VRFBG and VRFCP as the transceiver references, VRFDIG, VDIG and VGEN for the different digital sections of the platform, VIOHI, VIOLO for the different interfaces, VCAM for the camera module, VSIM1 for the SIM card, VESIM1 for the eSIM card, VMMC1 and VMCC2 for dual multimedia card support or peripheral supply, such as Bluetooth PA.

23.6.9.1 Data Structures

```
// switch mode regulator
typedef enum _MC13783_REGULATOR_SREG{
    SW1A = 0,
    SW1B,
    SW2A,
    SW2B,
    SW3,
} MC13783_REGULATOR_SREG;
typedef MC13783_REGULATOR_SREG PMIC_REGULATOR_SREG;

typedef UINT8 PMIC_REGULATOR_SREG_VOLTAGE;
```

```
/************************************
 * Switch regulator voltage settings type:
 *
 * SW_VOLTAGE_NORMAL
 * SW_VOLTAGE_DVS
 * SW_VOLTAGE_STBY
 *
 ************************************/
typedef enum _RR_REGULATOR_SREG_VOLTAGE_TYPE{
    SW_VOLTAGE_NORMAL=0,
    SW_VOLTAGE_DVS,
    SW_VOLTAGE_STBY,
 } RR_REGULATOR_SREG_VOLTAGE_TYPE;
typedef RR_REGULATOR_SREG_VOLTAGE_TYPE PMIC_REGULATOR_SREG_VOLTAGE_TYPE;


// standby input state H/L
typedef enum _MC13783_REGULATOR_SREG_STBY{
    LOW = 0,
    HIGH,
}MC13783_REGULATOR_SREG_STBY;
typedef MC13783_REGULATOR_SREG_STBY PMIC_REGULATOR_SREG_STBY;


/************************************************
// switch regulator modes:
//              1. OFF
//              2. PWM mode and no Pulse Skipping
//              3. PWM mode and pulse Skipping Allowed
//              4. Low Power PFM mode
************************************************/
typedef enum _MC13783_REGULATOR_SREG_MODE{
    SW_MODE_OFF,
    SW_MODE_PWM,
    SW_MODE_PULSESKIP,
    SW_MODE_PFM,
}MC13783_REGULATOR_SREG_MODE;
typedef MC13783_REGULATOR_SREG_MODE PMIC_REGULATOR_SREG_MODE;
```

```
// linear voltage regulator
typedef enum _MC13783_REGULATOR_VREG{
    VIOHI = 0,
    VIOLO,
    VDIG,
    VGEN,
    VRFDIG,
    VRFREF,
    VRFCP,
    VSIM,
    VESIM,
    VCAM,
    V_VIB,
    VRF1,
    VRF2,
    VMMC1,
    VMMC2,
} MC13783_REGULATOR_VREG;
typedef MC13783_REGULATOR_VREG PMIC_REGULATOR_VREG;


/*************************************************************************
// LOW_POWER
// VxMODE=1, Set Low Power no matter of VxSTBY and STANDBY pin
//
// LOW_POWER_CTL_BY_PIN
// VxMODE=0, VxSTBY=1, Low Power Mode is controlled by STANDBY pin
//
// LOW_POWER_DISABLED
// VxMODE=0, VxSTBY=0, Low Power Mode is disabled
*************************************************************************/
typedef enum _MC13783_REGULATOR_VREG_POWER_MODE{
    LOW_POWER_DISABLED = 0,
    LOW_POWER,
    LOW_POWER_CTRL_BY_PIN,
} MC13783_REGULATOR_VREG_POWER_MODE;
typedef MC13783_REGULATOR_VREG_POWER_MODE PMIC_REGULATOR_VREG_POWER_MODE;

typedef enum _MC13783_REGULATOR_VREG_VOLTAGE_VIOHI{
    VIOHI_2_775 = 0,   //output  2.775V,
 } MC13783_REGULATOR_VREG_VOLTAGE_VIOHI;

typedef enum _MC13783_REGULATOR_VREG_VOLTAGE_VIOLO{
    VIOLO_1_20V = 0,   //output  1.20V,
    VIOLO_1_30V,        //output   1.30V,
    VIOLO_1_50V,        //output   1.50V,
    VIOLO_1_80V,        //output   1.80V,
} MC13783_REGULATOR_VREG_VOLTAGE_VIOLO;
```

```
typedef enum _MC13783_REGULATOR_VREG_VOLTAGE_VRFDIG{
    VRFDIG_1_20V = 0,   //output  1.20V,
    VRFDIG_1_50V,        //output   1.50V,
    VRFDIG_1_80V,        //output   1.80V,
    VRFDIG_1_875V,        //output   1.875V,
} MC13783_REGULATOR_VREG_VOLTAGE_VRFDIG;

typedef enum _MC13783_REGULATOR_VREG_VOLTAGE_VDIG{
    VDIG_1_20V = 0,   //output  1.20V,
    VDIG_1_30V,        //output   1.30V,
    VDIG_1_50V,        //output   1.50V,
    VDIG_1_80V,        //output   1.80V,
} MC13783_REGULATOR_VREG_VOLTAGE_VDIG;

typedef enum _MC13783_REGULATOR_VREG_VOLTAGE_VGEN{
    VGEN_1_20V = 0,   //output  1.20V,
    VGEN_1_30V,        //output   1.30V,
    VGEN_1_50V,        //output   1.50V,
    VGEN_1_80V,        //output   1.80V,
} MC13783_REGULATOR_VREG_VOLTAGE_VGEN;

typedef enum _MC13783_REGULATOR_VREG_VOLTAGE_VRF{
    VRF2_1_875V = 0,   //output  1.875V,
    VRF2_2_475V,        //output   2.475V,
    VRF2_2_700V,        //output   2.700V,
    VRF2_2_775V,        //output   2.775V,
} MC13783_REGULATOR_VREG_VOLTAGE_VRF;

typedef enum _MC13783_REGULATOR_VREG_VOLTAGE_VRFCP{
    VRFCP_2_700V = 0,   //output  2.700V,
    VRFCP_2_775V,        //output   2.775V,
} MC13783_REGULATOR_VREG_VOLTAGE_VRFCP;

typedef enum _MC13783_REGULATOR_VREG_VOLTAGE_VRFREF{
    VRFREF_2_475V = 0,   //output  2.475V,
    VRFREF_2_600V,        //output   2.600V,
    VRFREF_2_700V,        //output   2.700V,
    VRFREF_2_775V,        //output   2.775V,
} MC13783_REGULATOR_VREG_VOLTAGE_VRFREF;

typedef enum _MC13783_REGULATOR_VREG_VOLTAGE_CAM{
                          //        1st silicon,  2nd silicon
    VCAM_1 = 0,   //output  1.50V,          1.5V.
    VCAM_2,        //output   1.80V,          1.80V
    VCAM_3,        //output   2.50V,          2.50V
    VCAM_4,        //output   2.80V,          2.55V
    VCAM_5,        //output    -                2.60V
    VCAM_6,        //output    -                2.80V
    VCAM_7,        //output    -                3.00V
    VCAM_8,        //output    -                TBD
} MC13783_REGULATOR_VREG_VOLTAGE_CAM;
```

```
typedef enum _MC13783_REGULATOR_VREG_VOLTAGE_SIM{
    VSIM_1_8V = 0,   //output = 1.80V
    VSIM_2_9V,   //output = 2.90V
} MC13783_REGULATOR_VREG_VOLTAGE_SIM;

typedef enum _MC13783_REGULATOR_VREG_VOLTAGE_ESIM{
    VESIM_1_8V = 0,   //output = 1.80V
    VESIM_2_9V,   //output = 2.90V
} MC13783_REGULATOR_VREG_VOLTAGE_ESIM;


typedef enum _MC13783_REGULATOR_VREG_VOLTAGE_MMC{
                    //        1st silicon,  2nd silicon
    VMMC_1,  //output    1.60V,           1.60V
    VMMC_2,  //output    1.80V,           1.80V
    VMMC_3,  //output    2.00V,           2.00V
    VMMC_4, //output     2.20V,           2.60V
    VMMC_5,  //output    2.40V,           2.70V
    VMMC_6,  //output    2.60V,           2.80V
    VMMC_7,  //output    2.80V,           2.90V
    VMMC_8,  //output    2.90V,           3.00V
} MC13783_REGULATOR_VREG_VOLTAGE_MMC;

typedef enum _MC13783_REGULATOR_VREG_VOLTAGE_VIB{
    V_VIB_1_3V = 0,   //output = 1.30V
    V_VIB_1_8V,   //output = 1.80V
    V_VIB_2_0V,   //output = 2.0V
    V_VIB_3_0V,   //output = 3.0V
} MC13783_REGULATOR_VREG_VOLTAGE_VIB;

typedef union {
    MC13783_REGULATOR_VREG_VOLTAGE_VIOHI viohi;
    MC13783_REGULATOR_VREG_VOLTAGE_VIOLO violo;
    MC13783_REGULATOR_VREG_VOLTAGE_VRFDIG vrfdig;
    MC13783_REGULATOR_VREG_VOLTAGE_VDIG vdig;
    MC13783_REGULATOR_VREG_VOLTAGE_VGEN vgen;
    MC13783_REGULATOR_VREG_VOLTAGE_VRF vrf;
    MC13783_REGULATOR_VREG_VOLTAGE_VRFCP vrfcp;
    MC13783_REGULATOR_VREG_VOLTAGE_VRFREF vrfref;
    MC13783_REGULATOR_VREG_VOLTAGE_CAM vcam;
    MC13783_REGULATOR_VREG_VOLTAGE_SIM vsim;
    MC13783_REGULATOR_VREG_VOLTAGE_ESIM vesim;
    MC13783_REGULATOR_VREG_VOLTAGE_MMC vmmc;
    MC13783_REGULATOR_VREG_VOLTAGE_VIB v_vib;
} MC13783_REGULATOR_VREG_VOLTAGE;
typedef MC13783_REGULATOR_VREG_VOLTAGE PMIC_REGULATOR_VREG_VOLTAGE;

typedef enum _MC13783_REGULATOR_ENABLE{
    DISABLE = 0,
    ENABLE = 1,
} MC13783_REGULATOR_ENABLE;
```

23.6.9.2 Switch Mode Regulator APIs

PmicSwitchModeRegulatorOn

Prototype `PMIC_STATUS PmicSwitchModeRegulatorOn (PMIC_REGULATOR_SREG regulator);`

Parameters: `regulator [in]`

Which switch mode regulator to turn on

Returns: *status*

PMIC_SUCCESS for success and PMIC_ERROR for failureThis function is used to turn on the switch mode regulator.

PmicSwitchModeRegulatorOff

Prototype `PMIC_STATUS PmicSwitchModeRegulatorOff (PMIC_REGULATOR_SREG regulator);`

This function is used to turn off the switch regulator

Parameters: `regulator [in]`

Which switch mode regulator to turn off

Returns: *status*

PMIC_SUCCESS for success and PMIC_ERROR for failure

PmicSwitchModeRegulatorSetVoltageLevel

Prototype

```
PMIC_STATUS PmicSwitchModeRegulatorSetVoltageLevel (PMIC_REGULATOR_SREG
regulator,
PMIC_REGULATOR_SREG_VOLTAGE_TYPE voltageType,
PMIC_REGULATOR_SREG_VOLTAGE voltage );
```

This function is to set the voltage level for the switch regulator.

Parameters: `regulator [in]`

The regulator to be set

`voltageType [in]`
SW_VOLTAGE_NORMAL/SW_VOLTAGE_LVS/SW_VOLTAGE_STBY

SW1 offers support for Dynamic Voltage-Frequency scaling. If this feature is activated, then assertion of the STANDBY input will automatically configure SW1 to output the voltage defined by the 3-bit field SW1X_STBY. If STANDBY=LOW, then assertion of the LVS input will automatically configure SW1 to output the voltage defined by the 3-bit field SW1X_LVS. These alternative bit fields would normally be programmed to a voltage lower than that encoded in the SW1X bit field. When STANDBY and LVS are both de-asserted, the output voltage will revert the that encoded by the SW1X field.

SW2 offers limited support for Dynamic Voltage-Frequency scaling. If this feature is activated, then assertion of the STANDBY input will automatically configure SW2 to output the voltage defined by the 3-bit field SW2_STBY.

If STANDBY=LOW, then assertion of the LVS2 input will automatically configure SW2 to output the voltage defined by the 3-bit SW2X_LVS field. When STANDBY and LVS2 are both de-asserted, the output voltage will revert to that encoded by the SW2X 3-bit field.

voltage [in]

The voltage to be set, it depends on different regulator.

Returns: *status*

PMIC_SUCCESS for success and PMIC_ERROR for failure

PmicSwitchModeRegulatorGetVoltageLevel

Prototype *PMIC_STATUS PmicSwitchModeRegulatorGetVoltageLevel (PMIC_REGULATOR_SREG regulator, PMIC_REGULATOR_SREG_VOLTAGE_TYPE voltageType, PMIC_REGULATOR_SREG_VOLTAGE* voltage);*

This function is to get the voltage settings.

Parameters: *regulator [in]*

The regulator to get voltage from

voltageType [in]
SW_VOLTAGE_NORMAL/SW_VOLTAGE_LVS/SW_VOLTAGE_STBY

voltage [out]

the pointer to get the value

Returns: *status*

PMIC_SUCCESS for success and PMIC_ERROR for failure

PmicSwitchModeRegulatorSetMode

Prototype *PMIC_STATUS PmicSwitchModeRegulatorSetMode (PMIC_REGULATOR_SREG regulator, PMIC_REGULATOR_SREG_STBY standby, PMIC_REGULATOR_SREG_MODE mode);*

This function is to set the switch mode regulator into synchronous rectifier mode or pulse skipping mode. The synchronous rectifier can be disabled (and pulse-skipping enabled) to improve low current efficiency. Software should disable synchronous rectifier / enable the pulse skipping for average loads less than approximately 30 mA, depending on the quiescent current penalty due to synchronous mode.

Parameters: *regulator [in]*

The regulator to be set

mode [in]

Synchronous rectifier mode or pulse skipping mode.

Returns: *status*

PMIC_SUCCESS for success and PMIC_ERROR for failure

PmicSwitchModeRegulatorGetMode

Prototype *PMIC_STATUS PmicSwitchModeRegulatorGetMode (PMIC_REGULATOR_SREG regulator, PMIC_REGULATOR_SREG_MODE* mode);*

This function get the current setting of regulator mode

Parameters: *regulator [in]*

The regulator to get voltage value from

mode [out]

Synchronous rectifier mode or pulse skipping mode.

Returns: *status*

PMIC_SUCCESS for success and PMIC_ERROR for failure

PmicSwitchModeRegulatorEnableSTBYDVFS

Prototype *PMIC_STATUS PmicSwitchModeRegulatorEnableSTBYDVFS (PMIC_REGULATOR_SREG regulator);*

This function is used to enable the standby or Dynamic Voltage-Frequency scaling.

Parameters: *regulator [in]*

The regulator to be set

Returns: *status*

PMIC_SUCCESS for success and PMIC_ERROR for failure

PmicSwitchModeRegulatorDisableSTBYDVFS

Prototype *PMIC_STATUS PmicSwitchModeRegulatorDisableSTBYDVFS (PMIC_REGULATOR_SREG regulator);*

This function is used to disable the standby or Dynamic Voltage-Frequency scaling.

Parameters: *regulator [in]*

The regulator to be set

Returns: *status*

PMIC_SUCCESS for success and PMIC_ERROR for failure

PmicSwitchModeRegulatorSetDVSSpeed

Prototype *PMIC_STATUS PmicSwitchModeRegulatorSetDVSSpeed (PMIC_REGULATOR_SREG regulator, UINT8 dvsspeed);*

This function is to set the DVS speed the regulator.

Parameters: *regulator [in]*
The regulator to be set
dvsspeed [in]
The speed settings for DVS
Returns: *status*
PMIC_SUCCESS for success and PMIC_ERROR for failure
Remarks: This function is only applicable to MC13783, it is a stub function here.

PmicSwitchModeRegulatorEnablePanicMode

Prototype `PMIC_STATUS PmicSwitchModeRegulatorEnablePanicMode(PMIC_REGULATOR_SREG regulator);`
This function is used to enable the panic mode.
Parameters: *regulator [in]*
The regulator to be set
Returns: *status*
PMIC_SUCCESS for success and PMIC_ERROR for failure
Remarks: This is a stub function here.

PmicSwitchModeRegulatorDisablePanicMode

Prototype `PMIC_STATUS PmicSwitchModeRegulatorDisablePanicMode(PMIC_REGULATOR_SREG regulator);`
This function is used to disable the panic mode.
Parameters: *regulator [in]*
the regulator to be set
Returns: *status*
PMIC_SUCCESS for success and PMIC_ERROR for failure
Remarks: This is a stub function here.

PmicSwitchModeRegulatorEnableSoftStart

Prototype `PMIC_STATUS PmicSwitchModeRegulatorEnableSoftStart(PMIC_REGULATOR_SREG regulator);`
This function is used to enable soft start.
Parameters: *regulator [in]*
The regulator to be set
Returns: *status*
PMIC_SUCCESS for success and PMIC_ERROR for failure
Remarks: This is a stub function.

PmicSwitchModeRegulatorDisableSoftStart

Prototype `PMIC_STATUS PmicSwitchModeRegulatorDisableSoftStart(PMIC_REGULATOR_SREG regulator);`

This function is used to disable soft start.

Parameters: `regulator [in]`

The regulator to be set

Returns: *status*

PMIC_SUCCESS for success and PMIC_ERROR for failure

Remarks: This is a stub function here.

23.6.9.3 Linear Voltage Regulator API's

PmicVoltageRegulatorOn

Prototype `PMIC_STATUS PmicVoltageRegulatorOn (PMIC_REGULATOR_VREG regulator);`

This function is used to turn on the voltage regulator

Parameters: `regulator [in]`

Which voltage regulator to turn on

Returns: *status*

PMIC_SUCCESS for success and PMIC_ERROR for failure

Remarks: MMC does not have on/off, just directly set the voltage level.

0V=OFF will return PMIC_INVALID_PARAMETER.

PmicVoltageRegulatorOff

Prototype `PMIC_STATUS PmicVoltageRegulatorOff (PMIC_REGULATOR_VREG regulator);`

This function is used to turn off the regulator

Parameters: `regulator [in]`

Which voltage regulator to turn off

Returns: *status*

PMIC_SUCCESS for success and PMIC_ERROR for failure

Remarks: MMC don't have on/off, just directly set the voltage level. 0V=OFF will return PMIC_INVALID_PARAMETER.

PmicVoltageRegulatorSetVoltageLevel

Prototype `PMIC_STATUS PmicVoltageRegulatorSetVoltageLevel (PMIC_REGULATOR_VREG regulator, PMIC_REGULATOR_VREG_VOLTAGE voltage);`

This function is used to set voltage level for the voltage regulator.

