# i.MX53 System Development User's Guide

**Supports**
i.MX53

MX53UG
Rev. 1
3/2011

freescale™
semiconductor

***How to Reach Us:***

**Home Page:**
www.freescale.com

**Web Support:**
http://www.freescale.com/support

**USA/Europe or Locations Not Listed:**
Freescale Semiconductor, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or
+1-480-768-2130
www.freescale.com/support

**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)
www.freescale.com/support

**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
+1-800 441-2447 or
+1-303-675-2140
Fax: +1-303-675-2150
LDCForFreescaleSemiconductor
@hibbertgroup.com



Document Number: MX53UG
Rev. 1, 3/2011

# Contents

| Paragraph Number | Title | Page Number |
|---|---|---|

## About This Guide



## Chapter 1 Design Checklist



## Chapter 2 i.MX53 Layout Recommendations

# Contents

| Paragraph Number | Title | Page Number |
|---|---|---|

## Chapter 3
## Understanding the IBIS Model



## Chapter 4
## Using the IOMUX Design Aid

# Contents

| Paragraph Number | Title | Page Number |
|---|---|---|

## Chapter 5
## Setting up Power Management



## Chapter 6
## Interfacing DDR2 and DDR3 Memories with the i.MX53 Processor



## Chapter 7
## Avoiding Board Bring-Up Problems

# Contents

| Paragraph Number | Title | Page Number |
|---|---|---|

## Chapter 8
## Using the Clock Connectivity Table

## Chapter 9
## Configuring JTAG Tools for Debugging



## Chapter 10
## Porting the On-Board-Diagnostic-Suite (OBDS) to a Custom Board



## Chapter 11
## Porting U-Boot from an i.MX53 Reference Board to an i.MX53 Custom Board



## Chapter 12
## Porting the Android Kernel

# Contents

| Paragraph Number | Title | Page Number |
|---|---|---|

# Contents

| Paragraph Number | Title | Page Number |
|---|---|---|

## Chapter 16
## Configuring the SPI NOR Flash Memory Technology Device (MTD) Driver



## Chapter 17
## Setting Up the Keypad Port (KPP)



## Chapter 18
## Supporting the i.MX53 Reference Board DISP0 LCD

# Contents

# Contents

| Paragraph Number | Title | Page Number |
|---|---|---|

# Figures

# Figures

| Figure Number | Title | Page Number |
|---|---|---|

# Figures

# Figures

| Figure Number | Title | Page Number |
|---|---|---|

# Tables

# Tables

# About This Guide

The i.MX53 multimedia applications processor (i.MX53) is Freescale Semiconductor, Inc.'s latest addition to a growing family of multimedia-focused products that offer high performance processing optimized for the lowest power consumption. The i.MX53 processors feature Freescale's advanced implementation of the ARM Cortex-A8™ core which operates at speeds as high as 1.2 GHz. The integrated memory controller is compatible with DDR2-800, LVDDR2-800, and DDR3-800 DRAM, as well as LPDDR2-800 in the PoP package.

This product is suitable for applications such as:

- Automotive navigation and entertainment
- High-end mobile Internet devices and high-end PDAs
- Tablets
- Smart mobile devices
- High-end portable media players with HD video capability
- Portable navigation devices
- Gaming consoles
- Industrial HMI

Freescale provides a number of tools that facilitate the rapid design-in of the i.MX53 Applications Processor for consumer, automotive, or industrial products. These tools include the i.MX53 software developer's kit (SDK), the i.MX53 Quick Start Board, the SABRE platform for tablets based on the i.MX53, and the SABRE platform for automotive infotainment. These tools allow the rapid prototyping of new products prior to commitment to production-level designs. Once you have determined the precise features, function and physical parameters of your product, these prototyping tools along with this document, the *i.MX53 System Development User's Guide*, aid you in the design, layout, and bring-up of your design.

Along with tips on designing your custom circuit board, this guide helps you customize Freescale provided software utilizing the development tools provided in the SDK. This guide assumes that you have access to generally available software tools (such as compilers, linkers and Make builders) as well as Freescale's Linux Target Image Builder (LTIB).

## Audience

This document is targeted to software and hardware engineers who desire to port the i.MX53 board support package (BSP) to customer-specific products. The audience is expected to have a working understanding of the ARM processor programming model, the C programming language, tools such as compilers and assemblers, and program build tools such as MAKE. Familiarity with the use of commonly available hardware test and debug tools such as oscilloscopes and logic analyzers is assumed. An understanding of the architecture of the i.MX53 application processor is also assumed.

## Organization

This guide is a compendium of application notes organized in two parts. The first part covers aspects of hardware design and bring-up, and the second focuses on software development.

Part I, "Hardware Design and Bring-up" covers topics that aid you in the design of a custom printed circuit board design utilizing the i.MX53. The following lists the chapters of Part I and provides a quick link to each:

- Chapter 1, "Design Checklist"—provides a design checklist that contains recommendations for optimal design for i.MX53-based systems.
- Chapter 2, "i.MX53 Layout Recommendations"—provides recommendations to assist design engineers with the correct layout of their i.MX53x-based system.
- Chapter 3, "Understanding the IBIS Model"—explains how to use the IBIS (input output buffer information specification) model.
- Chapter 4, "Using the IOMUX Design Aid"—explains how to use of the IOMUX system design aid (IOMux.exe), which facilitates the assignment of internal signals to external device balls/pins.
- Chapter 5, "Setting up Power Management"—discusses how to supply and interface the i.MX53 multimedia applications processor with two different power management integrated circuits (PMICs): DA9053 and LTC3589.
- Chapter 7, "Avoiding Board Bring-Up Problems"—provides recommendations for avoiding typical mistakes when bringing up a board for the first time.
- Chapter 8, "Using the Clock Connectivity Table"—explains how to use the i.MX53 clocking connectivity.
- Chapter 9, "Configuring JTAG Tools for Debugging"—explains how to configure JTAG tools for debugging."

Part II, "Software Development" aids you in software development for your product. The first four chapters are organized in the way a developer might approach the task of porting Freescale's SDK BSP to support their target product board. The remaining chapters deal with porting selected integrated I/O devices. The following lists the chapters of Part II and provides a quick link to each:

- Chapter 10, "Porting the On-Board-Diagnostic-Suite (OBDS) to a Custom Board
- Chapter 11, "Porting U-Boot from an i.MX53 Reference Board to an i.MX53 Custom Board"
- Chapter 12, "Porting the Android Kernel"
- Chapter 13, "Configuring the IOMUX Controller (IOMUXC)"
- Chapter 14, "Registering a New UART Driver"
- Chapter 6, "Interfacing DDR2 and DDR3 Memories with the i.MX53 Processor"
- Chapter 15, "Adding Support for the i.MX53 ESDHC"
- Chapter 16, "Configuring the SPI NOR Flash Memory Technology Device (MTD) Driver"
- Chapter 17, "Setting Up the Keypad Port (KPP)"
- Chapter 18, "Supporting the i.MX53 Reference Board DISP0 LCD"
- Chapter 19, " Connecting an LVDS Panel to an i.MX53 Reference Board"
- Chapter 20, " Supporting the i.MX53 Camera Sensor Interface CSI0"

- Chapter 21, "Porting Audio Codecs to a Custom Board"
- Chapter 22, "Porting the Fast Ethernet Controller Driver"
- Chapter 23, "Porting USB Host1 and USB OTG"

# Essential Reference

You should have access to an electronic copy of the latest version of the *i.MX53 Multimedia Applications Processor Reference Manual* (MCIMX53RM) and *i.MX53xD Applications Processors for Consumer Products* (IMX53CEC).

# Suggested Reading

This section lists additional reading that provides background for the information in this manual as well as general information about the architecture.

## General Information

The following documentation provides useful information about the ARM processor architecture and computer architecture in general:

- For information about the ARM Cortex-A8 processor see http://www.arm.com/products/processors/cortex-a/cortex-a8.php
- *Computer Architecture: A Quantitative Approach*, Fourth Edition, by John L. Hennessy and David A. Patterson
- *Computer Organization and Design: The Hardware/Software Interface*, Second Edition, by David A. Patterson and John L. Hennessy

## Related Documentation

Freescale documentation is available from the sources listed on the back cover of this manual; the document order numbers are included in parentheses for ease in ordering:

Additional literature is published as new Freescale products become available. For a current list of documentation, refer to www.freescale.com.

# Conventions

This document uses the following notational conventions:

| | |
|---|---|
| `Courier` | Used to indicate commands, command parameters, code examples, and file and directory names. |
| *Italics* | *Italics indicates* command or function parameters |
| **Bold** | Function names are written in bold. |
| cleared/set | When a bit takes the value zero, it is said to be cleared; when it takes a value of one, it is said to be set. |
| **mnemonics** | Instruction mnemonics are shown in lowercase bold |

| | |
|---|---|
| | Book titles in text are set in italics |
| sig_name | Internal signals are written in all lowercase |
| 0x0 | Prefix to denote hexadecimal number |
| 0b0 | Prefix to denote binary number |
| **r**A, **r**B | Instruction syntax used to identify a source GPR |
| **r**D | Instruction syntax used to identify a destination GPR |
| REG[FIELD] | Abbreviations for registers are shown in uppercase text. Specific bits, fields, or ranges appear in brackets. For example, MSR[LE] refers to the little-endian mode enable bit in the machine state register. |
| x | In some contexts, such as signal encodings, an unitalicized x indicates a don't care. |
| *x* | An italicized *x* indicates an alphanumeric variable |
| *n*, *m* | An italicized *n* indicates a numeric variable |

**NOTE**

In this guide, notation for all logical, bit-wise, arithmetic, comparison, and assignment operations follow C Language conventions.

## Signal Conventions

| | |
|---|---|
| $\overline{\text{PWR\_ON\_RESET}}$ | An overbar indicates that a signal is active when low |
| _b, _B | Alternate notation indicating an active-low signal |
| signal_name | Lowercase italics is used to indicate internal signals |

## Acronyms and Abbreviations

The following table defines the acronyms and abbreviations used in this document.

**Definitions and Acronyms**

| Term | Definition |
|---|---|
| Address Translation | Address conversion from virtual domain to physical domain |
| API | Application Programming Interface |
| ARM® | Advanced RISC Machines processor architecture |
| AUDMUX | Digital audio multiplexer—provides a programmable interconnection for voice, audio, and synchronous data routing between host serial interfaces and peripheral serial interfaces. |
| BCD | Binary Coded Decimal |
| Bus | A path between several devices through data lines. |
| Bus load | The percentage of time a bus is busy. |

**Definitions and Acronyms (continued)**

| Term | Definition |
| --- | --- |
| CODEC | Coder/decoder or compression/decompression algorithm—Used to encode and decode (or compress and decompress) various types of data. |
| CPU | Central Processing Unit—generic term used to describe a processing core. |
| CRC | Cyclic Redundancy Check—Bit error protection method for data communication. |
| CSI | Camera Sensor Interface |
| DMA | Direct Memory Access—an independent block that can initiate memory-to-memory data transfers. |
| DRAM | Dynamic Random Access Memory |
| EMI | External Memory Interface—controls all IC external memory accesses (read/write/erase/program) from all the masters in the system. |
| Endian | Refers to byte ordering of data in memory. Little Endian means that the least significant byte of the data is stored in a lower address than the most significant byte. In Big Endian, the order of the bytes is reversed. |
| EPIT | Enhanced Periodic Interrupt Timer—a 32-bit set and forget timer capable of providing precise interrupts at regular intervals with minimal processor intervention. |
| FCS | Frame Checker Sequence |
| FIFO | First In First Out |
| FIPS | Federal Information Processing Standards—United States Government technical standards published by the National Institute of Standards and Technology (NIST). NIST develops FIPS when there are compelling Federal government requirements such as for security and interoperability but no acceptable industry standards or solutions. |
| FIPS-140 | Security requirements for cryptographic modules—Federal Information Processing Standard 140-2(FIPS 140-2) is a standard that describes US Federal government requirements that IT products should meet for Sensitive, But Unclassified (SBU) use. |
| Flash | A non-volatile storage device similar to EEPROM, but where erasing can only be done in blocks of the entire chip. |
| Flash path | Path within ROM bootstrap pointing to an executable Flash application. |
| Flush | A procedure to reach cache coherency. Refers to removing a data line from cache. This process includes cleaning the line, invalidating its VBR and resetting the tag valid indicator. The flush is triggered by a software command. |
| GPIO | General Purpose Input/Output |
| Hash | Hash values are produced to access secure data. A hash value (or simply hash), also called a message digest, is a number generated from a string of text. The hash is substantially smaller than the text itself, and is generated by a formula in such a way that it is extremely unlikely that some other text will produce the same hash value. |
| I/O | Input/Output |
| ICE | In-Circuit Emulation |
| IP | Intellectual Property. |
| IPU | Image Processing Unit —supports video and graphics processing functions and provides an interface to video/still image sensors and displays. |

**Definitions and Acronyms (continued)**

| Term | Definition |
|---|---|
| IrDA | Infrared Data Association—a nonprofit organization whose goal is to develop globally adopted specifications for infrared wireless communication. |
| ISR | Interrupt Service Routine. |
| JTAG | JTAG (IEEE Standard 1149.1) A standard specifying how to control and monitor the pins of compliant devices on a printed circuit board. |
| Kill | Abort a memory access. |
| KPP | KeyPad Port—a 16-bit peripheral that can be used as a keypad matrix interface or as general purpose input/output (I/O). |
| line | Refers to a unit of information in the cache that is associated with a tag. |
| LRU | Least Recently Used—a policy for line replacement in the cache. |
| MMU | Memory Management Unit—a component responsible for memory protection and address translation. |
| MPEG | Moving Picture Experts Group—an ISO committee that generates standards for digital video compression and audio. It is also the name of the algorithms used to compress moving pictures and video. |
| MPEG standards | There are several standards of compression for moving pictures and video.<br>MPEG-1 is optimized for CD-ROM and is the basis for MP3.<br>MPEG-2 is defined for broadcast quality video in applications such as digital television set-top boxes and DVD.<br>MPEG-3 was merged into MPEG-2.<br>MPEG-4 is a standard for low-bandwidth video telephony and multimedia on the World-Wide Web. |
| MQSPI | Multiple Queue Serial Peripheral Interface—used to perform serial programming operations necessary to configure radio subsystems and selected peripherals. |
| MSHC | Memory Stick Host Controller |
| NAND Flash | Flash ROM technology—NAND Flash architecture is one of two flash technologies (the other being NOR) used in memory cards such as the Compact Flash cards. NAND is best suited to flash devices requiring high capacity data storage. NAND flash devices offer storage space up to 512-Mbyte and offer faster erase, write, and read capabilities over NOR architecture. |
| NOR Flash | See NAND Flash. |
| PCMCIA | Personal Computer Memory Card International Association—a multi-company organization that has developed a standard for small, credit card-sized devices, called PC Cards. There are three types of PCMCIA cards that have the same rectangular size (85.6 by 54 millimeters), but different widths. |
| Physical address | The address by which the memory in the system is physically accessed. |
| PLL | Phase Locked Loop—an electronic circuit controlling an oscillator so that it maintains a constant phase angle (a lock) on the frequency of an input, or reference, signal. |
| RAM | Random Access Memory |
| RAM path | Path within ROM bootstrap leading to the downloading and the execution of a RAM application |
| RGB | The RGB color model is based on the additive model in which Red, Green, and Blue light are combined in various ways to create other colors. The abbreviation RGB come from the three primary colors in additive light models. |

**Definitions and Acronyms (continued)**

| Term | Definition |
|---|---|
| RGBA | RGBA color space stands for Red Green Blue Alpha. The alpha channel is the transparency channel, and is unique to this color space. RGBA, like RGB, is an additive color space, so the more of a color you place, the lighter the picture gets. PNG is the best known image format that uses the RGBA color space. |
| RNGA | Random Number Generator Accelerator—a security hardware module that produces 32-bit pseudo random numbers as part of the security module. |
| ROM | Read Only Memory |
| ROM bootstrap | Internal boot code encompassing the main boot flow as well as exception vectors. |
| RTIC | Real-time integrity checker—a security hardware module |
| SCC | SeCurity Controller—a security hardware module |
| SDMA | Smart Direct Memory Access |
| SDRAM | Synchronous Dynamic Random Access Memory |
| SoC | System on a Chip |
| SPBA | Shared Peripheral Bus Arbiter—a three-to-one IP-Bus arbiter, with a resource-locking mechanism. |
| SPI | Serial Peripheral Interface—a full-duplex synchronous serial interface for connecting low-/medium-bandwidth external devices using four wires. SPI devices communicate using a master/slave relationship over two data lines and two control lines: *Also see SS, SCLK, MISO, and MOSI.* |
| SRAM | Static Random Access Memory |
| SSI | Synchronous-Serial Interface—standardized interface for serial data transfer |
| TBD | To Be Determined |
| UART | Universal Asynchronous Receiver/Transmitter—this module provides asynchronous serial communication to external devices. |
| UID | Unique ID–a field in the processor and CSF identifying a device or group of devices |
| USB | Universal Serial Bus—an external bus standard that supports high speed data transfers. The USB 1.1 specification supports data transfer rates of up to 12Mb/s and USB 2.0 has a maximum transfer rate of 480 Mbps. A single USB port can be used to connect up to 127 peripheral devices, such as mice, modems, and keyboards. USB also supports Plug-and-Play installation and hot plugging. |
| USBOTG | USB On The Go—an extension of the USB 2.0 specification for connecting peripheral devices to each other. USBOTG devices, also known as dual-role peripherals, can act as limited hosts or peripherals themselves depending on how the cables are connected to the devices, and they also can connect to a host PC. |
| Word | A group of bits comprising 32 bits |

# Part I
# Hardware Design and Bring-up

The chapters that follow cover topics that aid you in the hardware design, bring-up, and debug of your custom printed circuit board utilizing the i.MX53.

# Chapter 1
# Design Checklist

This chapter provides a design checklist for i.MX53-based systems. The design checklist contains recommendations for optimal design. Where appropriate, the checklist also provides an explanation so that users have a greater understanding of why certain techniques are recommended. All supplemental tables referenced by the checklist appear following the design checklist table.

**Table 1-1. Design Checklist**

| Check Box | Recommendation | Explanation/Supplemental Recommendations |
|---|---|---|
| **DDR Recommendations** | | |
| | **1.** Tie DDR_VREF to a precision external resistor divider with a resistor to GND and a resistor to NVCC_EMI_DRAM. | When using DDR, the nominal reference voltage must be half of the NVCC_EMI_DRAM supply. The resistors must be sized to account for the i.MX53 DDR_VREF input current plus memory input current. This current drawn from the divider affects the reference voltage. See Table 1-2.<br>Also consider:<br>• Shunting each resistor with a closely-mounted 0.1 μF capacitor. The decouple cap connected in parallel with the resistor connected to NVCC_EMI_DRAM may not be required. This depends on the layout and the additional supply.<br>• Bypassing Vref at source and destinations. |
| | **2.** Use the following values for the DRAM calibration input:<br>• For DDR2, connect 300 Ω 1% to GND.<br>• For DDR3, connect 240 Ω 1% to GND.<br>• For LPDDR2, connect 240 Ω 1% to GND.<br>• For LVDDR2, connect 300 Ω 1% to GND. | The DRAM_CALIBRATION input requires an external resistor used as reference during DRAM output buffer driver calibration. This resistor must be mounted close to the associated BGA ball. |

**Table 1-1. Design Checklist (continued)**

| Check Box | Recommendation | Explanation/Supplemental Recommendations |
|---|---|---|
| **EIM Recommendations** | | |
| | **3.** When EIM boot signals are used as the system's EIM signals or GPIO outputs after boot, use a passive resistor network to select the desired boot mode for development boards. | Because only resistors are used, EIM bus loads can cause current drain, leading to higher (false) supply current measurements. Each EIM boot signal should connect to a series resistor to isolate the bus from the resistors and/or switchers. See Figure 1-1 for the implementation. Each configured EIM boot signal sees either a 14.7 kΩ pull-down or a 4.7 kΩ pull-up. For each switch-enabled pulled-up signal, the supply is presented with a 10 kΩ current load.<br>The i.MX53 does not have on-chip keeper circuits on the external boot inputs, allowing freedom to size boot resistors larger than some previous i.MX devices. Production product is booted from the on-chip fuses and does not employ these external boot mode resistors. |
| | **4.** To reduce incorrect boot-up mode selections, do one of the following:<br>• Use EIM boot interface lines as processor outputs.<br>• If an EIM boot signal must be configured as an input, isolate the EIM signal from the target driving source with one analog switch and apply the logic value with a second analog switch. Alternately, peripheral devices with three-state outputs may be used. Ensure the output is high-impedance during the boot up interval. | Using EIM boot interface lines as inputs may result in a wrong boot up due to the source overcoming the pull resistor value.<br>A peripheral device may require the EIM signal to have an external or on-chip resistor to minimize signal floating. If the usage of the EIM boot signal affects the peripheral device, then an analog switch, open collector buffer, or equivalent should isolate the path. A pull-up or pull-down resistor at the peripheral device may be required to maintain the desired logic level. Review the switch or device data sheet for operating specifications. |
| | **5.** Ensure EIM boot interface lines used as outputs are not loaded down such that the level is interpreted as low during power up, when the intent is to be a high level, or vice versa. | — |
| **$I^2C$ Recommendations** | | |
| | **6.** Verify the target $I^2C$ interface clock rates. | Remember the bus can only operate as fast as the slowest peripheral on the bus. |
| | **7.** Verify the target $I^2C$ address range is supported and not conflicting with other peripherals. If there is an unavoidable address conflict, move the offending device to another $I^2C$ port. See Table 1-3. | The i.MX53 supports up to three $I^2C$ ports.<br>If it is undesirable to move a conflicting device to another $I^2C$ port, review the peripheral operation to see if it supports re-mapping the addresses. |
| | **8.** Do not place more than one set of pull-up resistors on the $I^2C$ lines. | This can result in excessive loading. Good design practice is to place a pair of pull-ups only on the schematic page that has the i.MX53 symbol. Do not place pull-ups on the pages with the $I^2C$ peripherals. |

**Table 1-1. Design Checklist (continued)**

| Check Box | Recommendation | Explanation/Supplemental Recommendations |
|---|---|---|
| **JTAG Recommendations** | | |
| | **9.** Do not use external pull-up or pull-down resistors on JTAG_TDO. | JTAG_TDO is configured with an on-chip keeper circuit such that the floating condition is eliminated if an external pull resistor is not present. An external pull resistor on JTAG_TDO is detrimental.<br>See Table 1-5 for a summary of the JTAG interface. |
| | **10.**Ensure that the on-chip pull-up/down configuration is followed If external resistors are used with non-JTAG_TDO signals. For example, do not use an external pull-down on an input that has on-chip pull-up. | External resistors can be used with non-JTAG_TDO signals, but they do not need to be used.<br>See Table 1-5 for a summary of the JTAG interface. |
| **Clock Amplifier (CAMP) Recommendations** | | |
| | **11.** After initialization, disable unused clock amplifiers (CAMPs) within the CCM registers (CCM_CCR[CAMPx_EN] field). | CKIH1 and CKIH2 are inputs feeding CAMPs (clock amplifiers) that have on-chip AC coupling, eliminating the need for external coupling capacitors. The CAMPs are enabled by default, but the main clocks feeding the on-chip clock tree are sourced from XTAL/EXTAL upon power up.<br>Using low jitter external oscillators to feed CKIH1 or CKIH2 is not required, but it can advantageous if low jitter or special frequency clock sources are required by modules driven by CKIH1 or CKIH2.<br>See CCM chapter in the i.MX53 Reference Manual for details on the respective clock trees. |
| | **12.** Tie CKIH1/CKIH2 to GND if they are unused. | If disabled, the on-chip CAMP output is low. |
| **TVDAC Recommendations** | | |
| | **13.** Tie TVDAC_VREF to an external 1.05 kΩ 1% resistor to GND. | TVDAC_VREF determines the Triple Video DAC (TVDAC) reference voltage. This resistor must be mounted close to the associated BGA ball. |
| | **14.** Bypass the TVDAC_COMP reference with an external 0.1 μF capacitor tied to GND. | This capacitor must be mounted close to the associated BGA ball.<br>If TV OUT is not used, float the COMP contact and ensure the DACs are powered down by software. |
| | **15.** External ESD (electro-static discharge) and EOS (electrical overstress) protection is required at the processor device contacts for the following signals:<br>• TVDAC_IOB<br>• TVDAC_IOG<br>• TVDAC_IOR<br>• TVCDC_IOB_BACK<br>• TVCDC_IOG_BACK<br>• TVCDC_IOR_BACK | TVDAC_IOB, TVDAC_IOG, TVDAC_IOR, TVCDC_IOB_BACK, TVCDC_IOG_BACK, and TVCDC_IOR_BACK are analog TV outputs.<br>If the TV outputs are not used, they may be floated or tied to GND. |

**Table 1-1. Design Checklist (continued)**

| Check Box | Recommendation | Explanation/Supplemental Recommendations |
|---|---|---|
| **Miscellaneous Signal Recommendations** | | |
| | **16.** Tie FASTR_ANA and FASTR_DIG connections to GND | FASTR_ANA and FASTR_DIG are reserved for Freescale manufacturing use only. |
| | **17.** Float TEST_MODE or tie it to GND. | TEST_MODE is for Freescale factory use only. This signal is internally connected to an on-chip pull-down device. |
| | **18.** Float the USB_H1_GPANAIO and USB_OTG_GPANAIO outputs. | USB_H1_GPANAIO and USB_OTG_GPANAIO are reserved for Freescale manufacturing use. |
| | **19.** Connect SVCC and SVDDGP to test pads to facilitate measurement of printed circuit board IR drop from regulator to load. | The SVCC and SVDDGP sense lines provide the ability to sense voltage levels at the BGA package on their respective supplies. SVCC is used to monitor VCC and SVDDGP for VDDGP. |
| | **20.** For Ethernet access, the MAC address may be stored in the processor's fuse bank 1. | — |
| **LVDS Recommendations** | | |
| | **21.** For the LVDS_BG_RES input:<br>• Connect 28 kΩ 1% to GND when the external resistor option is chosen.<br>• If LVDS is not used, this signal can be a no connect. | LVDS_BG_RES functions as reference for the LVDS band-gap circuit. This resistor must be mounted close to the associated BGA ball. |
| | **22.** Connect NVCC_LVDS_BG to a 2.5 V supply though a series 49.9 Ω 1% resistor. Mount this resistor close to the associated BGA ball. Mount a 0.01 μF decoupling capacitor near the NVCC_LVDS_BG BGA contact. | NVCC_LVDS_BG functions as a source for the LVDS band-gap circuit. |
| **USB Recommendations** | | |
| | **23.** USB_H1_RREFEXT and USB_OTG_RREFEXT require a separate external 6.04 kΩ 1% resistors to GND. | USB_H1_RREFEXT and USB_OTG_RREFEXT determine reference currents for USB PHY band gap references that generate driver current. RREFEXT values are critical as they affect most of transmitter parameters. Additional recommendations for resistor connection:<br>• The connection must be made through a short trace<br>• The resistance of the connection line should be as low as possible (< 1 Ω)<br>• Both of the RREFEXT resistors and connections should be placed away from noisy regions; Freescale recommends 2x to 3x adjacent keep out and GND plane immediately below the trace to reduce coupling. |
| | **24.** Do not connect the VBUS contacts on the processor directly to the VBUS contact on the associated USB connector. | The user must employ a series 47 Ω resistor followed with a 1 μF capacitor mounted directly at the processor VBUS BGA ball. In addition, external ESD (electrostatic discharge) and EOS (electrical overstress) protection is required at the VBUS BGA ball. |

**Table 1-1. Design Checklist (continued)**

| Check Box | Recommendation | Explanation/Supplemental Recommendations |
|---|---|---|
| | **25.** USB I/O D+, D–, and UID contacts on the i.MX device require external ESD (electro-static discharge) damage protection. | — |
| | **Power Recommendations** | |
| | **26.** Comply with the power-up and power-down sequence guidelines as described in the data sheet to guarantee reliable operation of the device. | Any deviation from these sequences may result in the following situations:<br>• Excessive current during power-up phase<br>• Prevention of the device from booting<br>• Irreversible damage to the i.MX53 processor (worst-case scenario) |
| | **27.** Bypass both VDD_DIG_PLL (sourced from the on-chip 1.2 V linear regulator) and VDD_ANA_PLL (sourced from the on-chip 1.8 V linear regulator) with separate ≥10 μF low-ESR capacitors to GND. | There is no need to drive these supplies externally, and external supplies are not recommended due to possible noise introduction, power-up sequence issues, or other complications. The bypass capacitor must be included whether VDD_DIG_PLL or VDD_ANA_PLL is sourced on-chip or driven externally.<br>Refer to the data sheet for the applicable external supply levels (if driven externally). A 0.1 μf or 0.22 μF capacitor can be added in parallel to each larger capacitor when an external voltage source is used. The 10 μF minimum value must take into account temperature and expected capacitor aging. |
| | **28.** VDD_REG must be decoupled with a 22 μF capacitor to GND. Mount the VDD_REG capacitor close to the associated BGA ball. If two capacitors are utilized, mount the smaller capacitor (such as 0.22 μF) closer to the associated ball. | VDD_REG is the power supply input for the on-chip linear voltage regulators that supply the PLL digital and analog sections. |
| | **29.** To configure CKIL and ECKIL as an oscillator, tie a 32.768 kHz crystal with <70 kΩ ESR (equivalent series resistance) and approximately 9 pF load between CKIL and ECKIL. Do not use an external biasing resistor. | The capacitors implemented on either side of the crystal are about twice the crystal load capacitor. To hit the target oscillation frequency, board capacitors need to be reduced to compensate for board and chip parasitic capacitance, so 15–16 pF could be employed. The integrated oscillation amplifier has an on-chip self-biasing scheme, but is high-impedance (relatively weak) to minimize power consumption.<br>Care must be taken to limit parasitic leakage from CKIL and ECKIL to either power or ground (> 20 MΩ) as this negatively affects the amplifier bias and causes a reduction of startup margin.<br>Use short traces between the crystal and the processor, with a ground plane under the crystal, load capacitors, and associated traces.<br>Typically CKIL and ECKIL should bias to approximately 0.5 V |

**Table 1-1. Design Checklist (continued)**

| Check Box | Recommendation | Explanation/Supplemental Recommendations |
|---|---|---|
| | **30.** If feeding an external clock into the device, CKIL can be driven DC-coupled with ECKIL floated. | The logic high level driven into CKIL should be approximately NVCC_SRTC_POW. Do not exceed NVCC_SRTC_POW or damage/malfunction may occur. The CKIL signal should not be driven if the NVCC_SRTC_POW supply is off. This can lead to damage or malfunction.<br>Driving ECKIL is allowed but is not optimum because ECKIL is the output of the on-chip amplifier. |
| | **31.** The user should place a 24 MHz fundamental-mode crystal across XTAL/EXTAL. The crystal must be rated for a maximum drive level of 100 μW or higher. An ESR of 80 Ω or less is recommended. Freescale BSPs (board support packages) software requires 24 MHz on EXTAL. | If no TV encoding is required, the tolerance limitation is due to USB and a crystal with tolerance up to ±225 ppm (includes aging) may be used. For use of standard definition TV-out, tolerance up to ±50 ppm may be used.<br>The crystal can be eliminated if an external oscillator is available. In this case, EXTAL must be directly driven by the external oscillator and XTAL is floated. The EXTAL signal level must swing from NVCC_OSC to GND. If the clock is used for USB, then there are strict jitter requirements: < 50 ps peak-to-peak below 1.2 MHz and < 100 ps peak-to-peak above 1.2 MHz for the USB PHY. The COSC_EN bit in the CCM (Clock Control Module) must be cleared to put the on-chip oscillator circuit in bypass mode which allows EXTAL to be externally driven. COSC_EN is bit 12 in the CCR register of the CCM. |
| | **Reset Recommendations** | |
| | **32.** A reset switch may be wired to the i.MX53 POR_B, which is a cold-reset negative-logic input that resets all modules and logic in the IC. | The POR_B input must be asserted at power-up and remain asserted until after the last power rail is at its working voltage. |
| | **33.** Typically, RESET_IN_B is wired to the JTAG reset signal.<br>Alternately, connect POR_B to JTAG reset. In this case assertion of JTAG reset reboots the processor (see Table 1-6). | RESET_IN_B is a warm reset negative logic input that resets all modules and logic except for the following:<br>• Test logic (JTAG, IOMUXC, DAP)<br>• SRTC<br>• Memory repair—Configuration of memory repair per fuse settings<br>• Cold reset logic of WDOG—Some WDOG logic is only reset by POR_B. See the WDOG chapter in the i.MX53 reference manual for details. |

**Table 1-1. Design Checklist (continued)**

| Check Box | Recommendation | Explanation/Supplemental Recommendations |
|---|---|---|
| **SATA Recommendations** | | |
| | **34.** The impedance calibration process requires connecting a 191 Ω 1% reference resistor on SATA_REXT to ground.<br>Mount this resistor close to the associated BGA ball. | Module calibration consists of learning which internal resistor calibration register state causes an internal, digitally trimmed calibration resistor to best match the impedance applied to the SATA_REXT. This calibration register value is then supplied to all internal Tx and Rx termination resistors.<br>For < 100 μs during the calibration process, up to 0.3 mW can be dissipated in the external SATA_REXT resistor. At other times, no power is dissipated in the SATA_REXT resistor. |
| | **35.** For BOOT_MODE1 and BOOT_MODE0, use one of the following options:<br>• Achieve logic 0 with any of these three options: tie to GND through any size external resistor, tie directly to GND, or float.<br>• For logic 1, the options are: tie directly to NVCC_RESET or tie to NVCC_RESET through an external resistor ≤ 10 kΩ. A value of ≤ 4.7 kΩ is preferred in high-noise environments.<br>• If switch control is desired, use 4.7 kΩ to 68 kΩ pull-down resistors and a SPST switch to NVCC_RESET. | Boot inputs BOOT_MODE1 and BOOT_MODE0 each have on-chip pull-down devices with a nominal value of 100 kΩ and a projected minimum of 60 kΩ. |

## 1.1 Boot Configuration Bus Isolation Resistors

Figure 1-1 shows the boot configuration bus isolation resistors referenced in recommendation 3.

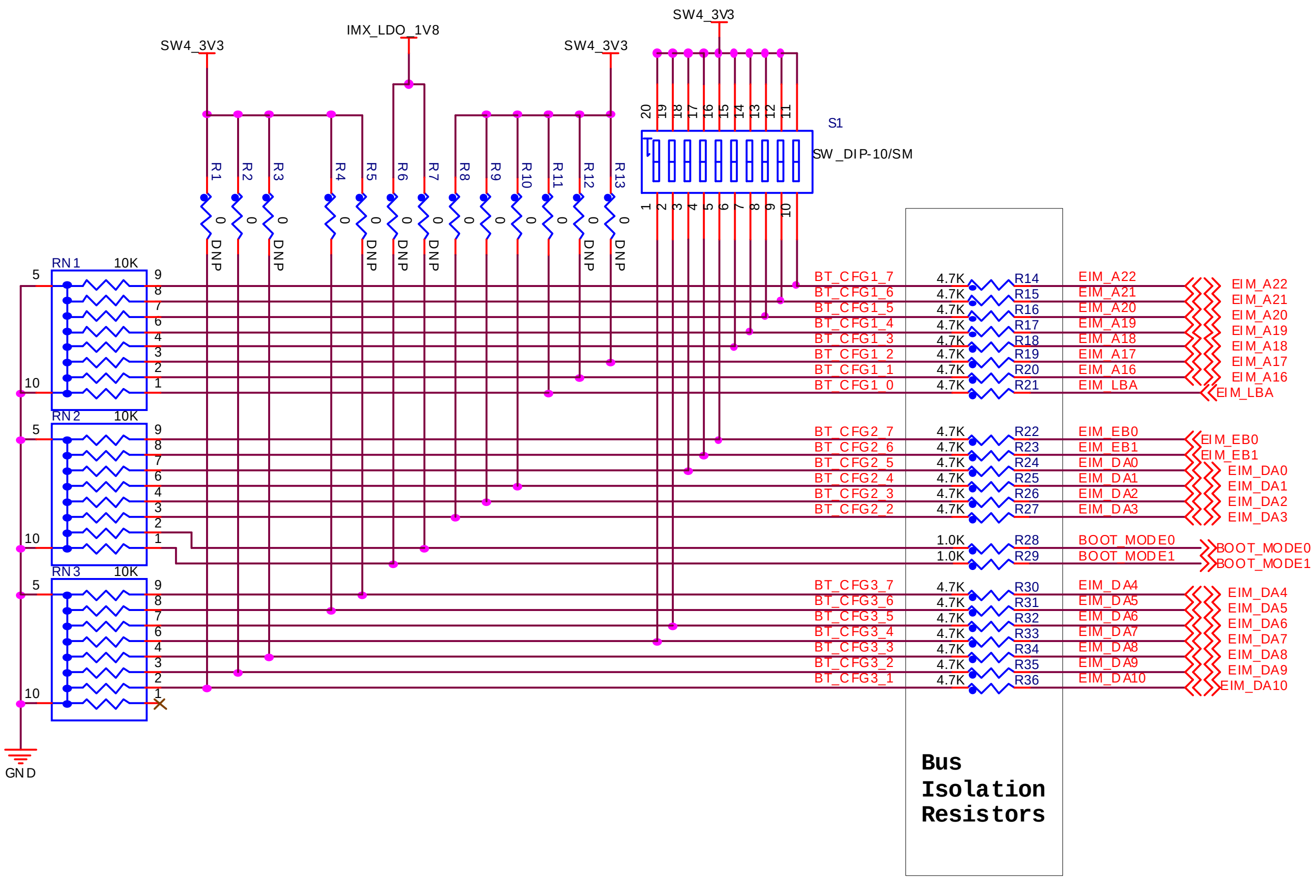



Figure 1-1. Boot Configuration Bus Isolation Resistors

## 1.2 DDR Reference Circuit

Table 1-2 is a resistor chart (see recommendation 1). The recommendations are appropriate for designs with DDR memory chips with a maximum Vref input current of 2 µA each.

Table 1-2. DDR Vref Resistor Sizing Guideline

| Number of DRAM Packages with 2 µA Vref Input Current | Resistor Divider Value (2 resistors) |
|---|---|
| 2 | 1.21 kΩ 1% |
| 2 | 1.54 kΩ 0.5% |
| 2 | 2.32 kΩ 0.1% |
| 4 | 768 Ω 1% |

**Table 1-2. DDR Vref Resistor Sizing Guideline (continued)**

| Number of DRAM Packages with 2 µA Vref Input Current | Resistor Divider Value (2 resistors) |
|---|---|
| 4 | 1 kΩ 0.5% |
| 4 | 1.5 kΩ 0.1% |

## 1.3 Avoiding I²C Conflicts

Table 1-3 shows a spreadsheet for avoiding avoid I²C conflicts (see recommendation 7).

**Table 1-3. I²C Bus Example Spreadsheet**

| Peripheral | Bus Activity Level | Speed (kbps) | Slave Addresses Supported on the Peripheral (hex) | Selected System Address (hex) |
|---|---|---|---|---|
| PMIC | Low | 400 | 68 | 68 |
| Port Expander | Low | 400 | 30, 32, 34 | 30 |
| AM-FM Tuner | Med | 250 | C0, C2, C4, C6 | C0 |
| A/D Converter | Med | 400 | 40, 42 | 40 |
| Audio CODEC | Low | 400 | 90, 92, 94, 96 | 90 |

Note that there are no slave address conflicts. However, the shaded cell does call out a potential bus speed issue. The AM-FM tuner limits the maximum bus rate to 250 kbps, and the bus data rate cannot exceed the slowest peripheral on the bus, regardless of which peripheral is being accessed.

If the system cannot tolerate the 250 kbps rate for proper operation, the AM-FM tuner must be moved to another I²C port. If the I²C bus rate exceeds the AM-FM tuner module's maximum bus rate, the I²C bus operation may fail or become unpredictable.

Assuming the system can function properly with a reduced bus rate of 250 kbps, Table 1-4 shows a possible I²C port usage scenario.

**Table 1-4. I²C Port Usage Scenario**

| i.MX53 I²C Ports | Ball Name | Function | Speed (kbps) |
|---|---|---|---|
| Port 1 | | | |
| Port 1 | | | |
| Port 2 | KEY_ROW3 | I2C2_SDA | 250 |
| Port 2 | EIM_EB2 | I2C2_SCL | 250 |
| Port 3 | | | |
| Port 3 | | | |

## 1.4 JTAG Signal Termination

Table 1-5 is a JTAG termination chart (see recommendation 9 and recommendation 10).

**Table 1-5. JTAG Interface Summary**

| JTAG Signal | i.MX53 I/O Type | On-Chip Termination to NVCC_JTAG or GND | External Termination |
|---|---|---|---|
| JTAG_TCK | Input | 100 kΩ pull-down | Not required<br>Can use 10 kΩ pull down |
| JTAG_TMS | Input | 47 kΩ pull-up | Not required<br>Can use 10 kΩ pull up |
| JTAG_TDI | Input | 47 kΩ pull-up | Not required;<br>Can use 10 kΩ pull up |
| JTAG_TDO | 3-state output | Keeper | Do not use pull up or pull down |
| JTAG_TRSTB | Input | 47 kΩ pull up | Not required<br>Can use 10 kΩ pull up |
| JTAG_MOD | Input | 100 kΩ pull up | Required<br>Use 0 to 6.8 kΩ pull down |

Table 1-6 shows additional JTAG signals that are not required for the processor's JTAG operation (see recommendation 33).

**Table 1-6. Additional JTAG Signals**

| JTAG Signal | System/Target Pin Type | Requirements or Recommendations | Discussion |
|---|---|---|---|
| JTAG_RST_B | Driven from ARM emulator | When utilized:<br>• Ensure the proper voltage levels.<br>• Ensure connection point is an open-drain or wired-OR (diode-OR) to alleviate contention.<br>• Install pull-up. | This signal allows the emulator to perform "Target Reset" from the emulator keyboard commands. |
| JTAG_DE_B | I/O from ARM emulator | Employ a 10 kΩ pull-up for general emulator usage because this signal is tagged as active-low logic. | This signal functions as a "Debug Request" and "Debug Acknowledge" between the emulator and target. Consult the emulator documentation for proper target design usage. |

# Chapter 2
# i.MX53 Layout Recommendations

This chapter provides recommendations to assist design engineers with the correct layout of their i.MX53x-based system. The majority of the chapter discusses the implementation of the DDR interface, but it also provides recommendation for power, the TV encoder, SATA, LVDS, reference resistors, and ESD and related emissions.

This chapter uses the i.MX53 Quick Start board as its reference when illustrating the key concepts. Refer to the existing i.MX53 Quick Start board layout files as a companion to this chapter.