Parameters: *regulator [in]*

Which switch mode regulator to be set

voltage [in]

The voltage value to be set to the register

Returns: *status*

PMIC_SUCCESS for success and PMIC_ERROR for failure

PmicVoltageRegulatorGetVoltageLevel

Prototype *PMIC_STATUS PmicVoltageRegulatorGetVoltageLevel (PMIC_REGULATOR_VREG regulator, PMIC_REGULATOR_VREG_VOLTAGE* voltage);*

This function is to get the current voltage settings of the regulator.

Parameters: *regulator [in]*

Which switch mode regulator to get the value from

voltage [out]

the pointer to storage the return value

Returns: *status*

PMIC_SUCCESS for success and PMIC_ERROR for failure

PmicVoltageRegulatorSetPowerMode

Prototype *PMIC_STATUS PmicVoltageRegulatorSetPowerMode (PMIC_REGULATOR_VREG regulator, PMIC_REGULATOR_VREG_POWER_MODE powerMode);*

This function is used to set low power mode for the regulator and whether to enter low power mode during STANDBY assertion or not.

Parameters: *regulator [in]*

Which switch mode regulator to be set

powerMode[in]

LOW_POWER

VxMODE=1, Set Low Power no matter of VxSTBY and STANDBY pin

LOW_POWER_CTL_BY_PIN

VxMODE=0, VxSTBY=1, Low Power Mode is controlled by STANDBY pin
LOW_POWER_DISABLED

VxMODE=0, VxSTBY=0, Low Power Mode is disabled.

Returns: *status*

PMIC_SUCCESS for success and PMIC_ERROR for failure

PmicVoltageRegulatorGetPowerMode

Prototype *PMIC_STATUS PmicVoltageRegulatorGetPowerMode (PMIC_REGULATOR_VREG regulator, PMIC_REGULATOR_VREG_POWER_MODE* powerMode);*

This function is to get the current power mode for the regulator

Parameters: *regulator [in]*

Which switch mode regulator to get the value from

powerMode [out]

Pointer to storage the powerMode get from the register

Returns: *status*

PMIC_SUCCESS for success and PMIC_ERROR for failure.

23.6.10 Battery Charger

23.6.10.1 Data Structures

```
typedef enum {
        BATT_MAIN_CHGR = 0,         // Main battery charger
        BATT_CELL_CHGR,             // CoinCell battery charger
        BATT_TRCKLE_CHGR            // Trickle charger
} BATT_CHARGER;

typedef enum {
    DUAL_PATH = 0,
    SINGLE_PATH,
    SERIAL_PATH,
    DUAL_INPUT_SINGLE_PATH,
    DUAL_INPUT_SERIAL_PATH,
    INVALID_CHARGER_MODE
}CHARGER_MODE;

typedef enum {
    LOW = 0,  //GND
    OPEN,     //HI Z
    HIGH      //VMC13783
}CHARGERMODE_PIN;
```

23.6.10.2 Battery Charger API (Compatible with SC55112 API)

PmicBatterEnableCharger

This function is used to start charging a battery. For different charger, different voltage and current range are supported.

The main battery charger supports a settable voltage and current. The coincell supports only a settable voltage. The trickel charger only a settable current.

Prototype *PMIC_STATUS PmicBatterEnableCharger(BATT_CHARGER chgr, UINT8 c_voltage, UINT8 c_current);*

Parameters: *chgr [in]*

Charger as defined in BATT_CHARGER

c_voltage [in]

Charging voltage. (main and coincell)

c_current [in]

Charging current. (main and trickle)

Returns: This function returns PMIC_SUCCESS if successful.

PmicBatterDisableCharger

This function turns off the selected charger. This is done by setting the curren level to zero for the main and trickele chargers. The coincell charger is disabled.

Prototype *PMIC_STATUS PmicBatterDisableCharger(BATT_CHARGER chgr)*

Parameters: *chgr [in]*

Charger as defined in BATT_CHARGER.

Returns: This function returns PMIC_SUCCESS if successful.

PmicBatterSetCharger

This function is used to change the charger setting.

Prototype *PMIC_STATUS PmicBatterSetCharger(BATT_CHARGER chgr, UINT8 c_voltage, UINT8 c_current);*

Parameters: *chgr [in]*

Charger as defined in BATT_CHARGER

c_voltage [in]

Charging voltage. (main and coincell)

c_current [in]

Charging current. (main and trickle)

Returns: This function returns PMIC_SUCCESS if successful.

PmicBatterGetChargerSetting

This function is used to retrieve what the charger setting are for the selected charger not what is measured

Prototype *PMIC_STATUS PmicBatterGetChargerSetting(BATT_CHARGER chgr, UINT8* c_voltage, UINT8* c_current);*

Parameters: *chgr [in]*

Charger as defined in BATT_CHARGER

**c_voltage [out]*

A pointer to what the charging voltage is set to. (main and coincell)

**c_current [out]*

A pointer to what the charging current is set to. (main and trickle)

Returns: This function returns PMIC_SUCCESS if successful.

PmicBatterGetChargeCurrent

This function is retrieves the main charger current. This value is obtained by reading a voltage between CHRGISNSP – CHRGISNSN. This corresponds to ADC channel 4.

Prototype *PMIC_STATUS PmicBatterGetChargeCurrent(UINT16* c_current);*

Parameters: **c_current [out]*

A pointer to what the measured charger current

Returns: This function returns PMIC_SUCCESS if successful.

PmicBatterEnableEol

This function enables End-of-Life comparator.

Prototype *PMIC_STATUS PmicBatterEnableEol(void);*

Parameters: None

Returns: This function returns PMIC_SUCCESS if successful.

PmicBatterDisableEol

This function disables End-of-Life comparator.

Prototype *PMIC_STATUS PmicBatterDisableEol (void);*

Parameters: None

Returns: This function returns PMIC_SUCCESS if successful.

PmicBatterLedControl

This function controls charge LED.

Prototype *PMIC_STATUS PmicBatterLedControl(BOOL on);*

Parameters: *on [in]*

If on is true, LED will be turned on, or otherwise the LED will be turned off.

Returns: This function returns PMIC_SUCCESS if successful.

PmicBatterSetReverseSupply

This function sets reverse supply mode.

Prototype `PMIC_STATUS PmicBatterSetReverseSupply(BOOL enable);`

Parameters: *enable [i]*

If enable is true, reverse supply mode is enable or otherwise the reverse supply mode is disabled.

Returns: This function returns PMIC_SUCCESS if successful.

PmicBatterSetUnregulated

This function sets limited charging mode on main battery charger. If this mode is selected the current is no longer controlled and it is only limited by what the charger can supply.

Prototype `PMIC_STATUS PmicBatterSetUnregulated(BOOL enable);`

Parameters: `enable [in]`

If enable is true, unregulated charging mode is enabled otherwise it is disabled.

Returns: This function returns PMIC_SUCCESS if successful.

23.6.10.3 Battery Charger API (MC13783 Native For Compatibility with SC55112)

These functions are just there for compatibility with the SC55112 API. This is an effort to maintain one PMIC API, regardless of which PMIC is being used. These are implemented as a stubs returning a PMIC_STATUS of PMIC_SUCCESS.

PmicBatteryEnableAdChannel5

This function enables use of AD channel 5 to read charge current on the SC55112 PMIC.

Prototype `PMIC_STATUS PmicBatteryEnableAdChannel5();`

Parameters: None.

Returns: PMIC_SUCCESS

PmicBatteryDisableAdChannel5

This function disables use of AD channel 5 to read charge current on the SC55112 PMIC.

Prototype `PMIC_STATUS PmicBatteryDisableAdChannel5();`

Parameters: None.

Returns: PMIC_SUCCESS;

PmicBatterySetCoincellCurrentlimit

This function limits the output current level of coincell charger on the SC55112 PMIC.

Prototype *PMIC_STATUS PmicBatterySetCoincellCurrentlimit (UINT8 coincellcurrentlevel);*

Parameters: *coincellcurrentlevel [IN]*

coincell current level

Returns: PMIC_SUCCESS;

PmicBatteryGetCoincellCurrentlimit

This function returns the output current limit of coincell charger on the SC55112 PMIC.

Prototype *PMIC_STATUS PmicBatteryGetCoincellCurrentlimit (UINT8* coincellcurrentlevel);*

Parameters: *coincellcurrentlevel [OUT]*

Pointer to coincell current level

Returns: PMIC_SUCCESS;

PmicBatterySetEolTrip

This function sets the end-of-life threshold on the SC55112 PMIC.

Prototype *PMIC_STATUS PmicBatterySetEolTrip (UINT8 eoltriplevel);*

Parameters: *eoltriplevel [IN]*

eol trip level

Returns: PMIC_SUCCESS;

PmicBatteryGetEolTrip

This function returns the end-of-life threshold on the SC55112 PMIC.

Prototype *PMIC_STATUS PmicBatteryGetEolTrip (UINT8* eoltriplevel);*

Parameters: *eoltriplevel [OUT]*

pointer to eol trip level

Returns: PMIC_SUCCESS;

23.6.10.4 Battery Charger API (MC13783 Native)

PmicBatterySetChargeVoltage

This function programs the output voltage of charge regulator.

Prototype `PMIC_STATUS PmicBatterySetChargeVoltage(UINT8 chargevoltagelevel);`

Parameters: `chargevoltagelevel [IN]`

voltage level

level 0 = 4.05V

1 = 4.10V

2 = 4.15V

3 = 4.20V

4 = 4.25V

5 = 4.30V

6 = 3.80V ... lowest setting

7 = 4.50V

Returns: PMIC_STATUS

PmicBatteryGetChargeVoltage

This function returns the output voltage of charge regulator.

Prototype `PMIC_STATUS PmicBatteryGetChargeVoltage(UINT8* chargevoltagelevel);`

Parameters: `chargevoltagelevel [OUT]`

pointer to voltage level

Returns: PMIC_STATUS

PmicBatterySetChargeCurrent

This function programs the charge current limit level to the main battery.

Prototype `PMIC_STATUS PmicBatterySetChargeCurrent (UINT8 chargecurrentlevel);`

Parameters: `chargecurrentlevel [IN]`

current level

level 0 = 0 mA (max value)

1 = 100 mA (max value)

... (in increment of 100 mA)

13 = 1300 mA (max value)

14 = 1800 mA (max value)

15 = disables the current limit

Returns: PMIC_STATUS

PmicBatteryGetChargeCurrent

This function returns the charge current setting of the main battery.

Prototype `PMIC_STATUS PmicBatteryGetChargeCurrent (UINT8* chargecurrentlevel);`

Parameters: `chargecurrentlevel [OUT]`

pointer to current level

Returns: PMIC_STATUS

PmicBatterySetTrickleCurrent

This function programs the current of the trickle charger.

Prototype `PMIC_STATUS PmicBatterySetTrickleCurrent(UINT8 tricklecurrentlevel);`

Parameters: `tricklecurrentlevel [IN]`

trickle current level

level 0 = 0 mA

1 = 12 mA

... (in addition of 12 mA per level)

6 = 72 mA

7 = 84 mA

Returns: PMIC_STATUS

PmicBatteryGetTrickleCurrent

This function returns the current of the trickle charger.

Prototype `PMIC_STATUS PmicBatteryGetTrickleCurrent (UINT8* tricklecurrentlevel);`

Parameters: `tricklecurrentlevel [OUT]`

pointer to trickle current level

Returns: PMIC_STATUS

PmicBatteryFETControl

This function programs the control mode and setting of BPFET and FETOVRD BATTFET and BPFET to be controlled by FETCTRL bit or hardware.

Prototype `PMIC_STATUS PmicBatteryFETControl(UINT8 fetcontrol);`

Parameters: `fetcontrol [IN]`

BPFET and FETOVRD control mode and setting

input = 0 (BATTFET and BPFET outputs are controlled by hardware)

= 1 (BATTFET and BPFET outputs are controlled by hardware)

= 2 (BATTFET low and BATTFET high, controlled by FETCTRL)

= 3 (BATTFET high and BATTFET low, controlled by FETCTRL)

Returns: PMIC_STATUS

PmicBatteryReverseDisable

This function disables the reverse mode.

Prototype `PMIC_STATUS PmicBatteryReverseDisable();`

Parameters: None

Returns: PMIC_STATUS

PmicBatteryReverseEnable

This function enables the reverse mode.

Prototype `PMIC_STATUS PmicBatteryReverseEnable();`

Parameters: None

Returns: PMIC_STATUS

PmicBatterySetOvervoltageThreshold

This function programs the overvoltage threshold value.

Prototype `PMIC_STATUS PmicBatterySetOvervoltageThreshold(UINT8 ovthresholdlevel);`

Parameters: `ovthresholdlevel [IN]`

overvoltage threshold level

High to low, Low to High (5.35V)

Returns: PMIC_STATUS

PmicBatteryGetOvervoltageThreshold

This function returns the overvoltage threshold value.

Prototype `PMIC_STATUS PmicBatteryGetOvervoltageThreshold (UINT8* ovthresholdlevel);`

Parameters: `ovthresholdlevel [OUT]`

pointer to overvoltage threshold level

Returns: PMIC_STATUS

PmicBatteryUnregulatedChargeDisable

This function disables the unregulated charge path. The voltage and current limits will be controlled by the charge path regulator.

Prototype `PMIC_STATUS PmicBatteryUnregulatedChargeDisable();`

Parameters: None

Returns: PMIC_STATUS

PmicBatteryUnregulatedChargeEnable

This function enables the unregulated charge path. The settings of the charge path regulator (voltage and current limits) will be overruled.

Prototype `PMIC_STATUS PmicBatteryUnregulatedChargeEnable();`

Parameters: None

Returns: PMIC_STATUS

PmicBatteryChargeLedDisable

This function disables the charging LED.

Prototype `PMIC_STATUS PmicBatteryChargeLedDisable();`

Parameters: None

Returns: PMIC_STATUS

PmicBatteryChargeLedEnable

This function enables the charging LED.

Prototype `PMIC_STATUS PmicBatteryChargeLedEnable();`

Parameters: None

Returns: PMIC_STATUS

PmicBatteryEnablePulldown

This function enables the 5k pull-down resistor used in the dual path charging.

Prototype `PMIC_STATUS PmicBatteryEnablePulldown();`

Parameters: None.

Returns: PMIC_STATUS.

PmicBatteryDisablePulldown

This function disables the 5k pull-down resistor used in the dual path charging.

Prototype `PMIC_STATUS PmicBatteryDisablePulldown();`

Parameters: None.

Returns: PMIC_STATUS.

PmicBatteryEnableCoincellCharger

This function enables the coincell charger.

Prototype `PMIC_STATUS PmicBatteryEnableCoincellCharger();`

Parameters: None

Returns: PMIC_STATUS

PmicBatteryDisableCoincellCharger

This function disables the coincell charger.

Prototype `PMIC_STATUS PmicBatteryDisableCoincellCharger();`

Parameters: None

Returns: PMIC_STATUS

PmicBatterySetCoincellVoltage

This function programs the output voltage level of coincell charger.

Prototype `PMIC_STATUS PmicBatterySetCoincellVoltage (UINT8 coincellvoltagelevel);`

Parameters: `votlagelevel [IN]`

voltage level

level 0 = 2.7V

1 = 2.8V

2 = 2.9V

... (in 100mV increment)

6 = 3.3V

Returns: PMIC_STATUS

PmicBatteryGetCoincellVoltage

This function returns the output voltage level of coincell charger.

Prototype `PMIC_STATUS PmicBatteryGetCoincellVoltage (UINT8* coincellvoltagelevel);`

Parameters: `voltagelevel [OUT]`

pointer to voltage level

Returns: PMIC_STATUS

PmicBatteryEnableEolComparator

This function enables the end-of-life function instead of the LOBAT.

Prototype `PMIC_STATUS PmicBatteryEnableEolComparator();`

Parameters: None

Returns: PMIC_STATUS

PmicBatteryDisableEolComparator

This function disables the end-of-life comparator function.

Prototype `PMIC_STATUS PmicBatteryDisableEolComparator();`

Parameters: None

Returns: PMIC_STATUS

PmicBatteryGetChargerMode

This function returns the charger mode (ie. Dual Path, Single Path, Serial Path, Dual Input Single Path and the Dual Input Serial Path).

Prototype `PMIC_STATUS PmicBatteryGetChargerMode(CHARGER_MODE *mode);`

Parameters: `mode [OUT]`

pointer to charger mode

Returns: PMIC_STATUS

Chapter 24
Secure Digital Host Controller

The Secure Digital Host Controller (SDHC) module supports Secure Digital Cards (SD) and Secure Digital I/O and Combo Cards (SDIO). The i.MX31 has two SDHC hardware modules. Each host controller supports only one card to be connected to it. On the 3-Stack board set, only host 1 is connected with the SD socket.

The SDHC driver provides the interface between the Microsoft SD Bus driver and the SDHC hardware.

NOTE

On the 3-Stack Rev 2.0 board, the SDIO function is not supported due to hardware limitations.

24.1 SDHC Driver Summary

24.1.1 MX31 SDHC Driver Summary

Table 24-1 identifies the source code location, library dependencies, and other BSP information.

Table 24-1. MX31 SDHC Driver Summary

Driver Attribute	Definition
Target Platform (TGTPLAT)	3DS
Target SOC (TGTSOC)	MX31
MXARM11 CSP Driver Path	..\CSP\ARM\FREESCALE\MXARM11\DRIVERS\SDHC
CSP Driver Path	N/A
CSP Static Library	mxarm11_sdhc.lib
Platform Driver Path	..\PLATFORM\<*tgtplat*>\SRC\DRIVERS\SDHC
Import Library	N/A
Driver DLL	sdhc.dll
Catalog Item	Third Party >BSPs >Freescale <tgtplat> >Device Drivers >SD Host Controller 1
SYSGEN Dependency	SYSGEN_SD_MEMORY=1 BSP_NIC_WLAN6060_SDIO=1 (This not required for 3-Stack rev 2.0 board)
BSP Environment Variables	BSP_SDHC1=1

24.2 Supported Functionality

The SDHC driver enables the 3-Stack board to provide the following software and hardware support:

- Supports the i.MX31 Secure Digital Host Controller
- Supports SD and SDIO cards
- Supports Power Management modes, full on and full off only

24.3 Hardware Operation

Refer to the chapter on the Secure Digital Host Controller (SDHC) in the *MCIMX31 and MCIMX31L Applications Processors Reference Manual* for detailed operation and programming information.

24.3.1 Conflicts with other SoC Peripherals

24.3.1.1 MX31 Peripheral Conflicts

In Alternate Mode 1, SDHC1 conflicts with Memory Stick (MS1). Configure the pins in Functional Mode to activate SDHC1 signals.

In Functional Mode, SDHC2 conflicts with PCMCIA. Configure the pins in Alternate Mode 1 to activate SDHC2 signals.