## 2.1 Basic Design Recommendations

The i.MX53 processor comes in a 19 × 19 mm package with 0.8 mm ball pitch. The ball-grid array contains 23 rows and 23 columns, making it a 529 ball BGA package. For detailed information about the package, see the i.MX53 data sheet.

Figure 2-1 provides an illustration of the ball-grid array and Figure 2-2 illustrates additional package information.

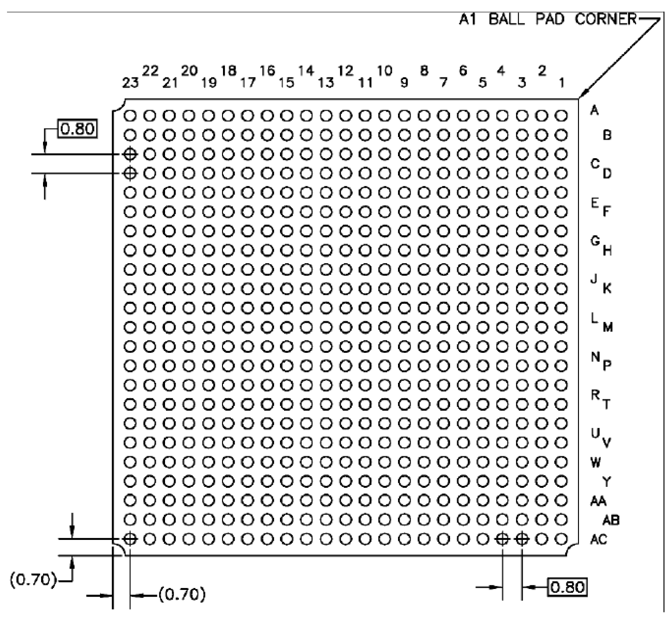



Figure 2-1. i.MX53 Ball-Grid Array

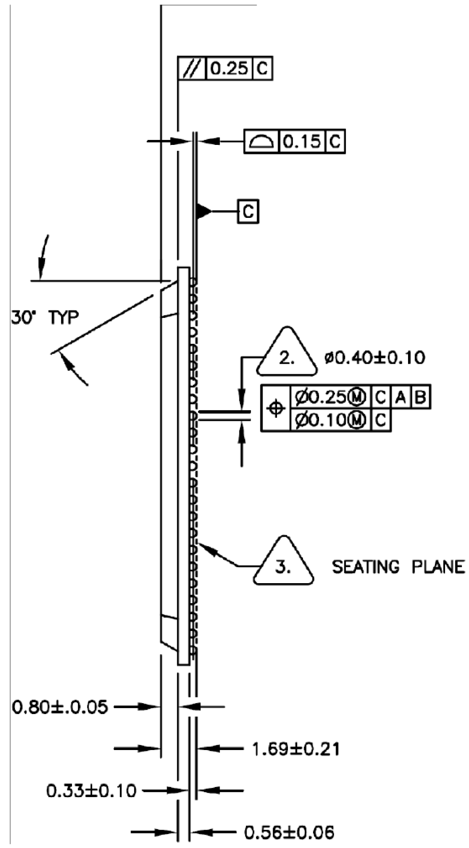



**Figure 2-2. i.MX53 Package Information**

Maintaining the recommended footprint of a 12 mils pad, which allows an air gap of 19.5-mils between pads, is critical for ease of fanout.

If using the Allegro tool, optimal practice is to use the footprint as created by Freescale. If not using the Allegro tool, use the Allegro footprint export feature (supported by many tools). If export is not possible, create the footprint as per the package mechanical dimensions outlined in the product data sheet.

### 2.1.1 Fanout

Figure 2-3 shows the fanouts for the i.MX53 for two different layers.

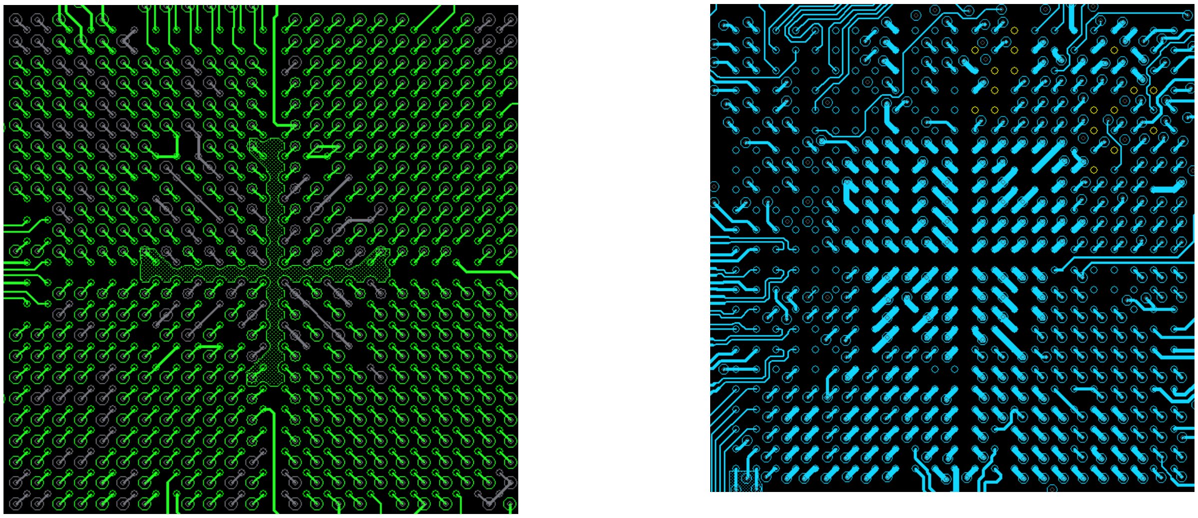


**Figure 2-3. i.MX53 Fanouts**

The fanout scheme creates a four quadrant structure that facilitates the placement of decoupling capacitors on the bottom side of the PCB. This keeps them closer to the power balls, which is critical for minimizing inductance and ensuring high-speed transient current demand by the processor.

A correct via size is critical for preserving adequate routing space. The recommended geometry for the via pads is: pad size 16 mils and drill 8 mils.

The constraints for the trace size may depend on a number of factors, such as the board stackup and associated di-electric and copper thickness, required impedance, and required current (for power traces). On the Freescale reference design, the minimum trace width of 3 mils is used for the DDR routing.

## 2.2 Stackup

High-speed design requires a good stackup in order have the right impedances for the critical traces. This also determines the constraints for routing and spacing.

The recommended stackup is 8-layers, with the layer stack as shown in Figure 2-4:

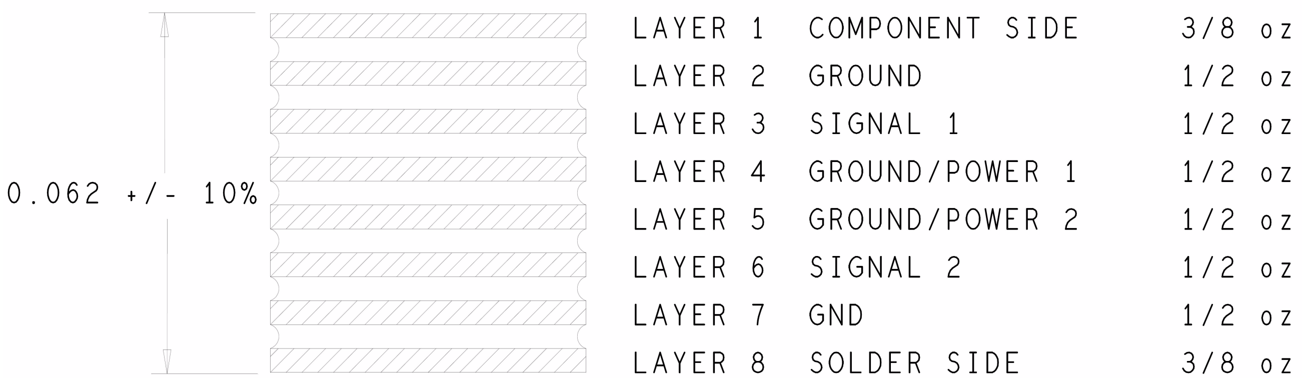



**Figure 2-4. Layer Stack**

Figure 2-5 shows a working stack-up implementation:

IMPEDANCE REQUIREMENTS

| Layers | Single Ended | | Single Ended | | Differential | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Trace Width (Mils) | Impedance (Ohms) | Trace Width (Mils) | Impedance (Ohms) | Trace Width (Mils) | Trace Pitch center-center (Mils) | Impedance (Ohms) | Trace Width (Mils) | Trace Pitch center-center (Mils) | Impedance (Ohms) |
| TOP | 8.50 | 50 | 3.25 | 75 | 6.25 | 4.75 | 90 | 5.00 | 5.00 | 100 |
| L2_GND | 3.25 | 50 | | | 3.75 | 6.25 | 90 | 3.00 | 6.00 | 100 |
| L3_SIGNAL_1 | 3.25 | 50 | | | 3.75 | 6.25 | 90 | 3.00 | 6.00 | 100 |
| L4_GND/PWR | 3.25 | 50 | | | 3.75 | 6.25 | 90 | 3.00 | 6.00 | 100 |
| L5_GND/PWR | 3.25 | 50 | | | 3.75 | 6.25 | 90 | 3.00 | 6.00 | 100 |
| L6_SIGNAL_2 | 3.25 | 50 | | | 3.75 | 6.25 | 90 | 3.00 | 6.00 | 100 |
| L7_GND | 3.25 | 50 | | | 3.75 | 6.25 | 90 | 3.00 | 6.00 | 100 |
| BOTTOM | 8.50 | 50 | 3.25 | 75 | 6.25 | 4.75 | 90 | 5.00 | 5.00 | 100 |
| | | | | | | | | | | |
| | | | | | | | | | | |

**Figure 2-5. Stackup Requirements**

## 2.3 DDR Connection Information

The DDR2 and DDR3 interface is one of the most critical for i.MX53 routing. It requires having the controlled impedance for the single ended traces at 50 Ω and for the differential pairs at 100 Ω.

Figure 2-6 shows the block diagram of the DDR2/DDR3 interface with the i.MX53 from the reference design boards.

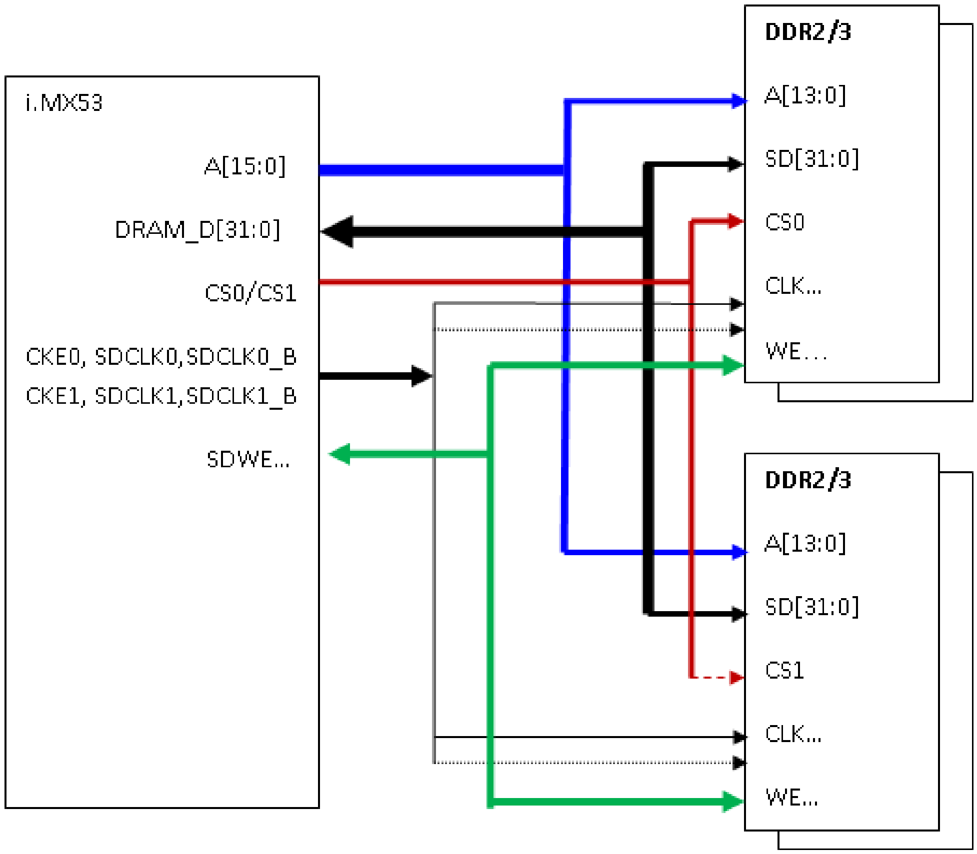



**Figure 2-6. Connection Between i.MX53 and DDR2 and DDR3**

Figure 2-7 illustrates the physical connection scheme for both top and bottom placement of the DDR chips. It is very important to place the memories as close to the processor as possible to reduce trace capacitance and keep the propagation delay to the minimum. Follow the reference board layout as a guideline for memory placement and routing.

Figure 2-7 shows the final placement of the memories and the decoupling capacitors. The blue figure shows the top layer and the red figure shows the bottom layer.

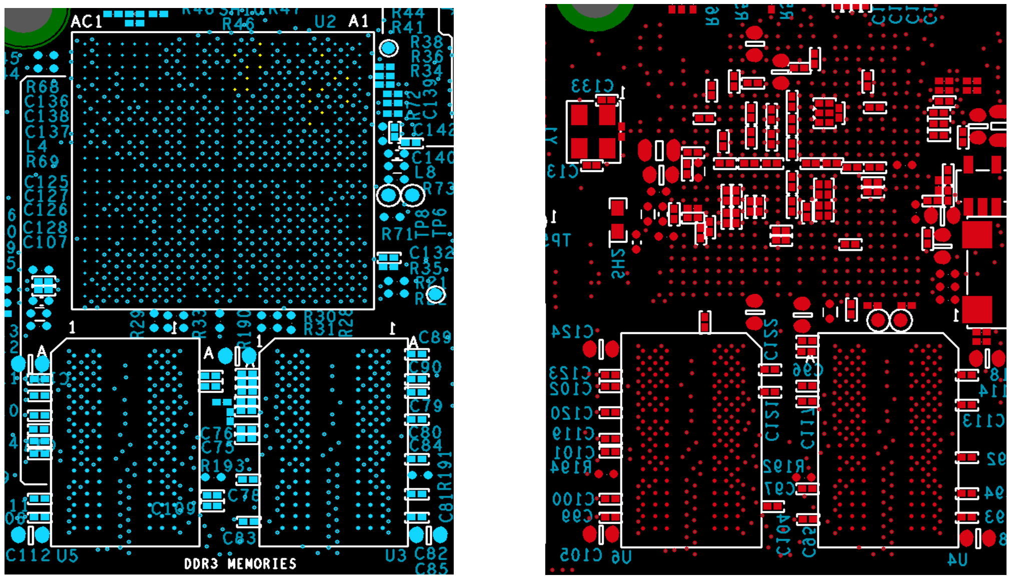



**Figure 2-7. Final Placement of Memories and Decoupling Capacitors**

## 2.4 DDR2 and DDR3 Routing Rules

DDR2 and DDR3 routing can be accomplished two different ways: routing all signals at the same length or routing by byte group.

Routing all signals at the same length can be more difficult at first because of the tight space between the DDR and the processor and the large number of required interconnects. However, it is the better way because it makes signal timing analysis straightforward. Table 2-1 explains how to route the signals by the same length.

**Table 2-1. DDR2/DDR3 Routing by the Same Length**

| Signals | Length | Considerations |
|---|---|---|
| Address and Bank | Clock length | Match the signals ± 25 mils of the value specified in the length column |
| Data and Buffer | Clock length | |
| Control signals | Clock length | |
| Clock | Lcritical (3 inches) | Match the signals of clocks signals ± 5 mils. |
| DQS and DQS_B | Clock length | Match the signals of DQS signals ± 10 mils of the value specified in the length column. |

Routing by byte group requires better control of the signals of each group. It is also a little more difficult for analysis and constraint settings. However, its advantage is that the constraint to match lengths can be applied to a smaller group of signals. This is often more achievable once the constraints are properly set. Table 2-2 explains how to route the signals by byte group.

**Table 2-2. DDR2/DDR3 Routing by Byte Group**

| i.MX53 Signals | Group | Length | | Considerations |
|---|---|---|---|---|
| | | Min | Max | |
| DRAM_SDCLK[1:0]<br>DRAM_SDCLK_B[1:0] | Clock | Short as possible | 2 inches | Match the signals ± 5 mils.<br>2 inches is recommended. |
| DRAM_A[15:0]<br>DRAM_SDBA[2:0]<br>DRAM_RAS<br>DRAM_CAS<br>DRAM_SDWE | Address and Command | Clock (min) – 200 | Clock (min)[1] | Match the signals ± 25 mils. |
| DRAM_D[7:0]<br>DRAM_DQM0<br>DRAM_SDQS0<br>DRAM_SDQS0_B | Byte Group 1 | — | Clock (min) | Match the signals of each byte group ± 25 mils.<br><br>All byte groups (1 to 4) matched ± 50 mils<br>Match the differential signals of DQS ± 10 mils. |
| DRAM_D[15:8]<br>DRAM_DQM1<br>DRAM_SDQS1<br>DRAM_SDQS1_B | Byte Group 2 | — | Clock (min) | |
| DRAM_D[23:16]<br>DRAM_DQM2<br>DRAM_SDQS2<br>DRAM_SDQS2_B | Byte Group 3 | — | Clock (min) | |
| DRAM_D[31:24]<br>DRAM_DQM3<br>DRAM_SDQS3<br>DRAM_SDQS3_B | Byte Group 4 | — | Clock (min) | |
| DRAM_CS[1:0]<br>DRAM_SDCKE[1:0]<br>DRAM_SDODT[1:0] | Control signals | Clock (min) – 200 | Clock (min) | Match the signals ± 50 mils. |

[1] Clock (min)—The shortest length of the clock group signals because this group has a ± 5 mil matching tolerance.

Finally, the impedance for the signals should be 50 Ω  for singled ended and 100 Ω for differential pairs.

## 2.5 Routing Topologies

The i.MX53 can handle up to 2 Gbytes of DRAM memory. The i.MX53 DDR routing needs to be separated into three groups: data, address, and control. Each group has its own method of routing from i.MX53 to DDR memory. The DDR layout has 1 Gbyte and 2 Gbyte options.

### 2.5.1 1 Gbyte Topologies

The 1 Gbyte option has four memories.

For good practice, adhere to the following recommendations:

- Have a balanced routing for the “T” connection.
- Avoid having many layer transitions.
- Do not cross split planes during the routing.

Figure 2-8 shows the topology for the ADDR/CMD/CTRL signals. It has a tree topology. Note the balanced T routing.

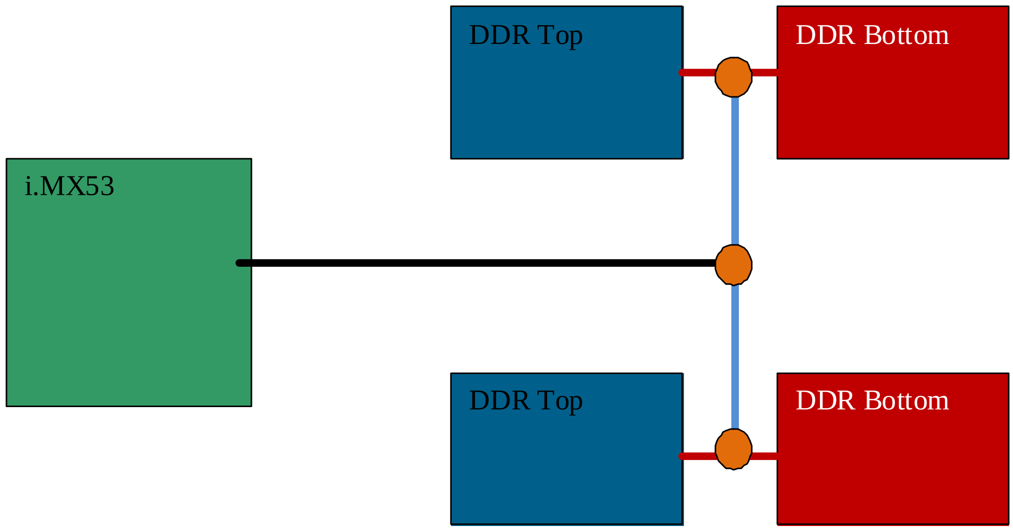



**Figure 2-8. Topology for ADDR/CMD/CTRL Signals**

The routing for the data groups depend on the bus size. Figure 2-9 shows the point-to-point connection, with routing by byte group.

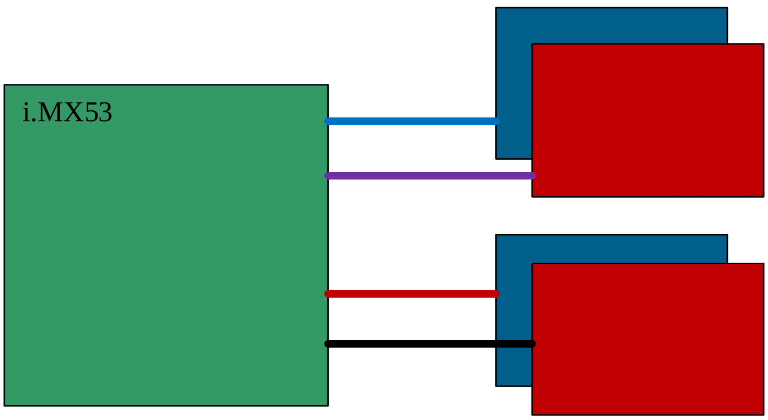



**Figure 2-9. Topology of Data Group, Point-to-Point Connection**

If the data bus is two byte groups by memory, the topology is fly-by, as shown in Figure 2-10.

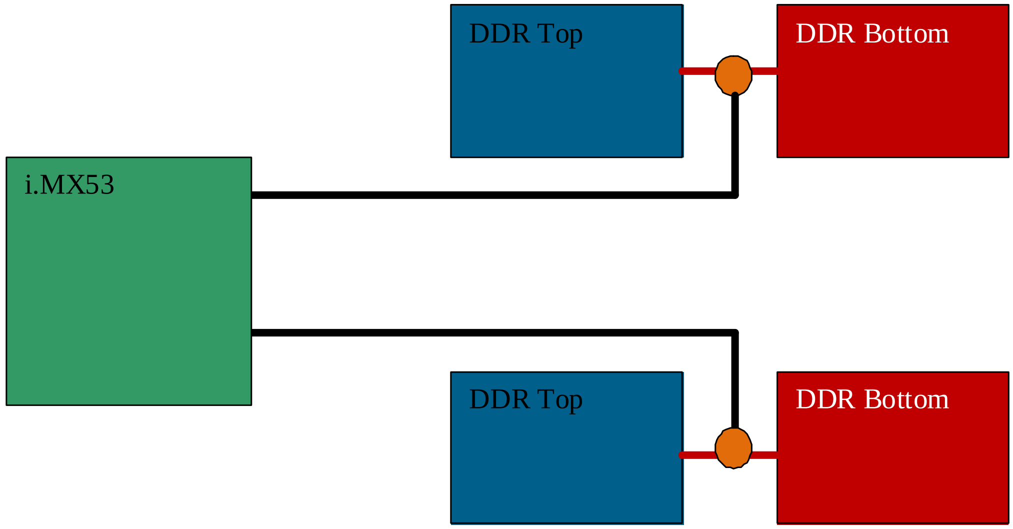



**Figure 2-10. Topology for Data Bus of Two Byte Groups by Memory**

Figure 2-11 shows the clock routing topology. Clock routing uses a fly-by topology. The i.MX53 provides two sets of clocks that are identical in timing and drive. This allows the user to select either clock pair to route to the DDR devices. Thus, routing and clock loading is minimized.

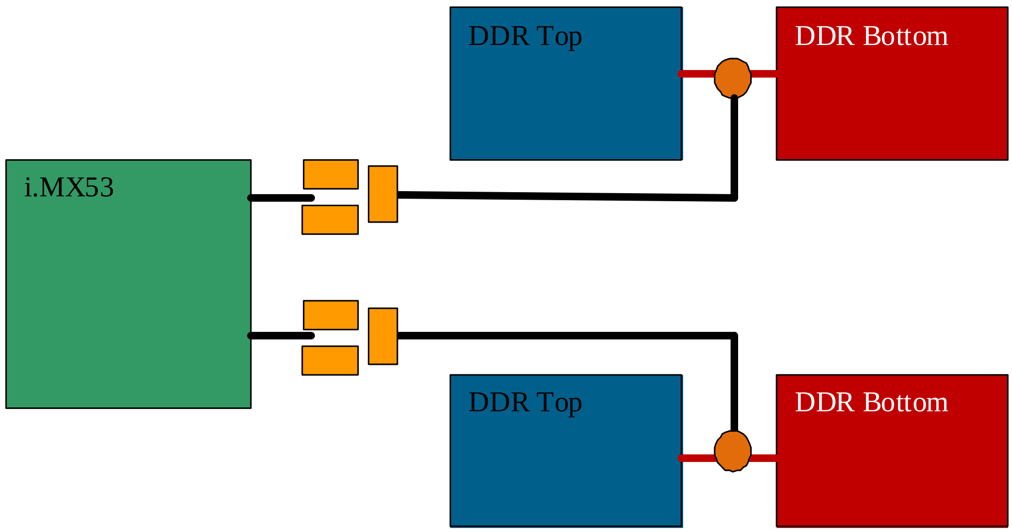



**Figure 2-11. Clock Routing Topology**

### 2.5.2 2 Gbyte Topologies

The following diagrams show the 2 Gbyte topologies. This option has eight memories and requires the addition of a termination resistor.

The ADDR/CMD signals should be routed as shown in Figure 2-12

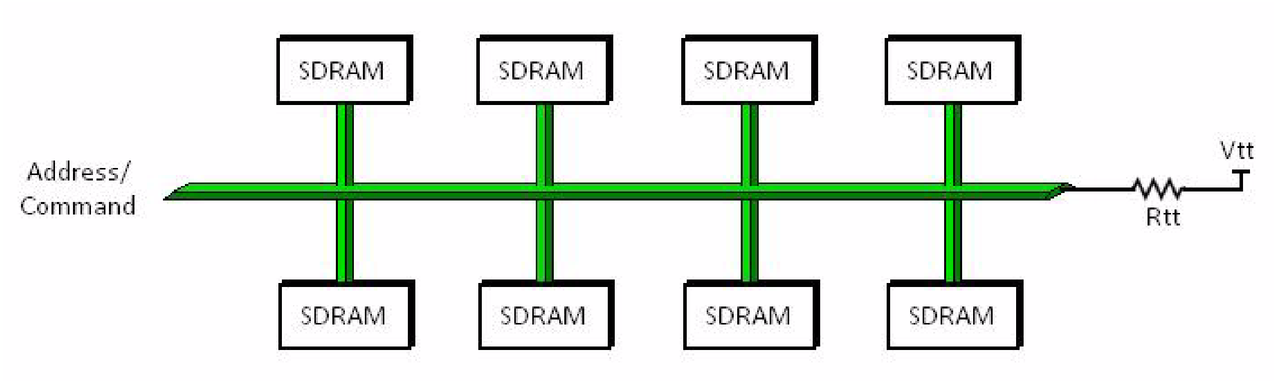



**Figure 2-12. ADDR/CMD Signal Routing**

Figure 2-13 shows the CTRL signals topology:

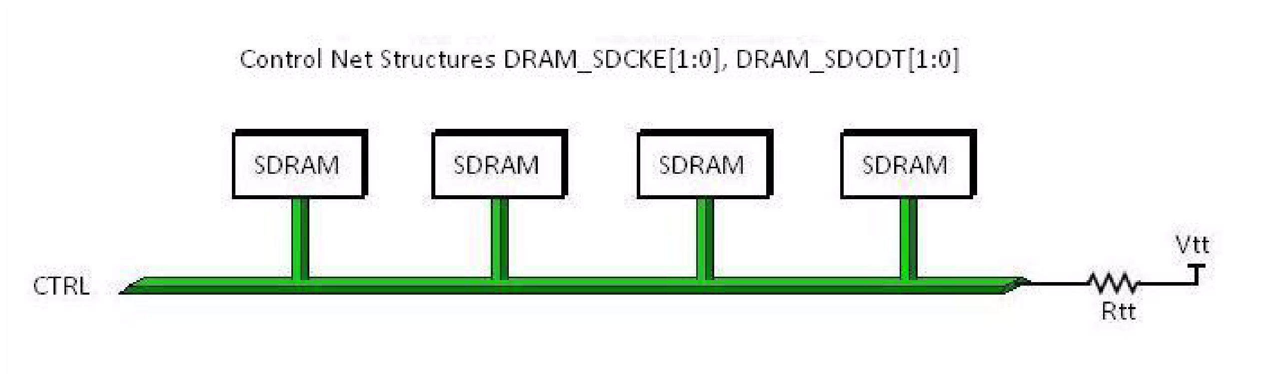



**Figure 2-13. CTRL Signal Topology**

Figure 2-14 shows the data bus routing topology.

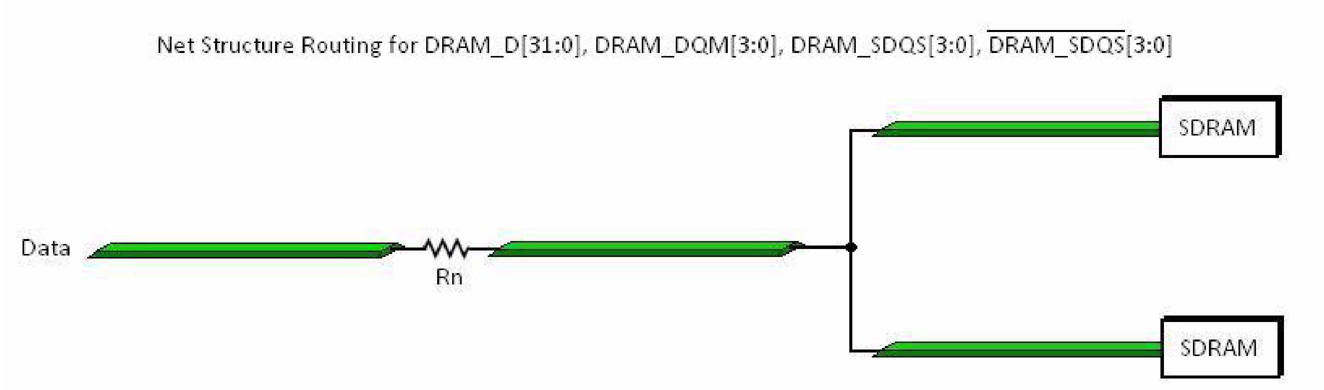



**Figure 2-14. Data Bus Routing Topology**

Figure 2-15 shows the clock routing topology.

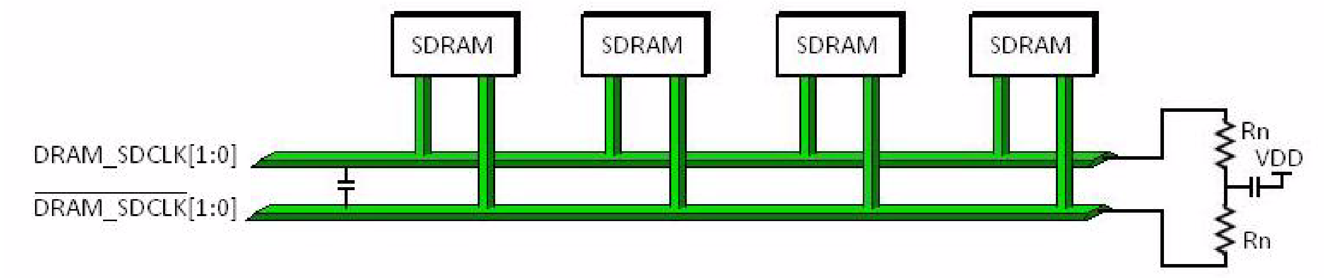
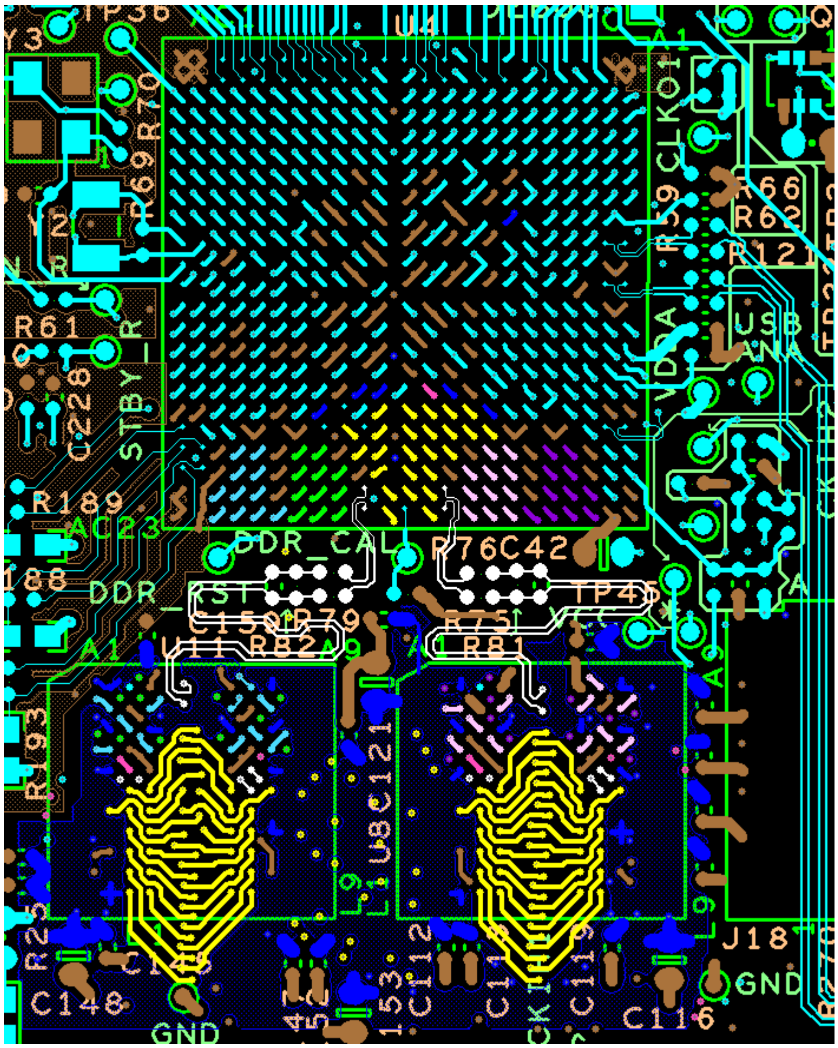



**Figure 2-15. Clock Routing Topology**

### 2.5.3 DDR2 Routing Examples

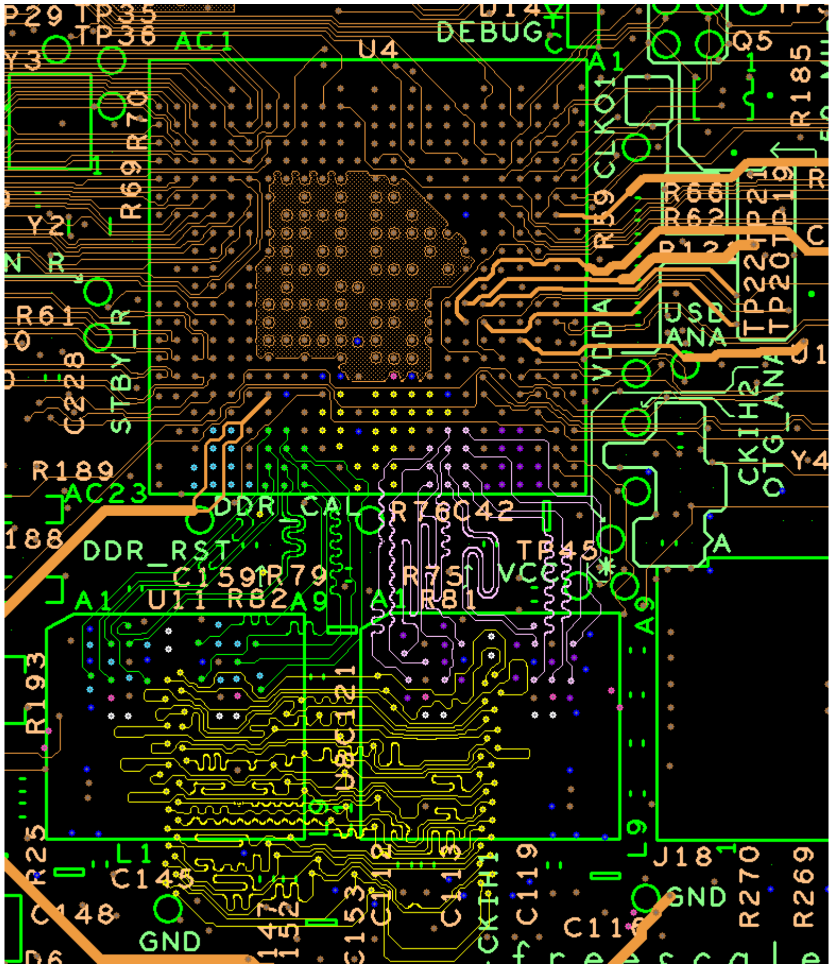
Figure 2-16–Figure 2-21 show examples for the routing of DDR2 memories. These figures are a guideline of the routing by layer.




**Color Legend**

| Color | Meaning | Color | Meaning |
|---|---|---|---|
| Yellow | ADD/CMD/CTRL Signals | Purple | Data Byte Group 1 |
| Soft Blue | Data Byte Group 3 | White | Clocks |
| Soft Pink | Data Byte Group 0 | Blue | DDR_1V8 |
| Green | Data Byte Group 2 | Pink | DDR_VREF |

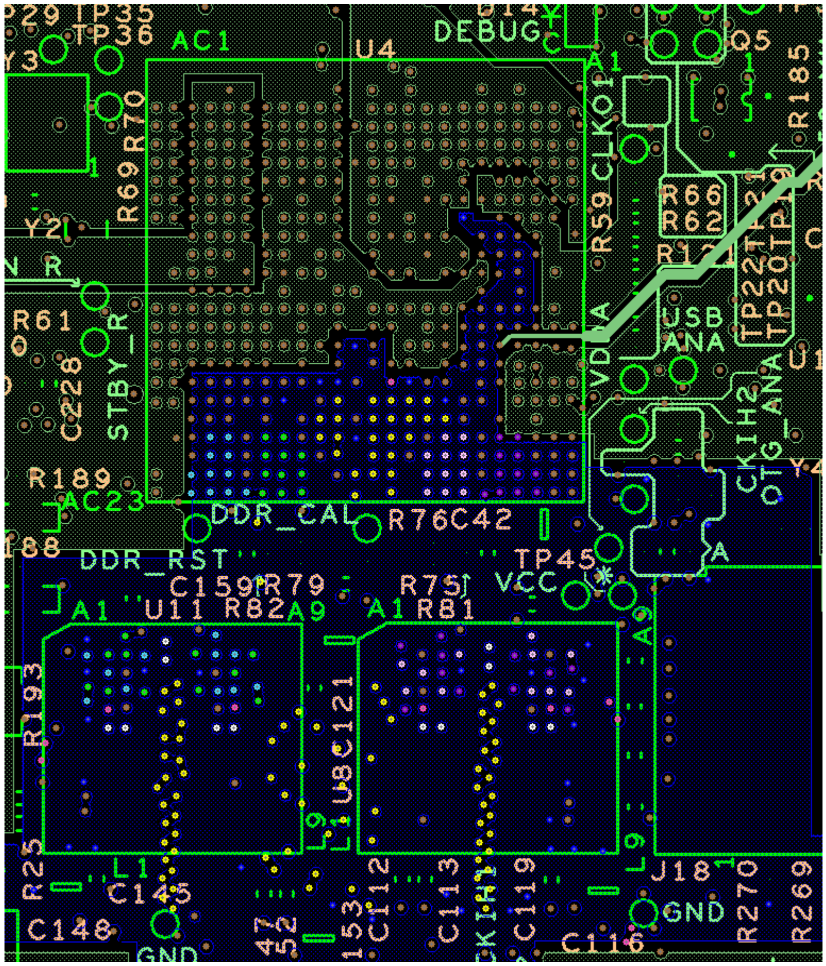
**Figure 2-16. Top DDR2 Routing**




**Color Legend**

| Color | Meaning | Color | Meaning |
|---|---|---|---|
| Yellow | ADD/CMD/CTRL Signals | Purple | Data Byte Group 1 |
| Soft Blue | Data Byte Group 3 | White | Clocks |
| Soft Pink | Data Byte Group 0 | Blue | DDR_1V8 |
| Green | Data Byte Group 2 | Pink | DDR_VREF |

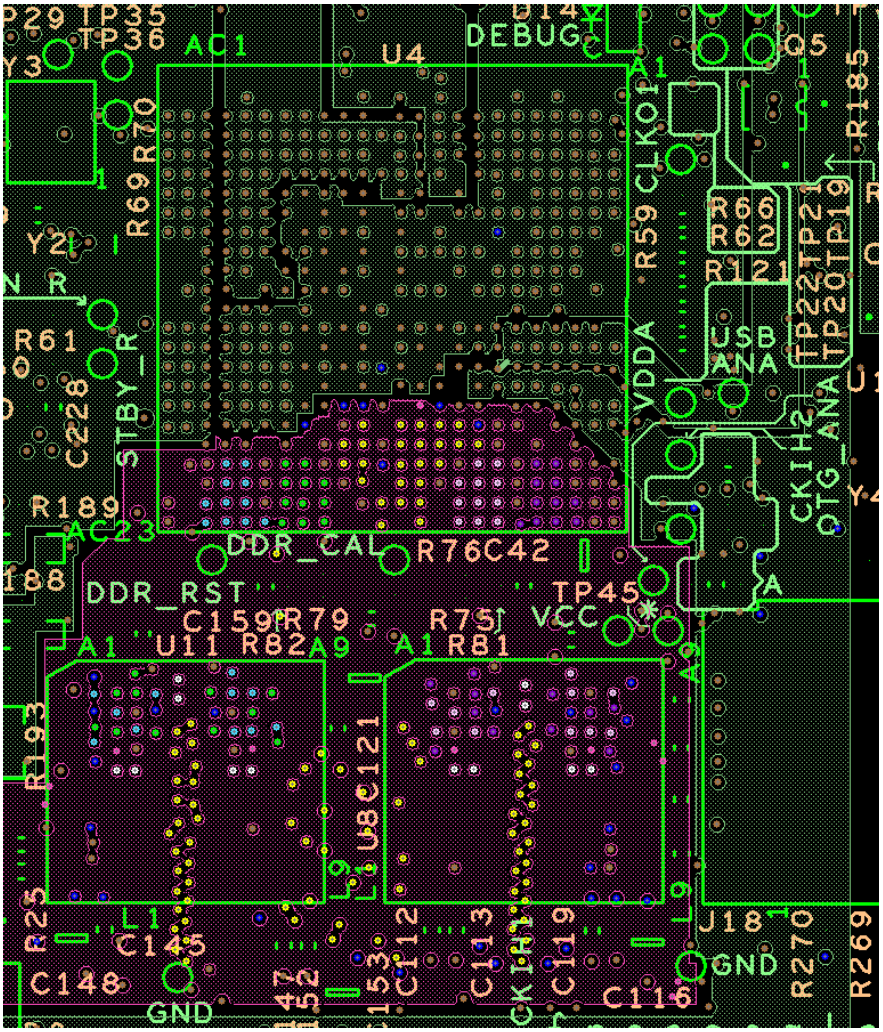
**Figure 2-17. Internal 1 DDR2 Routing**




**Color Legend**

| Color | Meaning | Color | Meaning |
|---|---|---|---|
| Yellow | ADD/CMD/CTRL Signals | Purple | Data Byte Group 1 |
| Soft Blue | Data Byte Group 3 | White | Clocks |
| Soft Pink | Data Byte Group 0 | Blue | DDR_1V8 |
| Green | Data Byte Group 2 | Pink | DDR_VREF |

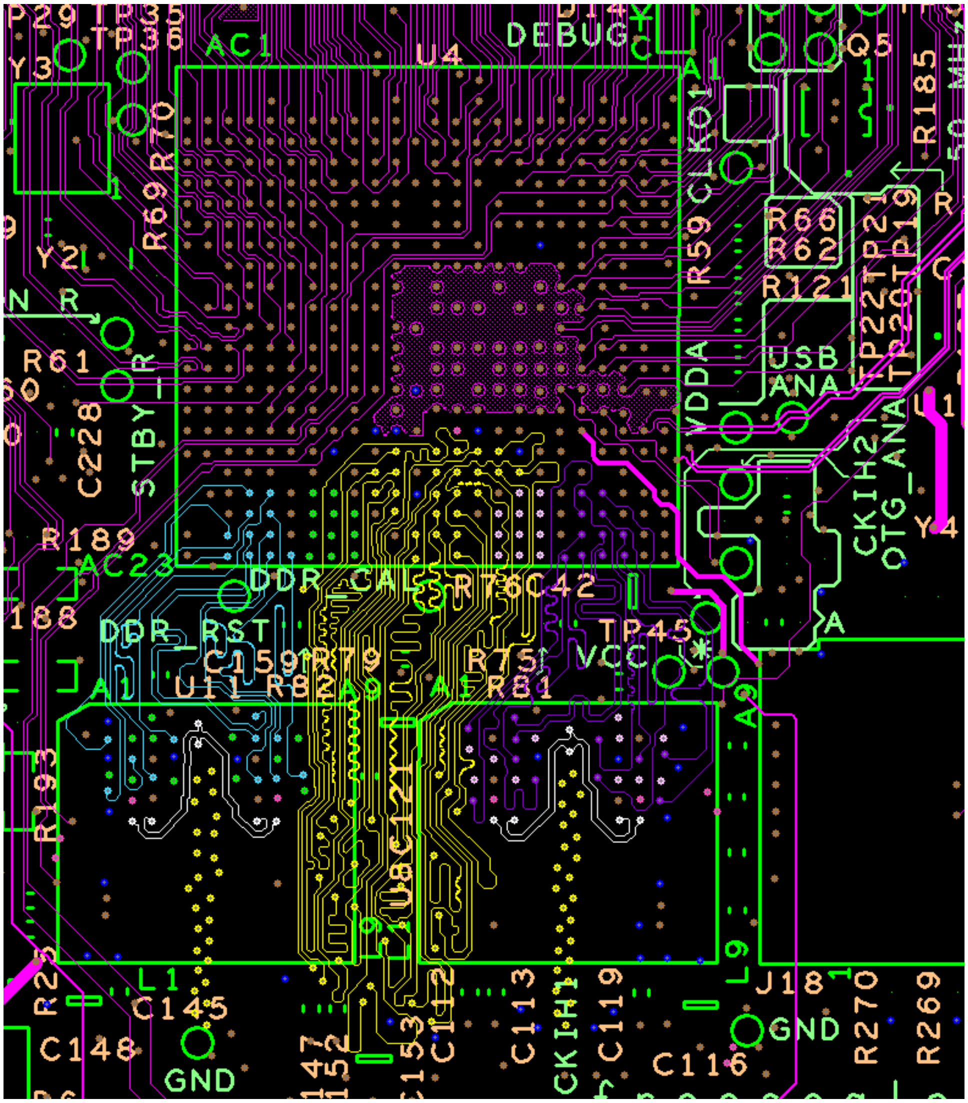
**Figure 2-18. Power Plane 1 DDR2 Routing**



**Color Legend**

| Color | Meaning | Color | Meaning |
|---|---|---|---|
| Yellow | ADD/CMD/CTRL Signals | Purple | Data Byte Group 1 |
| Soft Blue | Data Byte Group 3 | White | Clocks |
| Soft Pink | Data Byte Group 0 | Blue | DDR_1V8 |
| Green | Data Byte Group 2 | Pink | DDR_VREF |

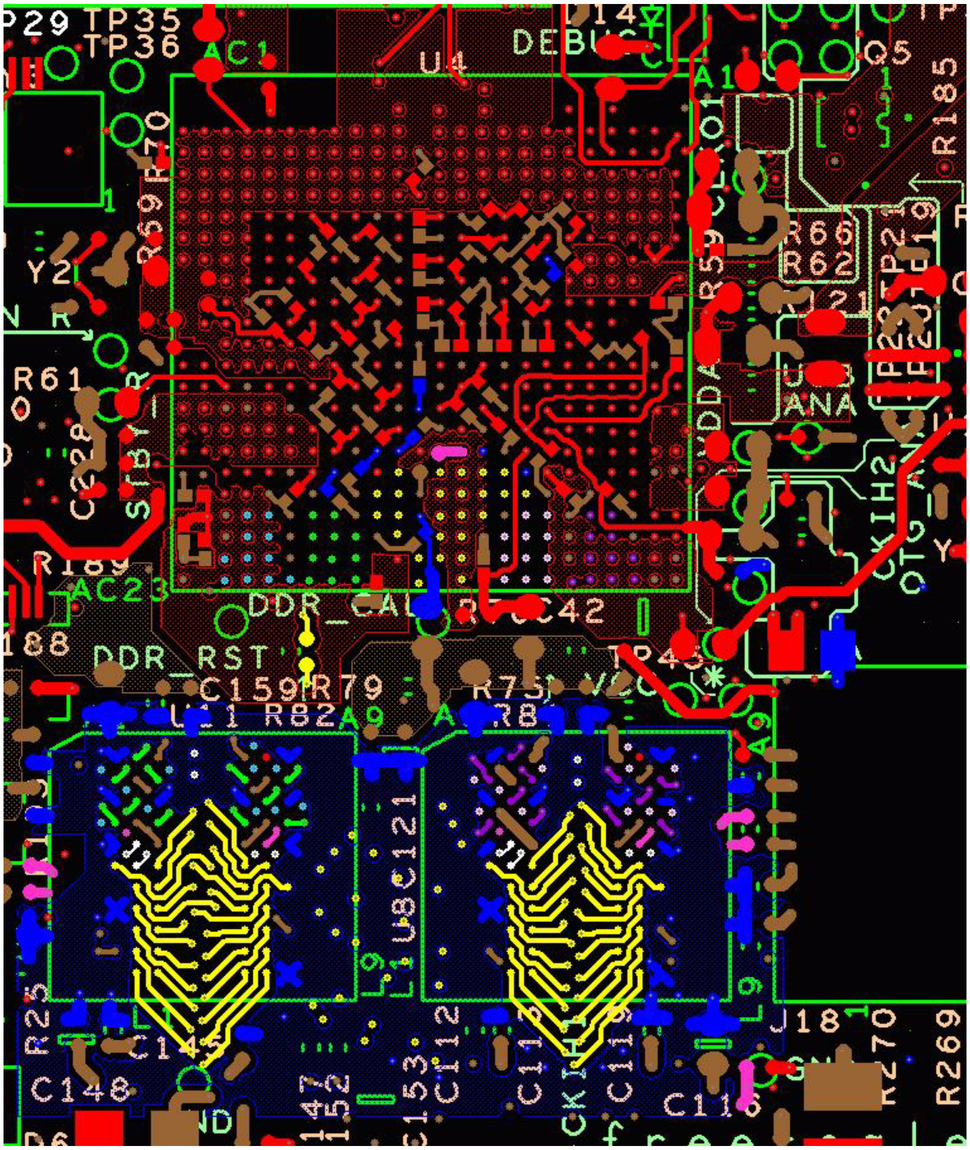
**Figure 2-19. Power Plane 2 DDR2 Routing**




**Color Legend**

| Color | Meaning | Color | Meaning |
|---|---|---|---|
| Yellow | ADD/CMD/CTRL Signals | Purple | Data Byte Group 1 |
| Soft Blue | Data Byte Group 3 | White | Clocks |
| Soft Pink | Data Byte Group 0 | Blue | DDR_1V8 |
| Green | Data Byte Group 2 | Pink | DDR_VREF |

**Figure 2-20. Internal 2 DDR2 Routing**



**Color Legend**

| Color | Meaning | Color | Meaning |
|---|---|---|---|
| Yellow | ADD/CMD/CTRL Signals | Purple | Data Byte Group 1 |
| Soft Blue | Data Byte Group 3 | White | Clocks |
| Soft Pink | Data Byte Group 0 | Blue | DDR_1V8 |
| Green | Data Byte Group 2 | Pink | DDR_VREF |

**Figure 2-21. Bottom DDR2 Routing**

Table 2-3 shows the total etch of the signals for the byte 0 and byte 1 groups.

**Table 2-3. Total Signal Etch (DDR2) [1]**

| Signals | Length (Mils) |
|---|---|
| DRAM_D0 | 667.16 |
| DRAM_D1 | 663.66 |
| DRAM_D2 | 666.01 |
| DRAM_D3 | 663.89 |
| DRAM_D4 | 662.69 |
| DRAM_D5 | 663.41 |
| DRAM_D6 | 668.31 |
| DRAM_D7 | 664.02 |
| DRAM_DQM0 | 663.5 |
| DRAM_SDQS0 | 663.62 |
| DRAM_SDQS0_B | 668.24 |
| DRAM_D8 | 668.57 |
| DRAM_D9 | 663.69 |
| DRAM_D10 | 664.28 |
| DRAM_D11 | 666.39 |
| DRAM_D12 | 664.75 |
| DRAM_D13 | 668.45 |
| DRAM_D14 | 664.65 |
| DRAM_D15 | 663.07 |
| DRAM_DQM1 | 664.08 |
| DRAM_SDQS1 | 667.66 |
| DRAM_SDQS1_B | 663.07 |
| DRAM_SDCLK0 | 1657.15 |
| DRAM_SDCLK0_B | 1655.22 |
| DRAM_SDCLK1 | 1657 |
| DRAM_SDCLK1_B | 1658.81 |

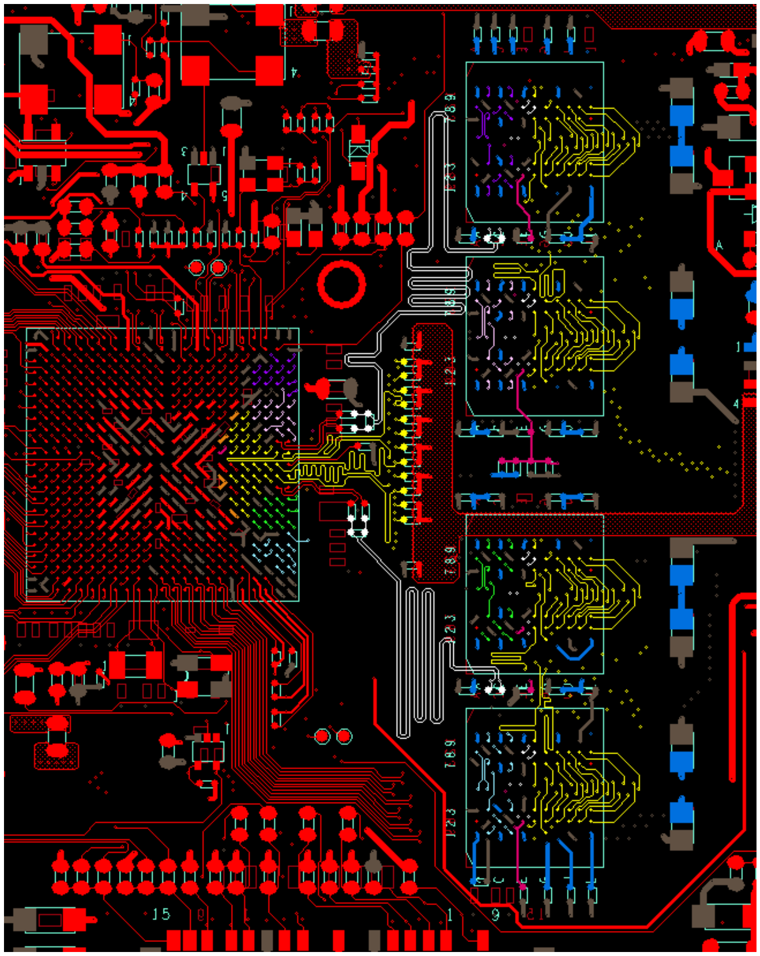
[1] Layout is an example, using 1000 mils for the clock.

### 2.5.4 2-Gbyte Routing Examples

Figure 2-22–Figure 2-27 show examples for the routing of 2-Gbyte DDR memories. These figures are a guideline of the routing by layer.

**NOTE**

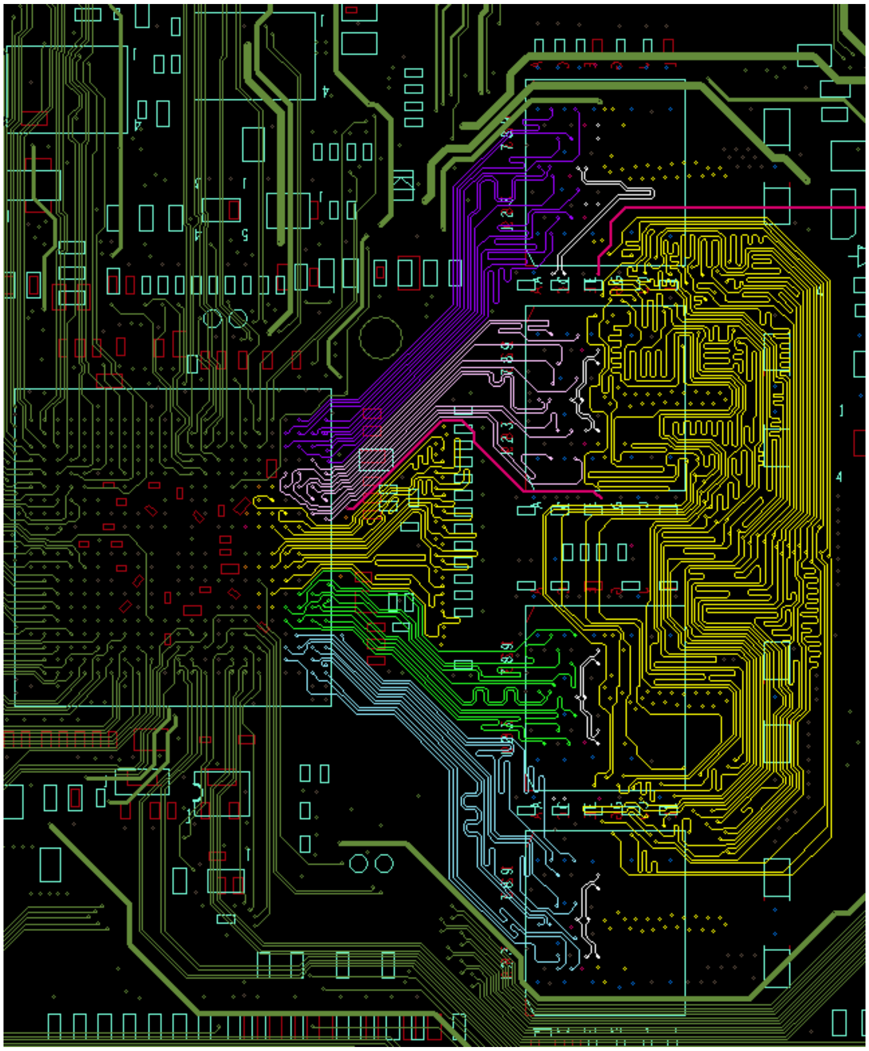
The Quick Start board referenced in the beginning of this chapter does not use 8 DDR chips. The following screen shots are from the validation board layout.



**Color Legend**

| Color | Meaning | Color | Meaning |
|---|---|---|---|
| Yellow | ADD/CMD/CTRL Signals | White | Clocks |
| Soft Blue | Data Byte Group 3 | Blue | DDR_1.5V |
| Soft Pink | Data Byte Group 0 | Pink | DDR_VREF |
| Green | Data Byte Group 2 | Orange | DDRQ_1.5V |
| Purple | Data Byte Group 1 | White | Clocks |

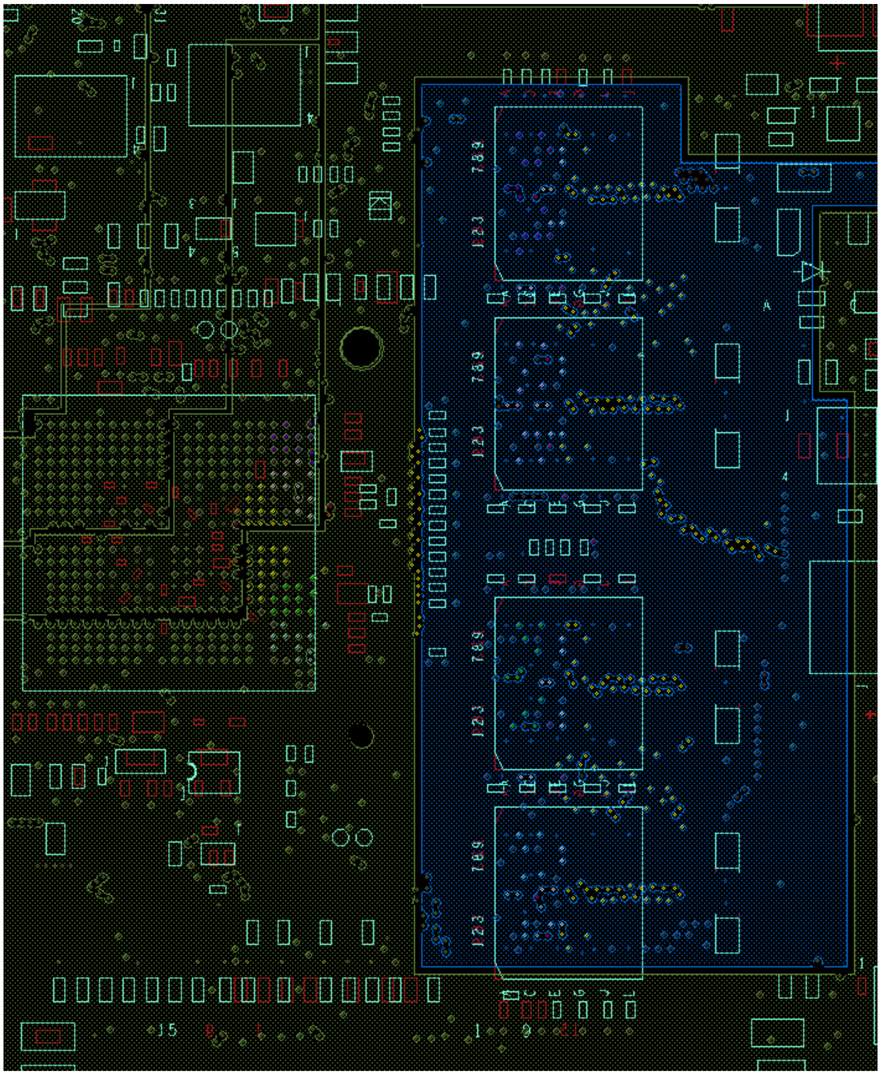
**Figure 2-22. Top 8-DDR3 Routing**



**Color Legend**

| Color | Meaning | Color | Meaning |
|---|---|---|---|
| Yellow | ADD/CMD/CTRL Signals | White | Clocks |
| Soft Blue | Data Byte Group 3 | Blue | DDR_1.5V |
| Soft Pink | Data Byte Group 0 | Pink | DDR_VREF |
| Green | Data Byte Group 2 | Orange | DDRQ_1.5V |
| Purple | Data Byte Group 1 | White | Clocks |

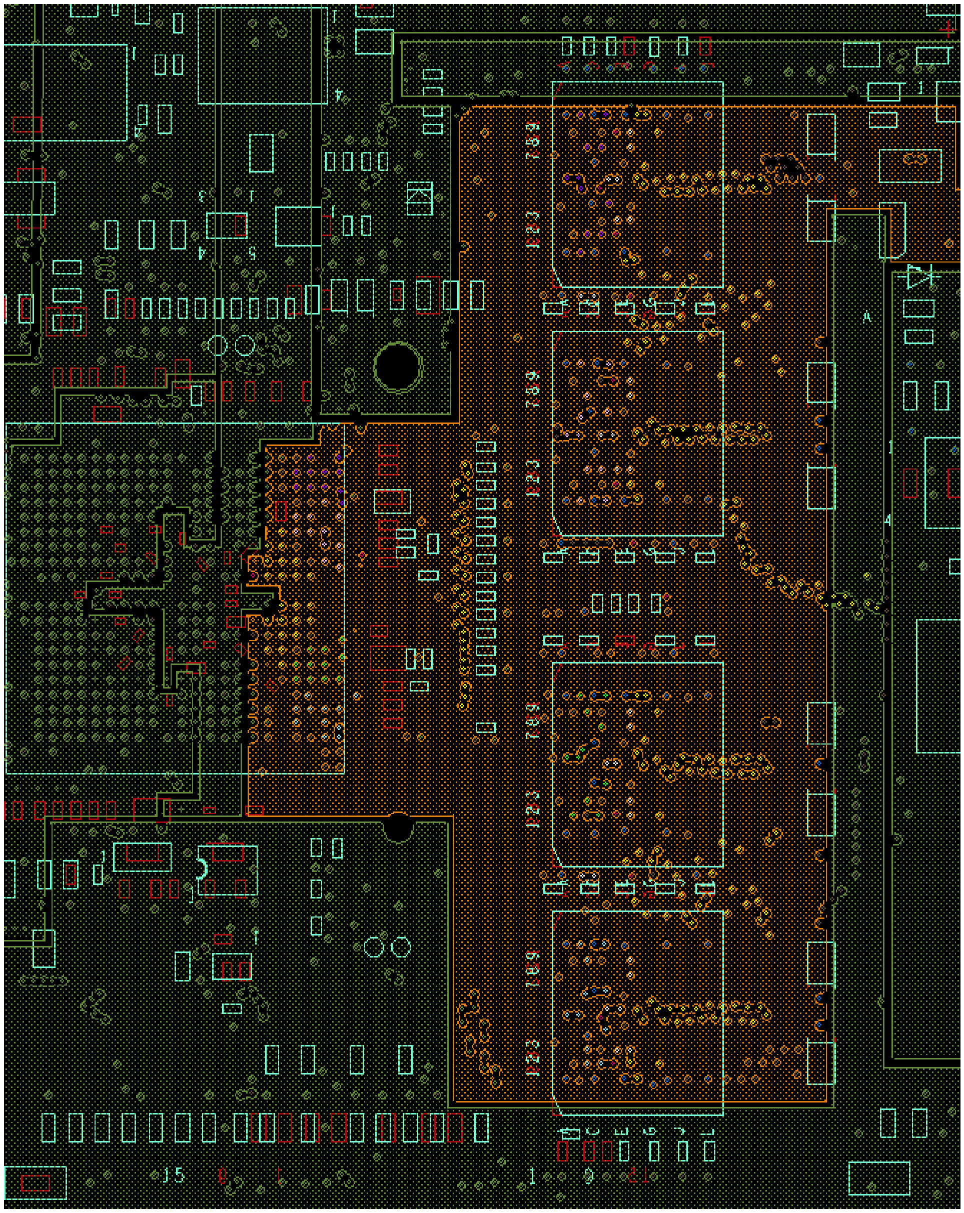
**Figure 2-23. Internal 1 8-DDR3 Routing**



**Color Legend**

| Color | Meaning | Color | Meaning |
|---|---|---|---|
| Yellow | ADD/CMD/CTRL Signals | White | Clocks |
| Soft Blue | Data Byte Group 3 | Blue | DDR_1.5V |
| Soft Pink | Data Byte Group 0 | Pink | DDR_VREF |
| Green | Data Byte Group 2 | Orange | DDRQ_1.5V |
| Purple | Data Byte Group 1 | White | Clocks |

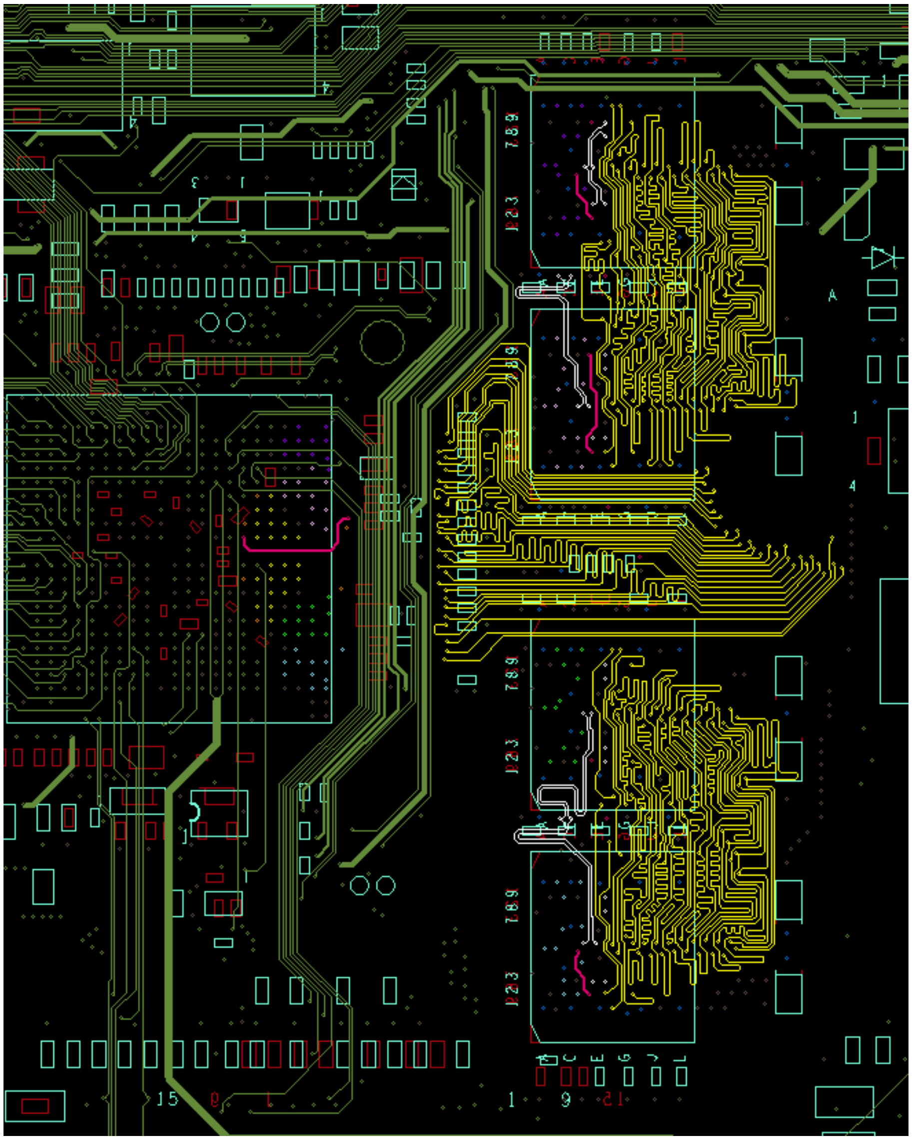
**Figure 2-24. Power Plane 1 8-DDR3 Routing**



**Color Legend**

| Color | Meaning | Color | Meaning |
|---|---|---|---|
| Yellow | ADD/CMD/CTRL Signals | White | Clocks |
| Soft Blue | Data Byte Group 3 | Blue | DDR_1.5V |
| Soft Pink | Data Byte Group 0 | Pink | DDR_VREF |
| Green | Data Byte Group 2 | Orange | DDRQ_1.5V |
| Purple | Data Byte Group 1 | White | Clocks |

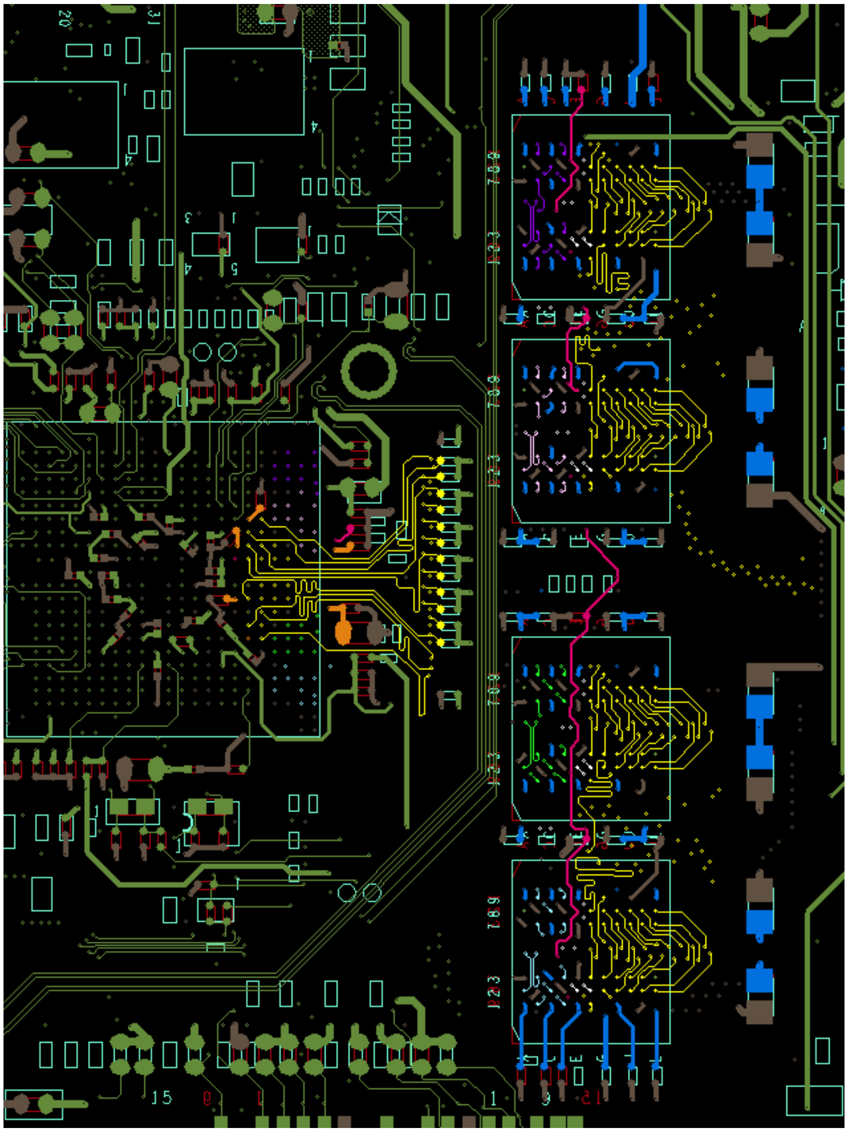
**Figure 2-25. Power Plane 2 8-DDR3 Routing**



**Color Legend**

| Color | Meaning | Color | Meaning |
|---|---|---|---|
| Yellow | ADD/CMD/CTRL Signals | White | Clocks |
| Soft Blue | Data Byte Group 3 | Blue | DDR_1.5V |
| Soft Pink | Data Byte Group 0 | Pink | DDR_VREF |
| Green | Data Byte Group 2 | Orange | DDRQ_1.5V |
| Purple | Data Byte Group 1 | White | Clocks |

**Figure 2-26. Internal 2 8-DDR3 Routing**



**Color Legend**

| Color | Meaning | Color | Meaning |
|---|---|---|---|
| Yellow | ADD/CMD/CTRL Signals | White | Clocks |
| Soft Blue | Data Byte Group 3 | Blue | DDR_1.5V |
| Soft Pink | Data Byte Group 0 | Pink | DDR_VREF |
| Green | Data Byte Group 2 | Orange | DDRQ_1.5V |
| Purple | Data Byte Group 1 | White | Clocks |

**Figure 2-27. Bottom 8-DDR3 Routing**

Table 2-4 shows the total etch of the signals for the byte 0 and byte 1 groups.

**Table 2-4. Total Signal Etch (DDR3)**

| Signals | Length (Mils) |
|---|---|
| DRAM_D0 | 616.034 |
| DRAM_D1 | 612.886 |
| DRAM_D2 | 613.808 |
| DRAM_D3 | 612.701 |
| DRAM_D4 | 617.177 |
| DRAM_D5 | 614.486 |
| DRAM_D6 | 614.743 |
| DRAM_D7 | 613.145 |
| DRAM_DQM0 | 612.794 |
| DRAM_SDQS0 | 615.633 |
| DRAM_SDQS0_B | 614.36 |
| DRAM_D8 | 615.063 |
| DRAM_D9 | 611.525 |
| DRAM_D10 | 616.758 |
| DRAM_D11 | 614.928 |
| DRAM_D12 | 614.521 |
| DRAM_D13 | 612.822 |
| DRAM_D14 | 612.794 |
| DRAM_D15 | 614.369 |
| DRAM_DQM1 | 614.705 |
| DRAM_SDQS1 | 610.26 |
| DRAM_SDQS1_B | 617.892 |
| DRAM_SDCLK0 | 1172.235 |
| DRAM_SDCLK0_B | 1174.757 |
| DRAM_SDCLK1 | 1176.5 |
| DRAM_SDCLK1_B | 1175.963 |

## 2.6 Power Recommendations

The following recommendations apply to the VREF voltage reference plane.

- Use 30 mils trace between de coupling cap and destination.
- Maintain a 25 mils clearance from other nets.
- Isolate VREF and/or shield with ground.

- Decouple using distributed 0.01 μF and 0.1 μF capacitors by the regulator, controller, and devices.
- Place one 0.1 μF near the source of VREF, one near the VREF pin on the controller, and two between the controller and the devices.

The following recommendations apply to the VTT voltage reference plane.

- Place the VTT island on the component side layer at the end of the bus behind the DRAM devices.
- Use a wide-island trace for current capacity.
- Place VTT generator as close to termination resistors as possible to minimize impedance (inductance).
- Place one or two 0.1 μF decoupling capacitor by each termination RPACK on the VTT Island to minimize the noise on VTT. Other bulk (10–22 μF) decoupling is also recommended to be placed on the VTT Island.

## 2.7 TV Encoder Recommendations

Use the following recommendations for the TV encoder.

- For the TV/VGA interface, the IOR, IOG, and IOB signals must have 75-Ω imepedance.

## 2.8 SATA Recommendations

Use the following recommendations for the SATA.

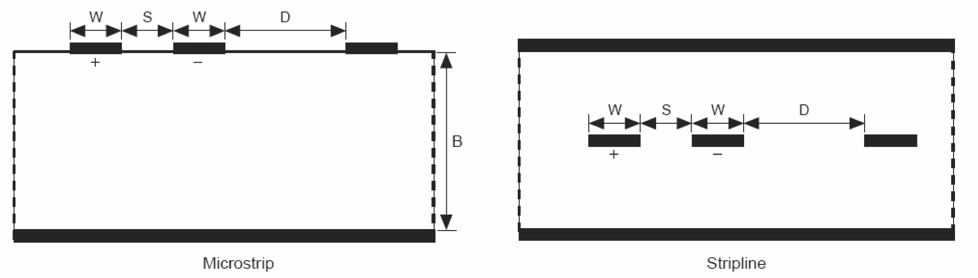
- SATA differential pairs should have a differential impedance of 100.
- Each differential pair should be length matched to ± 5 mils.
- Follow standard high-speed differential routing rules for signal integrity.

## 2.9 LVDS Recommendations

Use the following recommendations for the LVDS.

- Follow standard high-speed differential routing rules for signal integrity.
- Each differential pair should be length matched to ± 5 mils.

Figure 2-28 shows the dimensions of a stripline and microstrip pair. Figure 2-29 shows the differential pair routing.




For better coupling within a differential pair, make S < 2W, S < B, and D = 2S where:

W = width of a single trace in a differential pair
S = space between two traces of a differential pair
D = space between two adjacent differential pairs
B = thickness of the board

**Figure 2-28. Microstrip and Stripline Differential Pair Dimensions**

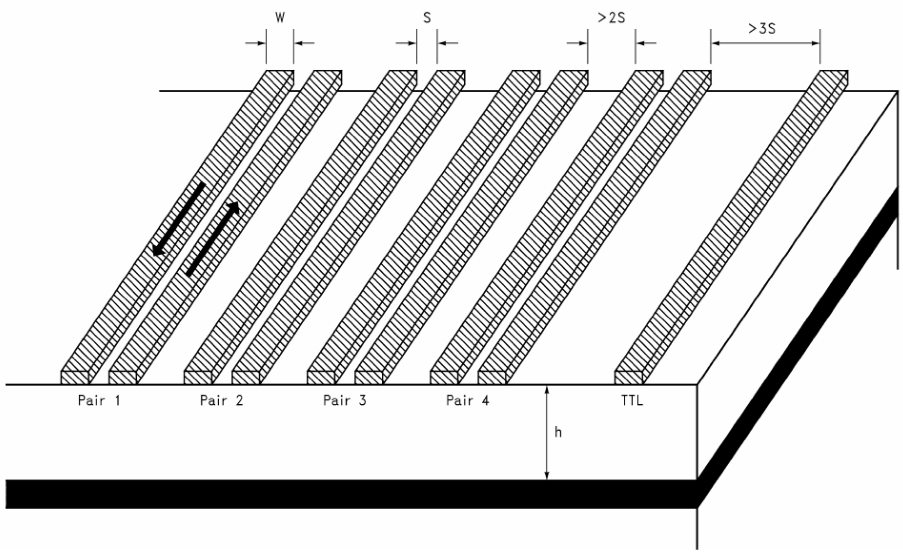



**Figure 2-29. Differential Pair Routing**

- The space between two adjacent differential pairs should be greater than or equal to twice the space between the two individual conductors.
- The skew between LVDS pairs should be within the minimum recommendation (± 100 mil).

## 2.10 Reference Resistors

Freescale reference designs have resistors that are used for reference in the interfaces. These resistors need to be in the following pins:

- USB_H1_RREFEXT

- USB_OTG_RREFEXT
- SATA_REXT
- LVDS_BG_RES
- TVDAC_VREF
- DRAM_CALIBRATION

Freescale recommends the use of a resistor of 1% tolerance or better with a connection that is made through a short trace. The resistance of the connecting trace should be as low as possible (< 1 Ω). The ground return should be short and direct to the ground plane.

**NOTE**

The reference resistor and the connection should be placed away from noisy regions. Noise induced on it may impact the internal circuit and degrade the interface signals.

## 2.11 ESD and Radiated Emissions Recommendations

The PCB design should use 6 or more layers, with solid power and ground planes and the recommendations for ESD immunity and radiated emissions performance are as follows:

- All components with ground chassis shields (USB jack, buttons, etc.) should connect the shield to the PCB chassis ground ring.
- Ferrite beads should be placed on each signal line connecting to an external cable. These ferrite beads must be placed as close to the PCB jack as possible.

**NOTE**

Ferrite beads should have a minimum impedance of 500 Ω at 100 MHz with the exception of the ferrite on USB_5V.

- Ferrite beads should NOT be placed on the USB D+/D– signal lines as this can cause USB signal integrity problems. For radiated emissions problems due to USB, a common mode choke may be placed on the D+/D– signal lines. However, in most cases, it should not be required if the PCB layout is satisfactory. Ideally, the common mode choke should be approved for high speed USB use or tested thoroughly to verify there are no signal integrity issues created.
- It is highly recommended that ESD protection devices be used on ports connecting to external connectors. Refer to the reference schematic (available on the Freescale website) for detailed information about ESD protection implementation on the USB and TV-E interfaces.

# Chapter 3
# Understanding the IBIS Model

This chapter explains how to use the IBIS (input output buffer information specification) model, which is an Electronic Industries Alliance standard for the electronic behavioral specifications of integrated circuit input/output analog characteristics. The model is generated in ACII text format and consists of multiple tables that capture current vs. voltage (IV) and voltage vs. time (VT) characteristics of the buffer. IBIS models are generally used to perform PCB-board-level signal integrity (SI) simulations and timing analyses.

The IBIS model's features are as follows:

- Supports fast chip-package-board simulation, with SPICE-level accuracy and faster than any transistor-level model
- Provides the following for portable model data
  - I/O buffers, series elements, terminators
  - Package RLC parasitics
  - Electrical board description

## 3.1 IBIS Structure and Content

An IBIS file contains the data required to model a component's input, output, and I/O buffers behaviorally in ASCII format. The basic IBIS file contains the following data:

- Header information regarding the file itself being modeled
- Information about the component, the package's electrical characteristics, and the pin-to-buffer model mapping (i.e., which pins are connected to which buffer models)
- The data required to model each unique input, output, and I/O buffer design on the component

IBIS models are component-centric, meaning they allow users to model an entire component rather than only a particular buffer. Therefore, in addition to the electrical characteristics of a component's buffers, an IBIS file includes the component's pin-to-buffer mapping and the electrical parameters of the component's package.

## 3.2 Header Information

The first section of an IBIS file provides the basic information about the file and its data. Table 3-1 explains the header information notation. Example 3-1 shows what header information looks like in an IBIS file.