24.4 Software Operation

The SDHC driver follows the Microsoft-recommended architecture for Secure Digital Host Controller drivers. The details of this architecture and its operation can be found in the Platform Builder Help under the heading “Secure Digital Card Driver Development Concepts”, or in the online Microsoft documentation at the following URL:

http://msdn.microsoft.com/en-us/library/ms894007.aspx

24.4.1 Required Catalog Items

24.4.1.1 SD Memory Card Support

Catalog > Device Drivers > SDIO > SD Memory

24.4.1.2 Wi-Fi SDIO Support

Catalog > Device Drivers > SDIO > SD Memory > SDIO WiFi (Not required for 3-Stack 1.0 and 1.1 board)

24.4.2 SDHC Registry Settings

24.4.2.1 MX31 SDHC Register Settings

The following registry keys are required to properly load the SDHC driver.

```
#if (defined BSP_SDHC1 || defined BSP_SDHC2)
[HKEY_LOCAL_MACHINE\Drivers\SDCARD\ClientDrivers\Class\SDMemory_Class]
   "BlockTransferSize"=dword:100  ; Overwrite from default 64 blocks.
;   "SingleBlockWrites"=dword:1  ; alternatively force the driver to use single block
access


[HKEY_LOCAL_MACHINE\System\StorageManager\Profiles\SDMemory]
    "Name"="SD Memory Card"
    "Folder"="SD Memory"
#endif

IF BSP_SDHC1
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\SDHC_ARM11_1]
    "Order"=dword:21
    "Dll"="sdhc.dll"
    "Prefix"="SDH"
    "ControllerISTPriority"=dword:64
    "Index"=dword:1
ENDIF ;BSP_SDHC1

IF BSP_SDHC2
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\SDHC_ARM11_2]
    "Order"=dword:21
    "Dll"="sdhc.dll"
    "Prefix"="SDH"
    "ControllerISTPriority"=dword:64
    "Index"=dword:2
ENDIF ;BSP_SDHC2
```

24.4.3 DMA Support

24.4.3.1 i.MX31 DMA Support

The SDHC driver supports the DMA and non-DMA modes of data transfer. The driver always defaults to DMA mode of transfer. The driver does not allocate or manage DMA buffers internally. All buffers are allocated and managed by the upper layers, the details of which are given in the request submitted to the driver. For every request submitted to it, the driver attempts to build a DMA Scatter Gather Buffer Descriptor list for the buffer passed to it by the upper layer. For cases where this list cannot be built, the driver falls back to the non-DMA mode of transfer. The default configuration is maintained in the file bsp_cfg.h using the parameters BSP_SDMA_SUPPORT_SDHC1 and BSP_SDMA_SUPPORT_SDHC2. A value of TRUE means DMA is the default mode, and for cases where DMA cannot be used, the driver falls back to a non-DMA mode. A value of FALSE means non-DMA mode is the default and DMA mode will not be attempted.

For the driver to attempt to build the Scatter Gather DMA Buffer Descriptors, the upper layer should ensure that the buffer meets the following criteria.

- Start of the buffer should be a word aligned address.
- Number of bytes to transfer should be word aligned.

Due to cache coherency issues arising due to processor and SDMA access of the memory, the above criteria is further stringent for the read or receive operation (it is not applicable for write or transmit):

- Start of the buffer should be a cache line size (32 bytes) aligned address.
- Number of bytes to transfer should be cache line size (32 bytes) aligned.

24.4.4 Power Management

The primary methods for limiting power in SDHC module is to gate off all clocks to the controllers and to cut off power to the card slot when no cards are inserted. When a card is inserted to any of the slots, that slot alone is powered and the clocks to that controller alone are gated on. While using memory cards, the clock to the host controller and the clock to memory cards are gated off when ever the controller is idle. For SDIO cards, both the clocks stay on all the time.

SDHC driver supports the full power on and full power off states. In full power off state, the clocks to the controllers and the power to the inserted cards are turned off. When powered on, all SD memory cards inserted before and after the power down will be detected and mounted. And after resume from power down, memory cards can work correctly.

24.4.4.1 PowerUp

This function is implemented to support resuming a memory card operation that was previously terminated by calling PowerDown() API. Power to the card is restored, clocks to the pertaining controller is restarted.

SDHC driver is notified of a device status change. This results in signaling the SD bus driver of a card removal followed by a card insertion. The card is re-initialized and is mounted so that the all operations scheduled during a power down resumes.

The details of this architecture and its operation can be found in the Platform Builder Help under the heading “Power On and Off Notifications for Secure Digital Card Drivers”, or in the online Microsoft documentation at the following URL:

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wceddk5/html/wce50conpoweronoffnotificationsforsecuredigitalcarddrivers.asp

Note that this function is intended to be called only by the Power Manager.

24.4.4.2 PowerDown

This function has been implemented to support suspending all currently active SD operations just before the entire system enters the low power state. Note that this function is intended to be called only by the Power Manager. This function gates off all clocks to the controllers and powers down all the card slots.

24.4.4.3 IOCTL_POWER_CAPABILITIES

N/A

24.4.4.4 IOCTL_POWER_GET

N/A

24.4.4.5 IOCTL_POWER_SET

N/A

24.5 Hardware and Software Limitation

For 3-Stack board set, the write-protection detect pin for the SD socket is not connected to a GPIO pin on the MCU chip. Therefore, the driver cannot detect the write-protection status of the inserted card through the GPIO, and the write-protection detect function is not implemented in the driver. This means that you can write data to a protected SD card, by enabling the write-protection switch.

This function can easily be enabled in the driver when you design your product and have free GPIO pins.

24.6 Unit Test

The SDHC driver is tested using the following tests, which are in the Windows CE 5.0 Test Kit (CETK).

- Storage Device Block Driver Read/Write Test
- Storage Device Block Driver API Test
- File System Driver Test
- Partition Driver Test
- Wi-Fi Test

24.6.1 Unit Test Hardware

Table 24-2 lists the required hardware to run the unit tests.

Table 24-2. Hardware Requirements to Run Unit Tests

Requirements	Description
SD Cards	SanDisk (128MB, 256MB, 512MB) Kingston (256MB) NCP (128MB, 256MB, 512MB) Transcend (128MB, 256MB, 512MB) Toshiba (1GB)
SDIO Cards	SanDisk Connect Wi-Fi (SDWSDB-000) (Not supported on 3-Stack rev 2.0 board)

24.6.2 Unit Test Software

Table 24-3 lists the required software to run the unit tests.

Table 24-3. Software Requirements to Run Unit Tests

Requirements	Description
tux.exe	Tux test harness, which is needed for executing the test
kato.dll	Kato logging engine, which is required for logging test data
tooltalk.dll	Library required by Tux.exe and Kato.dll. Handles the transport between the target device and the development workstation
rwtest.dll	Storage Device Block Driver Read/Write Test .dll file
disktest.dll	Storage Device Block Driver API Test .dll file
fsdtst.dll	File System Driver Test .dll file
msparttest.dll	Partition Driver Test .dll file
wzctooltest.dll	Wi-Fi Test .dll file

24.6.3 Building the Tests

All the above mentioned tests come pre-built as part of the CETK. No steps are required to build these tests. These test files are with the other required CETK files in the following location:

`[Drive]:\Program Files\Windows CE Platform Builder\5.00\cepb\wcetk\ddtk\armv4I`

24.6.4 Running the Tests

The following are the tests available and the test procedures for each of the tests. For information about the tests, see the appropriate subsections under “CETK Tests” in the Platform Builder Help, or view the Microsoft online documentation at the following URL:

http://msdn.microsoft.com/en-us/library/ms893193.aspx

NOTE

Because the SDIO function is not supported on 3-Stack Rev 2.0 board , the SDIO Wi-FI-related tests cannot be performed on these boards.

24.6.4.1 Storage Device Block Driver Read/Write Tests

Use the command line `tux -o -d rwtest -c “-z”` to run the tests. Note that this test tests only one card at a time.

24.6.4.2 Storage Device Block Driver API Tests

Use the command line `tux -o -d disktest -c “-z”` to run the tests. Note that this test tests only one card at a time.

24.6.4.3 File System Driver Test

Use command line `tux –o –d fsdtst –c "-p SDMemory –z"` to run the tests on an SD card.

Note that this test tests all the cards inserted and requires the cards to be formatted prior to running the test. For higher capacity cards, the test will take long time to complete, and hence it is recommended that the system power management (from control panel) should be configured so that the system does not enter suspend state during test execution.

24.6.4.4 Partition Driver Test

Use command line `tux –o –d msparttest –c "-z"` to run the tests.

Note that cards should be of size 256MB and higher. For higher capacity cards, the test will take long time to complete, and hence it is recommended that the system power management (from control panel) should be configured so that the system does not enter suspend state during test execution.

24.6.4.5 Wi-Fi Test

The Wi-Fi test suite requires 3 Wi-Fi access points to be present simultaneously, each configured for different encryption schemes. In case there is only 1 access point available, the tests can be split into 3 parts, depending upon encryption disabled, WEP 40-bit or WEP 104-bit respectively.

NOTE

Prior to running each part of the test, the system should be reset.

Follow the steps below.

Create an ad-hoc Wi-Fi link (SSID: CE-ADHOC1) on a WLAN supported laptop or other WLAN-supporting device.

Copy tux.exe, kato.dll, tooltalk.dll and wzctooltest.dll to the Windows folder of the 3-Stack board.

Insert an SDIO Wi-Fi card into the 3-Stack SD/MMC slot of the controller to be tested.

Part I:

Set up the WLAN access point – SSID: CEOPEN. WEP: disabled

On the 3-Stack board, run the command line `tux –o –d wzctooltest –x1-400,500,501`

Part II:

Setup the WLAN access point – SSID: CE-WEP40, WEP: 1/0x1234567890

On the 3-Stack board, run the command line `tux –o –d wzctooltest –x411-413,502,503,410,451-452,450`

Part III:

Setup the WLAN access point – SSID: CEWEP104, WEP: 1/qwertyuiopasd

On the 3-Stack board, run the command line `tux –o –d wzctooltest –x420`

Note that the access point should support complex encryption to run Part III. Also the Wi-Fi tests require `wzctool.exe` and `ipconfig.exe`, and hence the following catalog item needs to be included.

Catalog > Core OS > Windows CE Devices > Networking Features > Network Utilities

24.6.5 System Testing

The following system tests were performed to verify the operation of the SD and MMC memory cards.

- Use **Start** > **Settings** > **Control Panel** > **Storage Manager** to format and create partitions on the mounted memory cards.
- Establish ActiveSync connection over USB and transfer files to/from the memory cards.
- Write media files to memory storage. Use Windows Media Player to playback media files from memory storage.

The following system tests were performed to verify the operation of the SDIO Wi-Fi.

- Connect to a Wireless Access Point, and be able to lease a valid IP address.
- Ping other Wi-Fi devices in the range.
- Connect to the Internet and browse.
- Stream audio and video files from Internet websites or from local intranet sites.
- Access media files shared on a remote PC and play those files.

24.7 Secure Digital Card Driver API Reference

Detailed reference information for the Secure Digital Card driver may be found in Platform Builder Help under the heading “Secure Digital Card Driver Reference”, or in the online Microsoft documentation at the following URL:

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wceddk5/html/wce50lrfsecruredigitalcarddriverfunctions.asp

Chapter 25
Serial Driver

The i.MX31 board has five internal UARTs (Universal Asynchronous Receiver Transmitters): UART1, UART2, UART3, UART4 and UART5. The UART5 supports serial and slow infrared communication; the other UARTs support only serial communication. All of the UART modules are capable of standard RS-232 non-return-to-zero (NRZ) encoding formats, and UART5 in addition supports slow infrared modes. The serial port driver is implemented as a stream interface driver and supports all the standard I/O control codes and entry points. The serial port driver handles all the internal UARTs and the infrared I/O ports.

In the Windows CE 5.0 BSP implementation, the hardware-specific code that corresponds to the serial port driver's lower layer is implemented as the platform-dependent driver (PDD). This PDD will be linked with the Microsoft provided public serial MDD library (com_mdd2.lib) to form the complete serial port driver.

25.1 Serial Driver Summary

Table 25-1 identifies the source code location, library dependencies, and other BSP information:

Table 25-1. Serial Driver Summary

Driver Attribute	Definition
Target Platform (TGTPLAT)	3DS
Target SOC (TGTSOC)	MX31
MXARM11 CSP Driver Path	..\CSP\ARM\FREESCALE\MXARM11\DRIVERS\SERIAL
CSP Driver Path	..\CSP\ARM\FREESCALE\<TGTSOC>\DRIVERS\SERIAL
CSP Static Library	<TGTSOC>_serial.lib, mxarm11_serial.lib,Com_mdd2.lib
Platform Driver Path	..\PLATFORM\<tgtplat>\SRC\DRIVERS\SERIAL
Driver DLL	csp_serial.dll
Catalog Item for 3-Stack	Third Party > BSPs > Freescale <*tgtplat*> > Device Drivers > Serial > UART2 serial port hird Party > BSPs > Freescale <*tgtplat*> > Device Drivers > Serial > UART3 serial port
SYSGEN Dependency	N/A
BSP Environment Variables for 3-Stack	BSP_SERIAL_UART2 =1 BSP_SERIAL_UART3=1

25.2 Supported Functionality

The serial port driver enables the 3-Stack board to provide the following software and hardware support:

- Conforms to RS232 protocol standard

- Supports RTS/CTS hardware flow control function
- Supports up to 115200 BaudRate
- Supports internal UART controller
- Supports power management mode full on / full off

25.3 Hardware Operation

Refer to the chapter on the UART in the *MCIMX31 and MCIMX31L Applications Processors Reference Manual* for detailed operation and programming information.

25.3.1 Conflicts with Other SoC Peripherals

MX31 SDK has five internal UARTs: UART1, UART2, UART3, UART4 and UART5. In this UART1 and UART2 do not have conflicts with any other module and will be configured in functional mode. UART3 has conflicts with CSPI1 and CSPI3 modules and must be configured in alternate mode1. UART4 has conflicts with ATA and USB OTG modules and must be configured in alternate mode1. UART5 has conflicts with PCMCIA and USB modules and must be configured in alternate mode 2. Table 25-2 lists pins to be configured for serial driver for different UARTs.

Table 25-2. Pins to be Configured for Serial Driver

UART Port	Pins To Be Configured	I/O MUX Settings	Comment
UART1	RXD1, TXD1 RTS1, CTS1 DTR_DCE1 DSR_DCE1 RI_DCE1 DCD_DCE1	Functional Mode	not supported
UART2	RXD2 TXD2 RTS2 CTS2	Functional Mode	
UART3	CSPI3_MOSI CSPI3_MISO CSPI3_SCLK CSPI3_SPI_RDY	Functional Mode	not supported
UART4	ATA_CS0 ATA_CS1 ATA_DIOR ATA_DIOW	Alternate Mode1	not supported
UART5	PC_VS2 PC_BVD1 PC_BVD2 PC_RST	Alternate Mode 2	connect to GPS

25.4 Software Operation

The Serial driver follows the Microsoft-recommended architecture for serial drivers. For details of this architecture and its operation, see the Platform Builder Help:

Developing a Device Driver > **Windows CE Drivers** > **Serial Drivers** > **Serial Driver Development Concepts**

25.4.1 Power Management

Power management is not implemented in the serial driver.

25.4.2 MX31 Serial Registry Settings

The following registry keys are required to properly load the Serial driver.

```
; @CESYSGEN IF CE_MODULES_SERIAL
IF BSP_SERIAL_UART3
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\COM3]
   "DeviceArrayIndex"=dword:0
   "IoBase"=dword:5000C000
   "IoLen"=dword:D4
   "Prefix"="COM"
   "Dll"="csp_serial.dll"
   "Index"=dword:3
   "Order"=dword:9
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\COM3\Unimodem]
   "Tsp"="Unimodem.dll"
   "DeviceType"=dword:0
   "FriendlyName"="MGN COM3 UNIMODEM"
   "DevConfig"=hex: 10,00, 00,00, 05,00,00,00, 10,01,00,00, 00,4B,00,00, 00,00, 08, 00, 00,
00,00,00,00
ENDIF; BSP_SERIAL_UART3

IF BSP_SERIAL_UART2
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\COM2]
   "DeviceArrayIndex"=dword:0
   "IoBase"=dword:43F94000
   "IoLen"=dword:D4
   "Prefix"="COM"
   "Dll"="csp_serial.dll"
   "Index"=dword:2
   "Order"=dword:9
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\COM2\Unimodem]
   "Tsp"="Unimodem.dll"
   "DeviceType"=dword:0
   "FriendlyName"="mgn COM2 UNIMODEM"
   "DevConfig"=hex: 10,00, 00,00, 05,00,00,00, 10,01,00,00, 00,4B,00,00, 00,00, 08, 00, 00,
00,00,00,00
ENDIF  ; BSP_SERIAL_UART2
; @CESYSGEN ENDIF CE_MODULES_SERIAL
```

25.5 Unit Test

The serial driver is tested using the Serial Port Driver Test and Serial Communications Test included as part of the Windows CE 5.0 Test Kit (CETK). The Serial Port Test assesses whether the driver supports configurable device parameters, such as baud rate and data bits. The test also assesses additional functionality, such as COM port events, escape functions and timeouts.

25.5.1 Unit Test Hardware

The *<TGTSOC>* 3-Stack board with serial port is needed to run the unit tests. Serial ports are intended for communication between the i.MX31 and some 3-Stack peripherals. COM2 is connected to the Bluetooth interface, and COM3 is used for communication with the GPS daughter card

25.5.2 Unit Test Software

Table 25-3 lists the required software to run the unit tests.

Table 25-3. Software Requirements to Run Unit Tests

Requirements	Description
Tux.exe	Tux test harness, which is needed for executing the test
Kato.dll	Kato logging engine, which is required for logging test data
Tooltalk.dll	Library required by Tux.exe and Kato.dll. Handles the transport between the target device and the development workstation
SerDrvBvt.dll	Test .dll file for Serial Port Driver Test

25.5.3 Building the Serial Port Driver Tests

The serial port driver tests come pre-built as part of the CETK. No steps are required to build these tests. The `Pserial.dll` file is with the other required CETK files in the following location:

```
[Drive]:\Program Files\Windows CE Platform Builder\5.00\cepb\wcetk\ddtk\armv4I
```

25.5.4 Running the Serial Port Driver Test

The serial port driver test executes the command line `tux –o –d serdrvbvt -c “3”` to start the test on COM3.

CAUTION

Before performing this test, ensure that the GPS driver is excluded from the OS Design.

For detailed information about the Serial Port tests, see the Platform Builder Help:

Debugging and Testing > Windows CE Test Kit > CETK Tests > Serial Port Driver Test > Serial Port Driver Test Cases

Serial port tests are designed to test the serial port driver work properly and the API s behaves correctly, it should be pass all the test cases.

Table 25-4 describes the Serial Port driver test cases and its descriptions.