**Table 3-1. Header Information**

| Keyword | Required | Description |
|---|---|---|
| [IBIS Ver] | Yes | Version of IBIS Specification this file uses. |
| [Comment char] | No | Change the comment character. Defaults to the pipe (\|) character |
| [File Name] | Yes | Name of this file. All file names must be lower case. The file name extension for an IBIS file is .ibs |
| [File Rev] | Yes | The revision level of this file. The specification contains guidelines for assigning revision levels. |
| [Date] | No | Date this file was created |
| [Source] | No | The source of the data in this file. Is it from a data book? Simulation data? Measurement? |
| [Notes] | No | Component or file-specific notes. |
| [Disclaimer] | No | May be legally required |
| [Copyright] | No | The file's copyright notice |

**Example 3-1. Header Information**

```
[IBIS Ver]        4.0
[Comment Char]    |_char
[File Name]       imx53_19x19_to2_ver03.ibs
[File Rev]        03
[Date]            Wed Nov 24 13:15:11 IST 2010
[Source]
[Notes]           This model passed check w/t IBISCHK5 V5.0.3 utility.
                  No errors were reported.
                  8 warnings were reported: 4 of them relate to MIN DC
                  endpoints, 2 relate to MIN/MAX VI curves, 2 relate to
                  single model defined / model data reuse.
                  All these warnings can be waived.
```

## 3.3 Component and Pin Information

The second section of an IBIS file is where the data book information regarding the component's pinout, pin-to-buffer mapping, and the package and pin electrical parameters is placed. Table 3-2 explains the component and pin information notation, and Example 3-2 shows what it looks like in an IBIS file.

**Table 3-2. Component and Pin Information**

| Keyword | Required | Comment |
|---|---|---|
| [Component] | Yes | The name of the component being modeled. Standard practice has been to use the industry standard part designation. Note that IBIS files may contain multiple [Component] descriptions. |

**Table 3-2. Component and Pin Information (continued)**

| Keyword | Required | Comment |
|---|---|---|
| [Manufacturer] | Yes | The name of the component manufacturer |
| [Package] | Yes | This keyword contains the range (minimum, typical and maximum values) over which the packages' lead resistance, inductance, and capacitance vary (the R_pkg, L_pkg, and C_pkg parameters). |
| [Pin] | Yes | This keyword contains the pin-to-buffer mapping information. In addition, the model creator can use this keyword to list the R, L, and C data for each individual pin (R_pin, L_pin, and C_pin parameters). |
| [Package Model] | No | If the component model includes an external package model (or uses the [Define Package Model] keyword within the IBIS file itself), this keyword indicates the name of that package model. |
| [Pin Mapping] | No | This keyword is used if the model creator wishes to include information on buffer power and ground connections. This information may be used for simulations involving multiple outputs switching. |
| [Diff Pin] | No | This keyword is used to associate buffers that should be driven in a complementary fashion as a differential pair. |

**Example 3-2. Component and Pin Information**

```
[Component]        iMX53
[Manufacturer]     FREESCALE
[Package]
| variable      typ                 min                 max
R_pkg           0.2406012           0.0866665           0.3239850
L_pkg           3.72366nH           1.35844nH           6.31610nH
C_pkg           1.08882pF           0.40570pF           1.70556pF
|
[Pin] signal_name          model_name           R_pin     L_pin     C_pin
A1    GND                  GND                  NA        NA        NA
A2    GND                  GND                  NA        NA        NA
A3        GPIO_17        uhvio        0.314809        5.06836nH        1.34352pF
A4        GPIO_7        uhvio        0.306893        5.63292nH        1.11136pF
A5        GPIO_5        uhvio        0.295459        4.67265nH        1.18268pF
Model Information
The [Model Selector] keyword is a simple means by which several buffers can be made optionally
available for simulation at the same physical pin of the component.
[Pin] signal_name          model_name           R_pin     L_pin     C_pin
AA4   JTAG_TDI             gpio                 0.307112  5.51128nH 0.40126pF
…
[Model Selector] gpio
gpio_iods0hvf       GPIO, 2.7V, Low Drive, Fast SR
gpio_iods0hvs       GPIO, 2.7V, Low Drive, Slow SR
```

## 3.4 Model Information

The [Model Selector] keyword provides a simple means by which several buffers can be made optionally available for simulation at the same physical pin of the component, as shown in Example 3-3.

**Example 3-3. Model Selector Keyword Example**

```
[Pin] signal_name          model_name          R_pin     L_pin     C_pin
AA4   JTAG_TDI             gpio                0.307112  5.51128nH 0.40126pF
…
[Model Selector] gpio
gpio_iods0hvf       GPIO, 2.7V, Low Drive, Fast SR
gpio_iods0hvs       GPIO, 2.7V, Low Drive, Slow SR
……
```

The [Model] keyword starts the description of the data for a particular buffer. While a buffer model can appear quite complex, most buffers can be described using the following parameters and keywords:

- Parameters section
- Reference information
- IV keywords (see Figure 3-1)
- Vt keywords

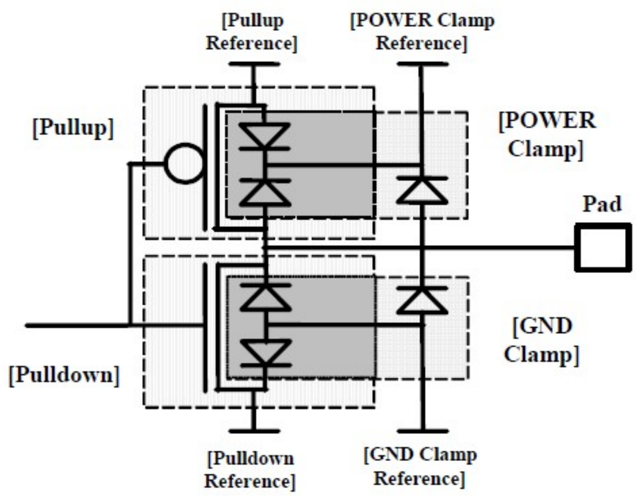



**Figure 3-1. Model IV Keywords' Structure**

### 3.4.1 Ramp and Waveform Keywords

Table 3-3 defines the keywords that provide the information about an output or I/O buffer, and Example 3-4 shows what they look like in an IBIS file.

**Table 3-3. Ramp and Waveform Keywords**

| Keyword | Required | Comment |
|---|---|---|
| [Ramp] | Yes | Basic ramp rate information, given as a dV/dt_r for rising edges and dV/dt_f for falling edges, see Equation 3-1. |
| [Rising Waveform] | No | The actual rising (low to high transition) waveform, provided as a VT table. |
| [Falling Waveform] | No | The actual falling (high to low transition) waveform, provided as a VT table. |

$$\frac{dV}{dt} = \frac{20\ \%\ \text{to}\ 80\%\ \text{voltage swing}}{\text{time taken to swing above voltage}}$$ ***Eqn. 3-1***

**NOTE**

The dV value is the 20% to 80% voltage swing of the buffer when driving into the specified load, R_load (for [Ramp], this load defaults to 50). For CMOS drivers or I/O buffers, this load is assumed to be connected to the voltages defined by the [Voltage Range] keyword for falling edges and to ground for rising edges.

**Example 3-4. Ramp and Waveform Keywords**

```
An example of ramp and waveform table from MX53 IBIS:
[Ramp]
| variable       typ                 min                 max
dV/dt_r  0.4717/0.2207n      0.4714/0.3252n      0.5647/0.1323n
dV/dt_f  0.4676/0.1382n      0.4749/0.2495n      0.6009/0.1270n
R_load = 50.0000
|
[Rising Waveform]
R_fixture= 50.0000
V_fixture= 0.0
V_fixture_min= 0.0
V_fixture_max= 0.0
|time     V(typ)              V(min)              V(max)
|
0.0S              67.3645uV          17.6030uV           38.0931uV
79.7191pS         67.3645uV          17.6030uV           38.0931uV
0.3871nS          67.3645uV          17.6030uV           38.0931uV
0.4395nS          67.3645uV          17.6030uV           38.0931uV
0.5059nS          67.3645uV          17.6030uV           38.0931uV
```

The [Ramp] keyword is always required, even if the [Rising Waveform] and [Falling Waveform] keywords are used. However, the VT tables under [Rising Waveform] and [Falling Waveform] are generally preferred to [Ramp] for the following reasons:

- VT data may be provided under a variety of loads and termination voltages
- VT tables may be used to describe transition data for devices as they turn on and turn off.

- [Ramp] effectively averages the transitions of the device, without providing any details on the shapes of the transitions themselves. All detail of the transition ledges would be lost.

The VT data should be included under two [Rising Waveform] and two [Falling Waveform] sections, each containing data tables for a Vcc-connected load and a Ground-connected load (although other loading combinations are permitted).

The most appropriate load is a resistive value corresponding to the impedance of the system transmission lines the buffer will drive (own impedance). For example, a buffer intended for use in a 60 Ω system is best modeled using a 60 Ω load (R_fixture).

Figure 3-2 shows how to interpret the model data.

- I_down [GND clamp] + [Power clamp] + [Pulldown]
- I_up [GND clamp] + [Power clamp] + [Pullup]
- I_recvr [GND clamp] + [Power clamp]

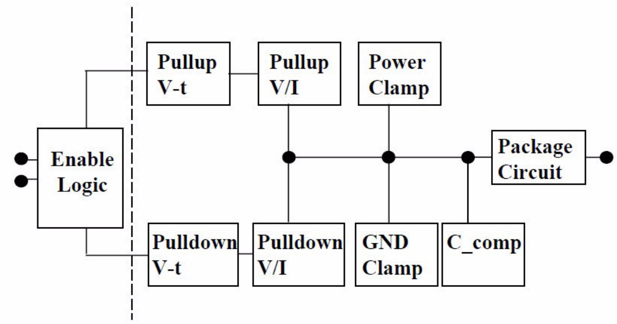



**Figure 3-2. Model Data Interpretation**

## 3.5 Model Golden Waveforms

Golden waveforms are a set of waveforms simulated using known ideal test loads. They are useful for verifying the accuracy of behavioral simulation results against the transistor level circuit model from which the IBIS model parameters originated.

Figure 3-3 shows a generic test load network.

```
|                                   V_term1
|                                o-----------o
|                                |           |
|                                \           \    receiver_model_name
|   ______                       /           /                         ______
|  |      |  NEAR       Rp1_near \           \ Rp1_far          FAR  |      |
|  | |\   |                      /           /                       | |\   |
|  | | \  |   Rs_near  Ls_near   |   _____   |     Ls_far  Rs_far    | | \  |
|  | |  >-|---o--/\/\--@@@@--o----o--O_____)--o----o--@@@@--/\/\--o---|-|  > |
|  | | /  |   |              |   |   Td      |    |              |   | | /  |
|  | |/   |   | C1_near      |   \   Zo      \    | C2_far       |   | |/   |
|  |______|  ===            ===  /           /   ===            ===  |______|
|             |      C2_near |   \           \    |       C1_far |
|             |              |   /           /    |              |
|             |              |   |  V_term2  |    |              |
|             o--------------o   o-----------o    o--------------o
|             |               Rp2_near    Rp2_far                |
```

**Figure 3-3. Generic Test Load Network**

Table 3-4 explains the golden waveform keywords.

**Table 3-4. Golden Waveform Keywords**

| Keyword | Required | Comment |
|---|---|---|
| [Test Data] | No | • Provides a set of golden waveforms and references the conditions under which they were derived.<br>• Useful for verifying the accuracy of behavioral simulation results against the transistor level circuit model from which the IBIS model parameters originated. |
| [Test Load] | Yes | • Defines a test load network and its associated electrical parameters for reference by golden waveforms under the [Test Data] keyword.<br>• If Test_load_type is Differential, the test load is a pair of the above circuits. If the R_diff_near or R_diff_far subparameter is used, a resistor is connected between the near or far nodes of the two circuits.<br>• If Test_load_type is Single_ended, R_diff_near and R_diff_far are ignored. |

## 3.6 Naming Conventions for Model Names and Usage in i.MX53 IBIS File

The model names are defined per each [Model selector]. The models may differ from each other by having different parameters—such as voltage, drive strength, mode of operation, and slew rate. The mode of operation, drive strength, and slew rate parameters are programmable by software (see Section 3.6.1, “[Model Selector] ddr,”–Section 3.6.5, “List of Pins Not Modeled in the i.MX53 IBIS File,” for further details).

### 3.6.1 [Model Selector] ddr

This model has the following parameters: voltage, mode of operation, drive strength, ODT enable/disable.

Mode of operation — Controlled by the IOMUXC_SW_PAD_CTL_GRP_DDR_TYPE[26:25] register in IOMUXC (IOMUX controller) DDR_SEL bits.

Drive strength — Controlled by bits 21–19 (DSE) of the following registers in IOMUXC (IOMUX controller): IOMUXC_SW_PAD_CTL_GRP_ADDDS, IOMUXC_SW_PAD_CTL_GRP_B0DS, IOMUXC_SW_PAD_CTL_GRP_B1DS, IOMUXC_SW_PAD_CTL_GRP_CTLDS, IOMUXC_SW_PAD_CTL_GRP_B2DS, IOMUXC_SW_PAD_CTL_GRP_B3DS, IOMUXC_SW_PAD_CTL_PAD_DRAM_SDODT0, IOMUXC_SW_PAD_CTL_PAD_DRAM_SDODT

ODT enable — Controlled by bits [18:16], [14:12], [10:8], and [6:4] in ODTCTRL in ESDCTL.

**Example 3-5. [Model Selector] ddr in IBIS File**

```
ddr2hs_sel00_ds111_mio        DDR, 1.8V, ddr2 mode, 43 Ohm driver impedance
ddr2hs_sel00_ds110_mio        DDR, 1.8V, ddr2 mode, 50 Ohm driver impedance
ddr2hs_sel00_ds101_mio        DDR, 1.8V, ddr2 mode, 60 Ohm driver impedance
```

Refer to the register description in the IOMUXC chapter in the i.MX53 reference manual for further details about this model.

### 3.6.2 [Model Selector] gpio

This model has the following parameters: voltage, drive strength, slew rate.

Drive strength — Controlled by the DSE bits (bits 2–1) in the IOMUXC_SW_PAD_CTL_PAD_<pad name>.

Slew rate — Controlled by the SRE bit (bit 0) in the IOMUXC_SW_PAD_CTL_PAD_<pad name>.

**Example 3-6. [Model Selector] gpio in IBIS File**

```
[Model Selector] gpio
gpio_iods0hvf           GPIO, 2.775V, Low Drive, Fast SR
gpio_iods0hvs           GPIO, 2.775V, Low Drive, Slow SR
gpio_iods0lvf           GPIO, 1.875V, Low Drive, Fast SR
gpio_iods0lvs           GPIO, 1.875V, Low Drive, Slow SR
```

Refer to the register description in the IOMUXC chapter in the i.MX53 reference manual for further details about this model.

### 3.6.3 [Model Selector] lvio

This model has no controllable parameters. Its associated pins are used as input only (for boot, reset) and cannot be configured.

The listed drive strength and slew rate options in the IBIS file have no meaning.

### 3.6.4 [Model Selector] uhvio

This model has the following parameters: voltage, drive strength, slew rate.

Drive strength — Controlled by DSE bits (bits 2-1) in the IOMUXC_SW_PAD_CTL_PAD_<pad name> in IOMUXC chapter that matches the pin name.

Voltage — The pin needs to be configured to match the voltage level that is supplied to it. There is an automatic voltage detection for these pins, but it is recommended to use the manual settings.

The voltage parameter is controlled by bit 18 (HVEOVERWRITE) and bit 17 (VDOEN) in the following registers in IOMUXC:

- IOMUXC_SW_PAD_CTL_PAD_NVCC_SD1
- IOMUXC_SW_PAD_CTL_PAD_NVCC_SD2
- IOMUXC_SW_PAD_CTL_PAD_NVCC_GPIO
- IOMUXC_SW_PAD_CTL_PAD_NVCC_PATA__0
- IOMUXC_SW_PAD_CTL_PAD_NVCC_PATA__2
- IOMUXC_SW_PAD_CTL_PAD_NVCC_FEC
- IOMUXC_SW_PAD_CTL_PAD_NVCC_NANDF
- IOMUXC_SW_PAD_CTL_PAD_NVCC_EIM__7
- IOMUXC_SW_PAD_CTL_PAD_NVCC_EIM__4
- IOMUXC_SW_PAD_CTL_PAD_NVCC_EIM__1
- IOMUXC_SW_PAD_CTL_PAD_NVCC_CSI__0
- IOMUXC_SW_PAD_CTL_PAD_NVCC_KEYPAD

**Example 3-7. [Model Selector] uhvio in IBIS File**

```
[Model Selector] uhvio
uhvio_iods0hvf      UHVIO, 3.3V, Low Drive
uhvio_iods0lvf      UHVIO, 1.875V, Low Drive
uhvio_iods1hvf      UHVIO, 3.3V, Medium Drive
uhvio_iods1lvf      UHVIO, 1.875V, Medium Drive
```

Refer to the register description in the IOMUXC chapter in the i.MX53 reference manual for further details about this model.

### 3.6.5 List of Pins Not Modeled in the i.MX53 IBIS File

Table 3-5 provides a list of analog or special interface pins that are not modeled in the i.MX53 IBIS file:

**Table 3-5. Unmodeled Analog or Special Interface Pins**

| Pin Name | |
|---|---|
| CKIL | TVDAC_COMP |
| ECKIL | TVDAC_IOB |
| EXTAL | TVDAC_IOG |
| XTAL | TVDAC_IOR |
| CKIH1 | TVDAC_VREF |
| CKIH2 | USB_H1_GPANAIO |
| DRAM_CALIBRATION | USB_H1_RREFEXT |
| FASTR_ANA | USB_H1_VBUS |
| FASTR_DIG | USB_OTG_GPANAIO |
| LVDS_BG_RES | USB_OTG_ID |
| TVCDC_IOB_BACK | USB_OTG_RREFEXT |
| TVCDC_IOG_BACK | USB_OTG_VBUS |
| TVCDC_IOR_BACK | SATA_REXT |

Table 3-6 provides a list of differential signals that are not represented in the current IBIS file. These signals require special treatment to be considered during PCB design. Complementary signals are shown in the same row.

**Table 3-6. Unmodeled Differential Signals**

| Differential Signal Name | |
|---|---|
| SATA_TXM | SATA_TXP |
| SATA_RXM | SATA_RXP |
| SATA_REFCLKM | SATA_REFCLKP |
| USB_H1_DN | USB_H1_DP |
| USB_OTG_DN | USB_OTG_DP |

## 3.7 Quality Assurance for the IBIS Models

The IBIS models are validated against the IBIS specification, which provides a way to objectively measure the correlation of model simulation results with reference transistor-level spice simulation or measurements.

Correlation — The process of making a quantitative comparison between two sets of I/O buffer characterization data, e.g. lab measurement vs. structural simulation or behavioral simulation vs. Structural simulation.

Correlation Level — A means for categorizing I/O buffer characterization data based on how much the modeling engineer knows about the processing conditions of a sample component and which correlation metric he or she used.

All models (GPIO, LPDDR2, UHVIO, LVDS, LVIO) have passed the following checks:

- Passes IBISCHK without errors or unexplained warnings
- Data for basic simulation checked
- Data for timing analysis checked
- Data for power analysis checked
- Correlated against Spice simulations

Validation reports can be provided upon demand.

## 3.8 References

Consult the following references for more information about the IBIS model.

- IBIS Open Forum (http://www.eda.org/ibis/)

  The IBIS Open Forum consists of EDA vendors, computer manufacturers, semiconductor vendors, universities, and end-users. It proposes updates and reviews, revises standards, and organizes summits. It promotes IBIS models and provides useful documentation and tools.
- IBIS specification (http://eda.org/pub/ibis/ver5.0/)

# Chapter 4
# Using the IOMUX Design Aid

This chapter explains how to use of the IOMUX system design aid (IOMux.exe), which facilitates the assignment of internal signals to external device balls/pins. The IOMUX design aid performs the following functions:

- Makes signal assignments for the supported i.MX device
- Identifies conflicts, allowing them to be resolved in real-time
- Allows notes or comments for each signal to be added to the list of assignments

Design configurations can be saved for future use and exported for use in schematics or software source code as supplementary documentation of  a system.

## 4.1 Application Requirements

The IOMUX application requires that the following be installed:

- Microsoft Windows XP or newer
- Microsoft's .NET Framework, version 3.5 or newer
- .NET 3.5 SP1

The IOMUX application currently supports the following devices in all available package variations:

- i.MX233
- i.MX25
- i.MX28
- i.MX35
- i.MX51
- i.MX53

## 4.2 IOMUX Tool Walkthrough

This section provides a brief walkthrough of the IOMUX tool. Users who are new to this tool may wish to read Section 4.5, "IOMUX Features Guide," prior to starting the walkthrough. The i.MX35 example shown is for illustration only. The steps in the process apply to all i.MX products in the preceding bullet list.

### 4.2.1 Identifying Signal Conflicts with the Signal Selection Pane

Figure 4-1 shows the application window after the following sequence:

1. Launch IOMux.exe application.
2. Select Device > i.MX35 TO2.1.
3. Check all UARTS (UART1, UART2, and UART3).
4. Expand the signals UART3_RXD_MUX and UART3_TXD_MUX.

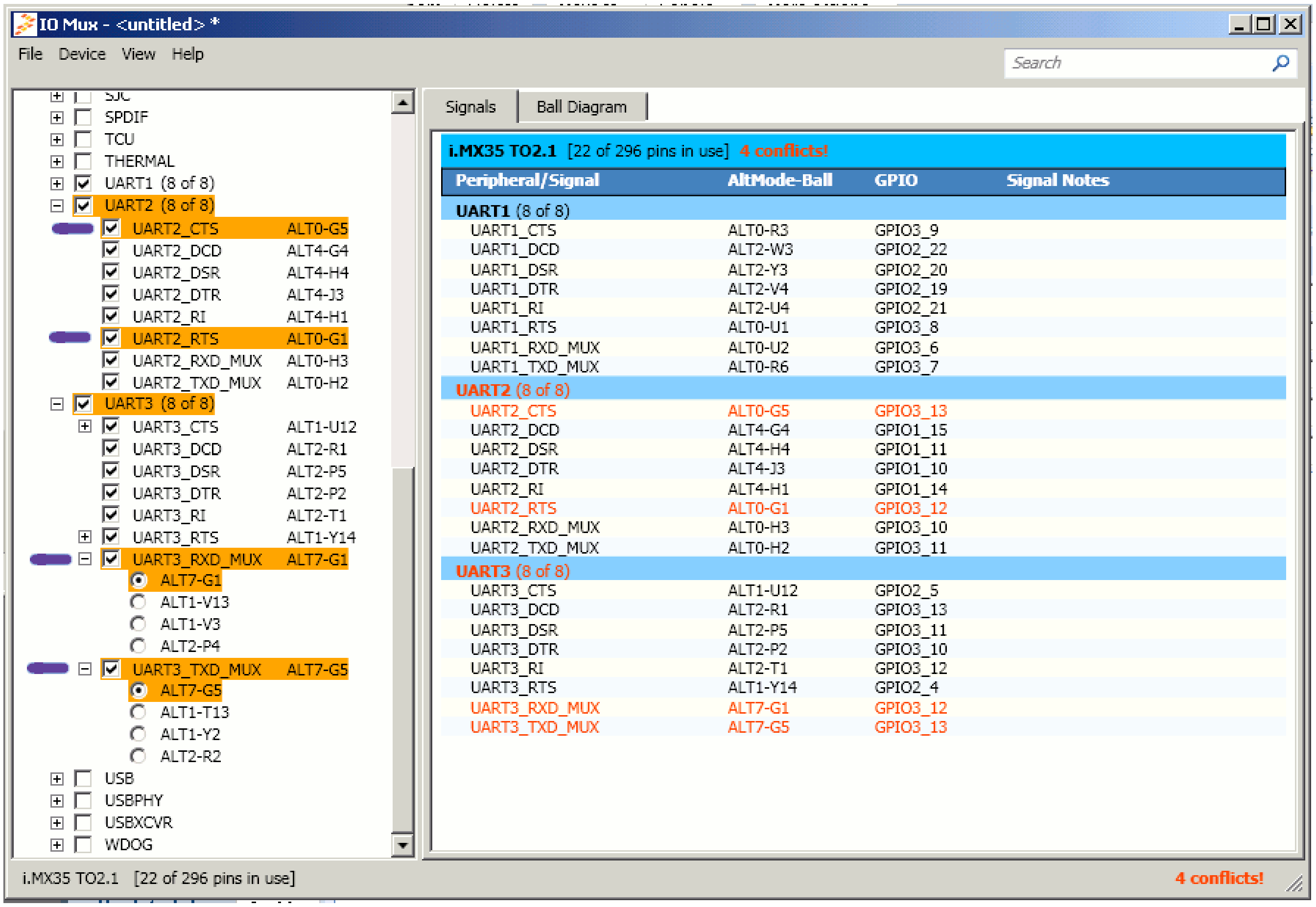



**Figure 4-1. Application Window after Expanding UARTs 2 and 3 of i.MX35 TO2.1**

The orange highlighting in the signal selection pane indicates conflicts between signals. This example shows four specific conflicts (marked by purple lines in the figure). The conflicts were detected immediately after selecting the second UART because the auto-detect conflicts option is enabled by default when the application starts up. Peripheral groups expand to show the conflicts at the signal level once they are detected. The status bar shows the exact number of conflicting signals in the lower right corner of the application window.

To see what other signals share the assigned ball for a signal, hover the mouse over a signal. Figure 4-2 illustrates this, using the UART3_TXD_MUX signal as an example. Note that UART2_CTS, which is one of the four conflicts highlighted in orange, is also bolded in the UART3_TXD_MUX pop-up list. The bolding indicates that UART2_CTS conflicts with UART3_TXD_MUX. All conflicts on a ball are bolded so that the user knows the conflicts exist.

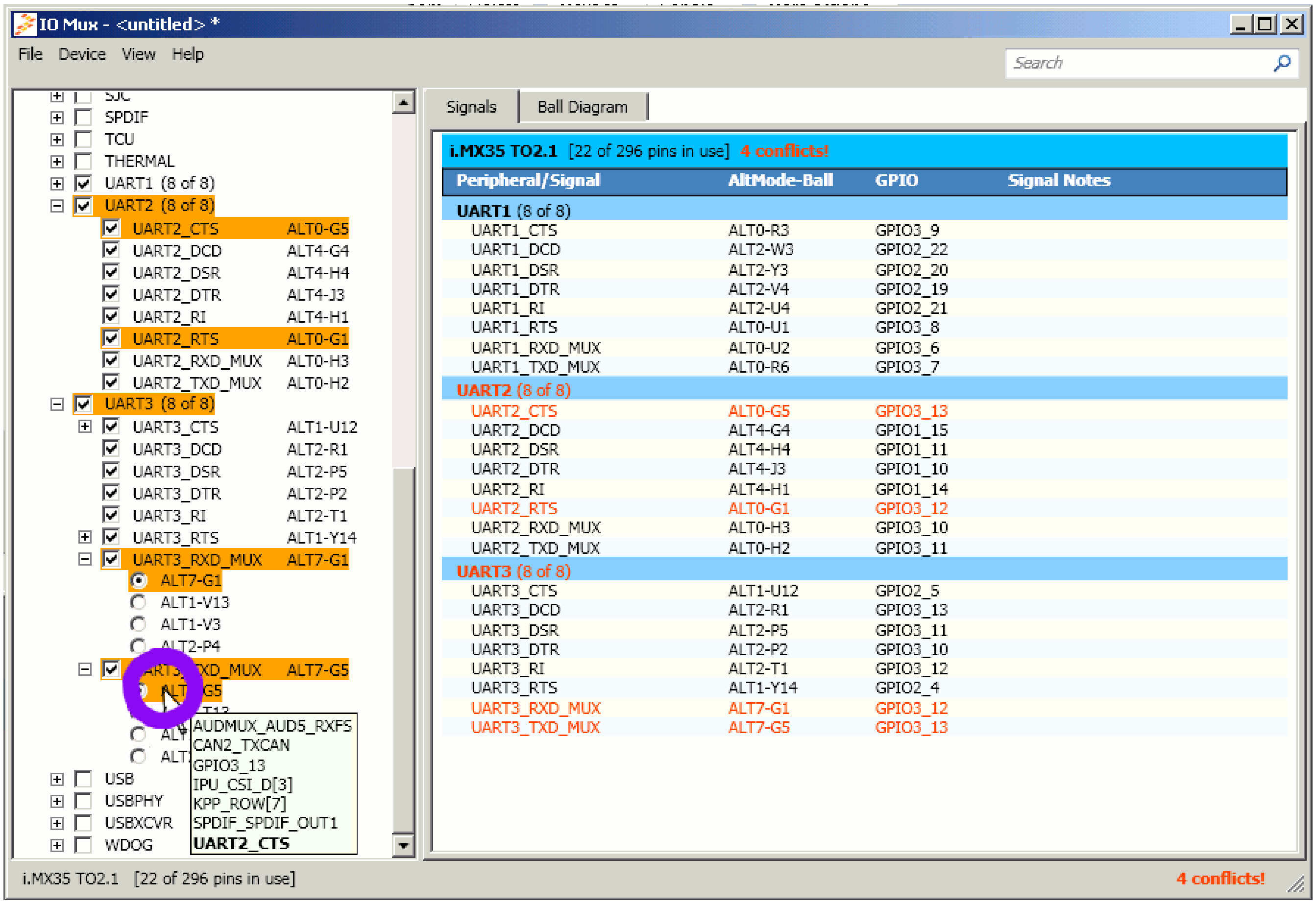



**Figure 4-2. Hovering Mouse over a Signal to Show Other Signals Sharing that Ball/Pin**

Hovering the mouse over the same signal, UART3_TXD_MUX, in the signals tab on the right-hand side of the screen is another way to show a list of signals shared on a given ball or pin (see Figure 4-3). Note that unlike in the signal selection pane, the list in the signals tab does not indicate conflicts.

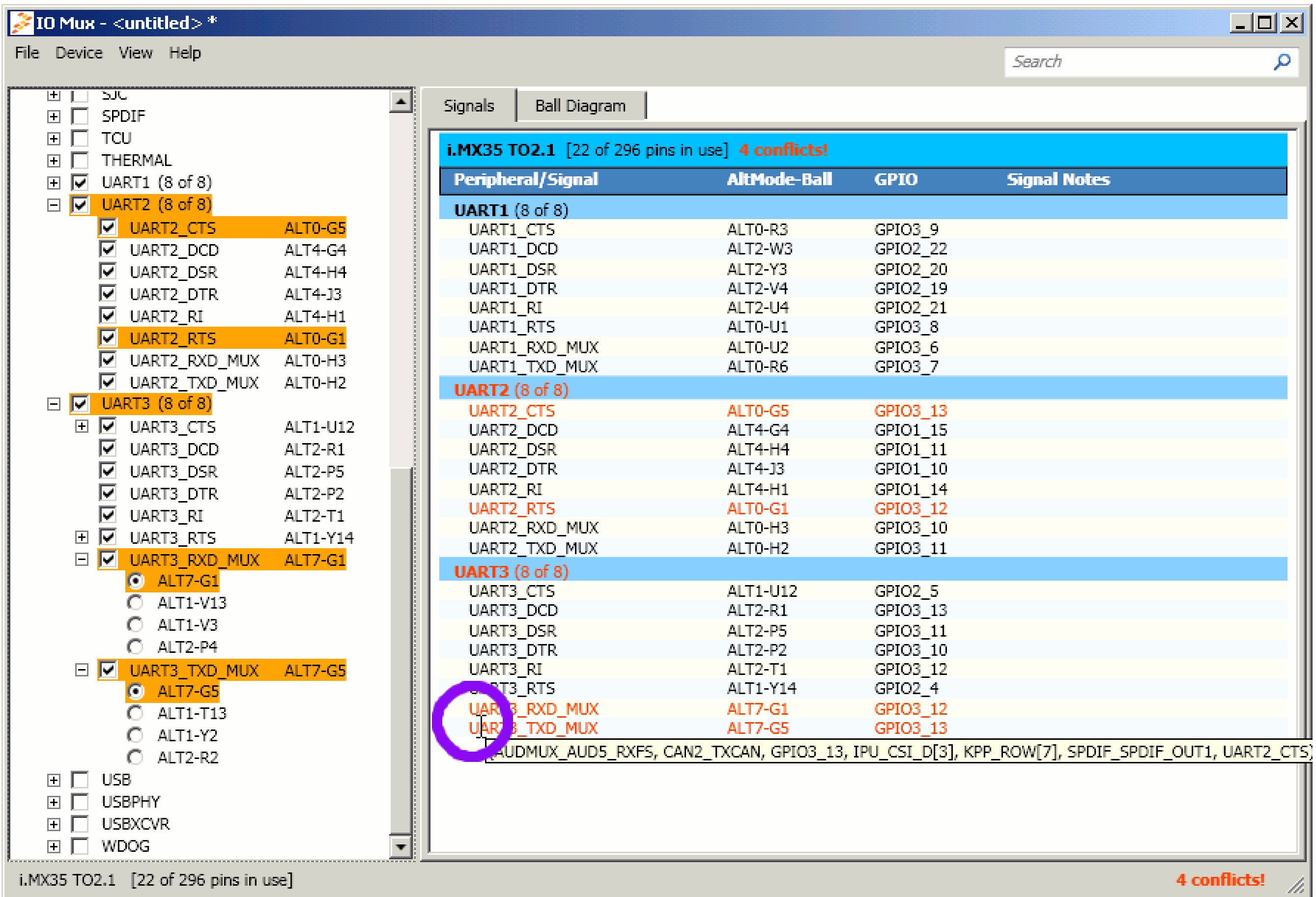


Figure 4-3. Hovering Mouse over a Signal in the Signals Tab

Conflicts are resolved by reassigning signals. However, before reassigning signals, it is important to check for potential conflicts. This is done by expanding signals in the signal selection pane. Figure 4-4 demonstrates this by expanding the signal UART3_RTS. Notice that the ball option U4 for UART3_RTS is highlighted in yellow, which indicates that selecting U4 for this signal would conflict with a previously assigned signal on that ball (which is not shown in the figure).

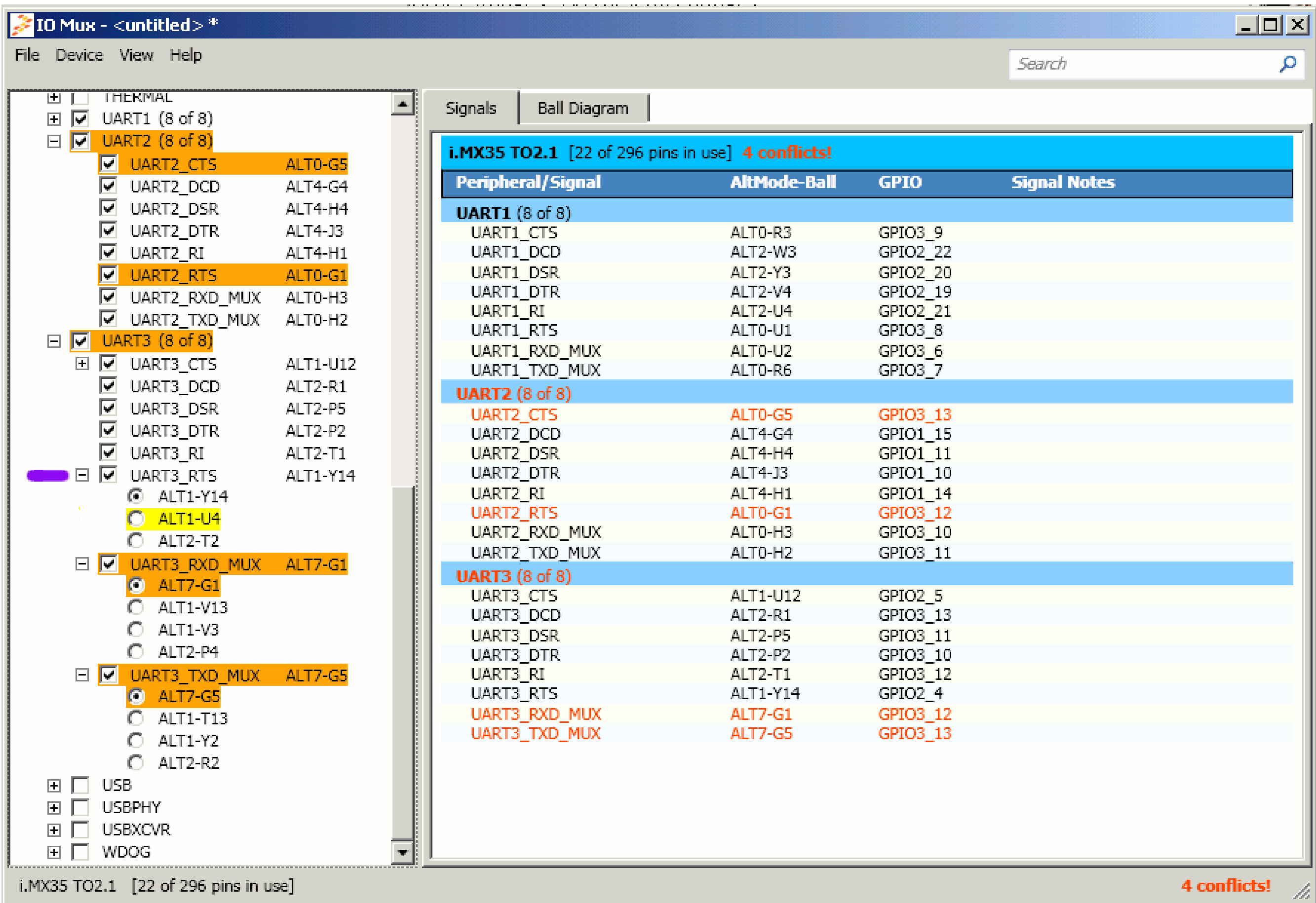



**Figure 4-4. Expanding UART3_RTS to Show Potential Conflicts**

In this example, the conflicts are resolved by assigning V13 to UART3_RXD_MUX and T13 to UART3_TXD, as shown in Figure 4-5. Notice that the original ball assignments for these two signals are now highlighted in yellow, which indicates that conflicts exist if the signals are assigned to those balls. All the conflicts in our example have now been resolved.

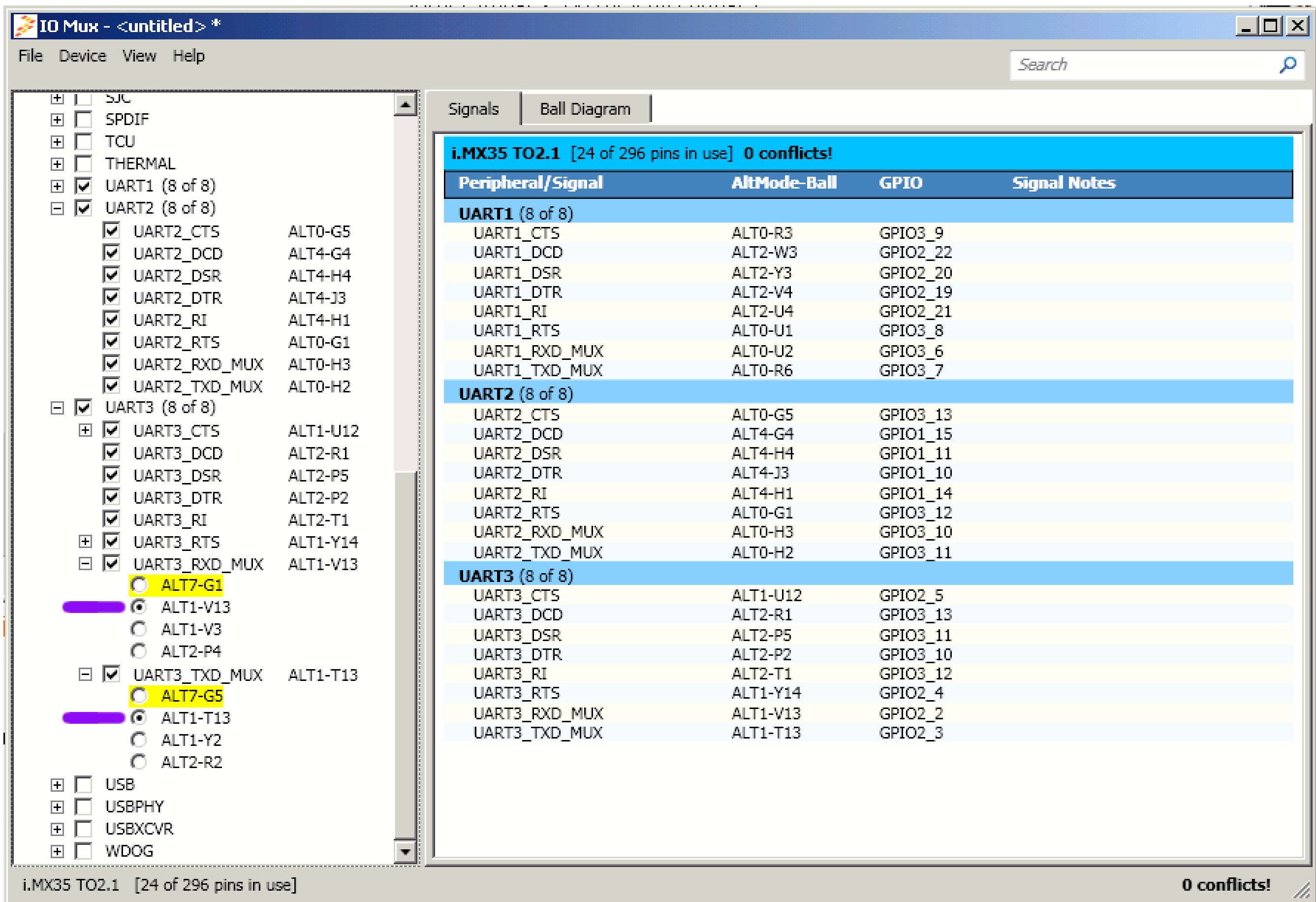



**Figure 4-5. Resolving Conflicts by Changing Ball Assignments**

**NOTE**

In an actual design, several iterations of re-assigning balls for signals may be required to resolve all the conflicts for chosen signals. In this example, all signals for each peripheral group were selected by checking the box next to the peripheral. In actual usage, only a subset of all the signals for a peripheral are selected.

## 4.2.2 Adding Comments with the Signals Tab

Adding information to the pin assignments allows users to explain why selections are made, what the pins connect to, and how they are intended to be used. Comments are added in the signals tab.

Figure 4-6 shows the signals tab after right-clicking on the UART2_TXD_MUX row. This brought up the contextual menu for entering a comment. Clicking on the menu brings up a text entry field where the user may enter text, which is circled in purple in the example.