Table 25-4. Serial Port Driver Test Cases

Test Case	Description
1001	Configures the port and writes data to the port at all possible baud rates, data bits, parities, and stop bits. This test fails if it cannot send data on the port with a particular configuration.
1002	Tests the **SetCommEvent** and **GetCommEvent** functions. This test fails if the driver does not properly support the **SetCommEvent** or **GetCommEvent** functions.
1003	Tests the **EscapeCommFunction** function. This test fails if the driver does not support one of the Microsoft Win32 **EscapeCommFunction** functions.
1004	Tests the **WaitCommEvent** function on the EV_TXEMPTY event. The test creates a thread to send data and waits for the EV_TXEMPTY event to occur when the thread finishes sending data. This test fails if the **WaitCommEvent** function behaves improperly or if the EV_TXEMPTY event does not signal appropriately.
1005	Tests the **SetCommBreak** and **ClearCommBreak** functions. This test fails if the driver does not properly support the **SetCommBreak** or **ClearCommBreak** functions.
1006	Makes the **WaitCommEvent** function return a value when the handle for the current COM port is cleared. This test fails if the **WaitCommEvent** function behaves improperly.
1007	Makes the **WaitCommEvent** function return a value when the handle for the current COM port is closed. This test fails if the **WaitCommEvent** function behaves improperly.
1008	Tests the **SetCommTimeouts** function and verifies that the **ReadFile** function properly times out when no data is received. This test fails if the COM timeouts do not function correctly.
1009	Verifies that previous Device Control Block (DCB) settings are preserved when the **SetCommState** function call fails with DCB settings that are not valid. This test fails if the serial port driver does not keep previous DCB settings when DCB settings that are not valid are passed to the driver.

25.6 Serial Driver API Reference

For detailed reference information for the Serial driver, see the Platform Builder Help:

Developing a Device Driver > **Windows CE Drivers** > **Serial Port Drivers** > **Serial Port Driver Reference**

25.6.1 Serial PDD Functions

Table 25-5 lists a mapping of Serial PDD functions to the functions used in the Serial driver:

Table 25-5. Mapping of Serial PDD Functions to Serial Driver Functions

PDD Function Pointer	Serial Driver Function
HWInit	SerSerialInit
HWPostInit	SerPostInit
HWDeinit	SerDeinit
HWOpen	SerOpen
HWClose	SerClose

Table 25-5. Mapping of Serial PDD Functions to Serial Driver Functions(Continued)

PDD Function Pointer	Serial Driver Function
HWGetIntrType	SL_GetIntrType
HWRxIntrHandler	SL_RxIntrHandler
HWTxIntrHandler	SL_TxIntrHandler
HWModemIntrHandler	SL_ModemIntrHandler
HWLineIntrHandler	SL_LineIntrHandler
HWGetRxBufferSize	SL_GetRxBufferSize
HWPowerOff	SerPowerOff
HWPowerOn	SerPowerOn
HWClearDTR	SL_ClearDTR
HWSetDTR	SL_SetDTR
HWClearRTS	SL_ClearRTS
HWSetRTS	SL_SetRTS
HWEnableIR	SerEnableIR
HWDisableIR	SerDisableIR
HWClearBreak	SL_ClearBreak
HWSetBreak	SL_SetBreak
HWXmitComChar	SL_XmitComChar
HWGetStatus	SL_GetStatus
HWReset	SL_Reset
HWGetModemStatus	SL_GetModemStatus
HWGetCommProperties	SerGetCommProperties
HWPurgeComm	SL_PurgeComm
HWSetDCB	SL_SetDCB
HWSetCommTimeouts	SL_SetCommTimeouts

25.6.2 Serial Driver Macros

N/A

25.6.3 Serial Driver Structures

25.6.3.1 UART_INFO

This structure contains information about the UART Module.

```
typedef struct {
    volatile PCSP_UART_REG    pUartReg;
```

```
    ULONG    sUSR1;
    ULONG    sUSR2;
    BOOL    bDSR;
    uartType_c    UartType;
    ULONG    ulDiscard;
    BOOL    UseIrDA;
    ULONG    HwAddr;
    EVENT_FUNC    EventCallback;
    PVOID    pMDDContext;
    DCB    dcb
    COMMTIMEOUTS    CommTimeouts;
    PLOOKUP_TBL    pBaudTable;
    ULONG    DroppedBytes;
    HANDLE    FlushDone;
    BOOL    CTSFlowOff;
    BOOL    DSRFlowOff;
    BOOL    AddTXIntr;
    COMSTAT    Status;
    ULONG    CommErrors;
    ULONG    ModemStatus;
    CRITICAL_SECTION    TransmitCritSec;
    CRITICAL_SECTION    RegCritSec
    ULONG    ChipID;
} UART_INFO, * PUART_INFO;
```

Members

pUartReg	Pointer to UART Hardware registers.
sUSR1	This value contains the UART status register.
sUSR2	This value contains the UART status register.
bDSR	This Boolean value keeps the DSR state.
UartType	This value contains the type of UART like DCE or DTE.
UlDiscard	This is used to discard the echo characters in IrDa Mode.
UseIrDA	This Boolean value determines the driver is in IR mode or not.
HwAddr	This value contains the hardware address of the UART Module.
EventCallback	This is a callback to the Model Device Driver.
pMDDContext	This contains the context of the UART, which will be the first parameter to the callback function.
dcb	This value contains the copy of Device Control Block.
CommTimeouts	This contains the copy of CommTimeouts structure used to get and set the timeout parameters for a communication device.
pBaudTable	Pointer to baud rate table.
DroppedBytes	This value contains the number of bytes dropped.
FlushDone	Handle to the flush done event.
CTSFlowOff	This Boolean value is used to store the CTS flow control state.
DSRFlowOff	This Boolean value is used to Store the DSR flow control state.

AddTXIntr	This Boolean value is used to fake a Tx interrupt.
Status	This value contains the comm status.
CommErrors	This value contains Win32 comm error status.
ModemStatus	This value shows the Win32 Modem status.
TransmitCritSec	This value is used as Critical Section for UART registers.
RegCritSec	This value is used as Critical Section for UART.
ChipID	This value contains Chip identifier (CHIP_ID_16550 or CHIP_ID_16450)

25.6.3.2 SER_INFO

This private structure contains the information about Serial.

```
typedef struct __SER_INFO {
    UART_INFO    uart_info;
    BOOL    fIRMode;
    DWORD    dwDevIndex;
    DWORD    dwIOBase;
    DWORD    dwIOLen;
    PCSP_UART_REG pBaseAddress;
    UINT8    cOpenCount;
    COMMPROP    CommProp;
    PHWOBJ    pHWObj;
    BOOL        useDMA;
    DDK_DMA_REQ     SerialDmaReqTx;
    DDK_DMA_REQ     SerialDmaReqRx;
    PHYSICAL_ADDRESS SerialPhysTxDMABufferAddr;
    PHYSICAL_ADDRESS SerialPhysRxDMABufferAddr;
    PBYTE       pSerialVirtTxDMABufferAddr;
    PBYTE       pSerialVirtRxDMABufferAddr;
    UINT8       SerialDmaChanRx;
    UINT8       SerialDmaChanTx;
    UINT8       currRxDmaBufId;
    UINT8       currTxDmaBufId;
    UINT        dmaRxStartIdx;
    UINT        availRxByteCount;
    UINT32      awaitingTxDMACompBmp;
    UINT32      dmaTxBufFirstUseBmp;
    UINT16      rxDMABufSize;
    UINT16      txDMABufSize;
} SER_INFO, *PSER_INFO;
```

Members

uart_info	This structure contains information about UART.
fIRMode	This Boolean value determines the module is FIR or serial.
dwDevIndex	This static value contains the device index value which will be read from registry.
dwIOBase	This static value contains the IO Base address of UART module which will be read from registry.
dwIOLen	This static value contains the IO length of UART Module which will be read from registry.

pBaseAddress	Pointer to the start address of the UART registers mapped.
cOpenCount	This value contains count of the concurrent open.
CommProp	Pointer to CommProp structure
pHWObj	Pointer to PDDs HWObj structure.
useDMA	This Boolean flag indicates if SDMA is to be used for transfers through this UART
SerialDmaReqTx	SDMA request line for Tx
SerialDmaReqRx	SDMA request line for Rx
SerialPhysTxDMABufferAddr	Physical address of Tx SDMA address.
SerialPhysRxDMABufferAddr	Physical address of Rx SDMA address.
pSerialVirtTxDMABufferAddr	Virtual address of Tx SDMA address.
pSerialVirtRxDMABufferAddr	Virtual address of Rx SDMA address.
SerialDmaChanRx	SDMA virtual channel indices for Rx
SerialDmaChanTx	SDMA virtual channel indices for Tx
currRxDmaBufId	Index of the buffer descriptor next expected to complete its SDMA in the Rx SDMA buffer descriptor chains.
currTxDmaBufId	Index of the buffer descriptor next expected to complete its SDMA in the Tx SDMA buffer descriptor chains.
dmaRxStartIdx	This variables keep the start index of byte to be delivered to MDD for Read.
availRxByteCount	This variable keeps the remaining bytes in the Rx SDMA buffer.
awaitingTxDMACompBmp	This indicates if an SDMA request is in progress on Tx SDMA buffer descriptor.
dmaTxBufFirstUseBmp	Indicator for first time use of a Tx SDMA buffer descriptor.
rxDMABufSize	Receive DMA buffer size
txDMABufSize	Transfer DMA buffer size

Chapter 26
Touch Panel Driver

The Touch screen interface provides all the circuitry required for the readout of a four-wire resistive touch screen. The Touch screen X plate is connected to TSX1 and TSX2, and the Y plate is connected to TSY1 and TSY2. A local supply ADREF will serve as reference.

26.1 Touch Panel Driver Summary

Table 26-1 identifies the source code location, library dependencies, and other BSP information.

Table 26-1. Touch Panel Driver Summary

Driver Attribute	Definition
Target Platform (TGTPLAT)	3DS
Target SOC (TGTSOC)	MX31
MXARM11 CSP Driver Path	..\ CSP\ARM\FREESCALE\MXARM11\DRIVERS\TOUCH
CSP Driver Path	N/A
CSP Static Library	mxarm11_touch.lib
Platform Driver Path	..\PLATFORM\<*tgtplat*>\SRC\DRIVERS\TOUCH
Import Library	N/A
Driver DLL	touch.dll
Catalog Item	Third Party > BSPs > Freescale <*tgtplat*> > Device Drivers > MBX> MC13783 Touch Driver
SYSGEN Dependency	SYSGEN_Touch = 1
BSP Environment Variables	BSP_PMIC_MC13783=1,BSP_TOUCH_MC13783=1

26.2 Supported Functionality

The touch panel conforms to the standards explained in the Platform Builder Help:

Developing a Device Driver > Windows CE Drivers > Touch Screen Drivers

26.3 Hardware Operations

The hardware consists of an LCD Panel. For correct functioning, ADC module is needed, used to generate the touch samples which after proper calculation can be converted to the x,y coordinates. The ADC module and the Touch Interrupt are part of PMIC. For details, see Chapter 16.

26.3.1 Conflicts with SoC Peripherals

There is a conflict with the GPIO pin with which the PMIC Interrupt is routed. So those pins cannot be used as standard GPIO. See Chapter 16 for PMIC Interrupt routing and conflicts.

26.3.2 Conflicts with 3-Stack

The Touch Driver needs a Timer to provide the necessary timings between different ADC Samples. EPIT2 is dedicated to the Touch Panel and therefore is not available for use by any other module.

26.4 Software Operations

The touch screen driver reads user input from the touch screen hardware and converts it to touch events that are sent to the Graphics, Windowing, and Events Subsystem (GWES). The driver also converts uncalibrated coordinates to calibrated coordinates. Calibrated coordinates compensate for any hardware anomalies, such as skew or nonlinear sequences.

For the touch screen driver to work properly it must submit points while the user's finger or stylus is touching the touch screen. When the user's finger or stylus is removed from the screen, the driver must submit at least one final event indicating that the user's finger or stylus tip was removed. The calibrated coordinates must be reported to the nearest one-quarter of a pixel.

The following steps detail the basic algorithm that are used to sample and calibrate the screen with the touch screen driver:

1) Call the TouchPanelEnable function to start the screen sampling.

2) Call the TouchPanelGetDeviceCaps function to request the number of sampling points.

For every calibration point, perform the following steps:

1) Call TouchPanelGetDeviceCaps to get a calibration coordinate. A cross hair appears on the screen. Touching the cross hair starts the calibration

2) Call the TouchPanelReadCalibrationPoint function to get calibration data.

3) Call the TouchPanelSetCalibration function to calculate the calibration coefficients.

26.4.1 Touch Driver Registry Settings

```
IF BSP_NOTOUCH !

[HKEY_LOCAL_MACHINE\HARDWARE\DEVICEMAP\TOUCH]
    "DriverName"="touch.dll"
    "MaxCalError"=dword:10

; For double-tap default setting
[HKEY_CURRENT_USER\ControlPanel\Pen]
    "DblTapDist"=dword:18
    "DblTapTime"=dword:637

[HKEY_LOCAL_MACHINE\HARDWARE\DEVICEMAP\TOUCH]
    "MaxCalError"=dword:7
```

```
    "CalibrationData"="539,520 280,259 280,778 793,781 794,259"

; For TouchPanel calibration. Note that the Windows Mobile PocketPC touchpanel
; calibration is handled automatically by the welcome.exe application so a
; separate "Launch" registry key is not required. Also, Windows Mobile
; SmartPhone does not support a touchpanel at all which means that this is
; not required for SmartPhone either.
#if !defined IMGTPC && !defined IMGPPC
[HKEY_LOCAL_MACHINE\init]
    "Launch80"="touchc.exe"
    "Depend80"=hex:14,00, 1e,00
#endif ; !defined IMGTPC && !defined IMGPPC

ENDIF ; BSP_NOTOUCH !
```

26.5 Unit Tests

26.5.1 Unit Test Hardware

The EPSON L4F00242T03 VGA LCD Panel is needed to run the unit tests. This is the display panel required for display of graphics data.

26.5.2 Unit Test Software

Table 26-2 lists the required software to run the unit tests.

Table 26-2. Software Requirements

Requirements	Description
Tux.exe	Tux test harness, which is needed for executing the test
Kato.dll	Kato logging engine, which is required for logging test data
Tooltalk.dll	Library required by Tux.exe and Kato.dll. Handles the transport between the target device and the development workstation
Touchtest.dll	The Test .dll File
Touch.dll	Touch Panel Driver

26.5.3 Building the Touch Panel Tests

Use the following command to run the touch test cases: `tux -o -d touchtest.dll -x <Test case id>`

For information about the test case IDs, see the Platform Builder Help:

Debugging and Testing > Windows CE Test Kit > CETK Tests > Touch Panel Test

Test case number 8007 was not performed because the required API TouchPanelInitializeCursor is not implemented in the driver.

26.6 Touch Panel API reference

For the complete API reference, see the Platform Builder Help:

Developing a Device Driver > Windows CE Drivers > Touch Screen Drivers > Touch Screen Driver Reference

Chapter 27
USB OTG Driver

The OTG USB driver provides High Speed USB 2.0 host and device support for the USB “On The Go” (OTG) port of the i.MX31. The OTG driver will automatically select either host or device functionality at any given time, depending on the USB cable/mini-plug configuration. This is achieved by the set of three drivers: USB OTG host controller driver, USB 3-Stack client driver and/or USB transceiver controller (“Full Function”) driver, which performs the host/function client switching.

The USB host driver can be configured for class support for mass storage, HID, printer, and RNDIS peripherals. The device/client portion can be configured to provide one of mass storage, serial, or RNDIS function.

The “Full Function” OTG transceiver driver automatically selects between the host or client driver. The host or client can also be configured as the only mode for the OTG port, using the “Pure Host” or “Pure Client” catalog item. All the OTG catalog items are exclusive. (See summary sections below).

27.1 USB OTG Driver Summary

27.1.1 OTG Client Driver Summary

Table 27-1. OTG Client Driver Summary

Driver Attribute	Definition
Target Platform (TGTPLAT)	3DS
Target SOC (TGTSOC)	MX31
CSP Driver Path	PUBLIC*\CSP\ARM\Freescale\MX31\Drivers\USBD PUBLIC*\CSP\ARM\Freescale\MX31\Drivers\USBFN
CSP Static Library	mxarm11_usbfn.lib mx31_ufnmddbase.lib
Platform Driver Path	\PLATFORM\3-Stack\SRC\DRIVERS\USBD
Import Library	N.A
Driver DLL	usbfn.dll
Catalog Item	High Speed OTG: Third Party > BSPs > Freescale 3-Stack: ARMV4I > Device Drivers > USB Devices > USB High Speed OTG Device To support only client/device mode, choose .. > High Speed OTG Port Pure Client Function.

Table 27-1. OTG Client Driver Summary(Continued)

Driver Attribute	Definition
SYSGEN Dependency	SYSGEN_USBFN=1
BSP Environment Variable	BSP_USB=1 BSP_USB_HSOTG_CLIENT=1

Note that USB clients require a function driver to be loaded. A client can only present one function. Only one of the function drivers should be configured through drag and drop. If more than one is configured, the Serial function will be default unless the registry is manually modified.

27.1.2 OTG Host Driver Summary

Table 27-2. OTG Host Driver Summary

Driver Attribute	Definition
Target Platform (TGTPLAT)	3DS
Target SOC (TGTSOC)	MX31
CSP Driver Path	PUBLIC*\CSP\ARM\Freescale\MX31\Drivers\USBH\EHCI PUBLIC*\CSP\ARM\Freescale\MX31\Drivers\USBH\EHCIPDD PUBLIC*\CSP\ARM\Freescale\MX31\Drivers\USBH\USB2COM
CSP Static Library	mx31_ehcdmdd.lib mx31_ehci_lib.lib mx31_hcd2lib.lib
Platform Driver Path	\PLATFORM*3-Stack*\SRC\DRIVERS\USBH\HSOTG
Import Library	N.A
Driver DLL	hcd_hsotg.dll
Catalog Item	Third Party > BSPs > Freescale 3-Stack: ARMV4I > Device Drivers > USB Devices > USB High Speed OTG Device To support only host mode, choose .. >High Speed OTG Port Pure Host Function.
SYSGEN Dependency	SYSGEN_USB=1
BSP Environment Variable	BSP_USB=1 BSP_USB_HSOTG_HOST=1

NOTE

The Host driver requires that a set of class drivers be loaded. See High Speed Host H2 (22.5) for class driver information.