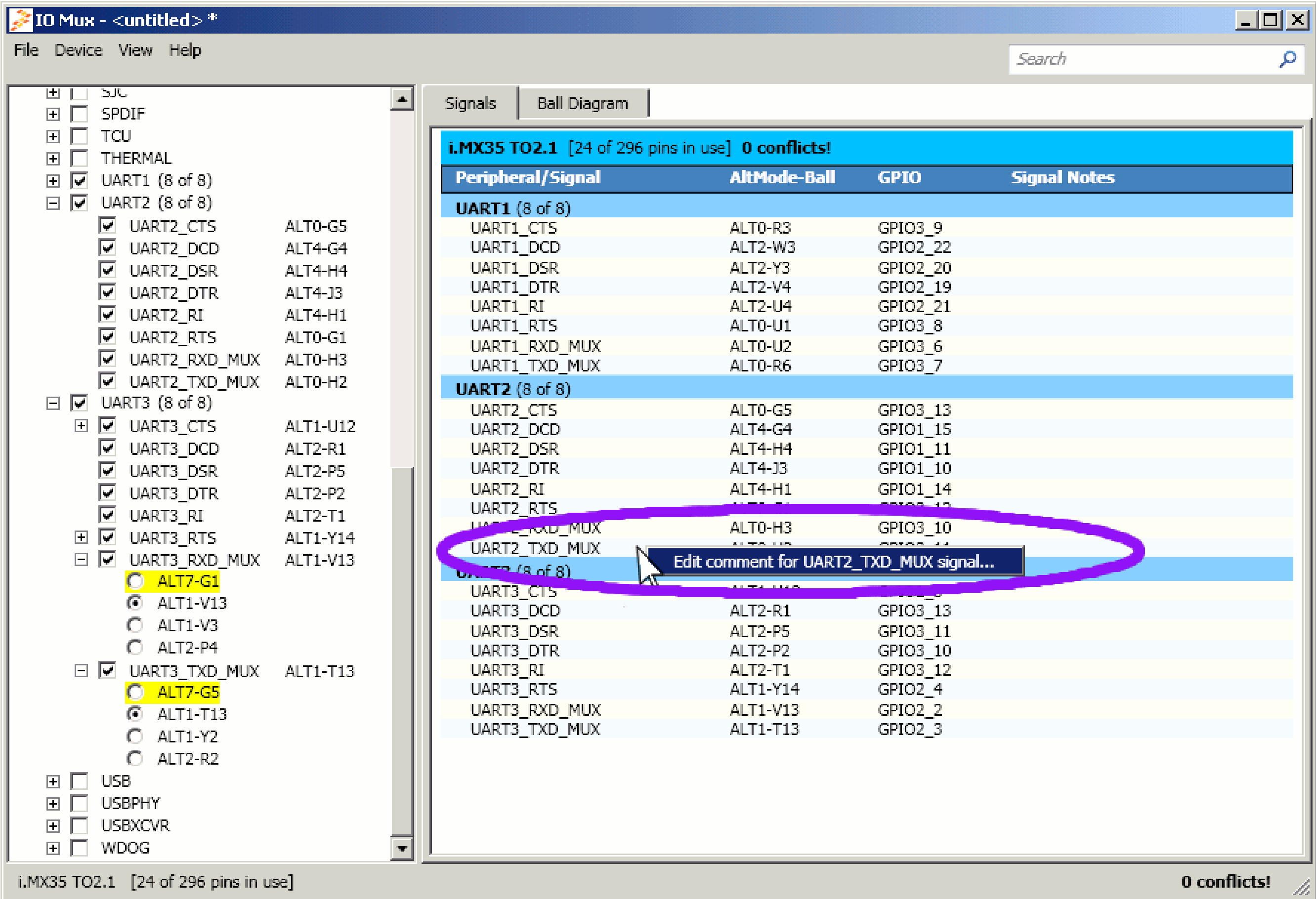



Figure 4-6. Signal Tab with Comment Entry Menu

After the user enters the desired text and clicks okay, the comment is added to the signals tab in the signal notes column, as shown in Figure 4-7.

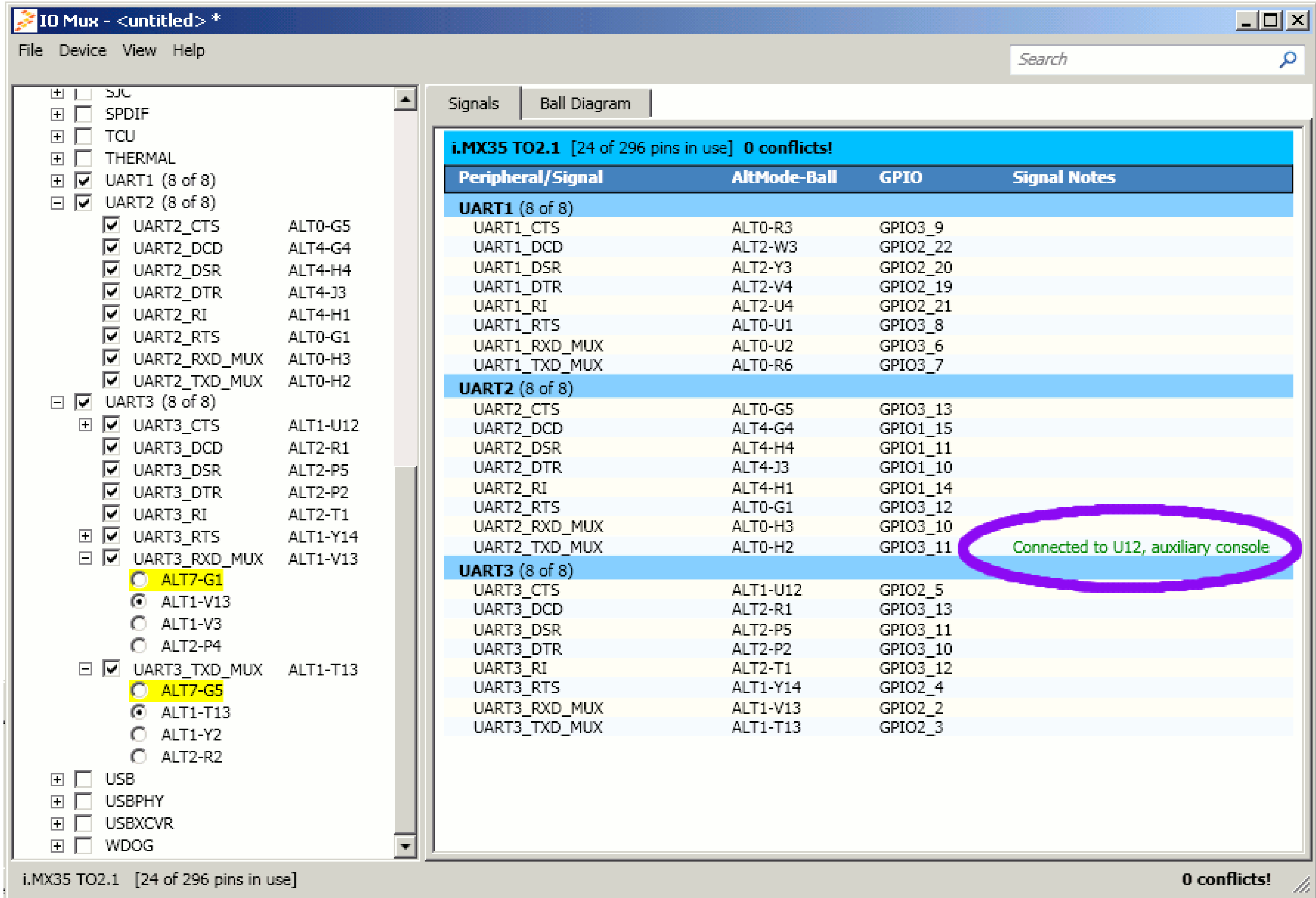



Figure 4-7. UART2_TXD_MUX with Note

Once finished with changes, users should use File > Save As to save the design for future use or reference. Figure 4-8 shows the dialog box. The native format for a design is XML. Users may choose to instead save the a new design in RTF or TXT formats (notice the “<untitled> *” in the title bar). However, the actual design is only saved if it is saved as an XML file. Navigate to a desired directory on the host machine and then change the design’s filename, if desired.

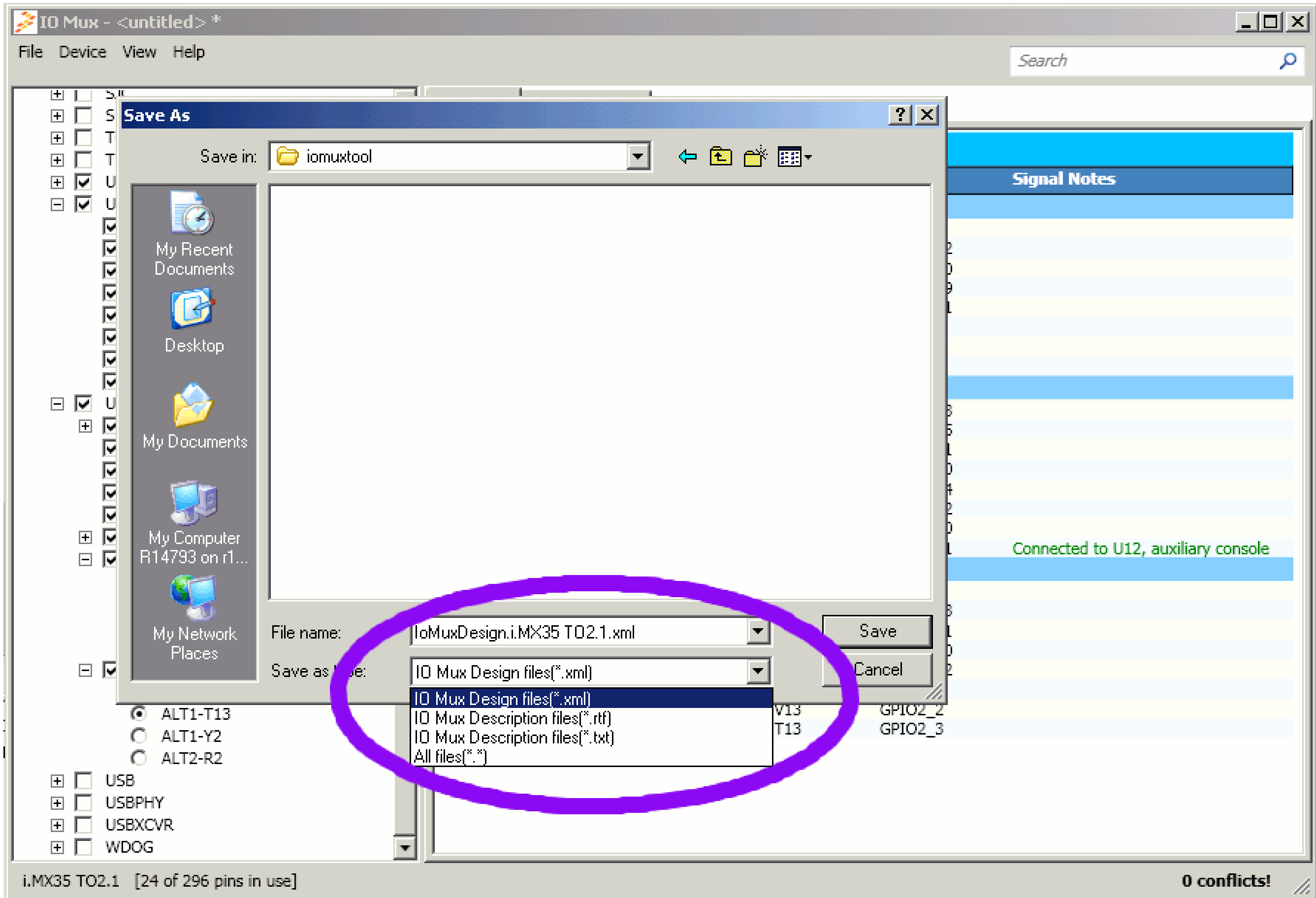



**Figure 4-8. File > Save Dialog Box, Showing the Three Formats for Saving Design Info**

## 4.3 Toggling the Alternate View of the Signals Tab

The alternate view of the signals tab may be seen by selecting View > Shared Signals, as shown in Figure 4-9. This view allows users to see which other signals may be routed to a particular ball, thus allowing users to quickly see which signals are sacrificed by any particular signal assignment.

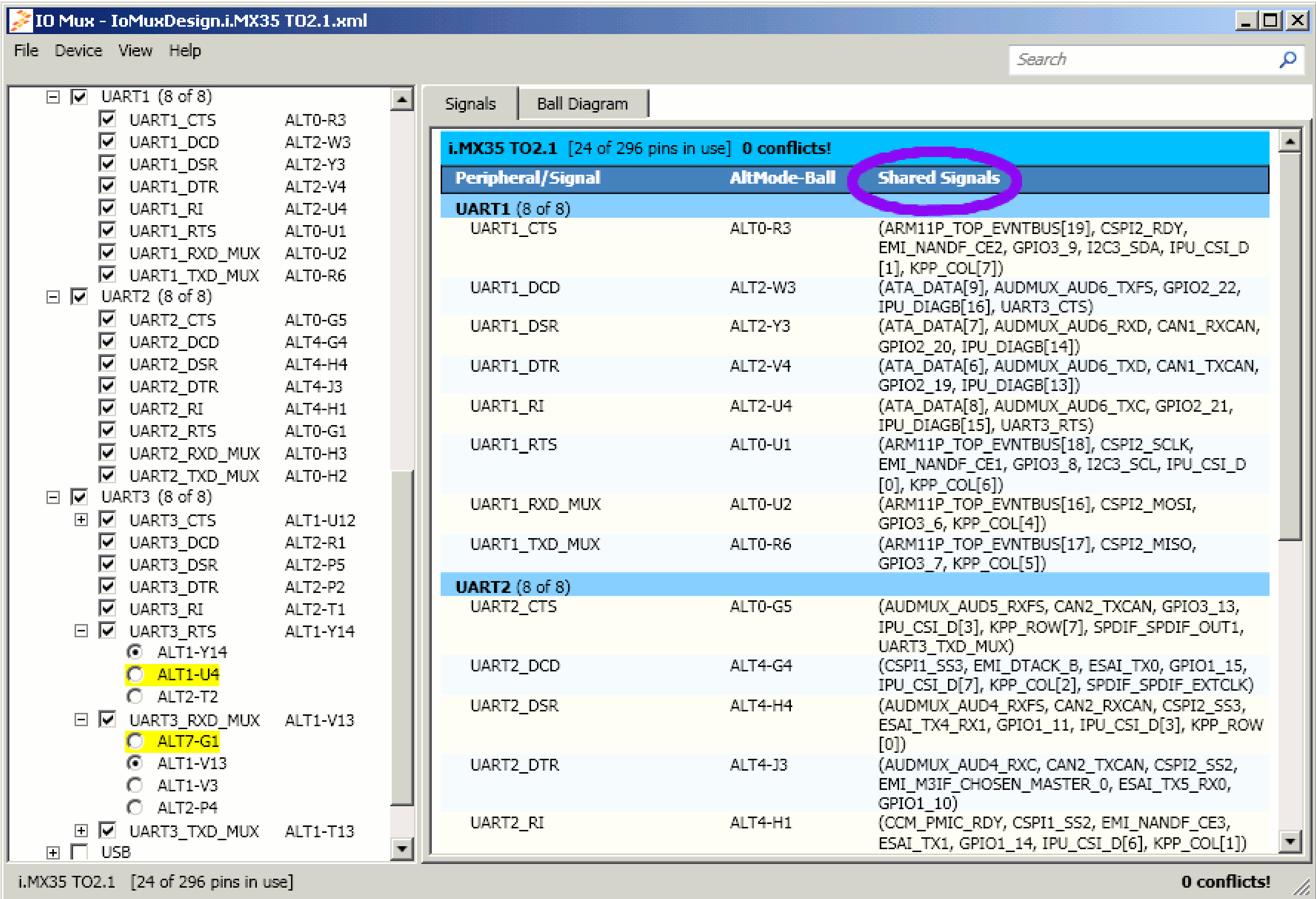



Figure 4-9. Display of Other Signals Available on an Assigned Ball

### 4.3.1 Finding Assigned Signal Locations with the Ball Diagram Tab

For some signals, proximity to other devices may be an issue because of critical routing. Use the ball diagram tab to see where assigned signals are located on the actual package.

Hovering the mouse over a ball produces a pop up list that shows all signals that are capable of being connected to that ball. The actual assigned signal is bolded, as shown in Figure 4-10. Note that the balls are color coded to show whether they are not muxed, available, in use, or conflicted.

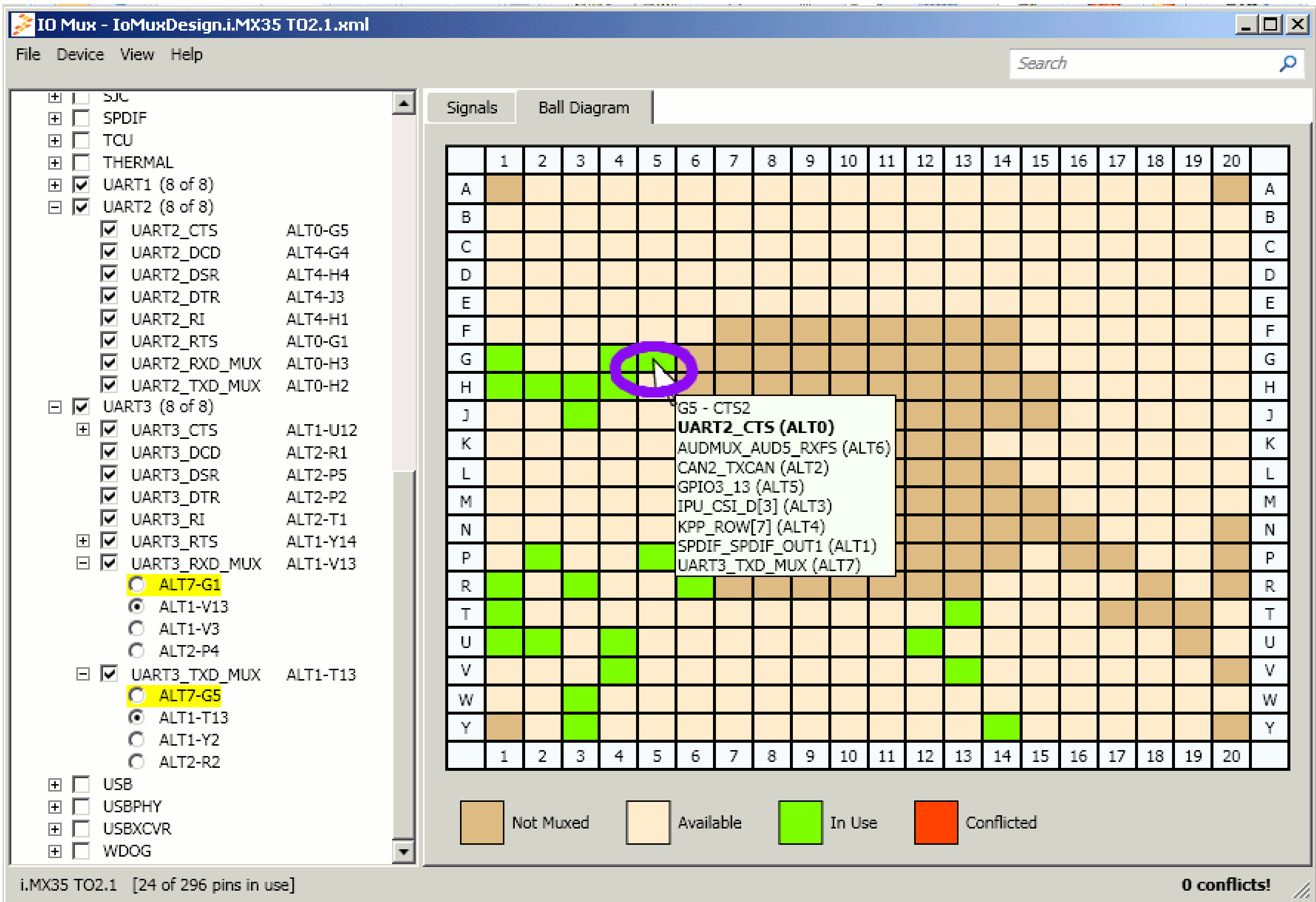



Figure 4-10. Choosing the Ball Diagram Tab Displays the Device's Ball Map

## 4.4 Using the Search Box to Find Specific Signals or Balls

Use the search box (circled in purple in Figure 4-11) to find a specific signal or ball. This conducts a basic substring search that highlights search hits in green, as shown in Figure 4-11.

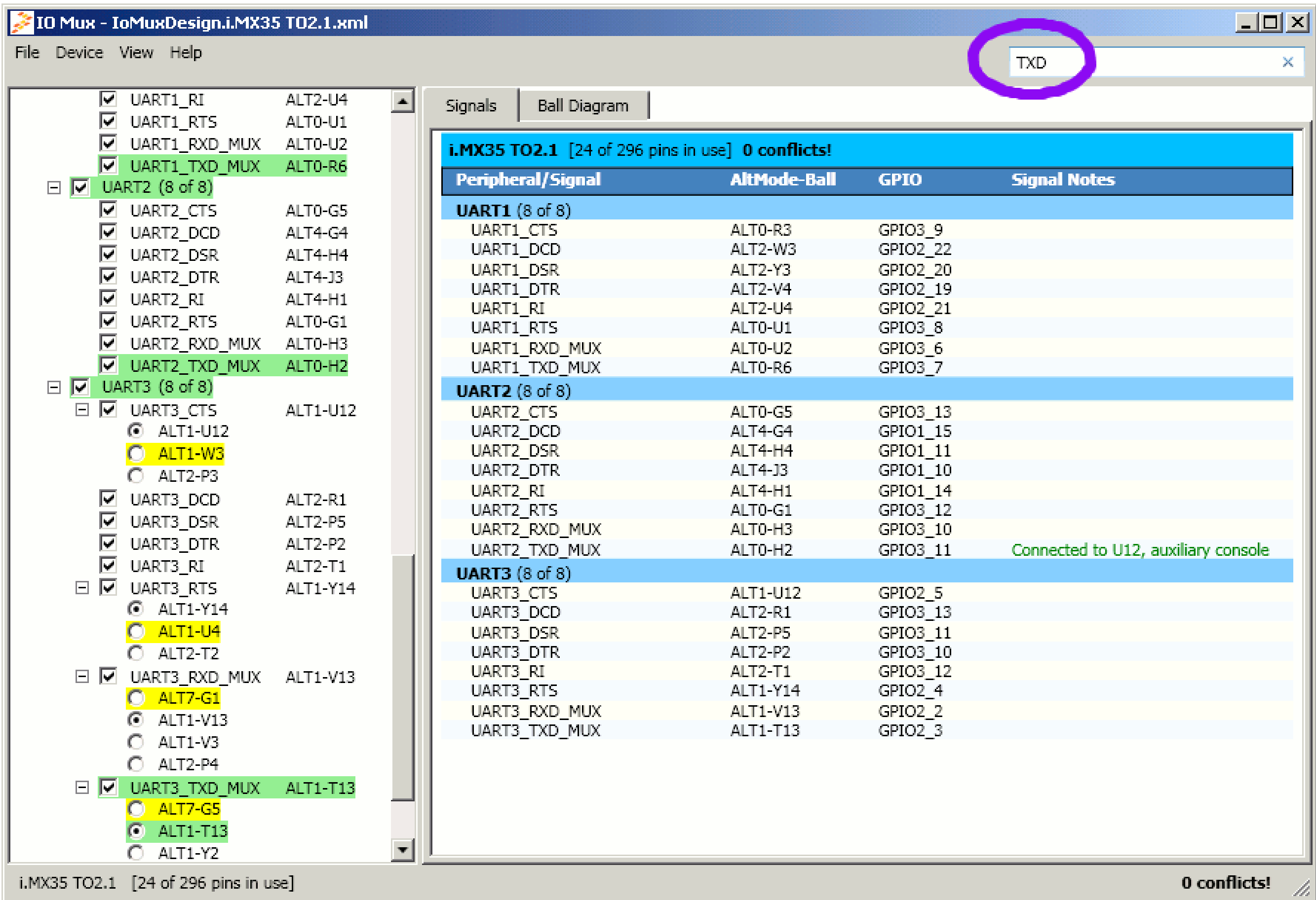


Figure 4-11. Basic Substring Text Searching

## 4.5 IOMUX Features Guide

This section provides a guide to the IOMUX features. Figure 4-12 shows a screenshot of the IOMUX application window with the addition of labels for the main areas. Each labeled area is discussed in greater detail in the following subsections.

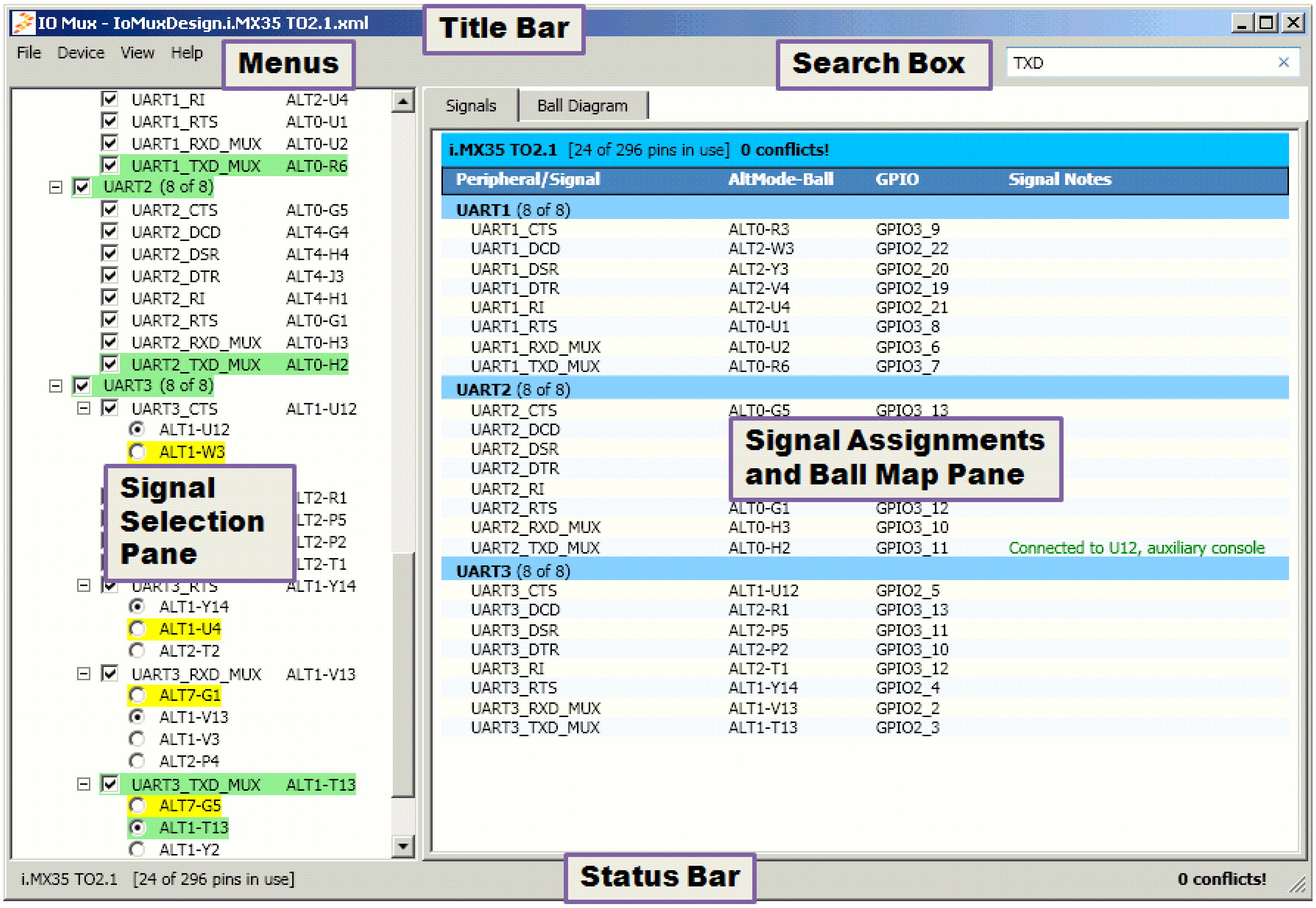



Figure 4-12. IOMux.exe Application Window Overview

### 4.5.1 Title Bar

The title bar shows the application name. If the application has been previously saved, the tool also indicates the name of the design to the left of the application name. If the application has not yet been saved, the tool says “<untitled>” in the same location. An asterisk to the right of the design name indicates that the design has pending changes that have not yet been saved.

### 4.5.2 Menus

The IOMUX application’s menu bar is located in the upper left-hand region of the application window. It contains the following options:

- File—used to open/save design files
- Device—used to select an i.MX device for a design

- View—used to change view settings
- Help—used to display application version information about the IOMUX application

#### 4.5.2.1 File Menu

Figure 4-13 provides a partial view of the application window, showing the expanded options for the file menu. Note that some options list a keystroke combination that can be used as an alternative to selecting it through the file menu.

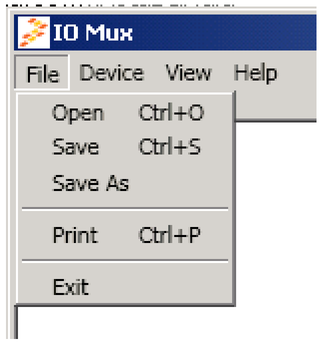



**Figure 4-13. Detailed View of File Menu**

| | |
|---|---|
| Open | Use this option to open a previously saved design file. The file format is a custom XML text file. The user should not edit or modify saved design files by hand because the application does not perform any error checking on input files. |
| Save | Use this option to save a design to the native XML text format. If the design is new and has never been saved before, a Windows save dialog box pops up, with a default filename inserted into the name field. The user may change the name if desired; the application issues a warning if the desired filename already exists and will be overwritten. |
| Save As | Use this option to save the design to a plain text file or to a rich text format (RTF) file. Use a plain text file for copying and pasting into an application's schematic, source code archive, or both. Use the RTF format file to take advantage of formatting that makes the design more readable. |
| Print | Use this option to send the list of assigned signals in the signal selection tab and a ball diagram to an attached Windows printer. Use a color printer to retain the color information in the tabs. Use a virtual printer that generates PDF files, such as Adobe's Distiller or PDF995, to print the design's assignments and a ball diagram to a PDF file. |
| Exit | Use this option to quit the application. The IOMUX application issues a warning to the user if a design has been modified but not yet saved. |

#### 4.5.2.2 Device Menu

Figure 4-14 provides a partial view of the application window, showing the expanded options for the device menu.

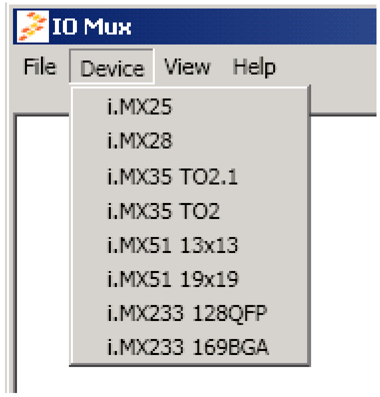



**Figure 4-14. Detailed View of the Device Menu**

Use the device menu to choose the desired device for your design. This menu only shows supported device and package options.

If the current design has unsaved changes and any item in the device menu is selected, the application issues a warning to the user that there are unsaved changes.

#### 4.5.2.3 View Menu

Figure 4-15 provides a partial view of the application window, showing the expanded options for the view menu.

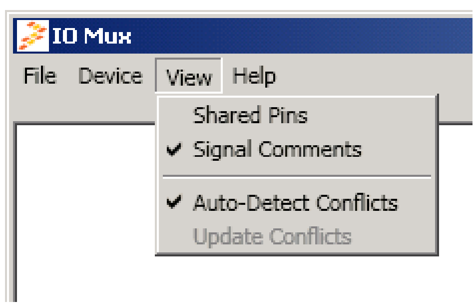



**Figure 4-15. Detailed View of the View Menu**

Use this menu to control the behavior and presentation of the design to the user.

Shared Pins

Signal Comments — The "Shared Pins" and "Signal Comments" options are mutually exclusive. These items select which column appears in the assigned signals tab. By default, "Signal Comments" is selected.

Auto-Detect Conflicts — This item toggles how the application checks a design for conflicts. If selected, the whole design is checked for conflicts any time a single signal assignment is updated. If unselected, a design is checked for conflicts until the Update Conflicts menu item is selected.

Update Conflicts — This item cannot be enabled unless the Auto-Detect option is unselected. Selecting “Update Conflicts” manually forces the design’s assignments to be checked. Usually this option is used for very large designs with many signals to speed up the selection.

#### 4.5.2.4 Help Menu

Figure 4-16 provides a partial view of the application window, showing the expanded options for the help menu. This menu contains a single item, which displays information about the IOMUX application.


**Figure 4-16. Detailed View of the Help Menu**

### 4.5.3 Search Box

The search box performs a simple textual search for full or partial signal names or package ball/pin IDs. Note that the text search is a basic string search. It is not case sensitive nor does it have options for refining the search criteria. Search hits are highlighted in green in the signal selection pane.

### 4.5.4 Signal Selection Pane

Once an i.MX device is selected, the signal selection pane is populated with all IOMUX options for that device and package combination. The signal selection pane is then used to assign signals and identify signal assignment conflicts.

By default, all signals are initially unassigned. Signals are assigned by clicking in the check box next to a peripheral group or, when expanded, an individual signal. For information about identifying and resolving conflicts, see the walkthrough in Section 4.2.1, “Identifying Signal Conflicts with the Signal Selection Pane.”

### 4.5.5 Signal Assignments and Ball Map Pane

The signal assignments and ball map pane allows the user to view either a textual summary of information or the ball map graphical display. Users select the options by choosing either the signals tab or the ball diagram tab.

#### 4.5.5.1 Signals Tab

The signals tab contains the list of all currently assigned signals in the current design. This tab is selected by default and can also be selected by clicking in the signals tab if the alternate ball diagram tab is displayed.

The following list explains each column's function:

| | |
|---|---|
| Peripheral/Signal | This column contains the peripheral and signal names, delimited by the first underscore in the name. The naming convention comes from the Excel spreadsheets that have previously been used to manually make IOMUX assignments. |
| ALT-Mode/Ball | This column contains the ALT-mode and package ball/pin assigned to each signal, separated by a hyphen. In the application code, the IOMUX register for the assigned ball/pin needs to be configured with the ALT-mode specified in the assignment. It should be noted that a SW_INPUT_SELECT register may also need to be assigned, depending on the signal (refer to the device's reference manual). |
| GPIO | This column is a handy reference for the hardware bring-up team for testing basic connectivity without having to actually set up the assigned signals as intended in the application. |
| Signal Notes | This column provides a means of annotating the specific signal assignments. Signal notes are added by right-clicking on an assignment row. Any information that the user feels is relevant may be inserted in the text box that pops up. Consider placing information that is helpful for other people that review the schematics in addition to information helpful for software developers. The output of this tool can be pasted into both the schematic files as well as the application source code. |
| Shared Signals | This column shows all of the other signals available to be muxed out on the assigned ball/pin. This information is identical to the information available by hovering the mouse over an assignment row in the same pane. |

**NOTE**

The signal notes column and shared signals column cannot be viewed simultaneously.

### 4.5.5.2 Ball Diagram Tab

The ball diagram tab provides a visual indication of where assigned signals enter or exit the device package. Each ball or pin is color coded according to a legend at the bottom of the pane (see Figure 4-10).

Hovering over a ball/pin brings up a pop-up that contains a list of available internal signals that may be assigned to that ball/pin for those package interconnects that the IOMUX controls. Note that dedicated signals, power, and ground connections are not shown; refer to the device's data sheet for complete package connection information.

## 4.5.6 Status Bar

The status bar is located at the bottom of the application window. On its left side, it indicates the device selected for the design and a count of the assigned and total signals available to the IOMUX for that design's device. On the right side, it shows a count of the conflicting assignments.

# Chapter 5
# Setting up Power Management

This chapter discusses how to supply and interface the i.MX53 multimedia applications processor with two different power management integrated circuits (PMICs): DA9053 from Dialog and LTC3589-1 from Linear Technology.

The extra components required for the interface are as follows:

- For the DA9053 interface, add an extra regulator RT8010 to supply the 3.3 V USB power domain
- The LTC3589-1 can supply all power rails except DDR. The DDR rail can be supplied by the LT3481. Both devices are available as automotive grade.

Section 5.6, "Additional Device Information," provides additional product information about DA9053 and LTC3589-1. It does not discuss RT8010 and LT3481 due to their simplicity.

## 5.1 i.MX53 Internal LDOs

The i.MX53 integrates two internal LDOs to supply each of the PLL voltage domains. These LDOs are supplied from the VDD_REG pin. They deliver 1.3 V for the VDD_DIG_PLL and 1.8 V for the VDD_ANA_PLL. The 1.3 V regulator outputs its voltage on the VDD_DIG_PLL pin. VDDA, VDDAL1, and VP can be supplied externally from this pin. Place a ferrite to avoid noise coupling between voltage domains.

The analog supply LDO can be software configured through the PLL1P8_VREG[4:0] bits for an output voltage from 1.5 to 2.275 V in 25 mV steps, the default being 1.8 V, and has an output current capability up to 125 mA. The digital supply LDO can be configured by means of the PLL1P2_VREG[4:0] bits from 0.8 to 1.575 V in 25 mV steps. Its default voltage at power up is 1.2 V and it is able to supply up to 125 mA. This LDO must be programmed to 1.3 V after boot up.

VDD_ANA_PLL can externally supply NVCC_RESET and VDD_DIG_PLL can be connected to VDDA, VDDAL1, and VP. The latter can also be supplied separately with an external regulator if desired.

eFuses on the i.MX53 control the LDO default state. By default, the following internal LDOs are used.

- LDO_DIS[0]—controls Analog PLL supply.
- LDO_DIS[1]—controls Digital PLL supply.

If either of these bits are cleared, the internal LDO source provides the respective PLL supply. If they are set, the fuse is blown and PLL power must be provided from the external pad, which is not recommended due to possible noise injections or other issues.

Figure 5-1 shows the internal LDOs.

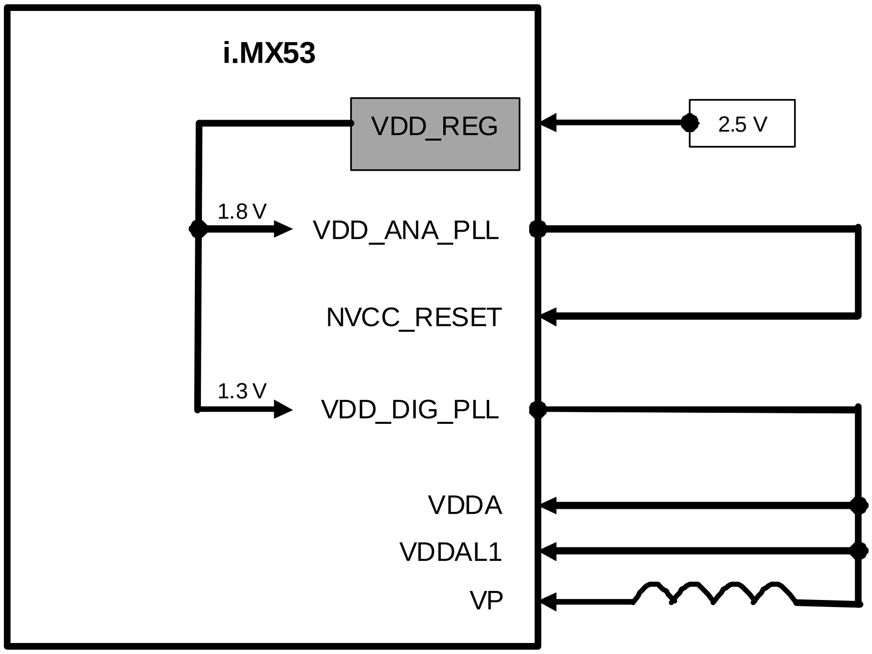



**Figure 5-1. Internal LDOs**

## 5.2 Interfacing the i.MX53 Processor with the DA9053

The power up sequence of the device is one time programmable and must be specified when ordering the device. Figure 5-2 shows the power-up sequence for this specific interface.

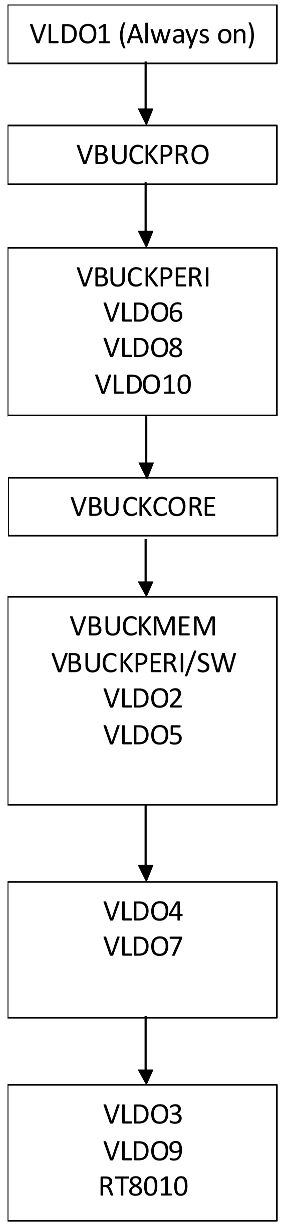



**Figure 5-2. Power-up Sequence**

Table 5-1 shows the i.MX53 voltage rails, their power requirements, and their associated DA9053 regulator. Most of the supply domains have flexible voltage and can be adjusted or supplied with a different regulator depending on application needs.

**Table 5-1. i.MX53 Voltage Rails and Associated DA9053 Regulator**

| Voltage Rail | Description | Nominal Voltage | Associated DA9053 Regulator | Voltage Set Point of DA9053 Regulator (V) | Current Capability (mA) | Power up Sequence Set at the DA9053 |
|---|---|---|---|---|---|---|
| VDDGP | ARM core supply voltage<br>fARM ≤ 400 MHz | 0.95 | VBUCKCORE | 1.1 | 2000 | 3 |
| | ARM core supply voltage<br>fARM ≤ 800 MHz | 1.1 | | | | |
| | ARM core supply voltage<br>fARM ≤ 1000 MHz | 1.25 | | | | |
| | ARM core supply voltage<br>Stop mode | 0.85 | | | | |
| VCC | Peripheral supply voltage | 1.3 | VBUCKPRO | 1.3 | 1000 | 1 |
| | Peripheral supply voltage—<br>stop mode | 0.95 | | | | |
| VDDA | Memory arrays voltage | 1.3 | LDO10 | 1.3 | 250 | 2 |
| | Memory arrays voltage—<br>stop mode | 0.95 | | | | |
| VDDAL1 | L1 cable memory arrays voltage | 1.3 | LDO6 | 1.3 | 150 | 2 |
| | L1 cable memory arrays<br>voltage—stop mode | 0.95 | | | | |
| VDD_DIG_PLL | PLL digital supplies<br>External regulator option | 1.3 | LDO2 | 1.3 | 100 | 4 |
| VDD_ANA_PLL | PLL analog supplies<br>External regulator option | 1.8 | LDO8 | 1.8 | 200 | 2 |
| NVCC_CKIH | ESD protection of the CKIH pins, Fuse read supply and 1.8 V bias for the UHVIO pads | 1.8 | LDO8 | 1.8 | 200 | 2 |
| NVCC_LCD | GPIO digital power supplies | 1.8 or 2.775 | LDO4 | 2.775 | 150 | 5 |
| NVCC_JTAG | | | LDO8 | 1.8 | 200 | 2 |
| NVCC_LVDS | LVDS interface supply | 2.5 | VBUCKPERI_SW | 2.475 | 1000 | 4 |
| NVCC_LVDS_BG | LVDS band gap supply | 2.5 | | | | |
| NVCC_EMI_DRAM | DDR supply DDR2 range | 1.8 | VBUCKMEM | 1.5 | 1000 | 4 |
| | DDR supply LP-DDR2 range | 1.2 | | | | |
| | DDR supply LV-DDR2 range | 1.55 | | | | |
| | DDR supply DDR3 range | 1.5 | | | | |

**Table 5-1. i.MX53 Voltage Rails and Associated DA9053 Regulator (continued) (continued)**

| Voltage Rail | Description | Nominal Voltage | Associated DA9053 Regulator | Voltage Set Point of DA9053 Regulator (V) | Current Capability (mA) | Power up Sequence Set at the DA9053 |
|---|---|---|---|---|---|---|
| VDD_FUSE | Fusebox program supply (Write only) | 3.15 | Ext DCDC[1] | 3.2 | 1000 | 6 |
| NVCC_NANDF | Ultra high voltage I/O (UHVIO) supplies | - | LDO8 | 1.8 | 200 | 2 |
| NVCC_SD1<br>NVCC_SD2<br>NVCC_PATA<br>NVCC_KEYPAD<br>NVCC_GPIO<br>NVCC_FEC<br>NVCC_EIM_ MAIN<br>NVCC_EIM_SEC | UHVIO_L<br>UHVIO_H<br>UHVIO_UH | 1.8<br>2.775<br>3.3 | LDO3 | 3.3 | 200 | 6 |
| NVCC_CSI | | | LDO8 | 1.8 | 200 | 2 |
| TVDAC_DHVDD<br>TVDAC_ AHVDDRGB | TVE digital and analog power supply, TVE-to-DAC level shifter supply, cable detector supply, analog power supply to RGB channel | 2.775 | LDO7 | 2.75 | 200 | 5 |
| | For GPIO use only, when TVE is not in use | 1.8 or 2.775 | | | | |
| NVCC_SRTC_ POW | SRTC Core and I/O supply (LVIO) | 1.3 | LDO1 | 1.3 | 40 | 0 |
| NVCC_RESET | LVIO | 1.8 or 2.775 | LDO8 | 1.8 | 200 | 2 |
| USB_H1_ VDDA25<br>USB_OTG_VDD A25 | USB_PHY analog supply, oscillator amplifier analog supply | 2.5 | VBUCKPERI_ SW | 2.475 | 1000 | 4 |
| NVCC_XTAL | | | VBUCKPERI | 2.475 | 1000 | 2 |
| USB_H1_ VDDA33<br>USB_OTG_ VDDA33 | USB PHY I/O analog supply | 3.3 | Ext DCDC[1] | 3.2 | 1000 | 6 |
| VDD_REG | Power supply input for the integrated linear regulators | 2.5 | VBUCKPERI | 2.475 | 1000 | 2 |

**Table 5-1. i.MX53 Voltage Rails and Associated DA9053 Regulator (continued) (continued)**

| Voltage Rail | Description | Nominal Voltage | Associated DA9053 Regulator | Voltage Set Point of DA9053 Regulator (V) | Current Capability (mA) | Power up Sequence Set at the DA9053 |
|---|---|---|---|---|---|---|
| VP | S-ATA PHY Core power supply | 1.3 | LDO5 | 1.3 | 100 | 4 |
| VPH | S-ATA PHY I/O supply voltage | 2.5 | VBUCKPERI_SW | 2.475 | 1000 | 4 |

[1] External DCDC part number is RT8010

### 5.2.1 Connecting Power and Communication Signals

Figure 5-3 shows the power connections required for the interface. Figure 5-4 shows how the communication signals must be connected between the i.MX53, DA9053, and required extra regulators.

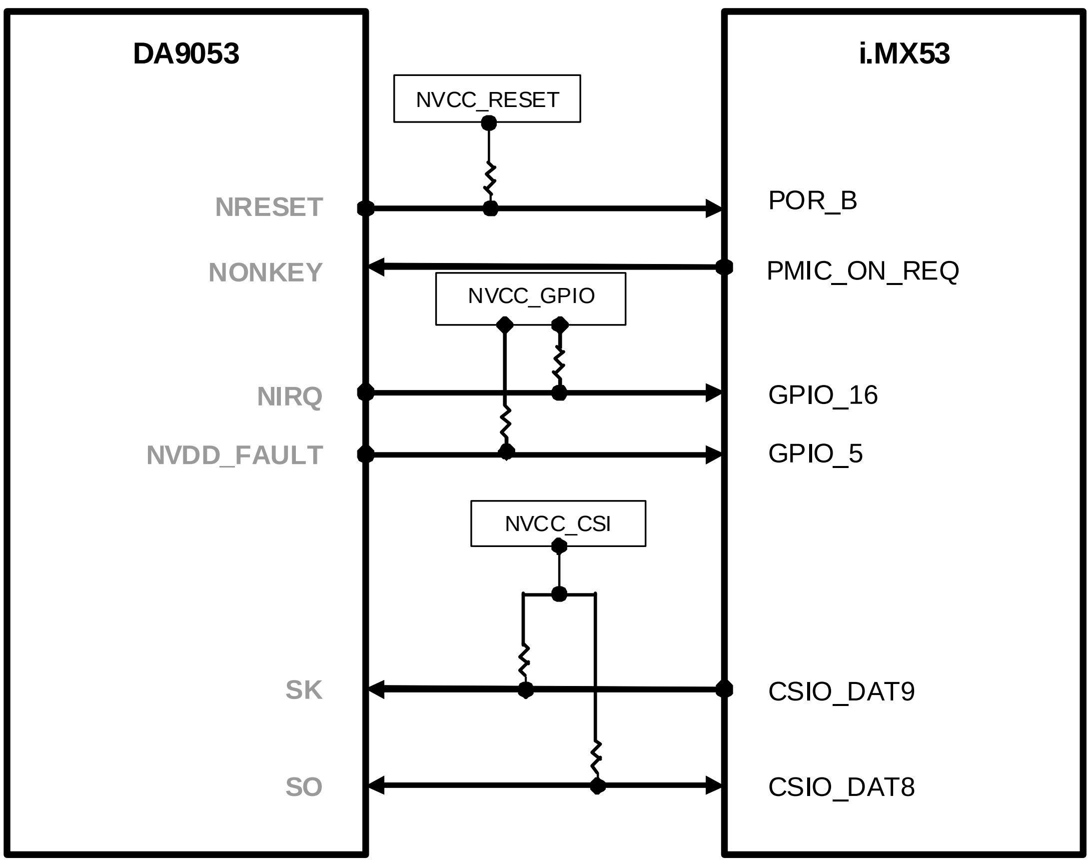



**Figure 5-3. Power Connections**

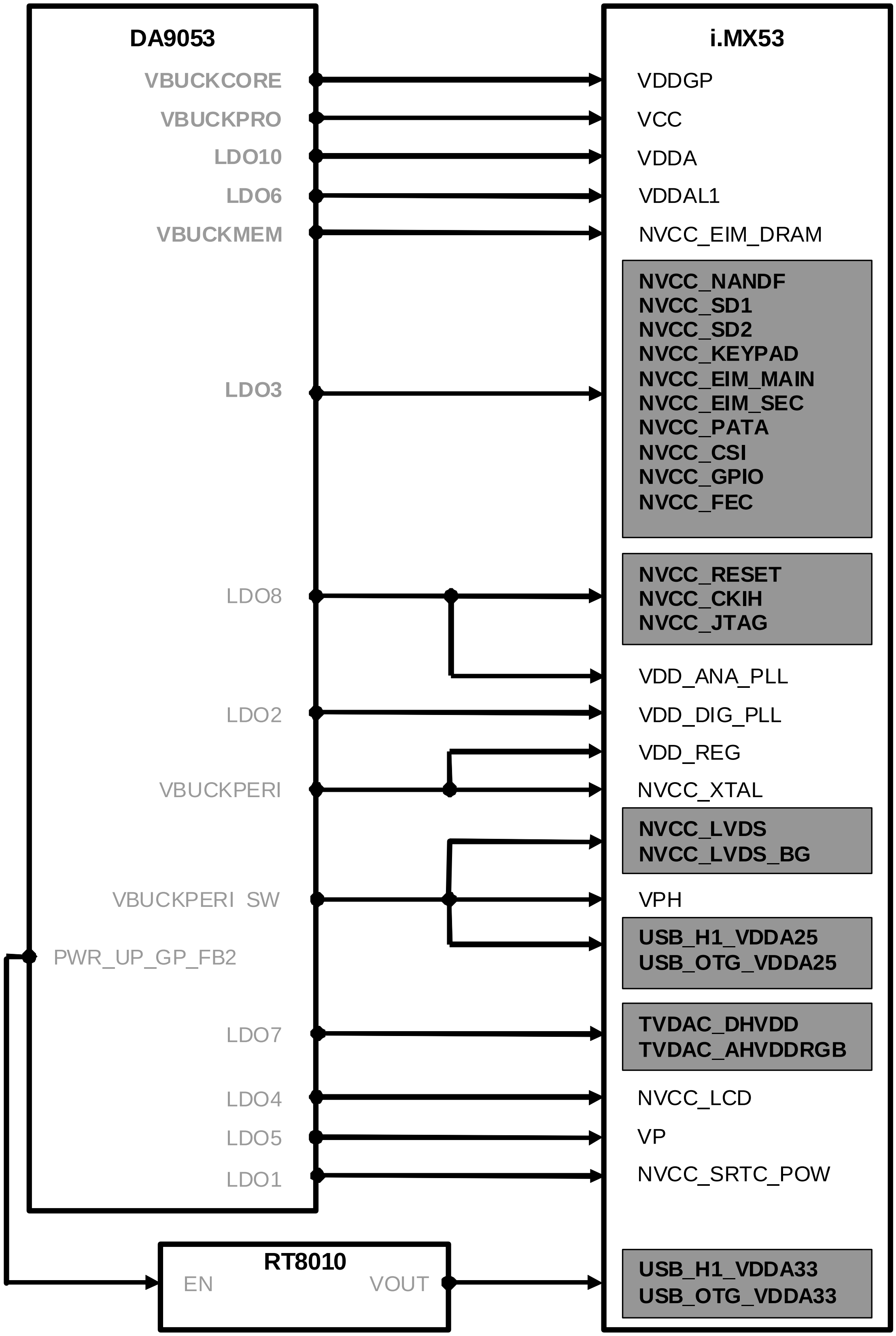



**Figure 5-4. Communication Signal Connections**

Figure 5-5 shows the power-up sequence of the interface that results from the connections shown in Figure 5-3 and Figure 5-4.

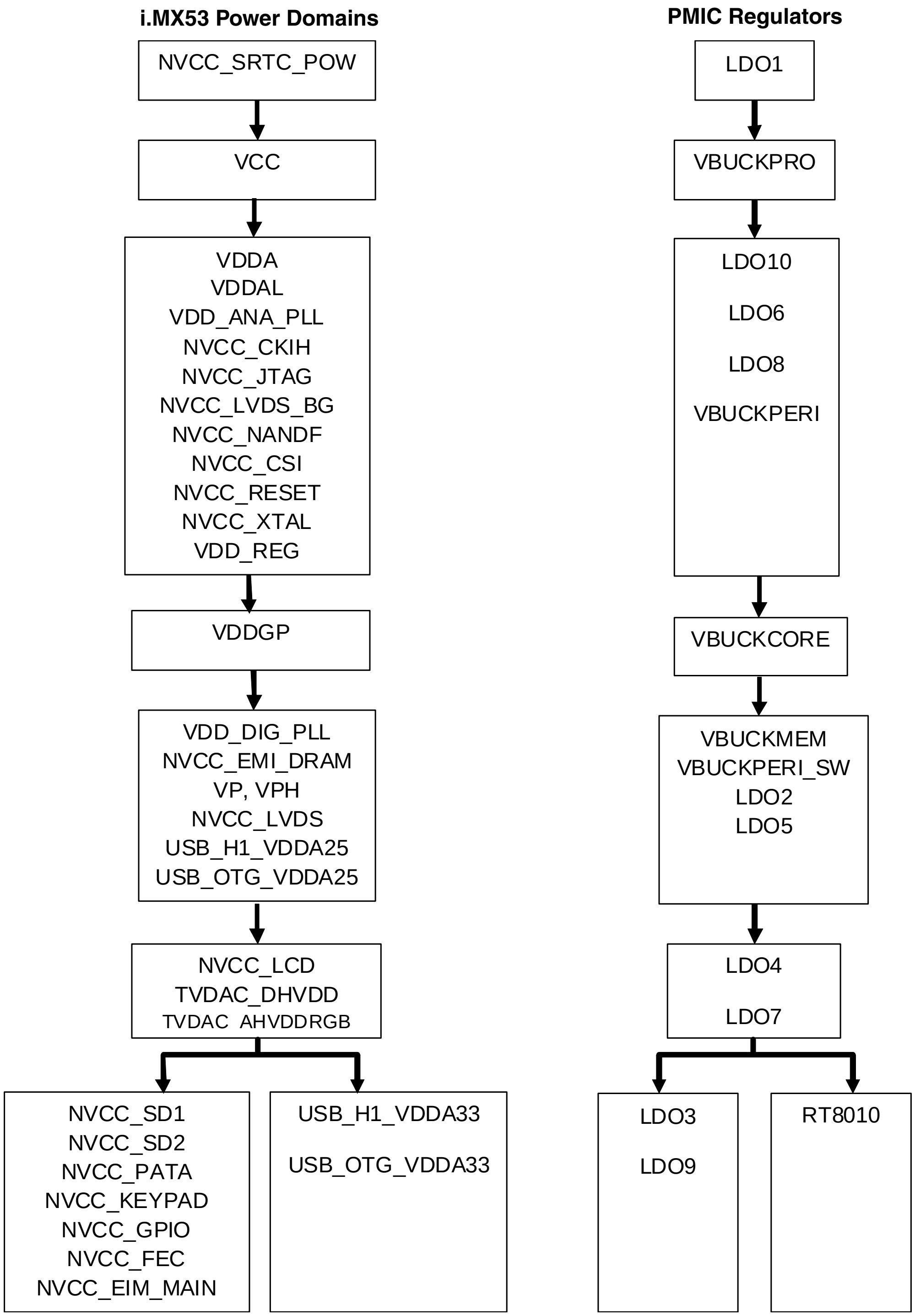
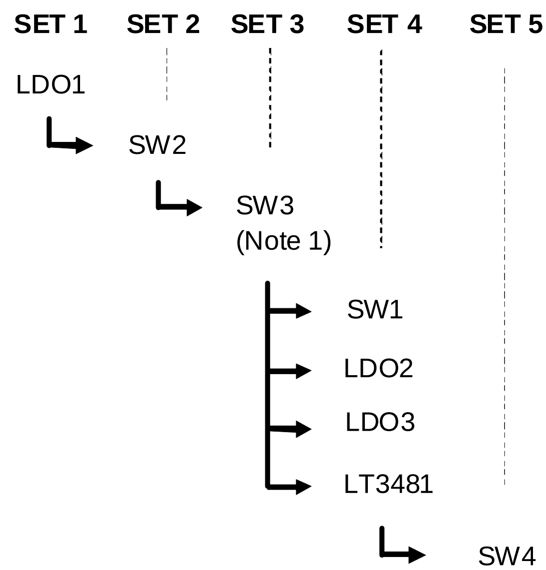



**Figure 5-5. Interface Power-up Sequence (DA9053)**

## 5.3 Interfacing the i.MX53 Processor with LTC3589-1

The LTC3589-1 has flexible options for enabling and sequencing the regulator enables. The regulators are enabled using input pins or the I²C serial port. To define a power-on sequence, tie the enable of the first regulator to be powered up to the wake pin. Connect the first regulator's output to the enable pin of the second regulator, and so on. One or more regulators may be started in any sequence. Each enable pin has a 200 µ (typical) delay between the pin and the internal enable of the regulator.

Figure 5-6 shows the power-up sequence for connecting the i.MX53 and the LTC3589-1 together as shown in this chapter, taking into account the required extra regulators (TPS73201 and LT3481). The TPS7320's EN pin controls its output voltage, and the LT3481's RUN/SS pin turns it on. Voltage sources are divided into four different sets, according to the time they turn on.




**Note:** i.MX53 internal regulators are turned on at this point

**Figure 5-6. Power-up Sequence**

### 5.3.1 Using the I²C Interface

The LTC3589-1 uses the standard I²C 2-wire interface for communication with bus masters. The two bus lines, SDA and SCL, must be high when the bus is not in use. External pull-up resistors or current sources are required on these lines.

The LTC3589-1 is both a slave receiver and slave transmitter. The I²C control signals, SDA and SCL, are scaled internally to the DVDD supply. DVDD should be connected to the same power supply as the bus pull-up resistors.

The I²C port has an under voltage lockout on the DVDD pin. When DVDD is below approximately 1 V, the I²C serial port is reset to power-on states and registers are set to default values.

### 5.3.2 I²C Acknowledge

The acknowledge signal is used for handshaking between the master and the slave. When the LTC3589-1 is written to, the LTC3589-1 acknowledges its write address and subsequent register address and data bytes. When the LTC3589-1 is read from, it acknowledges its read address and 8-bit status byte.

An acknowledge pulse (active low) generated by the LTC3589-1 lets the master know that the latest byte of information was transferred. The master generates the clock cycle and releases the SDA line (high) during the acknowledge clock cycle. The LTC3589-1 pulls down the SDA line during the write acknowledge clock pulse so that it is a stable Low during the high period of this clock pulse.

## 5.4 Interface Table

Table 5-2 shows the i.MX53 voltage rails, their power requirements, and their associated LTC3589-1 regulator in a typical application. Please note that system needs may vary according to the application, and voltage rails must be adjusted accordingly.

**Table 5-2. i.MX53 Voltage Rails and Associated LTC3589-1 Regulator**

| Voltage Rail | Description | Nominal Voltage | Associated LTC3589-1 Regulator | Voltage Set Point of LTC3589-1 Regulator (V) | Current Capability (mA) | Power up Sequence Set at the LTC3589-1 |
|---|---|---|---|---|---|---|
| VDDGP | ARM core supply voltage fARM ≤ 400 MHz | 0.95 | SW1 | 1.1 | 1600 | 3 |
| | ARM core supply voltage fARM 800 MHz | 1.1 | | | | |
| | ARM core supply voltage fARM 1000 MHz | 1.25 | | | | |
| | ARM core supply voltage Stop mode | 0.85 | | | | |
| VCC | Peripheral supply voltage | 1.3 | SW2 | 1.3 | 1200 | 1 |
| | Peripheral supply voltage Stop mode | 0.95 | | | | |
| VDDA | Memory arrays voltage | 1.3 | LDO2 | 1.3 | 250 | 3 |
| | Memory arrays voltage - Stop Mode | 0.95 | | | | |
| VDDAL1 | L1 cable memory arrays voltage | 1.3 | LDO2 | 1.3 | 250 | 3 |
| | L1 cable memory arrays voltage—stop mode | 0.95 | | | | |
| VDD_DIG_PLL | PLL Digital supplies External regulator option | 1.3 | Internal regulator | 1.3 | 125 | 2 |
| VDD_ANA_PLL | PLL Analog supplies External regulator option | 1.8 | Internal regulator | 1.8 | 125 | 2 |

**Table 5-2. i.MX53 Voltage Rails and Associated LTC3589-1 Regulator (continued)**

| Voltage Rail | Description | Nominal Voltage | Associated LTC3589-1 Regulator | Voltage Set Point of LTC3589-1 Regulator (V) | Current Capability (mA) | Power up Sequence Set at the LTC3589-1 |
|---|---|---|---|---|---|---|
| NVCC_CKIH | ESD protection of the CKIH pins, Fuse read supply and 1.8 V bias for the UHVIO pads | 1.8 | Note 1 | 1.8 | 125 | 2 |
| NVCC_LCD | GPIO digital power supplies | 1.8 or 2.775 | LDO3 | 2.8 | 250 | 3 |
| NVCC_JTAG | | | LDO3 | 2.8 | 250 | 3 |
| NVCC_LVDS | LVDS interface supply | 2.5 | SW3 | 2.5 | 1200 | 2 |
| NVCC_LVDS_ BG | LVDS Band gap supply | 2.5 | | | | |
| NVCC_EMI_ DRAM | DDR supply DDR2 range | 1.8 | Notes 2 and 3 | 1.8 | 5000 | 3 |
| | DDR supply LP-DDR2 range | 1.2 | | | | |
| | DDR supply LV-DDR2 range | 1.55 | | | | |
| | DDR supply DDR3 range | 1.5 | | | | |
| VDD_FUSE | FUSEBOX program supply (Write only) | 3.15 | LDO4 | 3.2 | 250 | 3 |
| NVCC_NANDF<br>NVCC_SD1<br>NVCC_SD2<br>NVCC_PATA<br>NVCC_KEYPAD<br>NVCC_GPIO<br>NVCC_FEC<br>NVCC_EIM_ MAIN<br>NVCC_EIM_ SEC<br>NVCC_CSI | Ultra High voltage I/O (UHVIO) supplies | — | SW4 | 3.3 | 1200 | 4 |
| | UHVIO_L | 1.8 | | | | |
| | UHVIO_H | 2.775 | | | | |
| | UHVIO_UH | 3.3 | | | | |
| TVDAC_DHVDD<br>TVDAC_ AHVDDRGB | TVE digital and analog power supply,<br>TVE-to-DAC<br>level shifter supply, cable detector supply, analog power supply to RGB channel | 2.75 | LDO3 | 2.8 | 250 | 3 |
| | For GPIO use only, when TVE is not in use | 1.8 or 2.775 | | | | |
| NVCC_SRTC_ POW | SRTC Core and I/O supply (LVIO) | 1.3 | LDO1 | 1.3 | 25 | 0 |

Table 5-2. i.MX53 Voltage Rails and Associated LTC3589-1 Regulator (continued)

| Voltage Rail | Description | Nominal Voltage | Associated LTC3589-1 Regulator | Voltage Set Point of LTC3589-1 Regulator (V) | Current Capability (mA) | Power up Sequence Set at the LTC3589-1 |
|---|---|---|---|---|---|---|
| NVCC_RESET | LVIO | 1.8 or 2.775 | Note 1 | 1.8 | 125 | 2 |
| USB_H1_ VDDA25 USB_OTG_ VDDA25 NVCC_XTAL | USB_PHY analog supply, oscillator amplifier analog supply | 2.5 | SW3 | 2.5 | 1200 | 2 |
| USB_H1_ VDDA33 USB_OTG_ VDDA33 | USB PHY I/O analog supply | 3.3 | SW4 | 3.3 | 1200 | 4 |
| VDD_REG | Power supply input for the integrated linear regulators | 2.5 | SW3 | 2.5 | 1200 | 2 |
| VP | SATA PHY Core power supply | 1.3 | LDO2 | 1.3 | 250 | 3 |
| VPH | SATA PHY I/O supply voltage | 2.5 | SW3 | 2.5 | 1200 | 2 |

[1] These domains are supplied by the internal regulator of the i.MX at the VDD_ANA_PLL pin.
[2] An external DCDC converter is needed to supply these domains.
[3] External part regulator number is LT3481.

## 5.5 Connecting Power and Communication Signals

Figure 5-7–Figure 5-10 show the required connections for interfacing LTC3589-1 and LT3481 with the i.MX53. Both power domains and communication signals blocks are specified for a more complete understanding of the interface.

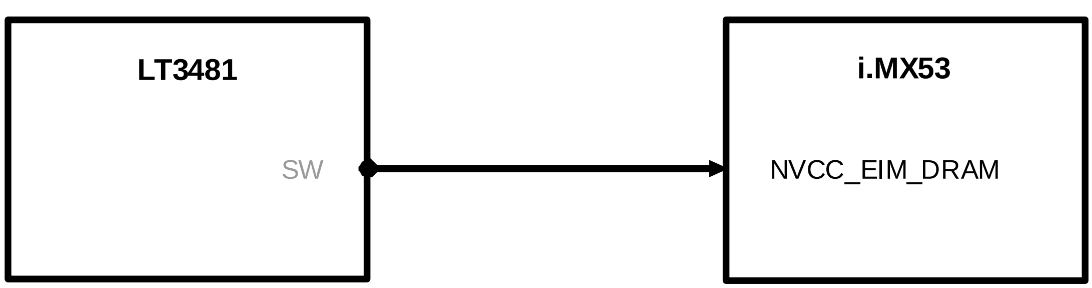



Figure 5-7. Power Connections Block (LT3481)

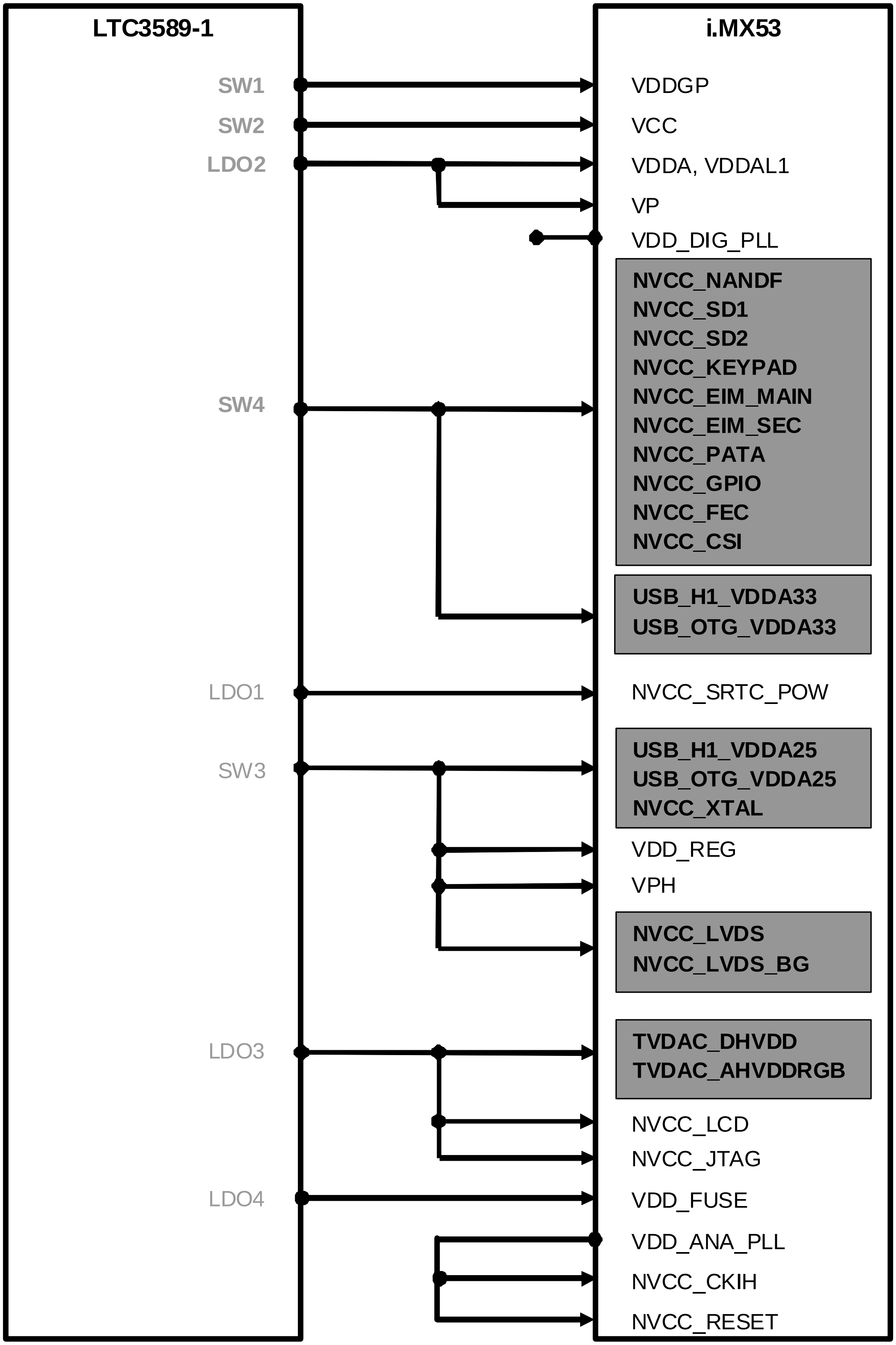



**Figure 5-8. Power Connections Block, cont. (LTC3589-1)**

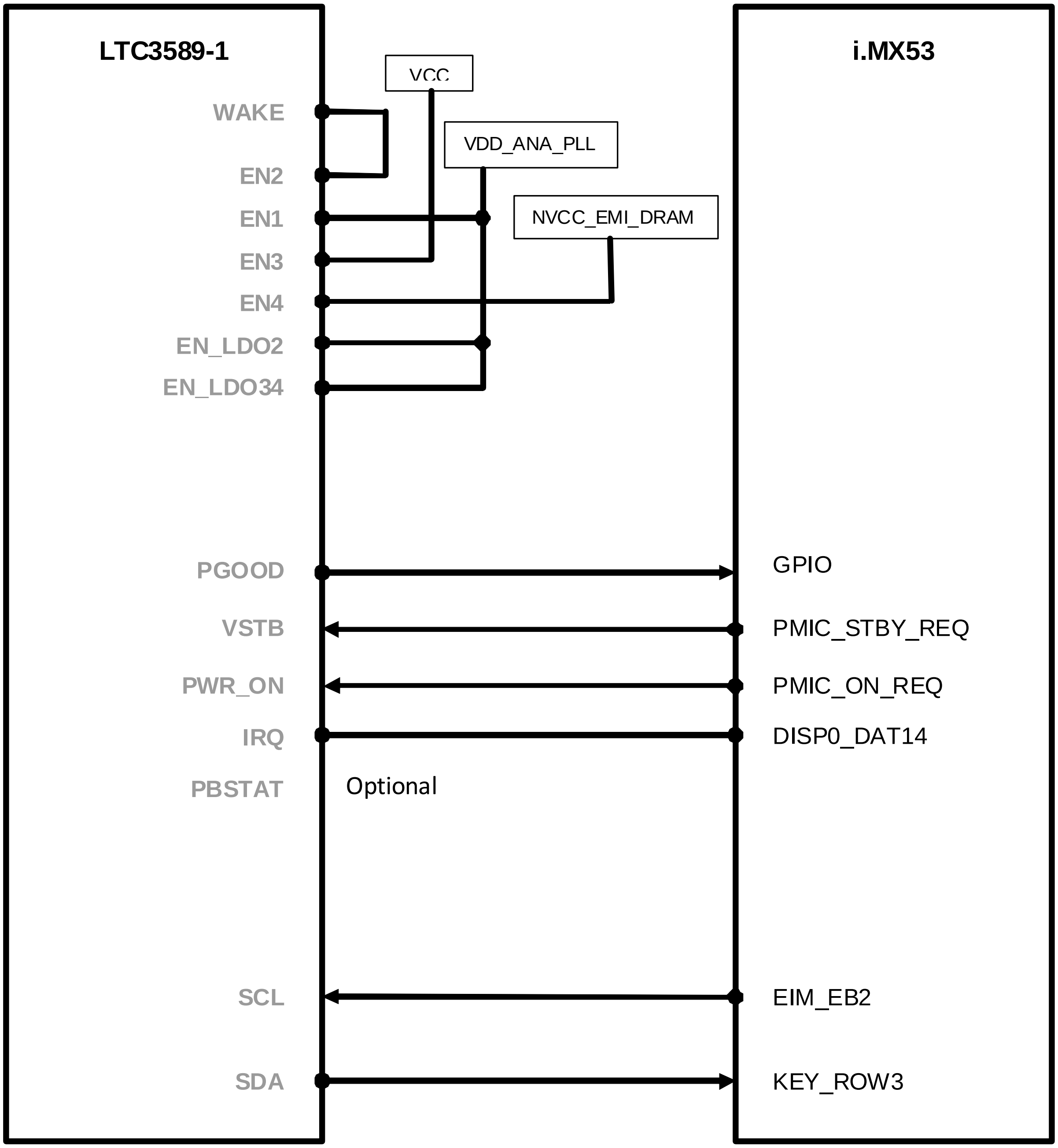



**Figure 5-9. Communication Signals Connections Block (LTC3589-1)**

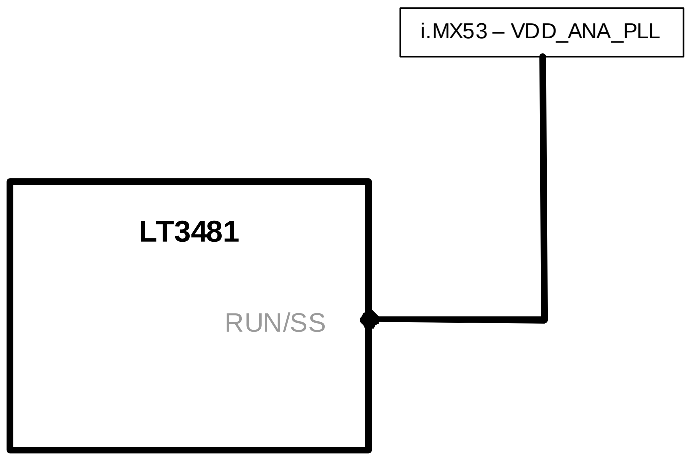



**Figure 5-10. Communication Signals Connections Block, cont. (TPS73201, LT3481)**

### 5.5.1 Powering-up the Interface

Figure 5-11 shows the interface's power-up sequence.

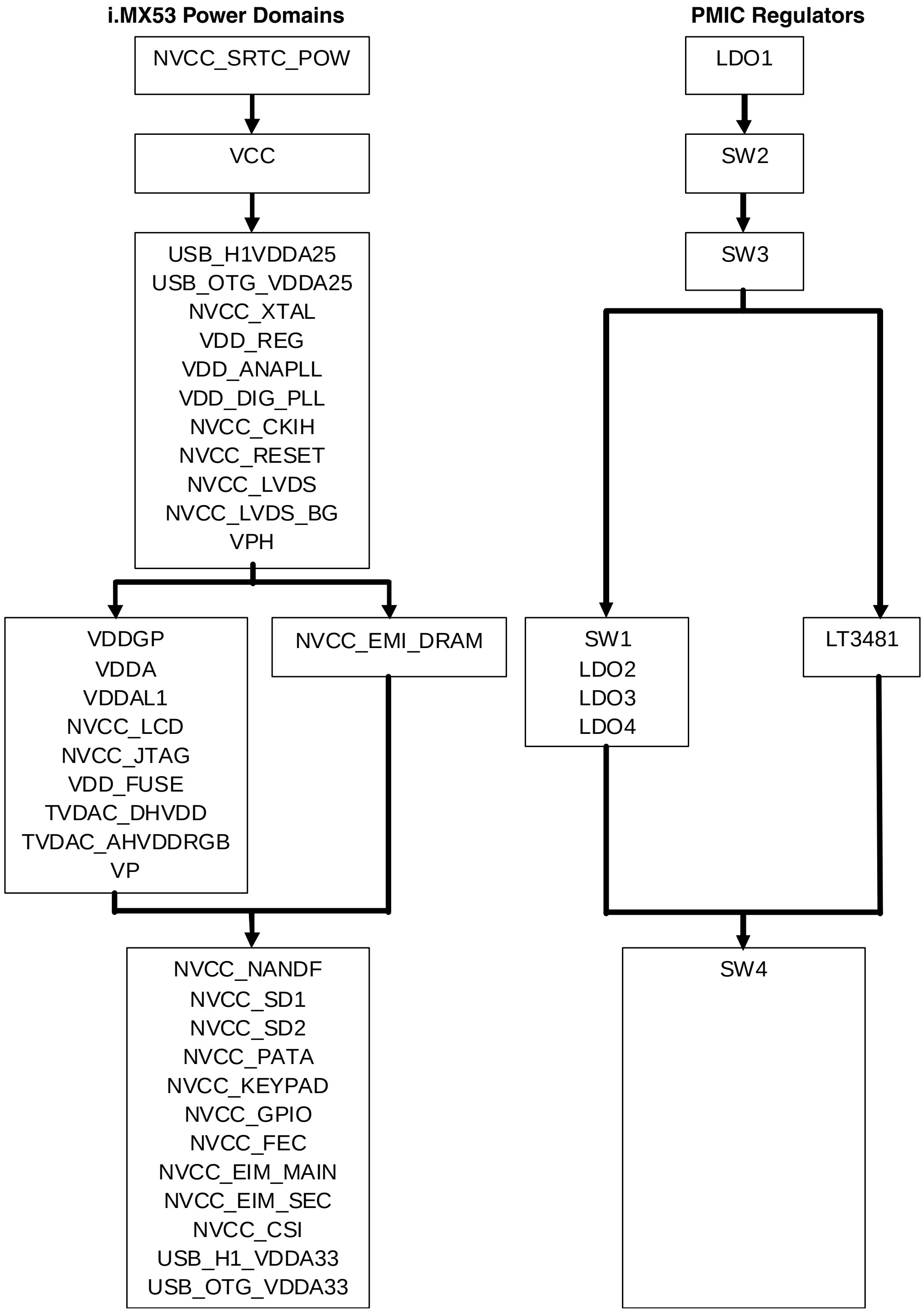



**Figure 5-11. Interface Power-Up Sequence (LTC3589-1)**

## 5.6 Additional Device Information

This section provides additional product information for the DA9053 PMIC subsystem and the LTC3589-1 PMIC subsystem.

### 5.6.1 DA9053

The DA9053 is a power management IC that includes the necessary sources to supply the i.MX53. That main features that enable this interface are:

- 4 buck converters and 10 programmable LDOs, capable of supplying all i.MX53 voltage domains
- Battery charger that supports DC and USB charging
- 32 KHz real time clock oscillator
- 10 Channel ADC and touch screen interface
- 3 string white LED driver
- 16 bit GPIO and dual serial control interfaces for communication with the processor

Figure 5-12 shows the DA9053 application block diagram.

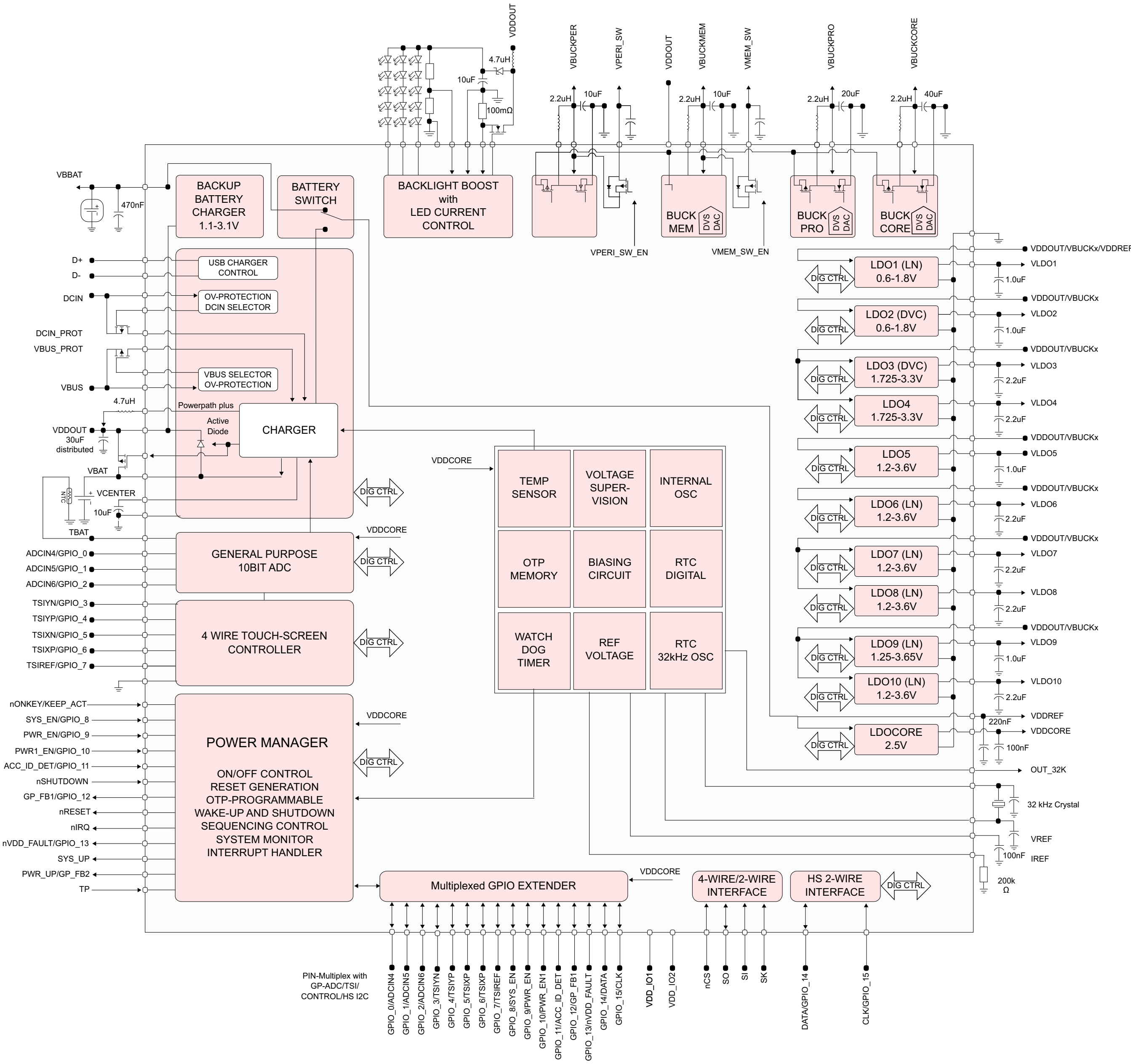


**Figure 5-12. DA9053 Typical Application Block Diagram**

Table 5-3 shows the generated supply domains.