27.1.3 OTG Transceiver Driver Summary (For HIGH-SPEED only)

Driver Attribute	Definition
Target Platform (TGTPLAT)	3DS
Target SOC (TGTSOC)	MX31
CSP Driver Path	PUBLIC*\CSP\ARM\Freescale\MX31\Drivers\USBXVR
CSP Static Library	mx31_xvc.lib
Platform Driver Path	PLATFORM\3-Stack\SRC\DRIVERS\USBXVR
Import Library	N/A
Driver DLL	imx_xvc.dll
Catalog Item	Third Party > BSPs > Freescale 3-Stack: ARMV4I > Device Drivers > USB Devices > USB High Speed OTG Device > High Speed OTG Port Full OTG Function
SYSGEN Dependency	SYSGEN_USBFN=1
BSP Environment Variable	BSP_USB=1 BSP_USB_HSOTG_CLIENT=1 BSP_USB_HSOTG_HOST=1 BSP_USB_HSOTG_XVC=1

27.2 Supported Functionality

The OTG driver enables the 3-Stack board to provide the following software and hardware support:

- The High Speed OTG driver supports USB specification 2.0.
- Driver is configured as client/peripheral by default, with one function driver defined as default. When nothing is connected to the OTG port, the port does not drive Vbus and awaits attachment to a host by raising its D+ signal. On attachment of a mini-A plug the driver switches to host mode as described below.
  - When a mini-B plug is connected to the OTG port, and the cable's opposite end is connected through mini-A plug to another OTG device, or through A-type plug to a host, then the OTG initiates operation as peripheral and responds to USB protocol from the host.
  - When a mini-A plug is connected to the OTG port and the cable's opposite end is connected through mini-B plug to another OTG device, then the OTG initialize/re-initializes itself into host mode and begins to act as a host. The OTG port remains in host mode whenever a mini-A plug is mated to the OTG socket connector.
- The OTG port as client/peripheral supports mass storage, RNDIS, and serial clients
- The OTG port as host supports mass storage, printer, HID, and RNDIS classes
- When nothing is attached to the OTG port, the driver configures the controller and transceiver into a low power state.
- When the system is suspended with nothing attached to the OTG port, the system wakes upon attachment of the port to a host or attachment of a device with mini-A plug.

- When the system is suspended while the OTG port is connected to a host or to a device with a mini-A plug, the system remains suspended when the OTG port connection is unplugged.
- When the system resumes after suspend, any attached devices are enumerated and their class drivers loaded appropriately.
- Data transfer rates on the client port exceed 40 Mbits/sec in Mass Storage client.

27.3 Hardware Operation

There is an OTG mini socket on the 3-Stack board (J10). The i.MX31 contains a USB 2.0 core for handling OTG, which is connected to the external transceivers through IO multiplexer (IOMUX). Options are possible on the i.MX31 hardware (such as routing signals on H1 signal port to OTG transceiver, with both controllers “offline”). This simply allows external peripherals direct connection to a transceiver. The OTG driver currently supports only one hardware MUX configuration.

The ID Pin Detect is supported through the Transceiver Driver (for High Speed OTG), and is constructed as a stream interface driver.

Please go through the sample reference implementation that is provided with Windows CE 5.0 installation first to get more detail understanding on how USB Host controller and USB Function controller driver are structured.

The i.MX31 processor supports speed translation within the USB 2.0 controller, a non-standard EHCI implementation. As a result software does not currently support full/low speed devices (aside from those non-FS hub devices directly connected to the OTG port).

The 3-Stack can supply a total of 100mA to attached devices on the OTG port and the default behavior does not need to be modified.

All bus powered hubs that have been tested require 500mA and therefore are not supported for use with the 3-Stack. Self-powered hubs are required to expand the number of USB sockets and also to support devices that require greater than 100mA (For Ex : Mini HDD devices should be connected through self powered Hub).

27.3.1 Conflicts with other Peripherals

The High Speed OTG port conflicts with UART4. The USB controller drivers coordinate in their management of the USB peripheral block clock and processor core voltage, as described in this chapter in Section 27.4.4, “Power Management”.

27.4 Software Operation

27.4.1 USB OTG Host Controller Driver

This driver enables the USB host functionality for the OTG port. It is part of the standard Windows USB software architecture as shown in Figure 27-1.

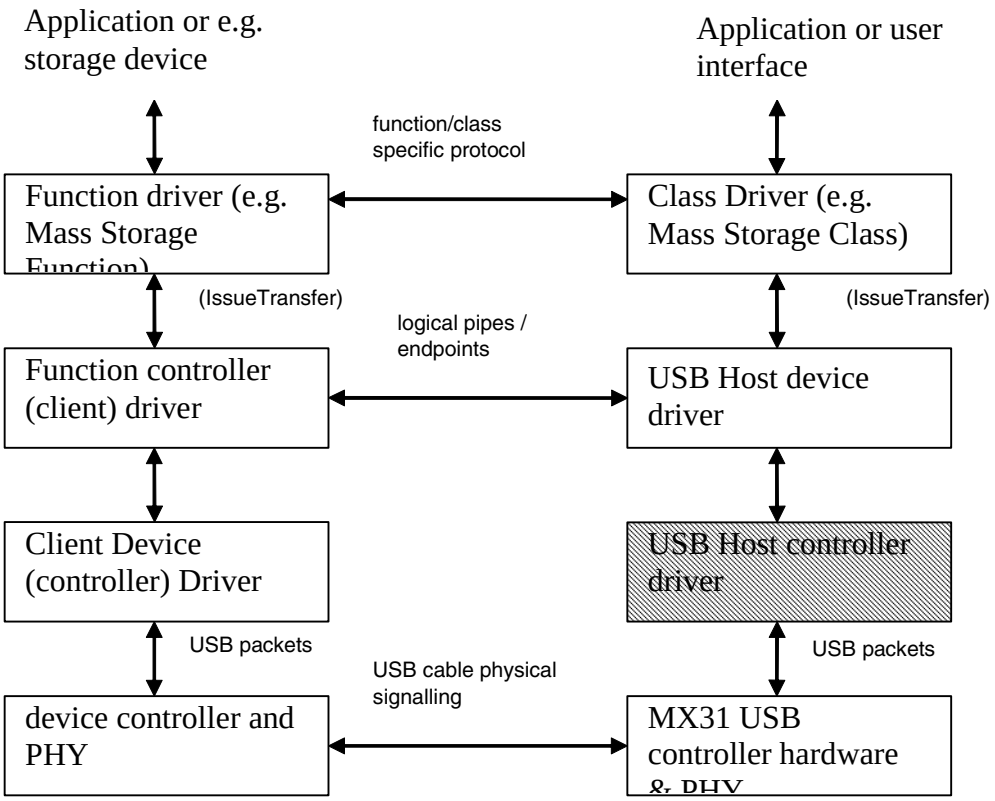

Figure 27-1. Windows USB Driver Architecture

For details of the Windows CE USB driver architecture and usage, see the Platform Builder Help:

Developing a Device Driver > **Windows CE Drivers** > **USB Host Drivers**

and

Developing a Device Driver > **Windows CE Drivers** > **USB Host Drivers** > **USB Host Controller Drivers** > **USB Host Controller Driver Development Concepts**

When transceiver mode is included, the host driver is activated when a USB Mini-A plug is connected to the Mini USB OTG socket. When Pure Host mode only is selected, the host driver is always in control of the relevant USB controller.

When a USB device is connected to the Mini USB OTG socket of the 3-Stack, the host controller driver enumerates and activates the appropriate class driver.

Windows CE 5.0 supports the following USB class drivers:

- Mass Storage -- SD cards, MMC cards, CF cards, HDD drive, thumb drive (disk-on-key).
- Note: some card reader firmware is not supported by the Microsoft standard Mass Storage class driver.
- HID -- Keyboard and mouse
- Printer
- RNDIS – Network Device Interface communication class

Hubs are supported, in all configurations with Full and Low Speed peripherals.

This version of the host controller driver has been verified against the following USB 2.0 vendor devices:

- Various disk-on-key including Kingston and Memorex
- HP HS self-powered and bus-powered hub
- External self-powered HS hard drive
- HS and FS card reader with CF and MMC storage. (Note: there are known issues formatting large CF cards some FS card readers)
- HP Photosmart 7450 printer and Lexmark E232 Laser Printer
- A variety of cameras, including Sony Cybershot and HP717
- Logitech keyboard and mouse

27.4.1.1 User Interface

As described above, user access to the USB host driver is through class drivers. For details about the Host Client Drivers, see the Platform Builder Help:

Developing a Device Driver > **Windows CE Drivers** > **USB Host Drivers** > **USB Host Controller Drivers** > **USB Host Client Drivers**

Where new class driver code is to be developed, refer to the Host client driver interface functions (for example, IssueBulkTransfer) as documented in:

Developing a Device Driver > **Windows CE Drivers** > **USB Host Drivers** > **USB Host Controller Drivers** > **USB Host Client Drivers** > **Host Client Driver Reference**

27.4.1.2 Host Controller Configuration

Configure the driver into the BSP build by dragging the appropriate catalog item for USB HS OTG. By default, host support is included along with peripheral/device and transceiver support. Select any additional classes to be supported from the Core OS catalog. See Registry Setting OTGSupport (below) for details on excluding OTG host support from the build.

The internal i.MX31 USB OTG signals can be multiplexed to a choice of pins on IC, as described for the IOMUX in the *MCIMX31 and MCIMX31L Applications Processors Reference Manual*.

27.4.1.3 Memory Configuration

The USB Host drivers (for all USB host ports) use DMA to perform all USB transfers. The physical memory for these transfer buffers is allocated as a pool at driver initialization. Unless physical addresses are specified in API accesses at the class-driver interface, the driver copies data between the user/class-provided data buffers and the DMA buffer from the driver physical memory pool.

The default DMA physical memory pool size is 128 kB. This value can be altered by registry setting PhysicalPageSize.

27.4.1.4 Vbus/Configured Power

USB provides a means to monitor the configured power of devices attached to a USB host. The host driver verifies that each attached device will not exceed the configured power limit.

This power limit is implemented through the platform-specific function, `BSPUsbhCheckConfigPower()`, as described in Section 27.4.1.8.1, "BSPUsbhCheckConfigPower," and located in the following directory:

```
PLATFORM\3DS\SRC\Drivers\USBH\Common\hwinit.c
```

This function must be modified to correspond with the platform hardware capabilities.

The i.MX31 3-Stack can supply a total of 100mA to attached devices on the OTG port and the default behavior does not need to be modified.

All bus powered hubs that have been tested require 500mA and therefore are not supported for use with the 3-Stack. Self-powered hubs are required to expand the number of USB sockets and also to support devices that require greater than 100mA.

27.4.1.5 Registry Settings

The USB OTG host controller settings are values located under the registry key:

```
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\HCD_HSOTG]
```

The values under this registry key are automatically included in the image through platform.reg. They do not normally require customization.

Default values are contained in `hsotg.reg`.

Table 27-3. Registry Settings

Value	Type	Content	Description
Dll	sz	hcd_hsotg.dll	Driver dynamic link library
OTGSupport	dword	01	This value must be set to 1 to enable host driver on the OTG. If no host support is required (client only) then this value can be set to 0, though the HCD_HSOTG key is not normally configured in the image at all when pure Host function is selected.
OTGGroup	sz	01	This unique string (example "00" to "99") is used to combine/correlate instances of the host, function, and transceiver driver within one USB OTG instance. Only one instance of the OTG is actually supported currently on the i.MX31 hardware.
HcdCapability	dword	4	HCD_SUSPEND_ON_REQUEST. Note: HCD_SUSPEND_RESUME is always assumed.
PhysicalPageSize	dword	20000	This value represents the number of bytes allocated for the physical memory pool of the OTG host driver, and defaults to 128kB. From this buffer, 75% are allocated for transfer descriptors and the remaining buffer is available for allocation to simultaneous transfers. In most cases, only one transfer is active at any time (for example, in the Mass Storage Class). A good value will be at least 3x as large as the largest data buffer transferred using IssueTransfer().

27.4.1.6 Host USB Test Modes

The USB 2.0 specification defines PHY-level test modes for USB host ports (see definitions in USB 2.0 specification section 7.1.20).

The i.MX31 USB host drivers support “Packet” test mode. The test mode is configured by compiling the BSP with the compilation flag OTG_TEST_MODE defined within `bsp_cfg.h`:

```
#define OTG_TEST_MODE
```

This configures the appropriate host controller within the platform-specific hardware initialization function (ConfigOTG()), located in:

```
PLATFORM\3DS\SRC\Drivers\USBH\Common\hwinit.c
```

The test mode is entered upon initialization, and cannot be exited. Normal USB operation is disabled when test mode support is compiled into the image.

27.4.1.7 Unit Test

The USB driver has many devices to be tested. Tests are performed manually and include connecting the devices, and confirming the attach, detach (on unplug) re-attach (on subsequent plug in of device), and transferring and verifying data (and/or functions).

To verify the RNDIS class device, a CEPC containing Netchip 2280 USB function is attached, and used to access a remote file server on the CEPC.

To verify the low-level transport for Bulk, Interrupt and Isochronous transfers, the CETK Host test kit is performed. This requires a CEPC configured with Netchip 2280 USB function and loopback driver.

27.4.1.7.1 USB Host Controller Driver Test

The information in this section is (c) 2001 Microsoft Corporation, and is documentation for the Windows CE 5.0 CETK USB Host tests.

For information, see the Platform Builder Help:

Debugging and Testing > **Windows CE Test Kit** > **CE Test Kit**

27.4.1.7.2 Building the Image to be Tested

To build the image to test:

1. Check out the RTM to be tested or install the MSI provided.
2. Add the following components from the catalog:
   - Freescale 3DS: ARMV4I - Device Drivers - USB Devices - USB High Speed Host 2
   - Core OS - Windows CE devices - Core OS Services- USB HOST Support; and all the sub-components of this catalog item (Sub-Components like USB Storage Class Driver)
   - Core OS - Windows CE devices - File Systems And Data store- Storage Manager; (Sub-Components: FAT File System, Partition Driver, Storage Manager control panel applet)
   - Device Drivers- USB Function-USB Function Clients - Serial.

27.4.1.7.3 Abstract

This test suite can be used to test USB host controller drivers that provide the same interface as Window CE USB host controller driver does (for more information, please see 22.4.1.1), also it can be used to verify whether a certain USB host controller (either stand alone card or onboard logic) can work with Windows CE or not.

The test setup and scenario is shown in Figure 27-2.

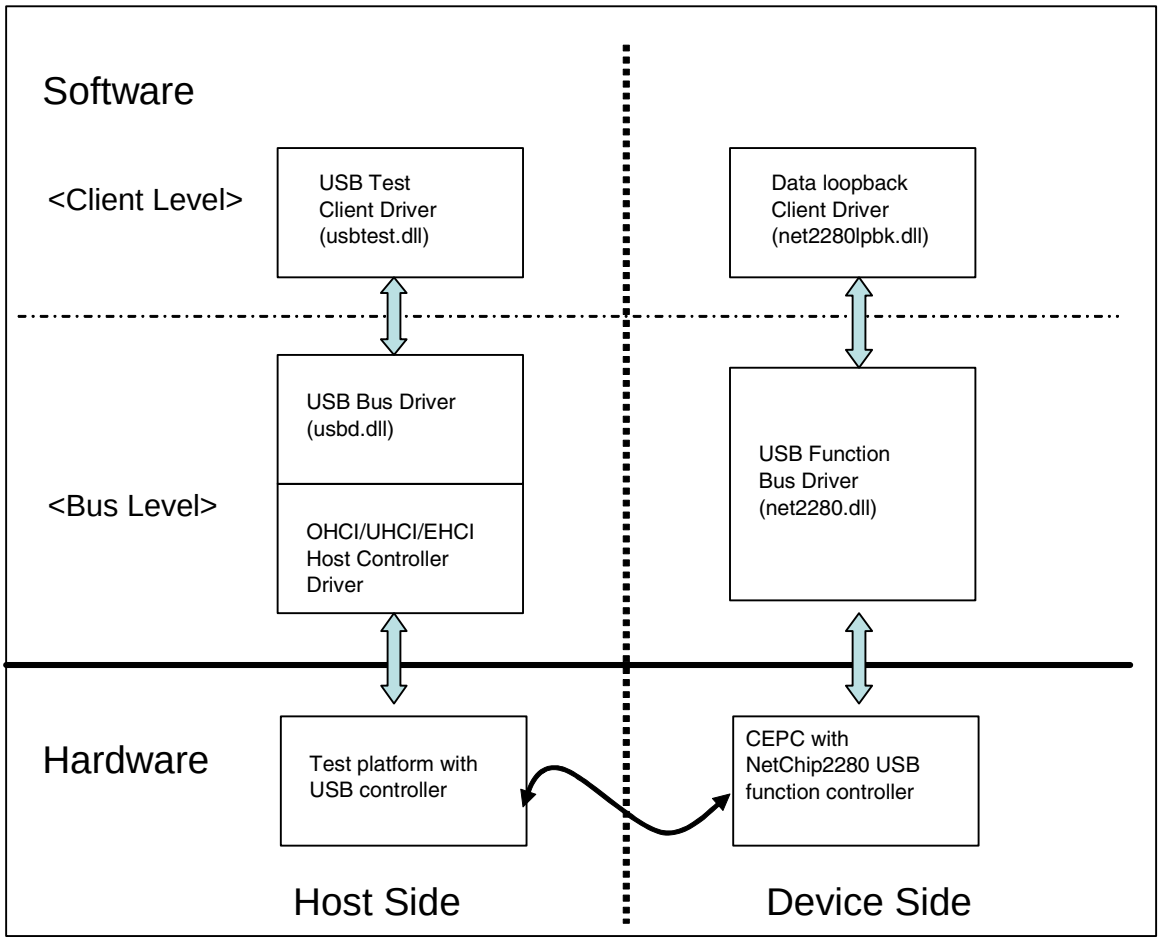

Figure 27-2. Test Setup and Scenario

This test suite acts as a client driver above USB bus driver (`usbd.dll`), it will be loaded when test device is connect to host through USB cable. Here, test device is a CEPC with a NetChip2280 USB function controller card in it. After this CEPC is booted up and net2280lpbk.dll got loaded, the whole CEPC will act as a generic USB data loopback device. USB test suite (the test client driver on the host side) can then streaming data or issue device requests to and from this data loopback device; this is how the USB host controller and its corresponding host controller drivers got exercised.

NetChip2280 USB function PCI controller card is a USB2.0 compatible USB function device, so it can be used to test both USB2.0 and USB1.1 host controllers (EHCI/OHCI/UHCI) and corresponding drivers.

Netchip2280 controller has 6 endpoints besides endpoint 0. The data loopback driver (net2280lpback.dll) configures these endpoints to be three pairs: one bulk IN/OUT pair, one Interrupt IN/OUT pair, and one Isochronous IN/OUT pair. The data loopback tests are done by sending data from host side to device side through OUT pipe, and receive it back through IN pipe, and then verify the data.

27.4.1.7.4 Hardware Requirements

- Test platform
- Host Controller Card (if not onboard logic)
- CEPC
- Netchip2280 Card
- USB cable

27.4.1.7.5 Required Software

Host side:

- Tux.exe
- Ddlx.dll
- Usbtest.dll
- Tooltalk.dll
- Kato.dll
- USB component (usbd.dll, EHCI/OHCI/UHCI host controller driver(s)) must be included in the run time image.