**Table 5-3. Generated Supply Domains**

| Regulator | Supplied pins | Supplied voltage | Supplied max current | External component | Notes |
|---|---|---|---|---|---|
| Buckcore | VBUCKCORE | V0.5–2.075 V ± 3% accuracy default 1.8 V | 2000 mA | 2.2/1.0 µH | DVC, 2 MHz, 25 mV steps<br>DVC ramp with controlled slew rate; pull-down resistor switch off |
| Buckpro | VBUCKPRO | 0.5–2.075 V ±3% accuracy default 1.2 V | 1000 mA | 2.2/1.0 µH | DVC, 2 MHz, 25 mV steps<br>DVC ramp with controlled slew rate; pull-down resistor switch off |
| Buckmem | VBUCKMEM; VMEM_SW | 0.925–2.475 V ±3% accuracy default 2.0 V | 1000 mA | 2.2/1.0 µH | DVC, 2 MHz, 25 mV steps<br>DVC ramp with controlled slew rate; second output with sequencer controllable switch; pull-down resistor switch off |
| Buckperi | VBUCKPERI_SW | 0.925–2.475 V ±3% accuracy default TBD | 1000 mA | 2.2/1.0 µH | 2 MHz, 25 mV steps<br>second output with sequencer controllable switch |
| Boost | Ext. FET | 5–25 V, regulated via current feedback | 50 mA | 4.7 µH | Current controlled boost converter for 3 strings of up to 6 serial white LEDs. Overvoltage protection via a voltage feedback pin. |
| LDO1 | VLD01 | 0.6–1.8 V ±3% accuracy default 1.2 V | 40 mA | 1.0 µF | High PSSR, low noise LDO, 50 mV steps, pull-down resistor switch off |
| LDO2 | VLD02 | 0.6–1.8 V ±3% accuracy default 1.2 V | 100 mA | 1.0 µF | DVC, digital LDO, 25 mV steps, DVC ramp with controlled slew rater, pull-down resistor switch off |
| LDO3 | VLD03 | 1.725–3.3 V ±3% accuracy default 2.85 V | 200 mA | 2.2 µF | Digital LDO, 25 mV steps, DVC with controlled slew rate, common supply with LDO4 |
| LDO4 | VLD04 | 1.725–3.3 V ±3% accuracy default 2.85 V | 150 mA | 2.2 µF | Digital LDO, 25 mV steps, optional hardware control from GPI1, common supply with LDO3 |
| LDO5 | VLD05 | 1.2–3.6 V ±3% accuracy default 3.1 V | 100 mA | 1.0 µF | Digital LDO, 50 mV steps, pull-down resistor switch off, optional hardware control from GPI2 |
| LDO6 | VLD06 | 1.2–3.6 V ±3% accuracy default 1.2 V | 150 mA | 2.2 µF | High PSSR, low noise, 50 mV steps |
| LDO7 | VLD07 | 1.2–3.6 V ±3% accuracy default 3.1 V | 200 mA | 2.2 µF | High PSSR, low noise, 50 mV steps, common supply with LDO8 |
| LDO8 | VLD08 | 1.2–3.6 V ±3% accuracy default 2.85 V | 200 mA | 2.2 µF | High PSSR, low noise, 50 mV steps, common supply with LDO7 |

**Table 5-3. Generated Supply Domains (continued)**

| Regulator | Supplied pins | Supplied voltage | Supplied max current | External component | Notes |
|---|---|---|---|---|---|
| LDO9 | VLD09 | 1.25–3.6 V ±3% accuracy default 2.5 V | 100 mA | 1.0 μF | High PSSR, low noise, 50 mV steps, TOP trimmed, optional hardware control from GPI12, common supply with LDO10 |
| LDO10 | VLD10 | 1.2–3.6 V ±3% accuracy default 1.8 V | 250 mA | 2.2 μF | High PSSR, low noise, 50 mV steps, common supply with LDO9 |
| Backup | VBBAT | 1.1–3.1 V ±3% accuracy default 3.0 V | 6 mA | 470 nF | 100/200 mV steps, configurable current limit between 100 and 6000 μA, reverse current protection |
| LDOCore | Internal PMIC supply | 2.5 V ±2% accuracy | 4 mA | 100 nF | Not for external use |

### 5.6.2 LTC3589-1

The LTCR3589 is a complete solution that fulfills the i.MX53 power management needs. It includes:

- 3 buck converters for the core, memory, and SoC rails
- A synchronous buck-boost regulator for I/O
- 3 250mA LDO regulators
- An $I^2C$ serial port for communication with processor
- Always Alive LDO regulator for RTC voltage domain

Figure 5-13 shows a typical application block guide.

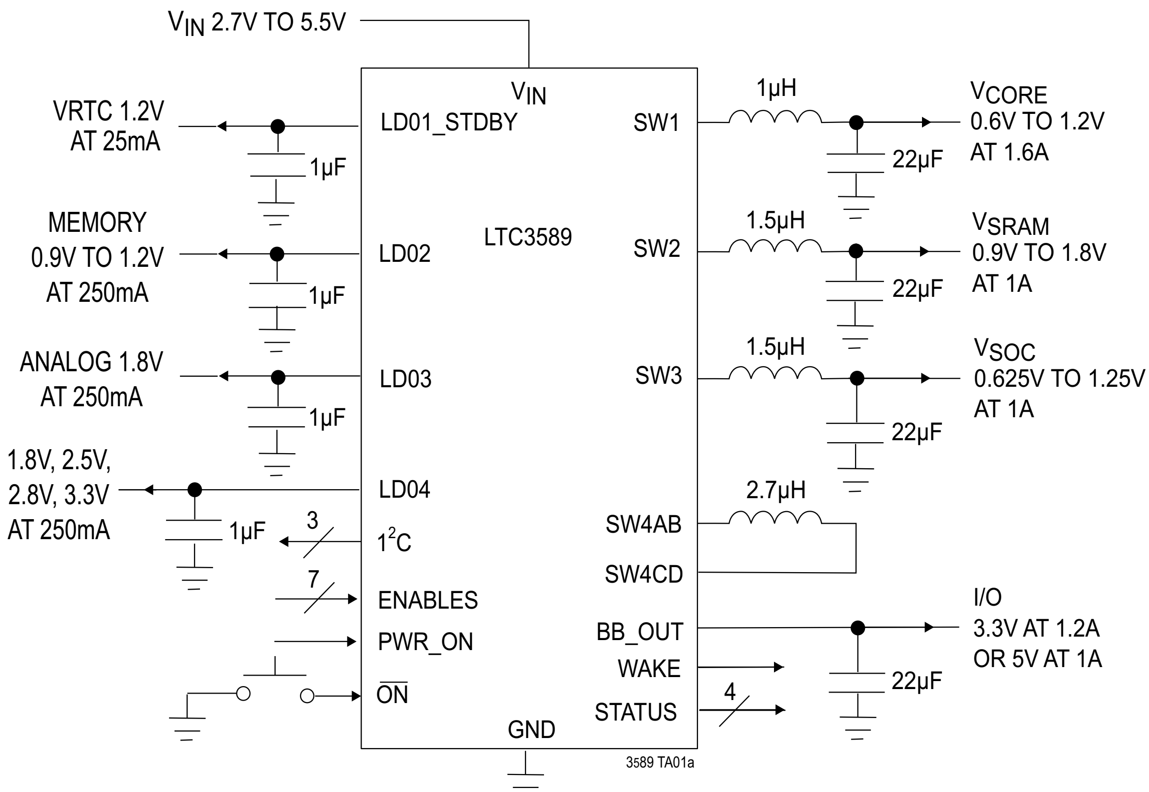



**Figure 5-13. LTC3589-1 Typical Application Block Guide**

Table 5-4 shows the supply domains.

**Table 5-4. LTC3589-1 Supply Domains**

| Source | Voltage (V) | Current (mA) |
|---|---|---|
| LDO1 (Always on) | Set with external resistor divider, as low as 0.8 | 25 |
| LDO2 | Set with external resistor divider and internal register configuration | 250 |
| LDO3 | 2.8 (Fixed) | 250 |
| LDO4 | 1.2, 1.8, 2.5, 3.2 (Configured via $I^2C$) | 250 |
| SW1 | Set with external resistor divider and internal register configuration from 0.6 to 1.2 | 1600 |
| SW2 | Set with external resistor divider and internal register configuration from 0.9 to 1.8 | 1000 |
| SW3 | Set with external resistor divider and internal register configuration from 0.625 to 1.25 | 1000 |
| SW4 | Set with external resistor divider, as low as 0.8 | 1200 |

# Chapter 6
# Interfacing DDR2 and DDR3 Memories with the i.MX53 Processor

This chapter explains the interface between the i.MX53 processor and DDR2 and DDR3 memories. It includes the routing guidelines, pictures, and examples.

## 6.1 i.MX53 SDRAM Controller Signals

The SDRAM controller can be interfaced with LPDDR2-S, DDR2, and DDR3 memories. The DDR controller from the i.MX53 uses the following signals to interface the memories:

- Data bus and its buffer control signals
  - DRAM_D0 – DRAM_D31.
  - DRAM_DQS0/DQS0_B - DRAM_DQS3/DQS3_B.
  - DRAM_DQM0 – DRAM_DQM3.
- Address bus and its bank control signals
  - DRAM_A0- DRAM_A15
  - DRAM_SDBA0- DRAM_SDBA2
- Control
  - DRAM_RAS
  - DRAM_CAS
  - DRAM_SDWE
  - DRAM_RESET
  - DRAM_CALIBRATION
  - DRAM_SDCKE0 - DRAM_SDCKE1
  - DRAM_CS0 – DRAM_CS1
  - DRAM_SDODT0 – DRAM_SDODT1
- Clock
  - DRAM_SDCLK_0
  - DRAM_SDCLK_0_B
  - DRAM_SDCLK_1
  - DRAM_SDCLK_1_B

Figure 6-1 shows the block diagram of the DDR2/DDR3 interfaced with the i.MX53 from the reference design boards.

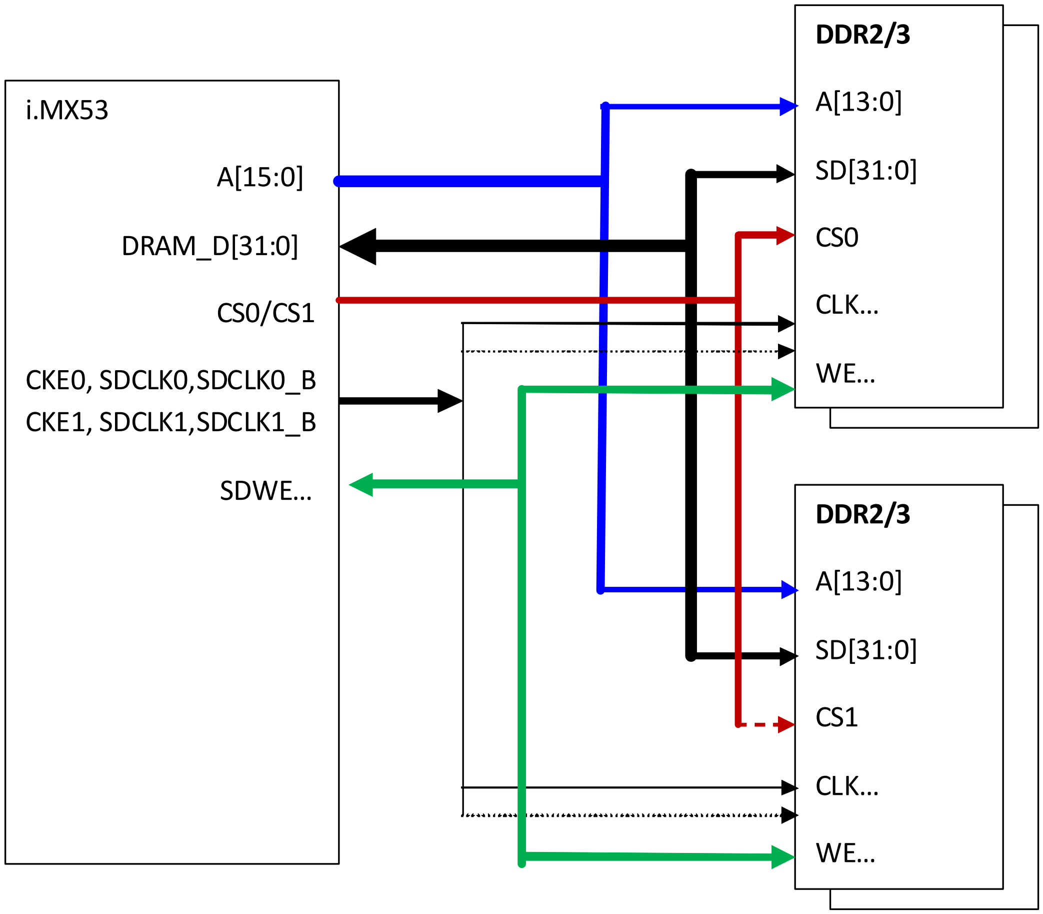



**Figure 6-1. Connection Between i.MX53 Processor and DDR2 and DDR3**

## 6.2 i.MX53 Memory Interface

Figure 6-2 shows the DDR2 connection. The DDR2 device is the H5PS2G83AFR.

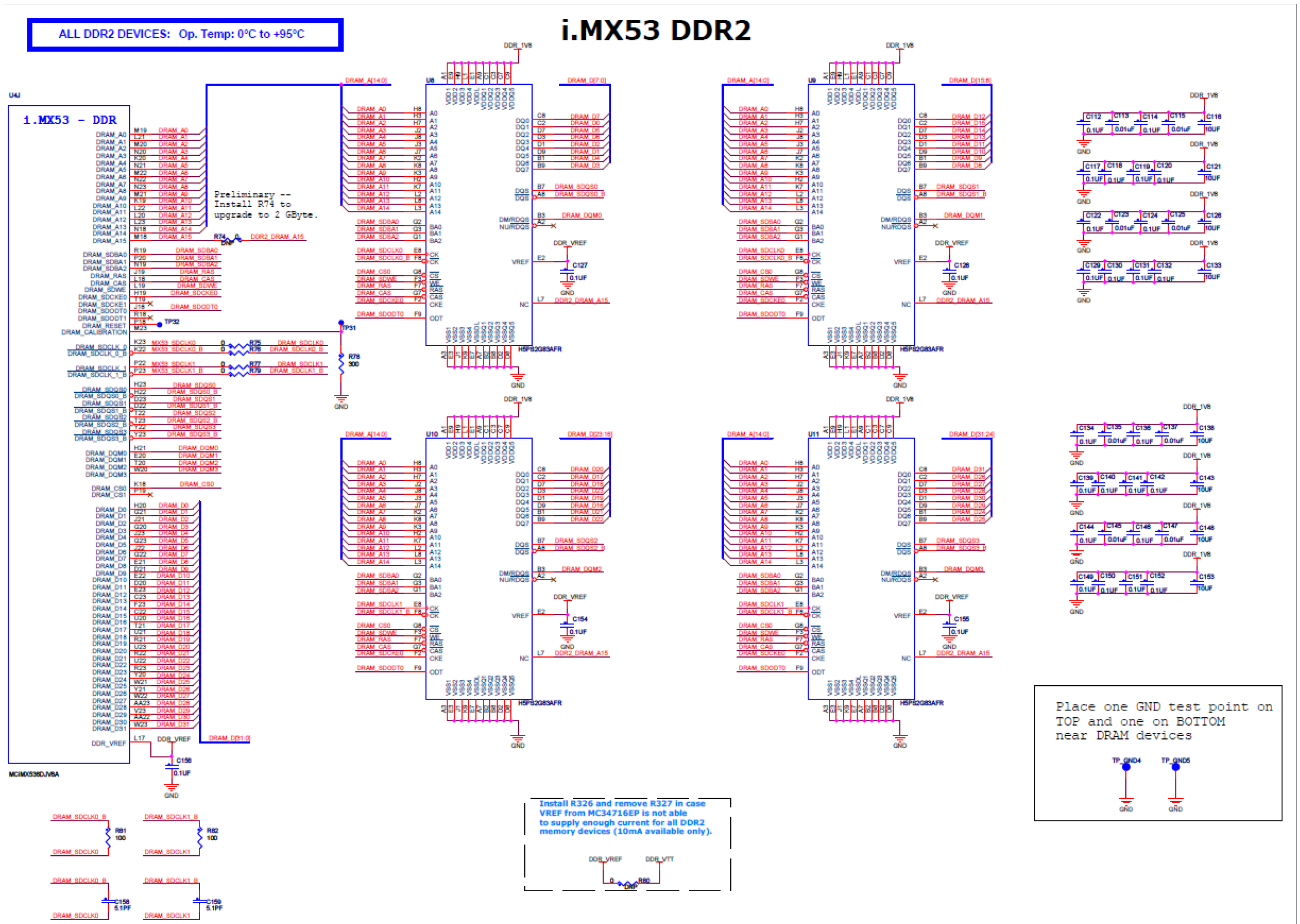



**Figure 6-2. DDR2 Memory Connection**

Figure 6-3 shows the DDR3 memory connections. The DDR3 device is the EDJ2116DASE.

The DDR2 and DDR3 memory connections differ in the following ways:

- RESET and VREF signals.
- DDR3 DQS signals are connected as differential pairs to the memory.

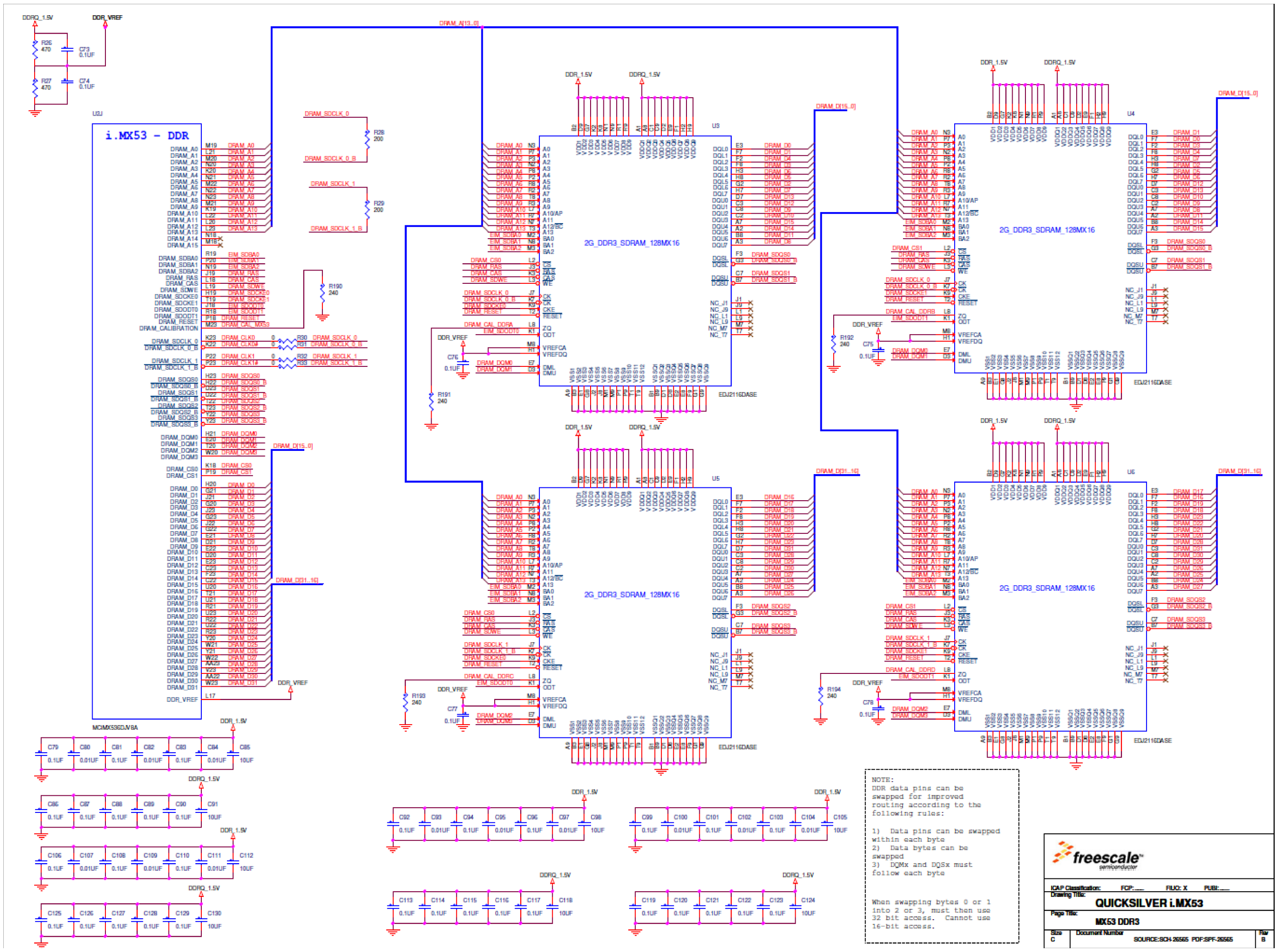



**Figure 6-3. DDR3 Memory Connection**

## 6.3 Configuring the DDR2 JTAG Script

The following code shows an example of how to configure the DDR2 memory for the i.MX53 processor:

**Example 6-1. DDR2 JTAG Script Configuration**

```
//*==============================================================================================
======
//* Copyright (C) 2010, Freescale Semiconductor, Inc. All Rights Reserved
//* THIS SOURCE CODE IS CONFIDENTIAL AND PROPRIETARY AND MAY NOT
//* BE USED OR DISTRIBUTED WITHOUT THE WRITTEN PERMISSION OF
//* Freescale Semiconductor, Inc.
//*==============================================================================================
======
// Initialization script for Rita CPU2 Board
// Version 1.0
//*==============================================================================================
======
```

```
wait = on

//*==================================================================================
======
// init ARM
//*==================================================================================
======

//*==================================================================================
======
// Disable WDOG
//*==================================================================================
======
setmem /16 0x53f98000 = 0x30

//*==================================================================================
======
// Program PLL2 to 300MHz
//*==================================================================================
======
setmem /32 0x63f84004 = 0x4         // disable PLL2 automatic restart
setmem /32 0x63f84000 = 0x1222
setmem /32 0x63f8400c = 0x3e7
setmem /32 0x63f84010 = 0xfa
setmem /32 0x63f84008 = 0x60

setmem /32 0x53fd4018 = 0x00015154
setmem /32 0x53fd4014 = 0x03119184  // switch peripherals to PLL3
pause 1
setmem /32 0x63f84000 = 0x1232      // restart PLL2
pause 1
setmem /32 0x53fd4014 = 0x01119184
setmem /32 0x53fd4018 = 0x00016154  // switch peripherals back to PLL2
pause 1
setmem /32 0x63f84004 = 0x6         // re-enable PLL2 automatic restart

//*==================================================================================
======
// Enable all clocks (they are disabled by ROM code)
//*==================================================================================
======
setmem /32 0x53fd4068 = 0xffffffff
setmem /32 0x53fd406c = 0xffffffff
setmem /32 0x53fd4070 = 0xffffffff
setmem /32 0x53fd4074 = 0xffffffff
setmem /32 0x53fd4078 = 0xffffffff
setmem /32 0x53fd407c = 0xffffffff
setmem /32 0x53fd4080 = 0xffffffff
setmem /32 0x53fd4084 = 0xffffffff

//*==================================================================================
======
// Initialization script for 32 bit DDR2 (CS0+CS1)
//*==================================================================================
======

// DDR2 IOMUX configuration
```

```
setmem /32 0x53fa8554 = 0x00380000  //IOMUXC_SW_PAD_CTL_PAD_DRAM_DQM3
setmem /32 0x53fa8558 = 0x00380040  //IOMUXC_SW_PAD_CTL_PAD_DRAM_SDQS3
setmem /32 0x53fa8560 = 0x00380000  //IOMUXC_SW_PAD_CTL_PAD_DRAM_DQM2
setmem /32 0x53fa8564 = 0x00380040  //IOMUXC_SW_PAD_CTL_PAD_DRAM_SDODT1
setmem /32 0x53fa8568 = 0x00380040  //IOMUXC_SW_PAD_CTL_PAD_DRAM_SDQS2
setmem /32 0x53fa8570 = 0x00200000  //IOMUXC_SW_PAD_CTL_PAD_DRAM_SDCLK_1 - boazp: weaker sdclk
to improve EVK DDR max frequency
setmem /32 0x53fa8574 = 0x00380000  //IOMUXC_SW_PAD_CTL_PAD_DRAM_CAS
setmem /32 0x53fa8578 = 0x00200000  //IOMUXC_SW_PAD_CTL_PAD_DRAM_SDCLK_0- boazp: weaker sdclk
to improve EVK DDR max frequency
setmem /32 0x53fa857c = 0x00380040  //IOMUXC_SW_PAD_CTL_PAD_DRAM_SDQS0
setmem /32 0x53fa8580 = 0x00380040  //IOMUXC_SW_PAD_CTL_PAD_DRAM_SDODT0
setmem /32 0x53fa8584 = 0x00380000  //IOMUXC_SW_PAD_CTL_PAD_DRAM_DQM0
setmem /32 0x53fa8588 = 0x00380000  //IOMUXC_SW_PAD_CTL_PAD_DRAM_RAS
setmem /32 0x53fa8590 = 0x00380040  //IOMUXC_SW_PAD_CTL_PAD_DRAM_SDQS1
setmem /32 0x53fa8594 = 0x00380000  //IOMUXC_SW_PAD_CTL_PAD_DRAM_DQM1
setmem /32 0x53fa86f0 = 0x00380000  //IOMUXC_SW_PAD_CTL_GRP_ADDDS
setmem /32 0x53fa86f4 = 0x00000200  //IOMUXC_SW_PAD_CTL_GRP_DDRMODE_CTL
setmem /32 0x53fa86fc = 0x00000000  //IOMUXC_SW_PAD_CTL_GRP_DDRPKE
setmem /32 0x53fa8714 = 0x00000000  //IOMUXC_SW_PAD_CTL_GRP_DDRMODE - CMOS mode
setmem /32 0x53fa8718 = 0x00380000  //IOMUXC_SW_PAD_CTL_GRP_B0DS
setmem /32 0x53fa871c = 0x00380000  //IOMUXC_SW_PAD_CTL_GRP_B1DS
setmem /32 0x53fa8720 = 0x00380000  //IOMUXC_SW_PAD_CTL_GRP_CTLDS
setmem /32 0x53fa8724 = 0x06000000  //IOMUXC_SW_PAD_CTL_GRP_DDR_TYPE - DDR_SEL=0
setmem /32 0x53fa8728 = 0x00380000  //IOMUXC_SW_PAD_CTL_GRP_B2DS
setmem /32 0x53fa872c = 0x00380000  //IOMUXC_SW_PAD_CTL_GRP_B3DS

// Initialize DDR2 memory - Hynix H5PS2G83AFR
setmem /32 0x63fd9088 = 0x2b2f3031
setmem /32 0x63fd9090 = 0x40363333
setmem /32 0x63fd9098 = 0x00000f00 //boazp: add 3 logic unit of delay to sdclk to improve EVK
DDR max frequency
setmem /32 0x63fd90F8 = 0x00000800
setmem /32 0x63fd907c = 0x01310132
setmem /32 0x63fd9080 = 0x0133014b
// Enable bank interleaving, RALAT = 0x3, DDR2_EN = 1
setmem /32 0x63fd9018 = 0x000016d0
// Enable CSD0 and CSD1, row width = 15, column width = 10, burst length = 4, data width = 32bit
setmem /32 0x63fd9000 = 0xc4110000
// tRFC = 78 ck, tXS = 82 ck, tXP = 2 ck, tXPDLL(tXARD) = 2 ck, tFAW = 14 ck, CAS latency = 5 ck
setmem /32 0x63fd900C = 0x4d5122d2
// tRCD = 5 ck, tRP = 5 ck, tRC = 23 ck, tRAS = 18 ck, tRPA = 1, tWR = 6 ck, tMRD = 2 ck, tCWL = 4 ck
setmem /32 0x63fd9010 = 0x92d18a22
// tDLLK(tXSRD) = 200 cycles, tRTP = 3 ck, tWTR = 3ck, tRRD = 3ck
setmem /32 0x63fd9014 = 0x00c70092
setmem /32 0x63fd902c = 0x000026d2
setmem /32 0x63fd9030 = 0x009f000e
setmem /32 0x63fd9008 = 0x12272000
setmem /32 0x63fd9004 = 0x00030012
setmem /32 0x63fd901c = 0x04008010
setmem /32 0x63fd901c = 0x00008032
setmem /32 0x63fd901c = 0x00008033
setmem /32 0x63fd901c = 0x00008031
setmem /32 0x63fd901c = 0x0b5280b0
setmem /32 0x63fd901c = 0x04008010
setmem /32 0x63fd901c = 0x00008020
setmem /32 0x63fd901c = 0x00008020
```

```
setmem /32 0x63fd901c = 0x0a528030  // BL = 4, CAS latency = 5, write recovery = 6
setmem /32 0x63fd901c = 0x03c68031
setmem /32 0x63fd901c = 0x00468031  // reduced drive strength, enable 50ohm ODT
setmem /32 0x63fd901c = 0x04008018
setmem /32 0x63fd901c = 0x0000803a
setmem /32 0x63fd901c = 0x0000803b
setmem /32 0x63fd901c = 0x00008039
setmem /32 0x63fd901c = 0x0b528138
setmem /32 0x63fd901c = 0x04008018
setmem /32 0x63fd901c = 0x00008028
setmem /32 0x63fd901c = 0x00008028
setmem /32 0x63fd901c = 0x0a528038 // BL = 4, CAS latency = 5, write recovery = 6
setmem /32 0x63fd901c = 0x03c68039
setmem /32 0x63fd901c = 0x00468039 // reduced drive strength, enable 50ohm ODT
setmem /32 0x63fd9020 = 0x00005800
setmem /32 0x63fd9058 = 0x00033337 // Enable 50ohm ODT
setmem /32 0x63fd901c = 0x00000000
```

## 6.4 Configuring the DDR3 JTAG Script

The following code shows an example of how to configure DDR3 memory for the i.MX53 processor:

**Example 6-2. DDR3 JTAG Script Configuration**

```
//*=============================================================================
======
//* Copyright (C) 2010, Freescale Semiconductor, Inc. All Rights Reserved
//* THIS SOURCE CODE IS CONFIDENTIAL AND PROPRIETARY AND MAY NOT
//* BE USED OR DISTRIBUTED WITHOUT THE WRITTEN PERMISSION OF
//* Freescale Semiconductor, Inc.
//*=============================================================================
======
// Initialization script for Rita Quick Silver Board, DDR3
// Version 1.0
//*=============================================================================
======

wait = on

//*=============================================================================
======
// init ARM
//*=============================================================================
======

//*=============================================================================
======
// Disable WDOG
//*=============================================================================
======
setmem /16 0x53f98000 = 0x30

//*=============================================================================
======
// Enable all clocks (they are disabled by ROM code)
```

```
//*=========================================================================================
======
setmem /32 0x53fd4068 = 0xffffffff
setmem /32 0x53fd406c = 0xffffffff
setmem /32 0x53fd4070 = 0xffffffff
setmem /32 0x53fd4074 = 0xffffffff
setmem /32 0x53fd4078 = 0xffffffff
setmem /32 0x53fd407c = 0xffffffff
setmem /32 0x53fd4080 = 0xffffffff
setmem /32 0x53fd4084 = 0xffffffff

//*=========================================================================================
======
// Initialization script for 32 bit DDR2 (CS0+CS1)
//*=========================================================================================
======

// DDR3 IOMUX configuration
//* Global pad control options */
setmem /32 0x53fa86f4 = 0x00000000  //IOMUXC_SW_PAD_CTL_GRP_DDRMODE_CTL for sDQS[3:0], 1=DDR2,
0=CMOS mode
setmem /32 0x53fa8714 = 0x00000000  //IOMUXC_SW_PAD_CTL_GRP_DDRMODE for D[31:0], 1=DDR2, 0=CMOS
mode
setmem /32 0x53fa86fc = 0x00000000  //IOMUXC_SW_PAD_CTL_GRP_DDRPKE

setmem /32 0x53fa8724 = 0x04000000  //IOMUXC_SW_PAD_CTL_GRP_DDR_TYPE - DDR_SEL=10
// setmem /32 0x53fa8724 = 0x00000000  //IOMUXC_SW_PAD_CTL_GRP_DDR_TYPE - DDR_SEL=00
// setmem /32 0x53fa8724 = 0x02000000  //IOMUXC_SW_PAD_CTL_GRP_DDR_TYPE - DDR_SEL=01
// setmem /32 0x53fa8724 = 0x06000000  //IOMUXC_SW_PAD_CTL_GRP_DDR_TYPE - DDR_SEL=11

//* Data bus byte lane pad drive strength control options */
setmem /32 0x53fa872c = 0x00300000  //IOMUXC_SW_PAD_CTL_GRP_B3DS
setmem /32 0x53fa8554 = 0x00300000  //IOMUXC_SW_PAD_CTL_PAD_DRAM_DQM3
setmem /32 0x53fa8558 = 0x00300040  //IOMUXC_SW_PAD_CTL_PAD_DRAM_SDQS3

setmem /32 0x53fa8728 = 0x00300000  //IOMUXC_SW_PAD_CTL_GRP_B2DS
setmem /32 0x53fa8560 = 0x00300000  //IOMUXC_SW_PAD_CTL_PAD_DRAM_DQM2
setmem /32 0x53fa8568 = 0x00300040  //IOMUXC_SW_PAD_CTL_PAD_DRAM_SDQS2

setmem /32 0x53fa871c = 0x00300000  //IOMUXC_SW_PAD_CTL_GRP_B1DS
setmem /32 0x53fa8594 = 0x00300000  //IOMUXC_SW_PAD_CTL_PAD_DRAM_DQM1
setmem /32 0x53fa8590 = 0x00300040  //IOMUXC_SW_PAD_CTL_PAD_DRAM_SDQS1

setmem /32 0x53fa8718 = 0x00300000  //IOMUXC_SW_PAD_CTL_GRP_B0DS
setmem /32 0x53fa8584 = 0x00300000  //IOMUXC_SW_PAD_CTL_PAD_DRAM_DQM0
setmem /32 0x53fa857c = 0x00300040  //IOMUXC_SW_PAD_CTL_PAD_DRAM_SDQS0

//* SDCLK pad drive strength control options */
setmem /32 0x53fa8578 = 0x00300000  //IOMUXC_SW_PAD_CTL_PAD_DRAM_SDCLK_0
setmem /32 0x53fa8570 = 0x00300000  //IOMUXC_SW_PAD_CTL_PAD_DRAM_SDCLK_1

//* Control and addr bus pad drive strength control options */
setmem /32 0x53fa8574 = 0x00300000  //IOMUXC_SW_PAD_CTL_PAD_DRAM_CAS
setmem /32 0x53fa8588 = 0x00300000  //IOMUXC_SW_PAD_CTL_PAD_DRAM_RAS
setmem /32 0x53fa86f0 = 0x00300000  //IOMUXC_SW_PAD_CTL_GRP_ADDDS for DDR addr bus
setmem /32 0x53fa8720 = 0x00300000  //IOMUXC_SW_PAD_CTL_GRP_CTLDS for CSD0, CSD1, SDCKE0,
SDCKE1, SDWE
```

```
setmem /32 0x53fa8564 = 0x00300040  //IOMUXC_SW_PAD_CTL_PAD_DRAM_SDODT1
setmem /32 0x53fa8580 = 0x00300040  //IOMUXC_SW_PAD_CTL_PAD_DRAM_SDODT0

// Initialize DDR3 memory - Micron MT41J128M16-187E
//** Keep for now, same setting as CPU3 board **//
//setmem /32 0x63fd904c = 0x01680172 //write leveling reg 0
//setmem /32 0x63fd9050 = 0x0021017f //write leveling reg 1
setmem /32 0x63fd9088 = 0x32383535 //read delay lines
setmem /32 0x63fd9090 = 0x40383538 //write delay lines
//setmem /32 0x63fd90F8 = 0x00000800 //Measure unit
setmem /32 0x63fd907c = 0x0136014d //DQS gating 0
setmem /32 0x63fd9080 = 0x01510141 //DQS gating 1

//* CPU3 Board setting
// Enable bank interleaving, Address mirror on, WALAT = 0x1, RALAT = 0x5, DDR2_EN = 0
//setmem /32 0x63fd9018 = 0x00091740 //Misc register:

//* Quick Silver board setting
// Enable bank interleaving, Address mirror off, WALAT = 0x1, RALAT = 0x5, DDR2_EN = 0
setmem /32 0x63fd9018 = 0x00011740 //Misc register


// Enable CSD0 and CSD1, row width = 14, column width = 10, burst length = 8, data width = 32bit
setmem /32 0x63fd9000 = 0xc3190000 //Main control register
// tRFC=64ck;tXS=68;tXP=3;tXPDLL=10;tFAW=15;CAS=6ck
setmem /32 0x63fd900C = 0x555952E3 //timing configuration Reg 0.
// tRCD=6;tRP=6;tRC=21;tRAS=15;tRPA=1;tWR=6;tMRD=4;tCWL=5ck
setmem /32 0x63fd9010 = 0xb68e8b63 //timing configuration Reg 1
// tDLLK(tXSRD)=512 cycles; tRTP=4;tWTR=4;tRRD=4
setmem /32 0x63fd9014 = 0x01ff00db //timing configuration Reg 2
setmem /32 0x63fd902c = 0x000026d2 //command delay (default)
setmem /32 0x63fd9030 = 0x009f0e21 //out of reset delays
// Keep tAOFPD, tAONPD, tANPD, and tAXPD as default since they are bigger than calc values
setmem /32 0x63fd9008 = 0x12273030 //ODT timings
// tCKE=3; tCKSRX=5; tCKSRE=5
setmem /32 0x63fd9004 = 0x0002002d //Power down control

//**********************************
//DDR device configuration:
//**********************************
//**********************************
// CS0:
//**********************************
setmem /32 0x63fd901c = 0x00008032 //write mode reg MR2 with cs0 (see below for settings)
// Full array self refresh
// Rtt_WR disabled (no ODT at IO CMOS operation)
// Manual self refresh
// CWS=5

setmem /32 0x63fd901c = 0x00008033 //write mode reg MR3 with cs0 .
setmem /32 0x63fd901c = 0x00028031 //write mode reg MR1 with cs0. ODS=01: out buff= RZQ/7 (see
below for settings)
// out impedance = RZQ/7
// Rtt_nom disabled (no ODT at IO CMOS operation)
// Aditive latency off
// write leveling disabled
// tdqs (differential?) disabled
```

```
setmem /32 0x63fd901c = 0x092080b0 //write mode reg MR0 with cs0 , with dll_rst0
setmem /32 0x63fd901c = 0x04008040 //ZQ calibration with cs0 (A10 high indicates ZQ cal long
ZQCL)

//***********************************
// CS1:
//***********************************
setmem /32 0x63fd901c = 0x0000803a //write mode reg MR2 with cs1 .
setmem /32 0x63fd901c = 0x0000803b //write mode reg MR3 with cs1
setmem /32 0x63fd901c = 0x00028039 //write mode reg MR1 with cs1. ODS=01: out buff= RZQ/7
setmem /32 0x63fd901c = 0x09208138 //write mode reg MR0 with cs1
setmem /32 0x63fd901c = 0x04008048 //ZQ calibration with cs1 (A10 high indicates ZQ cal long
ZQCL)
//***********************************

setmem /32 0x63fd9020 = 0x00001800 // Refresh control register
setmem /32 0x63fd9040 = 0x04b80003 // ZQ HW control
setmem /32 0x63fd9058 = 0x00022227 // ODT control register

setmem /32 0x63fd901c = 0x00000000

// CLKO muxing (comment out for now till needed to avoid conflicts with intended usage of
signals)
//setmem /32 0x53FA8314 = 0
//setmem /32 0x53FA8320 = 0x4
//setmem /32 0x53FD4060 = 0x01e900f0
```

## 6.5 Configuring the i.MX53 Registers for the Initialization Script

This section explains how to configure the registers of the i.MX53 for the initialization script, using values taken from the Micron MT41J128M16-187E memory data sheet as the example. Therefore, in this example CK = 2.5 ns.

### 6.5.1 Main Control Register

Figure 6-4 shows the main control register's bit fields, access, and reset values.

Access: Read/Write

| Bits | 31 | 30 | 29–27 | 26–24 | 23–22 | 21–20 | 19 | 18–17 | 16 | 15–0 |
|---|---|---|---|---|---|---|---|---|---|---|
| R/W | SDE_0 | SDE_1 | — | ROW | — | COL | BL | — | DSIZ | — |
| Reset | 0 | 0 | 0 0 0 | 0 1 1 | 0 0 | 0 1 | 0 | 0 0 | 1 | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 |

**Figure 6-4. Main Control Register**

The memory values are as follows:

- ROW = 14
- COL = 10

DDR3 only supports a burst length of 8. Therefore, BL = 8 burst length.

The SCh has a 32 bit data bus. Therefore, DSIZ = 32 bit width.

Enable the SDRAM controller as follows:

```
setmem /32 0x63FD9000 = 0xC3190000
```

### 6.5.2 Power Down Register

Figure 6-5 shows the power down register's bit fields, access, and reset values.

Access: Read/Write

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R/W | | PRCT_1 | | | | PRCT_0 | | | — | | | | | tCKE | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R/W | PWDT_1 | | | | PWDT_0 | | | | SLOW_PD | BOTH_CS_PD | tCKSRX | | | tCKSRE | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |

**Figure 6-5. Power Down Register**

Values are as follows:

- tCKE = 3 CK
- tCKSRE = greater than 5 CK
- tCKSRX = greater than 5 CK

Enable as follows:

```
setmem /32 0x63FD9004 = 0x0002002D
```

### 6.5.3 Timing Configuration 0 Register

Figure 6-6 shows the timing configuration 0 register's bit fields, access, and reset values.

Access: Read/Write

| | 31–24 | 23–16 | 15–13 | 12–9 | 8–4 | 3–0 |
|---|---|---|---|---|---|---|
| R/W | tRFC | tXS | tXP | tXPDLL | tFAW | tCL |
| Reset | 0 0 1 1 0 0 1 0 | 0 0 1 1 0 1 1 0 | 0 0 1 | 0 0 1 0 | 0 1 1 0 1 | 0 0 1 1 |

**Figure 6-6. Timing Configuration 0 Register**

Values are as follows:

- tRFC = 86 CK
- tXS = tRFC + 10 ns = 90 CK
- tXP = greater of 3 CK

- tXPDLL = greater of 10 CK
- tFAW= 15 CK
- tCL= 6

Enable as follows:

```
setmem /32 0x63FD900C = 0x555952E3
```

### 6.5.4 Timing Configuration 1 Register

Figure 6-7 shows the timing configuration 1 register’s bit fields, access, and reset values.

Access: Read/Write

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R/W | tRCD | | | tRP | | | tRC | | | | | tRAS | | | | |
| Reset | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 |

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R/W | tRPA | — | | | tWR | | | tMRD | | | | — | — | tCWL | | |
| Reset | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |

**Figure 6-7. Timing Configuration 1 Register**

Values are as follows:

- tRCD = 6 CK
- tRP = 6 CK
- tRC = 21 CK
- tRAS = 15 CK
- tRPA = tRP + 1
- tWR – 15 ns = 6CK
- tMRD = 12CK
- tCWL = 5CK

Enable as follows:

```
setmem /32 0x63FD9010 = 0xB68E8B63
```

### 6.5.5 Timing Configuration 2 Register

Figure 6-7 shows the timing configuration 2 register's bit fields, access, and reset values.