Device side:

- Lufldrv.exe
- Net2280lpbk.dll
- NetChip2280 USB function support (net2280.dll) must be included in the CEPC run time image.

27.4.1.7.6 Running the Test

To run the test:

1. Download runtime image to CEPC with Netchip2280 card on it.
2. After the system is booted up, run the command `s lufldrv`; the tester should verify that `net2280lpbk.dll` is loaded.
3. Download the runtime image to test the platform with USB host controller on it.
4. After the system is booted up, make sure there is NO connection between host side and device through USB cable. Then launch command `s tux –o –d ddlx –c “usbtest” “–xYYYY”`, where “`YYYY`” is the test case(s) you want to run.
5. The test will first remind you to make sure there is NO connection between host and device side. Then after 7 seconds, the test will ask you to connect two sides with USB cable.
6. After you have connected two sides with USB cable, the test main body will start to run.

7. After test(s) is(are) done, and if you want to run some other tests in the test suite, you do not need to disconnect two sides, just type the test command, and the tests will start directly. Of course, if you disconnected the USB connection before the second run, tests will ask you to connect again before start the real test part.

27.4.1.7.7 Test Cases

The test suite contains following test cases:

Test Case ID	1001-1315, 1501-1515
Test Case ID	1001-1315, 1501-1515
Test Description	Data loopback tests. Concerning the transfer type, there are five categories: 1) Bulk pipe loopback tests (tests with ID end with 1, like xxx1), 2) Interrupt pipe loopback tests (tests with ID end with 2, xxx2), 3) Isochronous pipe loopback tests (tests with ID end with 3, xxx3), 4) All pipe transfer simultaneously (tests with ID end with 4, xxx4), 5) All three types of transfer carry on simultaneously (tests with ID end with 5, xxx5)[1]. Concerning about how data is being transferred, there are also five categories: 1) Normal loopback tests (tests with ID start with 10, like 10), 2) loopback tests using physical memory (tests with ID start with 11, 11xx), 3) loopback tests using a part of allocated physical memory (tests with ID start with 12, 12xx), 4) Normal short transfer loopback tests ((tests with ID start with 13, 13xx), 5) Stress short transfer loopback tests ((tests with ID start with 15, 15xx), Also, both synchronous and asynchronous transfer methods (test cases like xx1x) are exercised using asynchronous transfer method, test cases like xx0x using synchronous method. So totally there are $5 \times 5 \times 2 = 50$ test cases.

[1] This category of tests is designed for testing some other USB function devices which have more endpoints than the host controller driver can handle. When using Netchip2280, it should be the same as category 4). Tester can just ignore this category.

Test Case ID	1401-1413
Test Description	Some additional data loopback tests. They mainly focus on testing APIs like GetTransferStatus(), AbortTransfer() and CloseTransfer().

Test Case ID	2001-2013
Test Description	Test related with Device requests.

Test Case ID	9001-9004
Test Description	These are some special tests that test APIs like SuspendDevice(), ResumeDevice() and DisableDevice().

Test Case ID	9005
Test Description	This is a test that stresses EP0 transfer (Vendor Transfer)

By default the data loopback device configures the endpoints with some often-used packet sizes; they are DWORD aligned, neither too big nor too small. By having all tests list above passed under this configuration is more than good enough for a BVT-type test pass. However, testers can change the packet sizes (these values are hard-coded in the source code for net2280lpbk.dll) for each endpoint by themselves and run these test cases again, this will be a good enrichment for testing.

This test suite has provided a way to change packet sizes of on NetChip2280 device on the fly. They are:

1. Test case 3001: Using some very small packet sizes in NetChip2280 device's full speed configuration.
2. Test case 3002: Using some very small packet sizes in NetChip2280 device's high speed configuration.
3. Test case 3003: Using some irregular packet sizes (like non DWORD-aligned size) in NetChip2280 device's full speed configuration.
4. Test case 3004: Using some irregular packet sizes (like non DWORD-aligned size) in NetChip2280 device's high speed configuration.
5. Test case 3005 (High Speed only): Using some very large packet sizes (like 2 × 1024 for Isochronous endpoints) in NetChip2280 device's full speed configuration. (Note: In the real world, Netchip2280 cannot handle transfers using such large packet size because its onboard FIFO buffer is small.).

What testers need to do is to run one of the test case above like running those normal tests, then after 15~20 seconds, through PB you should see usbtest.dll being unloaded and loaded again automatically. It means the packets sizes on netchip2280 side have already been changed. Then you can run those normal tests. You can use test case 3011(for full speed config) and 3012 (for high speed) to restore default packet sizes.

Another category test that is important for USB2.0 host controllers and drivers is called golden bridge tests, which means USB2.0 host controller is connected with a full speed (USB1.1) device. This is the only scenario that USB2.0 host controller will perform split transfers.

NetChip2280 can be forced to be a full speed device: In the test setup stage, instead of run s lufldrv to load loopback driver, run s lufldrv –f", this will force Netchip2280 to be configured as a full speed device.

Also testers are encouraged to do some manual tests. Here are some examples:

1. Plug in some real USB devices, suspend system, and then resume. USB devices should still be there.
2. Plug in some real USB devices, suspend system, unplug it, plug in another device, then resume. System should enumerate that new device properly.
3. Run one of the data transfer tests, in the middle of transfer stage, suspend the system (host side), then resume. Tests may fail, but system should not crash.
4. Run one of the data transfer tests, in the middle of transfer stage, disconnect the USB connection. Tests should fail, but system should not crash.

27.4.1.8 Platform-Specific API

27.4.1.8.1 BSPUsbhCheckConfigPower

This function is used to evaluate whether a device can be supported on the specified USB port.

Parameters:

UCHAR bPort [in] Unused. Each USB controller has only one port

DWORD dwCfgPower

[in] Power requirement (number of milliamps) requested by the device being evaluated for attachment support on this port

DWORD dwTotalPower

[in] current total power (number of milliamps) used by other previously attached devices on this port

Return Value: return TRUE if device requesting dwCfgPower can be safely attached

return FALSE if device cannot be attached

27.4.1.8.2 BSPUsbSetWakeUp

This function must do what is necessary to enable or disable wakeup on the USB port. For example, this function does not actually enable wake-up when a device is currently attached to the port.

Parameters:

BOOL bEnable [in] TRUE to enable wakeup, FALSE to disable wakeup

27.4.1.8.3 BSPUsbCheckWakeUp

This function evaluates the wake-up condition for the relevant USB port, and clears the condition and interrupt.

Parameters: None

Return Value: return TRUE when a wake-up condition was detected

return FALSE when no wake-up condition was present

27.4.1.8.4 SetPHYPowerMgmt

This function is called by the USB driver when transitioning to/from the suspended state (for example, during system suspend). The function must do what is necessary to place the transceiver hardware into a suspended (fSuspend == TRUE) or running (fSuspend == FALSE) state.

The standard implementation for the i.MX31 3-Stack board uses a ULPI-bus based ISP1504 transceiver for the HS OTG port, and this function configures the ULPI-bus for sleep state. If platform hardware uses other transceivers, this function would need to be modified appropriately.

Parameters:

BOOL fSuspend [in] TRUE: system/controller is going to suspend mode. FALSE: resuming

27.4.2 USB Client Driver

This driver enables the USB device functionality for the i.MX31. It gets activated when a USB Mini B connector is connected to the Mini USB OTG socket. When the i.MX31 3-Stack board is connected to a USB host system (ex: high speed or full speed port of PC), it is enumerated and according to the current configuration settings, and the appropriate class driver gets loaded on the PC. By default the 3-Stack board is configured for USB serial class. 3-Stack board can be configured as one of the following USB functions by setting the appropriate environment variable during build (drag/drop from the catalog).

Serial class -- Serial ActiveSync

Mass storage -- expose local storage (ATA hard disk, RAMDISK or other store) as USB drive

RNDIS class (Remote Network Driver Interface Specification)

27.4.2.1 User Interface

The USB client driver provides a standard Windows CE USB driver implementation. For an overview see section 33.4 and:

Developing a Device Driver > Windows CE Drivers > USB Function Drivers > USB Function Controller Drivers

User access to the USB client driver is through function drivers, such as Mass Storage or RNDIS.

For further details on these USB Function drivers, see the Platform Builder Help:

Developing a Device Driver > Windows CE Drivers > USB Function Client Drivers

Where new function driver code is to be developed, refer to the Function controller driver interface functions (for example, IssueTransfer) as documented in:

Developing a Device Driver > Windows CE Drivers > USB Function Controller Drivers > USB Function Controller Driver Reference

27.4.2.2 Client Driver Configuration

The OTG client driver is configured into the BSP build by dragging the appropriate catalog item. (See 22.1.1) When the “Pure Client” functionality is selected, the OTG port acts only as a device. When “Full OTG functionality” is selected, the OTG port can be either device or host (see transceiver driver configuration).

27.4.2.3 Registry Settings

The USB OTG function/client settings are values located under the registry key:

```
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\UFN]
```

The values under this registry key are automatically included in the image through platform.reg. They do not normally require customization.

Value	Type	Content	Description
Dll	sz	usbfn.dll	Driver dynamic link library
OTGSupport	dword	01	This value must be set to 1 to enable the function/client on the OTG. If no client support is required (host only) then this value can be 0, though the UFN key is not normally configured in the image at all when pure Host function is selected.
OTGGroup	sz	01	This unique string (example “00” to “99”) is used to combine/correlate instances of the host, function, and transceiver driver within one USB OTG instance. Only one instance of the OTG is actually supported currently on the i.MX31 hardware.

27.4.2.4 Device USB Test Modes

The USB 2.0 specification defines PHY-level test modes for USB device ports (see definitions in USB 2.0 specification section 7.1.20). This mechanism allows a host to configure a device into test mode by commanding the device with a specific SET_FEATURE request. Once test mode is entered, the device would not be able to leave test mode.

The device port does not by default support the USB test modes. Sample code for test mode support (SET_FEATURE on the device) is included in:

```
PUBLIC\COMMON\OAK\CSP\ARM\FREESCALE\MX31\DRIVERS\USBFN\CONTROLLER\MDD
```

In addition, USBFN_TEST_MODE_SUPPORT must be defined during compilation of the CSP USBD device driver library.

27.4.2.5 Unit Test

There is no CETK test case for USB client (function) drivers. The USB Function is tested by configuring the i.MX31 3-Stack board as either USB Serial function or USB Mass storage or RNDIS function and connecting it directly to a Host PC. The test verifies basic USB peripheral/client functionality, including attach, detach, and data transfer.

Separate images must be built and downloaded for each of the three peripheral function tests. See Section 27.4.1.7.2, “Building the Image to be Tested” for instructions.

27.4.2.5.1 Unit Test Hardware

Table 27-4 lists the required hardware to run the unit tests.

Table 27-4. Hardware Requirements

Requirements	Description
Host system	To test if control reaches the Host controller driver.
USB cable having Mini USB OTG plug A at one end and Mini USB OTG plug B on the other side.	For connecting between the host and the device.
ATA drive configured in UDMA mode 2 as DSK1	This is required as a storage device when the board is configured as mass storage class.

27.4.2.5.2 Unit Test Software

The ActiveSync 4.1 (and above) software is needed to perform the test. This is the host-side software that is required to be available for testing the Serial class functionality.

27.4.2.5.3 Running the USB Function Controller Driver Tests

The following table lists USB Function controller driver tests.

Table 27-5. USB Function Controller Driver Tests

Test Cases	Entry Criteria/Procedure/Expected Results
Board configured as USB Serial class and connected to a host system after the board boots up completely.	Entry Criteria: Make sure there is no mini USB OTG plug B is connected and the board is turned on and wait till the board boots-up completely. Procedure: 1. Connect the mini USB OTG plug B to the mini USB OTG socket. 2. Observe that the ActiveSync on the host side gets connected and is synchronized. 3. Copy files from Host system to the Mobile Device. Files will get copied. 4. Copy files from the Mobile Device to the Host system. Files gets copied. 5. Unplug the mini USB OTG plug B from the i.MX31 mini USB OTG socket to unload the Serial class driver. Expected Result: ActiveSync should get synchronized and copying of files should happen between the Host and the 3-Stack board.
Board configured as USB Mass storage client, with ATA drive as DSK1 mounted, and connected to a host system after the board boots up completely.	Entry Criteria: Make sure there is no mini USB OTG plug B is connected and the board is turned on and wait till the board boots-up completely. Procedure: 1. Connect the mini USB OTG plug B to the mini USB OTG socket. 2. Observe that a new disk in My Computer having as Removable Disk appearing in it. 3. Copy files from Host system to the new disk drive. Files will get copied. 4. Copy files from the new disk drive to the Host system. Files gets copied. 5. Unplug the mini USB OTG plug B from the 3-Stack mini USB OTG socket to unload the mass storage class driver. Expected Result: Files copied into mass storage client device match those copied out (when compared on Windows XP PC using file compare utility). Note that files will not be visible from within the 3-Stack system until the system has been reset. The file system should not be used inside the 3-Stack when it is being accessed through USB as a mass storage client.
Board configured as USB RNDIS client and connected to a host system after the board boots up completely. Browsing the Internet.	Entry Criteria: Make sure there is no mini USB OTG plug B is connected and the board is turned on and wait till the board boots-up completely. See to it that the NIC's local area connection is not having any IP address. Procedure: 1. Connect the mini USB OTG plug B to the mini USB OTG socket. 2. Observe that a new Local area connection in the Network and Dial up connections appears on the Windows XP machine. Bridge the NIC's local area connection and the RNDIS's local area connection. 3. Configure the bridge by giving IP address, Subnetmask, Default gateway, DNS, and so on. 4. On the 3-Stack board, a new Local area connection can be found in the Network and dial up connections. Configure the local area connection by giving IP address, Subnetmask, Default gateway, DNS, and so on. 5. In the Internet explorer on the 3-Stack board, configure the LAN settings as per the local area settings. Expected Result: Browsing the Internet should be possible.

27.4.2.6 Platform-Specific API

27.4.2.6.1 InitializeMux

This function is called to initialize the IOMUX connection within i.MX31, from USB controller to the appropriate device pins for the transceiver.

This function is implemented for the Pure Client situation.

Parameters int Speed

[in] Unused

Return Value return TRUE if device requesting dwCfgPower can be safely attached

27.4.2.6.2 HardwarePullupDP

This function is called by the USB client driver when D+ must be pulled-up, in preparation for connection to a USB host. The standard code configures for ISP1504/ISP1301 transceiver. It is possible to modify this routine to conditionally soft-disable USB connection.

Parameters

*CSP_USB_REGS *pRegs*

[in] pointer to the registers for the USB controller

Return Value return TRUE if D+ signal was pulled-up

27.4.3 USB Transceiver Driver (ID Pin Detect Driver -- XCVR)

This driver is responsible for detecting the type of USB connector plugged into the Mini USB OTG socket of 3-Stack. Upon detection the driver activates the USB host controller driver or USB function controller driver.

27.4.3.1 User Interface

There is no user interface to the transceiver driver. This driver merely manages the USB host or client drivers, which provide the appropriate programming API.

The driver can be configured through its platform-specific routines to provide different behavior for power management (wake-up, D+ soft connect, and so on).

27.4.3.2 Transceiver Driver Configuration

The transceiver driver is configured into the BSP automatically upon dragging and dropping the USB HS OTG catalog item. If transceiver functionality is not required, it can be disabled as described below.

27.4.3.3 Registry Settings

The USB OTG transceiver settings are values located under the registry key:

```
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\XVC]
```

The values under this registry key are automatically included in the image through `platform.reg`. They do not normally require customization.

Value	Type	Content	Description
Dll	sz	imx_xvc.dll	Driver dynamic link library
OTGSupport	dword	01	This value must be set to 1 to enable the transceiver-driven mode switching on the OTG. If no transceiver support is required (host or client only) then this value can be set to 0, though the XVC key will not normally be configured in the image when OTG Pure Host or OTG Pure Client is configured.
OTGGroup	sz	01	This unique string (example “00” to “99”) is used to combine/correlate instances of the host, function, and transceiver driver within one USB OTG instance. Only one instance of the OTG is actually supported currently on the i.MX31 hardware.

27.4.3.4 Unit Test

There is no CETK test case for USB Transceiver driver. The Transceiver driver is tested using the Mini USB OTG plug A and Mini USB OTG plug B. The test is done by manually plugging in the Mini USB OTG plug into the Mini USB OTG socket of 3-Stack board. The test verifies that the USB host or Function controller driver gets activated on cable plug-in.

27.4.3.4.1 Unit Test Hardware

Table 27-6 lists the required hardware to run the unit tests.

Table 27-6. Hardware Requirements

Requirements	Description
3-Stack board to act as a device.	3-Stack board is configured as USB Mass storage class.
USB LS Mouse	To test if control reaches the Host controller driver.
USB cable having A-type plug at one end and Mini USB OTG plug B on the other side. To plug in USB LS mouse, a USB extension cable having mini-A at one end and USB A-type socket at the other end	For connecting between the host and the device.

27.4.3.4.2 Running the Transceiver Test

The following table lists Transceiver tests:

Table 27-7. Transceiver Tests

Test Cases	Entry Criteria/Procedure/Expected Results
Idle case when no cable plugged in.	**Entry Criteria**: Make sure there is no mini USB OTG plug connected and the board is turned on and wait till the board boots-up completely. **Procedure**: When the board is powered and completely booted-up, the board should be idle (and as mass storage client, not verifiable). **Expected Result**: Device boots up and is stable.
Mass storage client visible from PC	**Entry Criteria**: Make sure there is no mini USB OTG plug connected and the board is turned on and wait till the board boots-up completely. **Procedure**: When the board is powered and completely booted-up, verify that board responds as a mass storage client when plugged into PC. **Expected Result**: New storage must be visible on PC.
Mini USB OTG plug-A connected to the mini USB OTG socket of 3-Stack board and mouse plugged into OTG port through this cable.	**Entry Criteria**: Unplug board from PC (in previous step). **Procedure**: 1. Connect the USB HID device (Mouse) which has a Mini USB OTG plug-A to it. The control goes to the USB Host controller driver. 2. The corresponding device gets enumerated and starts functioning. Ex: If a USB mouse is connected, on movement of the mouse, the pointer in the LCD screen is seen moving. **Expected Result**: Device should start functioning.

27.4.3.5 Platform-Specific API

The transceiver driver library code contains all i.MX31 chip-specific implementation, and is located in:

```
PUBLIC\Common\OAK\CSP\ARM\Freescale\MX31\Drivers\USBXVR
```

The transceiver driver operation can be customized through the platform-specific code located in:

```
PLATFORM\3-Stack\SRC\Drivers\USBXVR
```

The standard implementation located in hwinit.c is provided for the 3-Stack with ISP1504 transceiver attached to the High Speed OTG port.

Customizations would permit different power management and wake-up behavior, including when the device generates soft connect/disconnect (D+ pull-up) or what wake-up conditions are supported when nothing is attached to the OTG port.

The library USB transceiver code communicates with the platform-specific code through callback functions. Only one globally-defined specific routine (RegisterCallback) is required for using this interface.

Standard code is supplied for full transceiver operation using the 3-Stack hardware platform.

27.4.3.5.1 Structure BSP_USB_CALLBACK_FNS

Structure BSP_USB_CALLBACK_FNS is defined in MX31_usb.h. This is a structure containing all the USB callback functions as called by the USB CSP drivers. Currently only the transceiver driver (USBXVR) uses these callback functions. The callback functions are registered using RegisterCallback() (refer to 2 below).

```
typedef struct {
      // pfnUSBPowerDown - function pointer for platform to call during power down.
      // pfnUSBPowerUp - function pointer for platform to call during power up.
      // Parameter: 1) regs - USB registers
      //            2) pUSBCoreClk - pointer to Boolean to indicate the status of USB Core Clk
      //              if it is on or off. Platform is responsible to update this value if they
      change
      //              the status of USBCoreClk. [TRUE - USBCoreClk ON,  FALSE - USBCoreClk OFF]
      //            3) pPanicMode - pointer to Boolean to indicate the status of panic mode
      //              if it is on or off. Platform is responsible to update this value if they
      change
      //            the status of panic mode. [TRUE - PanicMode ON,  FALSE - USBCoreClk OFF]
      void (*pfnUSBPowerDown)(CSP_USB_REGS *regs, BOOL *pUSBCoreClk, BOOL *pPanicMode);
      void (*pfnUSBPowerUp)(CSP_USB_REGS *regs, BOOL *pUSBCoreClk, BOOL *pPanicMode);
      // pfnUSBSetPhyPowerMode - function pointer for platform to call when they want to
      suspend/resume the PHY
      // Parameter: 1) regs - USB registers
      //            2) bResume - TRUE - request to resume, FALSE - request suspend
      void (*pfnUSBSetPhyPowerMode)(CSP_USB_REGS *regs, BOOL bResume);
      } BSP_USB_CALLBACK_FNS;
```

27.4.3.5.2 pfnUSBPowerDown

This callback function is called during the Windows CE power down sequence. The actual platform specific power down routine should be registered as this callback function. Note: this function is only called if the system is in USB transceiver mode only (that is, when nothing is attached to the OTG port.).

There is no standard implementation for this callback, since by default the transceiver driver will automatically suspend the port when nothing is attached. This enables wake-up on device or host attachment, and enables the D+ pull-up during the suspended condition.

Parameters

*CSP_USB_REGS *regs*

[in] Mapped pointer to the USB registers in the i.MX31, from physical address space to a non-paged, process-dependent address space. This is mapped during the transceiver initialization routine (XVC_Init).

*BOOL *pUSBCoreClock*

[in/out] Pointer to a Boolean variable in transceiver to indicate whether the USB Core Clock has been stopped.

The platform-specific callback function must update this flag to reflect the current USB Core Clock status, if the status of the USB Core Clock is changed within the platform code (for example using DDKClockSetGatingMode()). This ensures consistency of the clock status within the CSP transceiver driver.

TRUE – USB Core Clock is running

FALSE – USB Core Clock is stopped

*BOOL *pPanicMode*

[in/out] Pointer to a Boolean variable to indicate whether the USB has requested for system voltage to remain in Panic Mode or not. The callback function must

update this flag to reflect the current Panic Mode status, if this status is changed within the platform code (for example using DDKClockEnablePanicMode(). This ensures consistency of the Panic Mode status within the CSP transceiver driver.

TRUE – Panic mode is currently requested for the USB

FALSE – Panic mode is not currently requested for the USB

27.4.3.6 pfnUSBPowerUp

Similar to pfnUSBPowerDown, this is called during the Windows CE power up sequence. The actual platform specific power up (resume) routine should be registered to this pointer. This is only called when USB is in transceiver mode (that is, when nothing is attached to the OTG port).

There is no standard implementation for this callback, since by default the transceiver driver will automatically suspend the port when nothing is attached and the port need not perform any wake-up activity until a device or host attachment is detected.

Parameters For parameter details see pfnUSBPowerDown, section 22.4.3.5.2.

27.4.3.6.1 pfnUSBSetPhyPowerMode

This function is called when the system is in USB transceiver mode, with no USB activity. With standard implementation on 3-Stack, if the system is in transceiver mode and there is no activity in USB port for 1 second, the transceiver driver will suspend the ULPI PHY (in this case, it is ISP1504, disable the USB Clock gating, and set the system to non-panic mode (allowing core voltage to drop).

When there is USB activity (for example, device attach), the transceiver driver sets the system to panic mode (requiring core voltage to stay high using DDKClockEnablePanicMode(), Supported for the i.MX31), enables USB Clock gating and puts the ULPI PHY transceiver to resume.

This callback function is responsible for handling the suspend and resume of ULPI PHY transceiver. The developer must register this pointer with the actual platform specific function for suspend and resume of ULPI PHY transceiver. Custom wake-up conditions can be enabled here.

Parameters

*CSP_USB_REGS *regs*

[in] Mapped pointer to the USB registers in the i.MX31, from physical address space to a non-paged, process-dependent address space. This is mapped during the transceiver initialization routine (XVC_Init).

BOOL resume

[in] This Boolean variable indicates whether the callback function must resume or suspend the ULPI PHY transceiver.
TRUE – callback function must resume transceiver activity
FALSE – callback function must suspend transceiver activity

27.4.3.6.2 RegisterCallback

This is used to register all the callback functions defined in BSP_USB_CALLBACK_FNS. This function is called by the USB driver during the initialization process of the transceiver driver (XVC_Init). The developer must implement a function by this name in their platform directory,

A standard implementation is provided for the ISP1504 transceiver of the 3-Stack.

When no callback function is required, those elements of the BSP_USB_CALLBACK_FNS structure should be initialized to NULL.

Parameters

*BSP_USB_CALLBACK_FNS *pFn*

[in/out] Pointer to BSP_USB_CALLBACK_FNS structure for the developer to register the callback function inside the BSP_USB_CALLBACK_FNS. The callback functions inside this structure will be used by the CSP transceiver code.

27.4.4 Power Management

There are four aspects of power management for the USB device drivers:

1. Special i.MX31 Vcore requirements
2. Clock gating to the USB peripheral block within the i.MX31
3. Setting the transceiver to a lower power mode or suspend
4. Transceiver auto-power-down on inactivity

The USB device driver(s) support an ON and OFF/standby (low power) state, with wake-up capability. The ON state is entered whenever a host or device is attached to the relevant USB port. The driver enters the standby state is automatically after timeout with no device or host attached to the USB port. As well, the standby state is entered when the system suspends. (In the latter case, system wake-up capability is enabled for the port).

27.4.4.1 Special i.MX31 Vcore Requirements

When ULPI-bus transceivers are used with the USB controller (for example, ISP1504 transceivers for High Speed OTG port and High Speed Host 2 port on the i.MX31/3-Stack), normal DVFS scaling of the i.MX31 Vcore must be suspended whenever there is potential of ULPI bus communication. This is the case whenever a device is connected (in host mode) or the device is connected to a host (in client mode). The USB OTG Transceiver driver, and USB Host and Client drivers constrain the DVFS behavior by calling DDKClockEnablePanicMode() whenever a device or host connection is detected, and calling DDKClockDisablePanicMode() when a timeout period expires with no device or host connected to the port. There is no user configuration required here; only the effect on DVFS (DVFC driver) behavior need be noted.

27.4.4.2 Clock Gating

The USB driver(s) for the various USB ports automatically manage clock gating to the i.MX31 USB controller cores. The drivers for the ports coordinate their use of the USB core clock, and when nothing is connected on any of the ports (all drivers are in their lowest power state) the clock is gated on or off using:

```
DDKClockSetGatingMode(DDK_CLOCK_GATE_INDEX_USBOTG, DDK_CLOCK_GATE_MODE_ENABLED_ALL)
DDKClockSetGatingMode(DDK_CLOCK_GATE_INDEX_USBOTG, DDK_CLOCK_GATE_MODE_DISABLED)
```

27.4.4.3 Transceiver Auto Power Down

The USB transceivers automatically enter a lower-power/suspended mode when no USB traffic is detected for several milliseconds. This internally sets a suspended state for the USB port. Software timeout is used to establish whether the driver power mode can be switched to its lowest power state (see below).

27.4.4.4 Transceiver Power Mode

Software timeout is used to establish whether the transceivers and their related bus (for example, ULPI-bus for ISP1504 connection to the i.MX31) needs to be set to a suspended condition. In the lowest-power state, the transceiver is configured to generate wake-up signalling on attachment of devices or host (to the OTG port). The transceiver driver provides callback routines to manage this transition.

27.4.4.5 PowerUp

Each of the OTG client, host and transceiver drivers have PowerUp routine associated. (For the host driver, this is referenced through the MDD to a function PowerMgmtCallback()).

For the host, the routine does the following:

- verify the wake-up conditions through the BSPUsbCheckWakeUp() platform routine
- stop the host controller
- suspend the relevant port
- set the PHY to low power mode using SetPHYPowerMgmt(TRUE) platform routine
- disable panic mode for the core voltage (DDKClockDisablePanicMode())
- gate the USB peripheral block clock

For the client, the routine does the following:

- ungate the USB peripheral block clock
- enable panic mode for the core voltage (DDKClockEnablePanicMode())
- force the port to resume
- disable the wake-up conditions
- enable the interrupts and start the USB controller

For the transceiver driver, the PowerUp routine calls the relevant platform-specific callback routine, pfnUSBPowerUp().

Under normal circumstances there is nothing to be done in this routine, since the OTG port is normally in a suspended state within the transceiver mode. (It is only in transceiver mode when nothing is connected to the port, and thus has already been automatically suspended).

27.4.4.6 PowerDown

As for the PowerUp routine, OTG client, host and transceiver drivers have PowerUp routine associated. (For the host driver, this is referenced through the MDD to a function PowerMgmtCallback()).

For the host, the routine does the following:

- verify the wake-up conditions through the BSPUsbCheckWakeUp() platform routine
- stop the host controller
- suspend the relevant port
- set the PHY to low power mode using SetPHYPowerMgmt(TRUE) platform routine
- disable panic mode for the core voltage (DDKClockDisablePanicMode())
- gate the USB peripheral block clock

For the client, the routine does the following:

- stop the USB controller
- clear any outstanding interrupts
- enable appropriate wake-up condition
- suspend the relevant port (suspends the PHY)
- disable core voltage panic mode (DDKClockDisablePanicMode())
- gate the USB peripheral block clock

For the transceiver driver, the PowerDown routine calls the relevant platform-specific callback routine, pfnUSBPowerDown().

Under normal circumstances there is nothing to be done in this routine, since the transceiver remains in its suspended state while nothing is connected to the port. Should any attachment have been made, the transceiver would wake through its wake-up mechanism and launch the appropriate (client or host) driver.

27.4.4.7 Suspend / Resume Operations

- Mass Storage Host / Client: Device will be mount automatically, but any unfinished browse / copy will be terminated
- ActiveSync Client : Once browsing into the content of device. A system suspend / resume will cause device will not be mounted until unplug and plug cable again
- HID Host : Client will be recognized again automatically

27.4.5 Function Drivers

The function drivers can be configured into the image through the Windows CE 5.0 Platform Builder catalog, and are located at:

Device Drivers -> USB Function > USB Function Clients

The default function driver is launched when the USB device port is attached to a host. This default function driver is selected by the registry key (the last instance of this value in reginit.ini will apply):

```
[HKEY_LOCAL_MACHINE\Drivers\USB\FunctionDrivers]
   "DefaultClientDriver"=- ; erase previous default
[HKEY_LOCAL_MACHINE\Drivers\USB\FunctionDrivers]
   "DefaultClientDriver"="Mass_Storage_Class"
```

or

```
   "DefaultClientDriver"="RNDIS"
```

or

```
   "DefaultClientDriver"="Serial_Class"
```

Unless the BSP is configured with persistent registry storage, it will only make sense to configure a single function driver into the image, and this one will become default.

NOTE

When no USB client functionality is included in the image (No OTG port, or OTG Pure Host only), then ensure that no function drivers have been configured. If function drivers are configured, then USB client driver libraries will also be included in the image through logic in:

`PUBLIC\CEBASE\OAK\Misc\winceos.bat`

27.4.5.1 Mass Storage Function

Table 27-8. Mass Storage Function

Driver Attribute	Definition
CSP Driver Path	N/A
CSP Static Library	N/A
Platform Driver Path	\PLATFORM\3-Stack\SRC\DRIVERS\USBMSFN
Import Library	USBMSFN_LIB.lib UFNCLIENTLIB.LIB
Driver DLL	usbmsfn.dll
Catalog Item	Device Drivers > USB Function > USB Function Clients > Mass Storage
SYSGEN Dependency	SYSGEN_USBFN_STORAGE

The Mass Storage function exposes a local data store as a USB peripheral storage device. This device used by default for this local data store is “DSK1”, and could be configured as ATA drive or RAMDISK or even a USB drive attached to a different USB host port, depending on the BSP configuration.

The name DSK1 is associated with the Mass Storage function through the value “DeviceName” under the key:

```
[HKEY_LOCAL_MACHINE\Drivers\USB\FunctionDrivers\Mass_Storage_Class]
```

which is imported by default when SYSGEN_USBFN_STORAGE is defined, from:

```
PUBLIC\Common\OAK\Files\common.reg
```

This default registry entry can (should) be copied into platform.reg and modified to over-ride the defaults. This will allow customizing the following values which must be properly configured for a commercial device:

```
[HKEY_LOCAL_MACHINE\Drivers\USB\FunctionDrivers\Mass_Storage_Class]
; idVendor must be changed. 045E belongs to Microsoft and is only to be used for
; prototype devices in your labs. Visit http://www.usb.org to obtain a vendor id.
   "idVendor"=dword:045E
   "Manufacturer"="Generic Manufacturer (PROTOTYPE--Remember to change idVendor)"
   "idProduct"=dword:FFFF
   "Product"="Generic Mass Storage (PROTOTYPE--Remember to change idVendor)"
   "bcdDevice"=dword:0
```

27.4.5.2 Serial Function

The primary use for Serial function is ActiveSync.

Table 27-9. Serial Function

Driver Attribute	Definition
CSP Driver Path	N/A
PUBLIC driver path	PUBLIC\Common\OAK\Drivers\USBFN\CLASS\SERIAL
CSP Static Library	N/A
Platform Driver Path	N/A
Export Library	serialusbfn.lib
Import Library	com_mdd2.lib serpddcm.lib ufnclientlib.lib
Driver DLL	SerialUsbFn.dll
Catalog Item	Device Drivers > USB Function > USB Function Clients > Serial Client
SYSGEN Dependency	SYSGEN_USBFN_SERIAL

NOTE

ActiveSync has been tested using connection to PC with ActiveSync version 4.1 installed. Please see www.microsoft.com to download the latest ActiveSync software for the PC.

In some cases, DEBUGCHK may be triggered during attachment to ActiveSync in DEBUG builds.

When SYSGEN_USBFN_SERIAL is defined, the default registry entry is automatically included from:

```
PUBLIC\Common\OAK\FILES\common.reg
```

This registry entry can (should) be copied into platform.reg and modified to over-ride the defaults. This will allow customizing the following values which must be properly configured for a commercial device:

```
[HKEY_LOCAL_MACHINE\Drivers\USB\FunctionDrivers\Serial_Class]
; idVendor must be changed. 045E belongs to Microsoft and is only to be used for
; prototype devices in your labs. Visit http://www.usb.org to obtain a vendor id.
   "idVendor"=dword:045E
   "Manufacturer"="Generic Manufacturer (PROTOTYPE--Remember to change idVendor)"
   "idProduct"=dword:00ce
   "Product"="Generic Serial (PROTOTYPE--Remember to change idVendor)"
   "bcdDevice"=dword:0
```

27.4.5.3 RNDIS Function

The RNDIS function allows communication over USB to be supplied to the Ethernet NDIS interface of protocol stack.

Table 27-10. RNDIS Function

Driver Attribute	Definition
CSP Driver Path	N/A
CSP Static Library	N/A
Platform Driver Path	N/A
PUBLIC Driver Path	PUBLIC*\OAK\Drivers\USBFN\Class\RNDIS
Import Library	ndis.lib
Driver DLL	RNDISFN.DLL
Catalog Item	Device Drivers > USB Function > USB Function Clients > RNDIS Client
SYSGEN Dependency	SYSGEN_USBFN_ETHERNET

Note: RNDIS function has been tested using PC RNDIS class driver as located at:

```
PUBLIC\Common\OAK\Drivers\ETHDBG\Rndismini\HOST\usb8023.inf
%WINDIR%\System32\drivers\usb8023.sys
```

When SYSGEN_USBFN_ETHERNET is defined, the default registry entry is automatically included from:

```
PUBLIC\Common\OAK\FILES\common.reg
```

This registry entry can (should) be copied into platform.reg and modified to over-ride the defaults. This will allow customizing the following values which must be properly configured for a commercial device:

```
[HKEY_LOCAL_MACHINE\Drivers\USB\FunctionDrivers\RNDIS]
; idVendor must be changed. 045E belongs to Microsoft and is only to be used for
; prototype devices in your labs. Visit http://www.usb.org to obtain a vendor id.
"idVendor"=dword:045E
"Manufacturer"="Generic Manufacturer (PROTOTYPE--Remember to change idVendor)"
"idProduct"=dword:0301
"Product"="Generic RNDIS (PROTOTYPE--Remember to change idVendor)"
"bcdDevice"=dword:0
```

27.4.6 Class Drivers

All host ports (OTG Host, High Speed Host (H2), and Full Speed Host (H1)) support the same class drivers, and this configuration is common to all host ports.

Class drivers must also be configured for the USB host ports. Class driver configuration is common to all host ports; there is no port-specific configuration to be completed on any class driver.

Table 27-11 identifies the standard Microsoft-supplied drivers. To use them, drag them from the catalog.

Table 27-11. Standard Microsoft-Supplied Drivers

Class Driver	Configuration Flag	Catalog Item
HID	SYSGEN_USB_HID	Core OS>Windows CE devices >Core OS Services > USB Host Support > USB Human Input Device (HID) Class Driver
Printer	SYSGEN_USB_PRINTER	.. > USB Printer Class Driver (and see additional configuration in section 33.5.2.1 below)
Keyboard	SYSGEN_USB_HID _KEYBOARD	.. > Keyboard HID Device (and see additional configuration in section)
	SYSGEN_USB_HID _MOUSE	.. > Mouse HID Device (and see additional configuration in section 33.5.2.2 below)
RNDIS	SYSGEN_ETH_USB_HOST	.. > USB Remote NDIS Class Driver
Storage	SYSGEN_USB_STORAGE	.. > USB (mass) Storage Class Driver

Drag all the class drivers required for the USB Host class.

NOTE

When no USB host ports are configured in the image, ensure that no class drivers are selected, otherwise host libraries will be included by default from logic in:

```
PUBLIC\CEBASE\OAK\Misc\winceos.bat
```

27.4.6.1 Printer

For printer support, you will also require printer driver/protocol support. For example, you may need to include PCL.

Catalog Item	Configuration Flag	Catalog Item
PCL	SYSGEN_PCL	Device Drivers > Printer Devices > PCL Printer Driver

For more information, see the Platform Builder Help:

Developing a Device Driver > **Windows CE Drivers** > **USB Host Drivers** > **USB Host Client Drivers** > **USB Host Printer Client Driver**

27.4.6.2 HID Mouse

For mouse support, you will also require the cursor in order to test/use the mouse.

Catalog Item	Configuration Flag	Catalog Item
HID	SYSGEN_CURSOR	Core OS > Windows CE devices > shell and User Interface > User Interface > Mouse

27.4.6.3 HID Keyboard

The 3-Stack Keyboard key mapping conflicts with that used for the HID keyboard. When USB keyboard support is to be included, remove the 3-Stack Keyboard and include the appropriate stub keyboard and `keyboard .dll` as described below.

Remove Item	Remove Catalog Item
Keyboard	Third Party > Freescale 3-Stack: ARMV4I > Device Drivers > Input Devices > Keypad

Include stub keyboard:

Catalog Item	Configuration Flag	Catalog Item
NOP Stub Keyboard	BSP_KEYBD_NOP	Device Drivers > Input Devices > Keyboard/Mouse > NOP (Stub) Keyboard/Mouse English

And include the appropriate `keyboard .dll`. For example, define SYSGEN_KBD_US and add the following lines in your `platform.bib` (immediately before the FILES section):

```
IF BSP_KEYBD_NOP
    kbdmouse.dll    $(_FLATRELEASEDIR)\KbdnopUs.dll                    NK  SH
ENDIF; BSP_KEYBD_NOP
```

27.5 IRAM Patch

The USB link in i.MX31 use very specific data structures, that is, QH, TD; data memory will also be prepared before a transfer is primed. In original design, the memory is allocated in DRAM. HC will lock the EMI AHB of DRAM when it performs the transfer. The lock will prevent other module from accessing the DRAM. This causes problem for IPU module, which has strict real-time requirements. If HC occupy the DRAM AHB for too long a time, underflow happens for IPU module, which results in LCD display flicker.

The problem is serious when complex codec is used, which gives IPU very high loadings. The IC should have wider bandwidth or a smarter arbiter to avoid such problem in the root.

A software walkaround is to move all USB related data structures and data memory to IRAM area. This way, USB and IPU will not get such conflicts. The trade-off is (as IRAM area total size is 16K) that all data needed cannot be pulled into it simultaneously. Large transfer needs to be split into smaller ones and prime them one by one. Enough area cannot be reserved for too many devices as well.

The detailed implementation of IRAM patch is out of the scope of this document. To enable the patch, simply add an environment variable “BSP_USB_IRAM_PATCH” = “1” in project settings. As described, attach two more devices in one USB port to weaken the effect of this patch.

27.6 Basic Elements for Driver Development

This section provides details of the basic elements for driver development in the 3-Stack BSP.

27.6.1 BSP Environment Variables

Table 27-12. BSP Environment Variables

Names	Definition
BSP_USB	Set to configure USB in BSP
BSP_USB_HSOTG_XVC	Set to enable Full OTG functionality (transceiver host-client switching) on the High Speed OTG port
BSP_USB_HSOTG_CLIENT	Set to include USB client functionality on High Speed OTG port
BSP_USB_HSOTG_HOST	Set to include USB host functionality on High Speed OTG port.

Pin conflicts between default driver implementations for the i.MX31 pin muxing (platform-specific implementation) mean certain configurations are mutually exclusive, as listed in the following table.

Functionality yes = Required no = Not permitted — = Don't care	BSP_ATA	BSP_CSPIBUS	BSP_USB	BSP_USB_HSOTG_XVC	BSP_USB_HSOTG_CLIENT	BSP_USB_HSOTG_HOST
ATA disk drive	yes	no				
High Speed OTG Port full function (Host + Client)			yes	yes	yes	yes
High Speed OTG Port Pure Host only			yes			yes
High Speed OTG Port Pure Client only			yes		yes	
Full Speed Host (H1)	no	no				
High Speed Host (H2)	no	no				

27.6.2 Dependencies of Drivers

Table 27-13 summarizes the Microsoft defined environment variables used in the BSP.

Table 27-13. Microsoft defined Environment Variables

Names	Definition
SYSGEN_USBFN_SERIAL	Set to support serial class for USB Function controller
SYSGEN_USBFN_STORAGE	Set to support mass storage class for USB Function controller
SYSGEN_USBFN_ETHERNET	Set to support RNDIS class for USB Function controller
SYSGEN_CURSOR	Set to support mouse cursor
SYSGEN_FATFS	Set to support FAT16 file system
SYSGEN_PCL	Set to support PCL printing
SYSGEN_PRINTING	Set to support printer

Table 27-13. Microsoft defined Environment Variables(Continued)

Names	Definition
SYSGEN_STOREMGR	Set to support storage manager
SYSGEN_UDFS	Set to support Universal Disc File System
SYSGEN_USB	Set to support USB driver
SYSGEN_USB_HID	Set to support Human Interface driver (HID) class
SYSGEN_USB_HID_CLIENTS	Set to support HID clients
SYSGEN_USB_HID_KEYBOARD	Set to support HID keyboards (keyboard stub and associated .dll are required)
SYSGEN_USB_HID_MOUSE	Set to support HID mouse
SYSGEN_USB_PRINTER	Set to support Printer (printer driver support, such as PCL (SYSGEN_PCL), may be required)
SYSGEN_USB_STORAGE	Set to support storage medium

Chapter 28
WLAN Driver

The WLAN driver is used to drive APM6628 module to implement Wi-FI functionality. WLAN module exchanges data with i.MX31 through SDHC 2 port. The APM6628 module adopts Unifi V3 solution of Cambrige Silicon Radio company

28.1 WLAN Driver Summary

28.1.1 WLAN Client Driver Summary

WLAN driver is provided in binary form instead of source codes.

Table 28-1 provides a summary of the source code location, library dependencies, and other BSP information.

Table 28-1. WLAN Client Driver Summary

Driver Attribute	Definition
Target Platform (TGTPLAT)	3DS
Target SOC (TGTSOC)	MX31
CSP Driver Path	N/A
CSP Static Library	N/A
Platform Driver Path	..\PLATFORM\<TGTPLAT > \SRC\DRIVERS\wifi\csr
Import Library	N.A
Driver DLL	ufmp.dll, sta.xbv, loader.xbv, ufmib.dat, u

Table 28-1. WLAN Client Driver Summary(Continued)

Driver Attribute	Definition
Catalog Item	Third Party > BSPs > Freescale <TGTPLAT> > Device Drivers > WiFi. Core OS > Windows CE devices > Communication Services and Networking > Networking > Local Area Network [LAN] > Wireless LAN (802.11) STA - Automatic Configuration and 802.1x Core OS > Windows CE devices > Communication Services and Networking > Communication Services and Networking > Networking Features> Extensible Authentication Protocol Core OS > Windows CE devices > Security > Authentication Services (SSPI)> NTLM Core OS > Windows CE devices > Security > Authentication Services (SSPI)> Schannel (SSL/TLS) Core OS > Windows CE devices > Security > Microsoft Certificate Enrollment Tool Sample Core OS > Windows CE devices >I nternet Client Services > Internet Explorer 6 for Windows CE Components > Windows Internet Services Core OS > Windows CE device > Communication Services and Networking > Networking Features > Network Utilities (IpConfig, Ping, Route)
SYSGEN Dependency	SYSGEN_ETH_80211 SYSGEN_EAP SYSGEN_AUTH_NTLM SYSGEN_AUTH_SCHANNEL SYSGEN_ENROLL SYSGEN_WININET
BSP Environment Variable	BSP_WIFI_CSR=1

The Recommended Catalog Items listed in Table should be included in the OS design in order to provide Wi-FI functionality.

28.2 Requirements

The Wi-FI driver must satisfy all of the following requirements:

1. Correctly drive Bluetooth module in APM6628 chip
2. Can support scanning and connect to 802.11b AP with open security
3. Can support scanning and connect to 802.11b/g AP with open security
4. Can support scanning and connect to 802.11b/g AP with WEP(64/128/256) security
5. Can support scanning and connect to 802.11b/g AP with WPA-PSK security
6. Can support scanning and connect to adhoc laptop with open security
7. Can support scanning and connect to adhoc laptop with WEP security
8. Can support scanning and connect to 802.11g-only AP with open security

28.3 Hardware Operation

The Wi-FI client driver exchanges data and commands between SD stack and Wi-FI hardware through SDIO port.

The ID Pin Detect is supported through the Transceiver Driver (for High Speed OTG), and is constructed as a stream interface driver.

28.3.1 Conflicts with Other Peripherals

Wi-FI shares one reset pin with the Bluetooth module in the 3-Stack board. Normally, the Wi-FI module will start early than the Bluetooth. It will reset the whole APM6628 chip upon startup.

28.4 Software Operation

The overall software architecture with WLAN Unifi driver for APM 6628 is depicted in Figure 28-1.

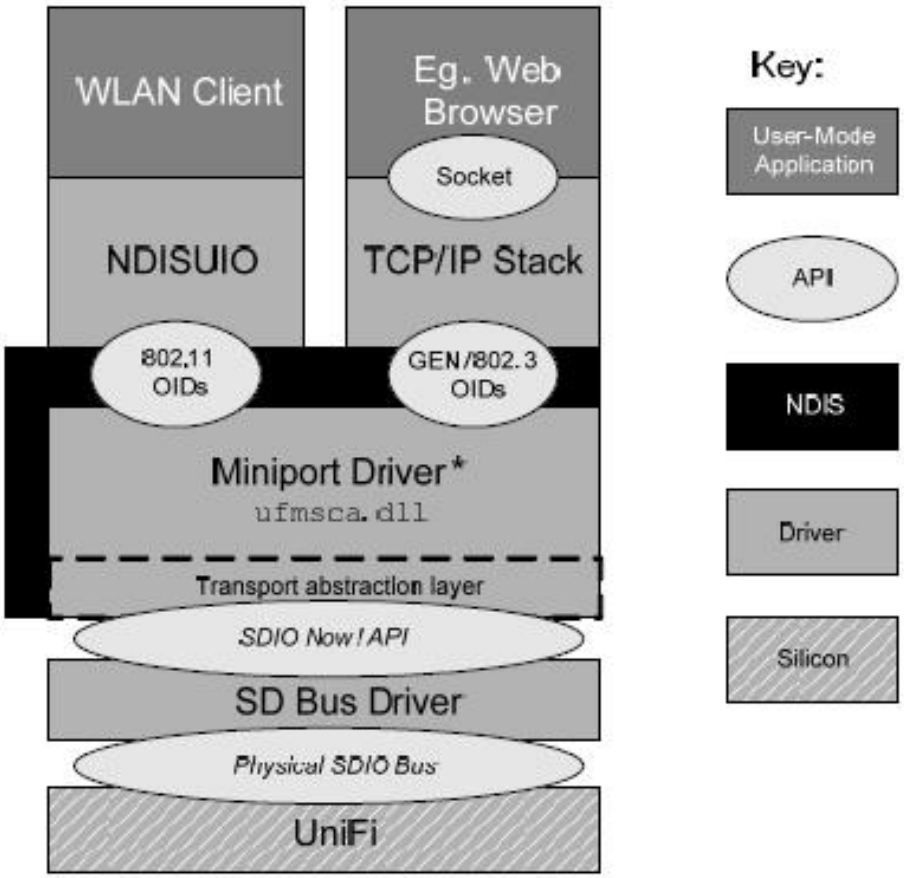

Figure 28-1. Software Architecture with WLAN Unifi Driver

The main component is the miniport driver. The driver provides the means to configure the UniFi device for connecting to a wireless network to send and receive data. A suitable wireless client, such as Microsoft Wireless Zero Configuration can:

- Actively scan for wireless networks in the local area.
- Connect to an unsecured or WEP-enabled infrastructure or ad-hoc network.
- Connect to WPA-enabled networks using pre-shared key (PSK).

- Start an unsecured or WEP-enabled ad-hoc network (IBSS).

The device driver conforms to the following:

- The NDIS 5.1 specification (defined by the IEEE 802.11 Network Adapter Design Guidelines for Windows XP) for integration into the Windows operating system
- The UniFi Host Interface Protocol Specification for the exchange of signal primitives with the UniFi WLAN card.
- Network connections are set up using a wireless LAN client. The client issues a set of NDIS defined 802.11 OIDs to the miniport driver so that it can configure the UniFi device appropriately

28.4.1 Wi-FI Registry setting

The following registry keys are required to properly load and configure WLAN driver

[HKEY_LOCAL_MACHINE\Comm\ufmp]

"IstPriority"=dword:288

"Prefix"="NDL"

"ImagePath"="ufmp.dll"

"Group"="NDIS"

"Dll"="ufmp.dll"

"DisplayName"="CSR UniFi Wireless LAN"

[HKEY_LOCAL_MACHINE\Comm\ufmp\Linkage]

"Route"=multi_sz:""

[HKEY_LOCAL_MACHINE\Comm\ufmp1]

"ImagePath"="ufmp.dll"

"Group"="NDIS"

"Display Name"="CSR UniFi Wireless LAN"

[HKEY_LOCAL_MACHINE\Comm\ufmp1\Parms]

"PowerMobileMode"=dword:0

"BusType"=dword:0

"BusNumber"=dword:0

[HKEY_LOCAL_MACHINE\Drivers\SDCARD\ClientDrivers\Custom\MANF-032A-CARDID-0001-FUNC-1]

"Instance0"="ufmp:ufmp1"

"Prefix"="NDL"

"Dll"="ufmp.dll"

"DbgLevel"=dword:1.

28.5 Unit Test

WLAN test includes CETK test and manual WLAN connection without protection.

28.5.1 Unit Test Hardware

Table 28-2 lists the required hardware on the 3-Stack board to run the unit tests.

Table 28-2. Hardware Requirements

Requirement	Description
3 access points	The Access point supports open/wep/wpa-psk.
laptop	Used to setup adhoc network.
3-Stack board	Test board.

28.5.2 Unit Test Software

Table 28-3 lists the required software on the 3-Stack board to run the unit tests.

Table 28-3. Software Requirements

Requirement	Description
Tux.exe	Tux text harness, which is required for executing the test.
Kato.dll	Kato logging engine, which is required for logging test data.
Tooltalk.dll	Application required by Tux.exe and Kato.dll. Handles the transport between the target device and the development workstation.
ufmp.dll	Test ufmp.dll file for the test client

28.5.3 Running the WLAN Driver CETK Tests

The Wi-Fi test suite requires 3 Wi-Fi access points to be present simultaneously, each configured for different encryption schemes. In case there is only 1 access point available, the tests can be split into 3 parts, depending upon encryption disabled, WEP 40-bit or WEP 104-bit respectively. Follow the steps below.

1. Create an ad-hoc Wi-Fi link (SSID: CE-ADHOC1) on a WLAN supported laptop or other WLAN-supporting device.

2. Copy tux.exe, kato.dll, tooltalk.dll and wzctooltest.dll to the Windows folder of the 3-Stack board.
3. Part I:
   — Set up the WLAN access point - SSID: CEOPEN. WEP: disabled
   — On the 3-Stack board, run the command line tux -o -d wzctooltest -x1-400,500,501
4. Part II:
   — Setup the WLAN access point - SSID: CE-WEP40, WEP: 1/0x1234567890
   — On the 3-Stack board, run the command line tux -o -d wzctooltest -x411-413,502,503,410,451-452,450
5. Part III:
   — Setup the WLAN access point - **SSID: CEWEP104, WEP: 1/qwertyuiopasd**
   — On the 3-Stack board, run the command line tux -o -d wzctooltest -x420
   — Note that the access point should support complex encryption to run Part III. Also the Wi-Fi tests require wzctool.exe and ipconfig.exe, and hence the following catalog item needs to be included: **Catalog** > **Core OS** > **Windows CE Devices** > **Networking Features** > **Network Utilities**
   — Note that before running each Part, system should be reset.

28.5.4 Test the WLAN Communication without Protection

This test just covers the practical functionality of the Wireless LAN driver to connect to any public wireless network for internet access. For this test it is required to have a 3-Stack board and any wireless access point (maybe a wireless router) without any protection to the internet access. The test is considered passed if the user can access http://www.google.com and view its contents.

To run the test:

1. Turn on the board.
2. If Windows CE 5.0 was correctly configured and if the Wireless driver could be successfully loaded, Windows CE 5.0 displays a Window listing the available wireless networks.
3. Select the wireless network without protection that you can use to navigate through the Internet.
4. Open Internet Explorer from the desktop icon.
5. Navigate through the Internet to http://www.google.com.

Appendix A 3-Stack Frequently Asked Questions

A.1 How to Deal with Different Resolutions of the Display Panel?

The DirectDraw display driver does not support dynamic resolution change. You can specify the screen resolution in either of two locations during boot up:

- The `sdc.c` file specifies the hardware-level register settings. Set the width and height using the variables PANEL_INFO.width and PANEL_INFO.height; these parameters identify to the SDC the size of the LCD panel. For more information about the register settings, see the chapter on the Image Processing Unit (IPU) in the *MCIMX31 and MCIMX31L Applications Processors Reference Manual*.
- The `sdc.h` file specifies the actual panel size in the software settings. Set the size values using `SCREEN_PIX_WIDTH` and `SCREEN_PIX_HEIGHT`.

NOTE

If the user selects a large resolution, such as VGA (640x480), remember to adjust the video memory size, which is set in the registry. For details, see Section 4.3.3.2 Display Registry Setting.

A.2 How to Deal with Different Display Interface Formats?

The current driver setting uses the RGB565 interface format. It is recommended that you use the RGB565 interface, as it uses much less memory than the RGB888 interface, and it also provides a high quality color representation.

Currently, the display driver supports only RGB565 and YUV422 surface. Better performance cannot be obtained using the RGB666 and RGB888 display interfaces.

If the display panel does not support the RGB565 format, use other RGB format. Map RGB565 to RGBXXX by changing three registers: DI_DISP3_B0_MAP, DI_DISP3_B1_MAP, DI_DISP3_B2_MAP. These control the DI bus mapping unit. The setting of these registers is in the function InitializeSDC in the `sdc.c` file. For further information about settings, see section 44.4.6.4 Bus Mapping Unit in the *MCIMX31 and MCIMX31L Applications Processors Reference Manual*.