Access: Read/Write

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | — | | | | | | | tDLLK | | | | | | | | | — | | | | | | | tRTP | | | tWTR | | | tRRD | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |

**Figure 8. ESDCTL Timing Configuration Register 2(ESDCFG2)**

Values are as follows:

- tDLLK = 512 CK
- tRTP = Greater than 4 CK
- tWTR = Greater than 4 CK
- tRRD = Greater than 4 CK

Enable as follows:

```
setmem /32 0x63FD9014 = 0x01FF00DB
```

# Chapter 7
# Avoiding Board Bring-Up Problems

This chapter provides recommendations for avoiding typical mistakes when bringing up a board for the first time. These recommendations consist of basic techniques that have proven useful in the past for detecting board issues and address the three most typical bring-up pitfalls: power, clocks, and reset. A sample bring-up checklist is provided at the end of the chapter.

## 7.1 Using a Voltage Report to Avoid Power Pitfalls

Using incorrect voltage rails is a common power pitfall. To help avoid this mistake, create a basic table called a voltage report prior to bringing up your board. This table helps validate that your supplies are coming to the expected level.

To create a voltage report, list the following:

- Your board voltage sources
- Default power-up values for the board voltage sources
- Best place on the board to measure the voltage level of each supply

Be careful when determining the best place to measure each supply. In some cases, a large voltage drop (IR drop) on the board may cause you to measure inaccurate levels depending on the location you take your measurement. The following guidelines help prevent this:

- Measure closest to the load (in this case the i.MX53 processor).
- Make two measurements: the first after initial board power-up and the second while running a heavy use-case that stresses the i.MX53.

The supplies that are powering the i.MX53 should all meet the DC electrical specifications as listed in the i.MX53 data sheet.

Table 7-1 shows a sample voltage report table. Blank cells would be filled in after measuring.

**Table 7-1. Sample Voltage Report**

| Regulator | Net Name on Schematic | Default Power Up (V) | Measured Voltage (V) | Measurement Point | Comment |
|---|---|---|---|---|---|
| — | VBAT | 12 | | J1 pin 1 | |
| Wall supply | 5V_MAIN | 5 | | J2 pin 4 | |
| Switcher 1 | 1V8_MAIN | 1.8 | | C11 | 0402, near U3, inch below j37 |
| Switcher 2 | 3V3_MAIN | 3.3 | | C14 | 0603, right next to C11 |

## 7.2 Using a Current Monitor to Avoid Power Pitfalls

Excessive current can cause damage to the board. Avoid this problem by using a current-limiting laboratory supply that has a current read-out to power the main power to the board when bringing up the board for the first time. This allows the main power to be monitored, which makes it easy to detect any excessive current.

## 7.3 Checking for Clock Pitfalls

Problems with the external clocks are another common source of board bring-up issues. Ensure that all of your clock sources are running as expected.

The EXTAL/XTAL and the ECKIL/CKIL clocks are the main clock sources for 24 MHz and 32 kHz reference clocks respectively on the i.MX53. Although not required, the use of low jitter external oscillators to feed CKIH1 or CKIH2 on the i.MX53 can be an advantage if low jitter or special frequency clock sources are required by modules driven by CKIH1 or CKIH2. See the CCM chapter in the i.MX53 reference manual for details.

When checking crystal frequencies, use an active probe to avoid excessive loading. A parasitic probe typically inhibits the 32.768 kHz oscillator from starting up. Use the following guidelines:

- CKIL clock should be running at 32.768 kHz (can be generated internally or applied externally)
- EXTAL/EXTAL should be running at 24 MHz (used for the PLL reference)
- CKIH1/CKIH2 can be used as oscillator inputs for low jitter special frequency sources.
- CKIH1 and CKIH2 are optional.

In addition to probing the external input clocks, you can check internal clocks by outputting them at the debug signals CLKO1 and CLKO2. See the CCM chapter in the i.MX53 reference manual for more details about which clock sources can be output to those debug signals.

## 7.4 Avoiding Reset Pitfalls

Follow these guidelines to ensure that you are booting using the correct boot mode.

- During initial power on while asserting the POR_B reset signal, ensure that both your reference clocks are active before releasing POR_B.
- Follow the recommended power-up sequence specified in the i.MX53 reference manual.

The GPIOs and internal fuses control the i.MX53 boots. For a more detailed description about the different boot modes, refer to the system boot chapter of the i.MX53 reference manual.

## 7.5 Sample Board Bring-Up Checklist

Table 7-2 provides a sample board bring-up checklist. Note that the checklist incorporates the recommendations described in the previous sections. Blank cells should be filled in during bring-up as appropriate.

**Table 7-2. Board Bring-Up Checklist**

| Checklist Item | Details | Owner | Findings & Status |
|---|---|---|---|
| **Note: The following items must be completed serially.** | | | |
| 1. Perform a visual inspection. | Check major components to make sure nothing has been misplaced or rotated before applying power. | | |
| 2. Verify all i.MX53 voltage rails. | Confirm that the voltages match the data sheet's requirements. Be sure to check voltages not only at the voltage source, but also as close to the i.MX53 as possible (like on a bypass capacitor). This reveals any IR drops on the board that will cause issues later.<br>Ideally all of the i.MX53 voltage rails should be checked, but VDDGP, VCC, and VDDA are particularly important voltages. These are the core logic voltages and must fall within the parameters provided in the i.MX53 data sheet.<br>NVCC_SRTC_POW, NVCC_XTAL, NVCC_CKIH, NVCC_RESET, NVCC_JTAG, and NVCC_EMI_DRAM are also critical to the i.MX53 boot up. | | |
| 3. Verify power up sequence. | Verify that power on reset (POR) is de-asserted (high) after all power rails have come up and are stable. Refer to the i.MX53 data sheet for details about power up sequencing. This is an important process as many complex processors are sensitive to the proper power up sequencing. | | |
| 4. Measure/probe input clocks (32 kHz, others). | Without a properly running clock, the i.MX53 will not function properly. Look for jitter and noise. | | |
| 5. Check JTAG connectivity (RV-ICE). | This is one of the most fundamental and basic access points to the i.MX53 to allow the debug and execution of low level code. | | |
| **Note: The following items may be worked on in parallel with other bring up tasks.** | | | |
| Access internal RAM. | Verify basic operation of the i.MX53 in system. The on-chip internal RAM starts at address 0xF800_0000 and is 128 Kbytes in density. Perform a basic test by performing a write-read-verify to the internal RAM. No software initialization is necessary to access internal RAM. | | |
| Verify CLKO outputs (measure and verify default clock frequencies for desired clock output options) if the board design supports probing of the CLKO pin. | This ensures that the corresponding clock is working and that the PLLs are working.<br>Note that this step requires chip initialization, for example via the JTAG debugger, to properly set up the IOMUX to output CLKO and to set up the clock control module to output the desired clock. Refer to the reference manual for more details. | | |

**Table 7-2. Board Bring-Up Checklist (continued)**

| Checklist Item | Details | Owner | Findings & Status |
|---|---|---|---|
| Measure boot mode frequencies. Set the boot mode switch for each boot mode and measure the following, (depending on system availability:<br>• NAND (probe CE to verify boot, measure RE frequency)<br>• SPI-NOR (probe slave select and measure clock frequency)<br>• MMC/SD (measure clock frequency) | This verifies the specified signals' connectivity between the i.MX53 and boot device and that the boot mode signals are properly set.<br>Refer to section "Boot Modes for the i.MX53" for details about configuring the various boot modes. | | |
| Run basic DDR initialization and test memory. | 1 Assuming the use of a JTAG debugger, run the DDR initialization and open a debugger memory window pointing to the DDR memory map starting address.<br>2 Try writing a few words and verify if they can be read correctly.<br>3 If not, recheck the DDR initialization sequence and whether the DDR has been correctly soldered onto the board.<br>It is also recommended that users recheck the schematic to ensure that the DDR memory has been connected to the i.MX53 correctly. | | |

# Chapter 8
# Using the Clock Connectivity Table

This chapter explains how to use the i.MX53 clocking connectivity. This information can help users save power by disabling clocks to unused modules.

Table 8-1 describes the available clock sources and lists the maximum frequencies that are supported by design. In some cases if maximum frequency is used, users need to divide the clock inside the module in order to meet the protocol requirements. The clock controller module (CCM) generates and drives the clock sources.

For information about how the root clocks are generated, see the clock generation diagrams in the CCM chapter of the i.MX53 reference manual. In some cases, the CCM does not generate the clock, and the clock may come directly from the IO pad.

**Table 8-1. Clock Roots**

| Clock Root Name (from CCM) | Description | Target Frequency [MHz] |
|---|---|---|
| arm_clk_root | Root of ARM high frequency | 1000 |
| arm_axi_clk_root | Root for ARM AXI clock | 200 |
| emi_slow_clk_root | Root for EMI slow arbitrator | 133 |
| debug_apb_clk_root | Root for debug busses of ARM | 200 |
| ddr_clk_root | Root of DDR clock | 400 |
| enfc_clk_root | Root for NFC controller | 66.5 |
| vpu_axi_clk_root | Root for VPU AXI clock | 200 |
| vpu_rclk_root | Root for reference clock for VPU | 66.5 |
| spdif0_clk_root | Root of SPDIF-0 clock | 66.5 |
| spdif1_clk_root | Root of SPDIF-1 clock | 66.5 |
| ahb_clk_root | Root of AHB clock | 133 |
| ipg_clk_root | Root of IPG clock | 66.5 |
| asrc_clk_root | — | 66.5 |
| perclk_root | Root of PERCLK | 66.5 |
| usboh3_clk_root | Root of USB clock | 66.5 |
| esdhc1_clk_root | Root clock for ESDHC-1 and MSHC-1 | 104 |
| esdhc2_clk_root | Root clock for ESDHC-2 | 104 |
| esdhc3_clk_root | Root for ESDHC-3 clock | 104 |
| esdhc4_clk_root | Root for ESDHC-4 clock | 104 |

**Table 8-1. Clock Roots (continued)**

| Clock Root Name (from CCM) | Description | Target Frequency [MHz] |
|---|---|---|
| ssi1_clk_root | Root for SSI-1 clock | 66.5 |
| ssi2_clk_root | Root for SSI-2 clock | 66.5 |
| ssi3_clk_root | Root for SSI-3 clock | 66.5 |
| usb_phy_clk_root | Root for USB_PHY (24 Mhz) | 24 |
| ieee_cemx_clk_root | Root for IEEE RTC clock | 66.5 |
| tve_216_54_clk_root | Root for TVE (216/54 Mhz) | 297 |
| di0_clk_root | Root for DI0 clock (for IPU) | 170 |
| di1_clk_root | Root for DI1 clock (for IPU) | 170 |
| ipg_clk_sync_ieee_root | Root for sync signal of IEEE RTC module | 66.5 |
| ldb_di0_serial_clk_root | Root clock for LDB bridge | 595 |
| ldb_di1_serial_clk_root | Root clock for LDB bridge | 595 |
| ecspi_clk_root | Root for CSPI clock | 66.5 |
| uart_clk_root | Root for UART perclk | 66.5 |
| ipu_hsp_clk_root | Root for IPU_HSP clock | 200 |
| gpu_clk_root | Root for GPU clock | 200 |
| gpu2d_clk_root | Root for GPU2D clock | 200 |
| esai_clk_root | Root for ESAI serial clock | 66.5 |
| can_clk_root | Root for FLEXCAN serial clock | 66.5 |
| pgc_clk_root | Root for PGC clock of GPC | 66.5 |
| wrck_clk_root | Root for WRCK clock | 25 |
| firi_clk_root | Root for FIRI clock | 66.5 |
| ckil_sync_clk_root | Root for CKIL clock after sync | 0.032 |

Clock connectivity is described in the in the “System Clocks Connectivity” section in the CCM chapter of the i.MX53 reference manual. This section contains a series of tables that describe the clock inputs of each module and which clock is connected to it. In most cases, the clocks are CCM root clocks as listed in Table 8-1. However, some clocks come from IO pins (mainly though IOMUX) and not from CCM.

Clock gating is done with the low power clock gating (LPCG) module based on a combination of the clock enable signals. For more information about how the clock gating signals are logically combined, refer to the LPCG section in the CCM chapter of the i.MX53 reference manual.

**NOTE**

In some cases, a clock is part of a protocol and is sourced from a pad (mainly through IOMUX). Such clocks do not appear in the clock connectivity table. They are found in the “External Signals and Pin Multiplexing” chapter.

# Chapter 9
# Configuring JTAG Tools for Debugging

This chapter explains how to configure JTAG tools for debugging. The JTAG module is a standard JEDEC debug peripheral. It provides debug access to important hardware blocks, such as the ARM processor and the system bus, which can give users access and control over the entire SoC. Because of this, unsecured JTAG modules are vulnerable to JTAG manipulation, a known hacker's method of executing unauthorized program code, getting control over secure applications, and running code in privileged modes. To properly secure the system, unauthorized JTAG usage must be strictly forbidden.

To prevent JTAG manipulation while allowing access for manufacturing tests and software debugging, the i.MX53 processor incorporates a secure JTAG controller for regulating JTAG access. The secure JTAG controller provides four different JTAG security modes, which are selected by an e-fuse configuration. For more information about the security modes, see the "Security" section in the "System JTAG Controller (SJC)" chapter of the i.MX53 reference manual.

**NOTE**

By default all parts are shipped with security disabled.

The JTAG port must be accessible during platform initial validation bring-up and for software debugging. It is accessible in all development kits from Freescale. Multiple tools are available for accessing the JTAG port for tests and software debugging. Freescale recommends use of the ARM JTAG tools for compatibility with the ARM core. However, the JTAG chain described in the following sections should work for non-ARM JTAG tools. For more information about non-ARM tools, contact the third party tool vendors for support.

## 9.1 Accessing Debug with a JTAG Scan Chain (ARM tools)

This section shows how to use the ARM tools to connect to the i.MX53 processor, using a JTAG scan chain. The example uses the RealView ICE (RVI) and RVDS ARM tools. RVI provides the hardware interface between the host PC and the JTAG port on the development kit (see http://www.arm.com/products/tools/rvi-and-rvt2.php for more information). RVDS is the software development kit that runs on the host PC. Its primary components consist of the ARM compiler, an Eclipse based IDE, and the RealView Debugger (for more information, see http://www.arm.com/products/tools/software-development-tools.php).

**NOTE**

Users must have the latest recommended ARM firmware installed on their RVI box to be able to connect to the Cortex-A8 on the i.MX53.

Once the latest firmware is installed, follow these steps to configure the JTAG scan chain on the RVI box:

1. Connect RVI to the i.MX53 board using the JTAG ribbon cable.
2. Using the order shown below, configure the scan chain with the following connections: TDI → Unknown → Unknown → ARMCS-DP → Cortex-A8 (see Figure 9-1).
   a) Add Device > Custom Device > UNKNOWN > IR Length = 5
   b) Add Device > Custom Device > UNKNOWN > IR Length = 4
   c) Add Device > Registered Device > CoreSight > ARMCS-DP
   d) Add Device > Registered Device > Cortex > Cortex-A8

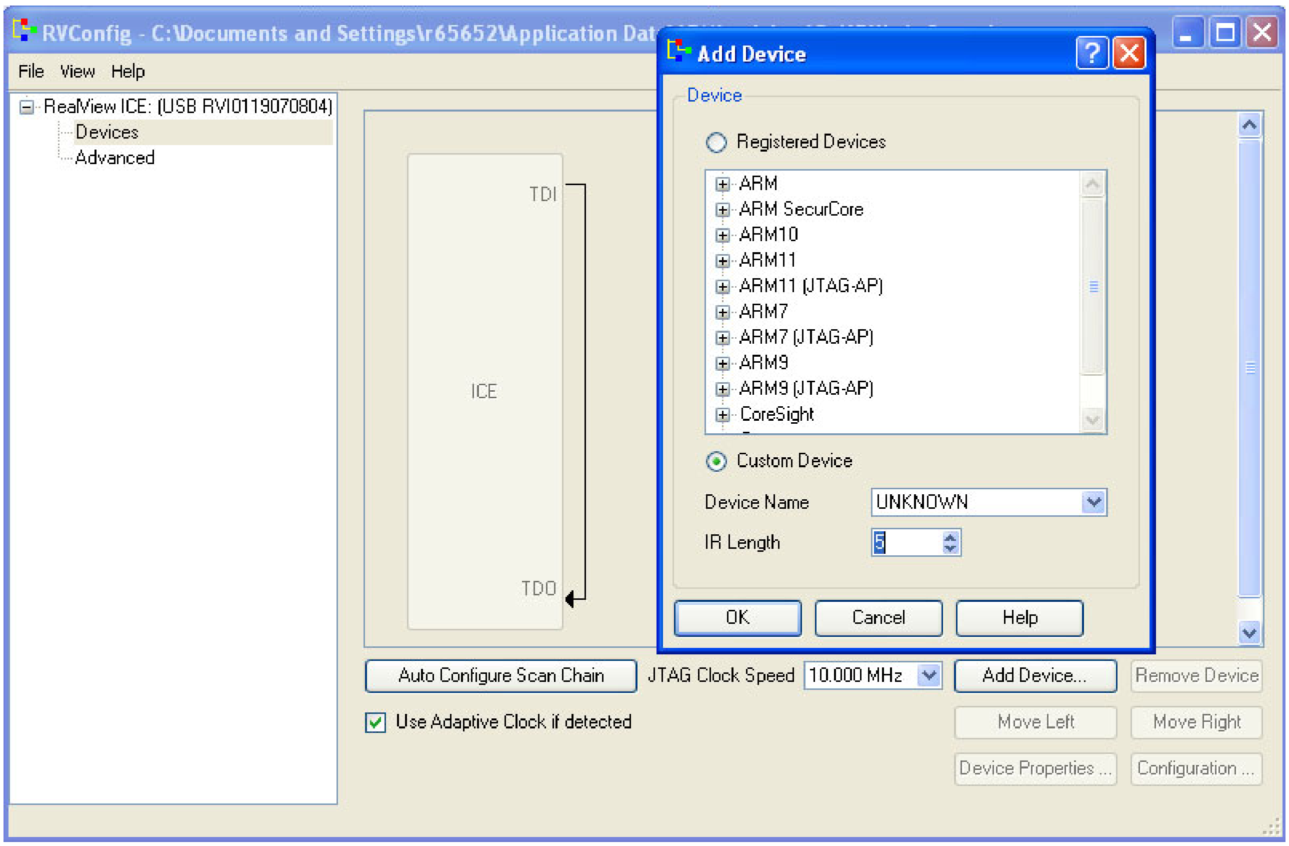



**Figure 9-1. Example of Adding a Device**

3. Update the CoreSight base address as follows:
   a) Right click on Cortex-A8 Device.
   b) Select configuration.
   c) Set CoreSight base address to = 0xC0008000.

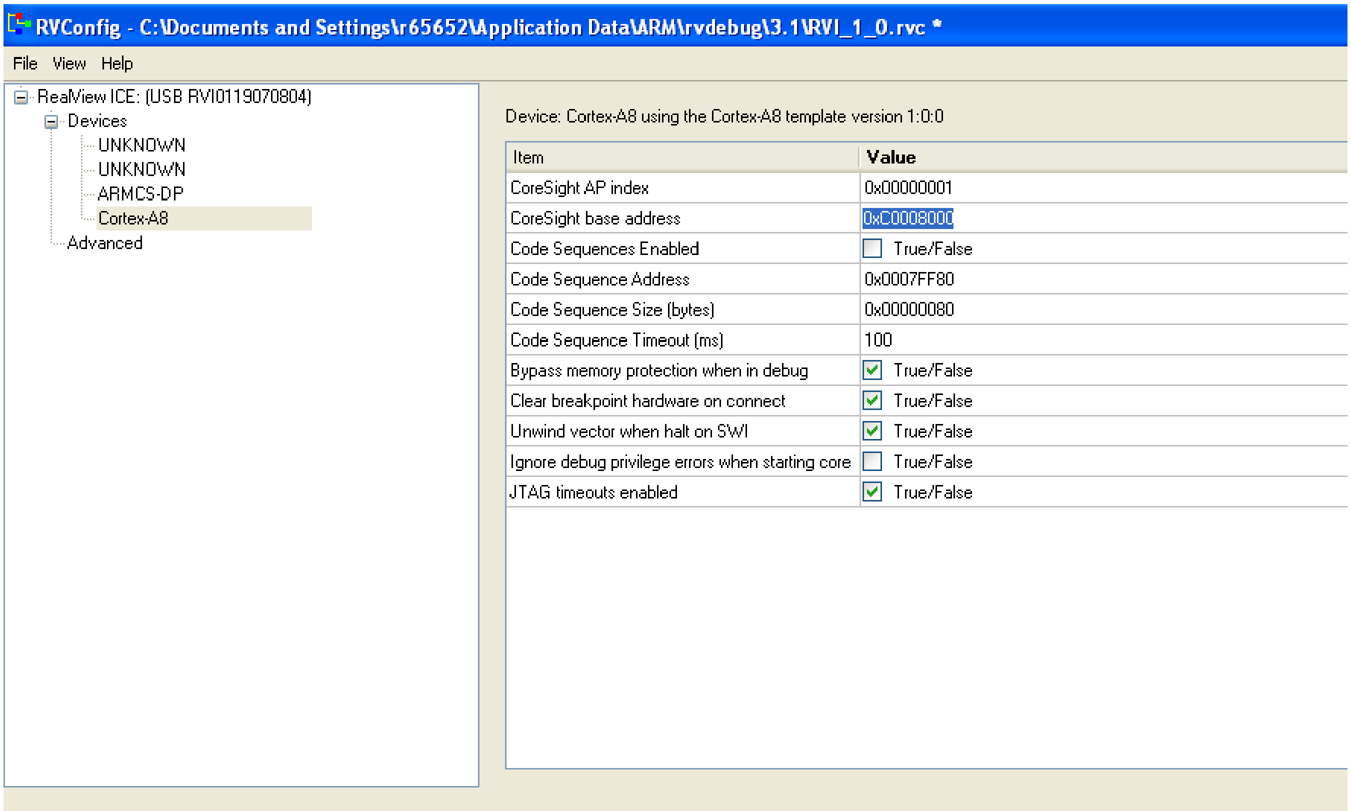



**Figure 9-2. Updating the CoreSight Base Address**

4. Save the configuration.

After following the recommended steps, the RVDS JTAG scan chain should look like Figure 9-3. Note this screenshot shows the resulting scan chain when using ARM RVDS v3.1 tools.

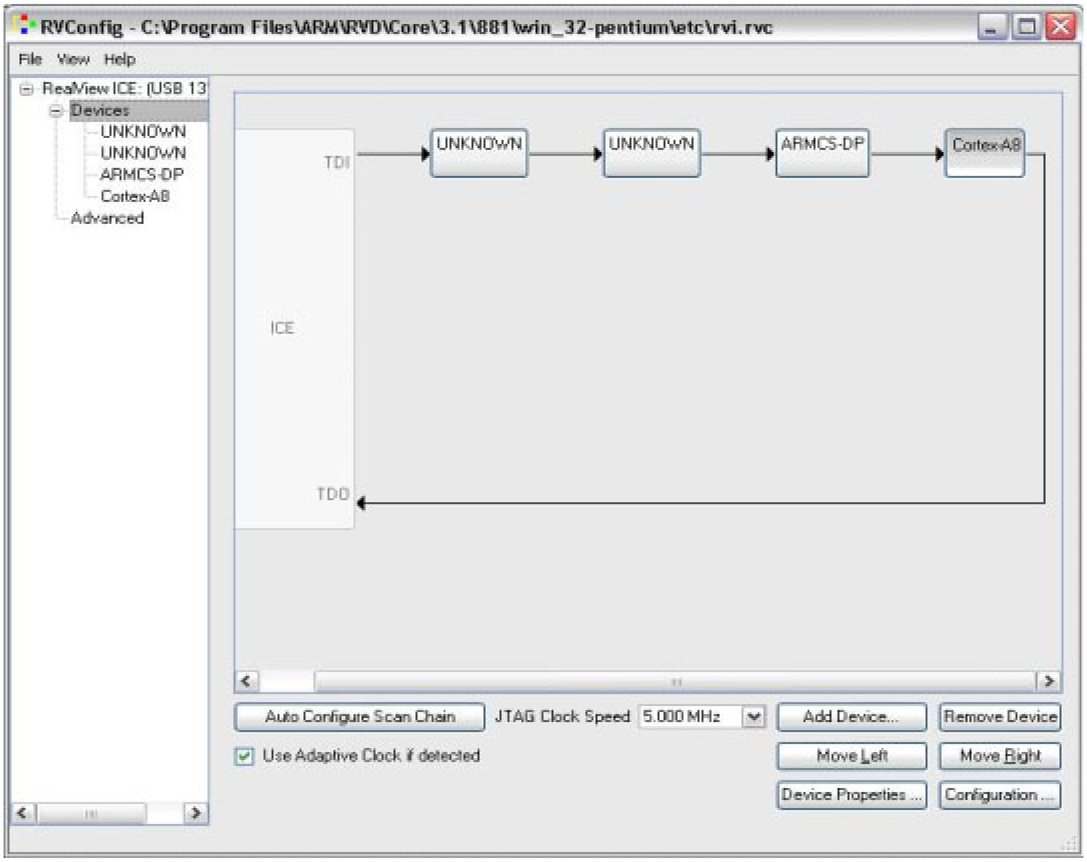



**Figure 9-3. i.MX/Cortex-A8 RVDS JTAG Scan Chain**

After setting up the JTAG scan chain, RVI can connect to the i.MX53's core. This is the only required step; no initialization scripts are necessary.

Once connected, test code can be loaded immediately into the internal RAM space, which starts at 0xF800_0000 (for more details refer to the i.MX53 memory map in the i.MX53 reference manual). Additionally, ARM provides ".bcd" files for some i.MX products, which can be used with RVDS to provide enumerated views of registers and/or peripherals on the target hardware along with the entire memory map of the target processor. Available ".bcd" configuration files are located at http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0182l/Bjefhigi.html

## 9.2 Accessing Debug with a JTAG Scan Chain (other JTAG tools)

The JTAG scan chain described in Section 9.1, "Accessing Debug with a JTAG Scan Chain (ARM tools)," is not specific to ARM tools. It can be used with any JTAG tool to connect to the i.MX53 processor. The IR lengths of each component in the JTAG scan chain are provided so that the steps can be repeated when using a different tool.

# Part II
# Software Development

The chapters that follow aid you in software development for your product utilizing the i.MX53 Board Support Package.

# Chapter 10
# Porting the On-Board-Diagnostic-Suite (OBDS) to a Custom Board

The on-board diagnostic suite (OBDS) is a set of validation software used during the board bring up phase and also to validate the boards produced during mass manufacturing for defects. OBDS is run to test out specific IP blocks of the i.MX53 SoC and the associated hardware on the board.

In a typical scenario, a basic set of the hardware components are tested to be functional, prior to engaging the software team to bring up the bootloader and the BSP.

Prior to reading this document, be familiar with the following chapters in the *i.MX53 Applications Processor Reference Manual*:

- Chapter 1, “Introduction”
- Chapter 9, “Power Management”
- Chapter 4, “External Signals and Pin Multiplexing”
- Chapter 6, “System Debug”
- Chapter 18, “Clock Control Module (CCM)”
- Chapter 43, “IOMUX Controller (IOMUX)”

## 10.1 Supported Components

The OBDS package for Freescale’s i.MX53 reference board provides support for the following SoC internal functional blocks:

- Debug UART test
- DDR test
- Audio Out test
- IPU LCD display test
- $I^2C$ connectivity test to the PMIC (MC13892 or LTC2495 depending on the EVK version) MMC/SD test fir SD Slot 2
- LED test
- Ethernet Loopback test
- SPI-NOR test (EVK only)
- USB HUB test (EVK only)
- NAND Flash Device ID test

## 10.2 Customizing OBDS for Specific Hardware

This section explains how to customize the OBDS for the following hardware modules:

- Section 10.2.1, "UART (serial port) Test"
- Section 10.2.2, "DDR Test"
- Section 10.2.3, "Audio Test"
- Section 10.2.4, "IPU Display Test"
- Section 10.2.5, "I$^2$C Test"
- Section 10.2.6, "SD/MMC Test"
- Section 10.2.7, "LED Test"
- Section 10.2.8, "Ethernet (FEC) Loopback Test"
- Section 10.2.9, "SPI-NOR Test"

### 10.2.1 UART (serial port) Test

The UART port is the primary communications channel between the reference board and host PC. The UART test tests the transmission capabilities of the serial port and verifies its receive capabilities by prompting the user to input a character from the host PC to the serial port. Typing the character "X" exits this test and moves to the next test.

On the i.MX53 reference boards (with the exception of the ARD), the UART1 TXD and RXD pins are routed to the CSI0_DAT10 and CSI0_DAT11 pins via the IOMUX (see the `~/diag-obds/src/mx53/hardware.c` file). In addition, the file `mx53.c` defines the "debug_uart" variable to UART1 as `static struct hw_module *debug_uart = &uart1;` If a different UART port is used, make the required IOMUX changes to the routine `debug_uart_iomux()`, using the following code:

```
void debug_uart_iomux(void)
{
// UART1 mux'd on CSI0_DAT10 and CSI0_DAT11
writel(0x2, IOMUXC_SW_MUX_CTL_PAD_CSI0_DAT10);
writel(0x2, IOMUXC_SW_MUX_CTL_PAD_CSI0_DAT11);     // daisy chain setup
writel(0x1, IOMUXC_UART1_IPP_UART_RXD_MUX_SELECT_INPUT);
}
```

### 10.2.2 DDR Test

The DDR test verifies the interface connectivity between the i.MX53 and the DDR memory. This test should not be confused with a stress test that validates robust signal integrity of the interface. Instead, this test ensures the proper assembly of the memory and i.MX53 by testing for opens and shorts on the interface.

Each of the i.MX53 reference boards use a different DDR configuration. If the custom board implements a DDR that has a different configuration than the reference boards, refer to the data sheet of the specific DDR and make the necessary changes to the DDR configurations in the `~/diag-obds/src/include/mx53/plat_startup.inc` file. The routine sets up the IOMUX and DDR specific configurations.

### 10.2.3 Audio Test

The audio test first performs $I^2C$ communications between the i.MX53 and the SGTL5000 audio codec. The test then outputs audio data via the SSI/I2S interface to the audio codec. The `~/diag-obds/src/drivers/audio` folder contains the files that implement the audio test.

If a different SSI port is used, make the necessary IOMUX changes to the `~/diag-obds/src/mx53/hardware.c` file.

### 10.2.4 IPU Display Test

This test outputs an image to the WVGA display (Chungwa CLAA070VC01 7-inch WVGA TFT LCD). The test also gives the user two options to test an LVDS display (either the AUO T150XG01V02 15-Inch XGA Panel or CHIMEI M216H1-L01 21-Inch HD1080 Panel) and the VGA output.

Refer to the hardware.c file for changes in IOMUX when a different panel is used. In particular, note which display interface (DI0 or DI1) is being used. The i.MX53 reference board configures the DI0 pins from D0–D23. To enable a different display, add the display timing information in the **di_config()** routine in the `~/src/drivers/ipu/ipu_di.c` file. The display's data sheet provides the information for the different parameters.

### 10.2.5 $I^2C$ Test

This tests performs an $I^2C$ communications test with one or more devices on the $I^2C$ bus (reads back the device ID). ~/src/drivers/i2c folder contains the driver for the I2C module. Refer to hardware.c for IOMUX setup. The IOMUX on the i.MX53 reference boards are configured to route I2C2 signal to the keypad COL3 and ROW3 pins.

If another $I^2C$ port is needed, add a new entry for the other $I^2C$ IOMUX settings at hardware.c and change the $I^2C$ device test code depending on the $I^2C$ devices on the custom board.

### 10.2.6 SD/MMC Test

This test performs a read/write test to the MMC/SD card plugged into the SD slot. This test configures and uses the ESDHCV3-3 module on the i.MX53 reference boards, with the exception of the ARD which uses ESDHCV2-2. The `~/drivers/src/mmc` folder contains the files necessary to test the MMC/SD port.

### 10.2.7 LED Test

This test verifies the functionality of the on-board debug LED by prompting the user to visually inspect the LED to verify that the LED has been turned on and then off. The test function is located in `src/mx53/mx53.c`.

If another GPIO is used to drive the LED, change the following function inside `src/mx53/mx53.c` accordingly:

```
int gpio_led_test(void)
```

### 10.2.8 Ethernet (FEC) Loopback Test

The test requires a loopback Ethernet cable, which is described in the OBDS user guide. There is only one FEC in the i.MX53 SoC. No customization is required and code from OBDs can be run as-is.

### 10.2.9 SPI-NOR Test

This test verifies the interface between the i.MX53 ECSPI-1 module and the SPI-NOR flash. The `~/src/driver/spinor` folder contains the files necessary to test the SPI-NOR flash available on the i.MX53 reference board, using the i.MX53 ECSPI-1 module and ECSPI-1 SS1. Change `src/drivers/spinor/imx_spi_nor.c` when using a different SPI model. See the following example implementation for the Atmel AT45DB321D SPI NOR Flash.

```
struct chip_id AT45DB321D_id =
{     .id0 = 0x01, // Atmel AT45DB321D
      .id1 = 0x27,
      .id2 = 0x1f
}
There are also calls that are specific to the Atmel flash:
•        spi_nor_status_atmel
•        spi_nor_write_atmel
```

If another CSPI port is used to connect to the SPI, the calls to ECSPI-1 needs to be created in `src/drivers/spi/imx_ecspi.c`. For example:

```
platform_init()
...
imx_spi_nor.base = ECSPI1_BASE_ADDR;
imx_spi_nor.freq = 25000000;
imx_spi_nor.ss_pol = IMX_SPI_ACTIVE_LOW;
imx_spi_nor.ss = 1;
imx_spi_nor.fifo_sz = 32;
imx_spi_nor.us_delay = 0;
spi_init_flash = imx_ecspi_init;
...
```

Change this to ECSPI2_BASE_ADDR when connecting the SPI NOR to CSPI-2. The IOMUX settings for the other CSPI port need to be added in `hardware.c`.

# Chapter 11
# Porting U-Boot from an i.MX53 Reference Board to an i.MX53 Custom Board

This chapter provides a step-by-step guide that explains how to add i.MX53 custom board support to U-Boot. This developer's guide is based on U-Boot version rel_imx_2.6.35_10.12.01_RC4, which is present as a package on the LTIB-based Linux BSP at http://opensource.freescale.com/git?p=imx/uboot-imx.git.

For an introduction to the use of U-Boot firmware with i.MX processors, read AN4173, "U-Boot for i.MX51 Based Designs," which is available on the Freescale website.

## 11.1 Obtaining the Source Code for the U-Boot

The following steps explain how to obtain the source code.

1. Install LTIB as usual. Make sure you **deselect** U-Boot from compilation.
2. Manually unpack `u-boot: ./ltib -m prep -p u-boot`.

The U-Boot code is now located at `rpm/BUILD/u-boot-<version number>`. The guide will now refer to the U-Boot main directory as `<UBOOT_DIR>` and assumes that your shell working directory is `<UBOOT_DIR>`.

## 11.2 Preparing the Code

The following steps explain how to prepare the code.

1. Make a copy of the board directory, using the following instruction:
   ```
   $cp -R board/freescale/mx53_<reference board name> board/freescale/mx53_<custom board name>
   ```
2. Copy the existing `mx53_<reference board name>.h` board configuration file as `mx53_<custom board name>.h`, using the following instruction.
   ```
   $cp include/configs/mx53_<reference board name>.h include/configs/mx53_<custom board name>.h
   ```
3. Create one entry in `<UBOOT_DIR>/Makefile` for the new i.MX53-based configuration. This file is in alphabetical order. The instruction to use is as follows:
   ```
   mx53_<custom board name>_config      : unconfig
     @$(MKCONFIG) $(@:_config=) arm arm_cortexa8 mx53_<custom board name> freescale mx53
   ```

**NOTE**

U-Boot project developers recommend adding any new board to the MAKEALL script and to run this script in order to validate that the new code has not broken any other's platform build. This is a requirement if you plan to submit a patch back to the U-Boot community. For further information, consult the U-Boot README file.

4. Rename board/freescale/mx53_<custom board name>/mx53_<reference board name>.c as board/freescale/mx53_<custom board name>/mx53_<custom board name>.c.
5. Adapt any fixed paths. In this case, the linker script board/freescale/mx53_<custom board name>/u-boot.lds has at least two paths that must be changed
   — Change board/freescale/mx53_<reference board name>/flash_header.o to board/freescale/mx53_<custom board name>/flash_header.o
   — Change board/freescale/mx53_<reference board name>/libmx53_<reference board name>.a to board/freescale/mx53_<custom board name>/libmx53_<custom board name>.a
6. Change the line COBJS := mx53_<reference board name>.o (inside board/freescale/mx53_<custom board name>/Makefile) to COBJS := mx53_<custom board name>.o

**NOTE**

The remaining instructions build the U-Boot manually and do not use LTIB.

7. Create a shell script under <UBOOT_DIR> named build_u-boot.sh.

   The file's contents are now:

```
#!/bin/bash
export ARCH=arm
export CROSS_COMPILE=<path to cross compiler/prefix> (e.g.
PATH:/opt/freescale/usr/local/gcc-4.1.2-glibc-2.5-nptl-3/arm-none-linux-gnueabi/b
in/arm-none-linux-gnueabi-
export PATH=$PATH:<path to compiler>

make mx53_<custom board name>_config
make
```

8. Compile U-Boot using $./build_u-boot.sh
9. If everything is correct, you should now u-boot.bin as proof that your build setup is correct and ready to be customized.

The new i.MX53 custom board that you have created is an exact copy of the i.MX53 reference board, but the boards are two independent builds. This allows you to proceed to the next step: customizing the code to suit the new hardware design.

## 11.3 Customizing the i.MX53 Custom Board Code

The new i.MX53 custom board is part of the U-Boot source tree, but it is a duplicate of the i.MX53 reference board code and needs to be customized.

The DDR technology is a potential key difference between the two boards. If there is a difference in the DDR technology between the two boards, the DDR initialization needs to be ported. DDR initialization is coded in the DCD table, inside the boot header of the U-Boot image. When porting bootloader, kernel or driver code, you must have the schematics easily accessible for reference.

### 11.3.1 Changing the DCD Table for i.MX53 DDR3 Initialization

Initializing the memory interface requires configuring the relevant I/O pins with the right mode and impedance and initializing the ESDCTL module.

1. To port to the custom board, the appropriate DDR initialization needs to be used. This is the same initialization as would be used in a JTAG initialization script.
2. Open the file `board/freescale/mx53_<custom board name>/flash_header.S`
3. Modify all MXC_DCD_ITEM macros to match the memory specifications.

This is the new `board/freescale/mx53_<custom board name>/flash_header.S` customized for DDR3.

**NOTE**

> If you change the number of MXC_DCD_ITEM lines in the DCD table, you must update the value of the dcd_hdr and write_dcd_cmd labels according to the number of items.

### 11.3.2 Booting with the Modified U-Boot

If the DCD table (board/freescale/mx53_<custom board name>/flash_header.S) was modified successfully, you can compile and write `u-boot.bin` to an SD card. To test this, insert the SD card into the SD card socket of the CPU board and power cycle the board.

A message like this should be printed in the console:

```
U-Boot 2009.08 (Jul 29 2010 - 15:17:24)

CPU:   Freescale i.MX53 family 1.0V at 800 MHz
Board: Unkown board id1:11
Boot Reason: [POR]
Boot Device: SD
I2C:   ready
DRAM:   1 GB
MMC:   FSL_ESDHC: 0, FSL_ESDHC: 1
Card did not respond to voltage select!
MMC init failed
In:    serial
Out:   serial
Err:   serial
Net:   FEC0
<reference board name>: U-Boot >
```

### 11.3.3 Further Customization at System Boot

To further customize your U-Boot board project, use the first function that system boot calls on:

```
start_armboot in "lib_arm/board.c".
board_init()
```

All board initialization is executed inside this function. It starts by running through the **init_sequence[]** array of function pointers.

The first board dependent function inside **init_sequence[]** array is **board_init()**. **board_init()** is implemented inside `board/freescale/mx53_<custom board name>.c`.

At this point the most important tip is the following line of code:

```
...
gd->bd->bi_arch_number = MACH_TYPE_MX53_<reference board name>; /* board id for Linux */
...
```

To customize your board ID, go to the registration process at http://www.arm.linux.org.uk/developer/machines/

This tutorial will continue to use MACH_TYPE_MX53_<reference board name>.

### 11.3.4 Customizing the Printed Board Name

To customize the printed board name, use the **checkboard()** function. This function is called from the **init_sequence[]** array implemented inside `board/freescale/mx53_<custom board name>.c`. There are two ways to use **checkboard()** to customize the printed board name from `Board: Unknown board id1:11` to `Board: MX53 CPU3 on <custom board name>2`: the brute force way or by using a more flexible identification method if implemented on the custom board.

To customize the brute force way, delete the call to **identify_board_id( )** inside **checkboard()** and replace `printf("Board: ");` with `printf("Board: MX53 CPU3 on <custom board>\n");`

If this replacement is not made, the custom board may use another identification method. The identification can be detected and printed by implementing the function **__print_board_info()** according to the identification method on the custom board.

Alternatively, if the custom board provides a method to detect the board type via an external signal this can be detected in the identify_board_id() function.

Once this has been done, recompile U-Boot and deploy u-boot.bin to the SD card. The new prompt message should be as follows:

```
U-Boot 2009.08 (Jul 30 2010 - 14:44:00)

CPU:   Freescale i.MX53 family 1.0V at 800 MHz
Board: MX53 CPU3 on <custom board name>
Boot Reason: [POR]
Boot Device: SD
I2C:   ready
DRAM:   1 GB
MMC:   FSL_ESDHC: 0, FSL_ESDHC: 1
Card did not respond to voltage select!
MMC init failed
In:    serial
Out:   serial
```

```
Err:   serial
Net:   FEC0
Reference Board: U-Boot >
```

# Chapter 12
# Porting the Android Kernel

Android releases for the i.MX53 processor are divided into three main parts: the bootloader (U-Boot or redboot), the kernel, and the Android framework. This chapter explains how to port an Android kernel to any platform that is based on the i.MX53 chip. The easiest way to apply kernel modifications to any i.MX platform is to use an existing Android release either for the i.MX51 or i.MX53 processor.

## 12.1 Patching the Android Kernel

Before configuring the Android kernel, locate the BSP patches in the imx-android-rX folder. This folder contains all BSP patches needed for the different i.MX platforms. It also contains patches for some of the libraries implemented on the hardware abstraction layer. Apply the relevant patches to the kernel.

## 12.2 Configuring Android Release for Customized Platforms

Once the patches have been applied to the kernel, go to `myandroid/kernel_imx/path`. Use the option `make imx5_android_defconfig` to prepare the configuration for your i.MX53 platform.

### 12.2.1 Enabling and Disabling Default Resources

Users can add or remove resources that are enabled by default on the EVK board configuration by entering `make menuconfig` under `myandroid/kernel_imx`. Figure 12-1 shows the menu option screen.

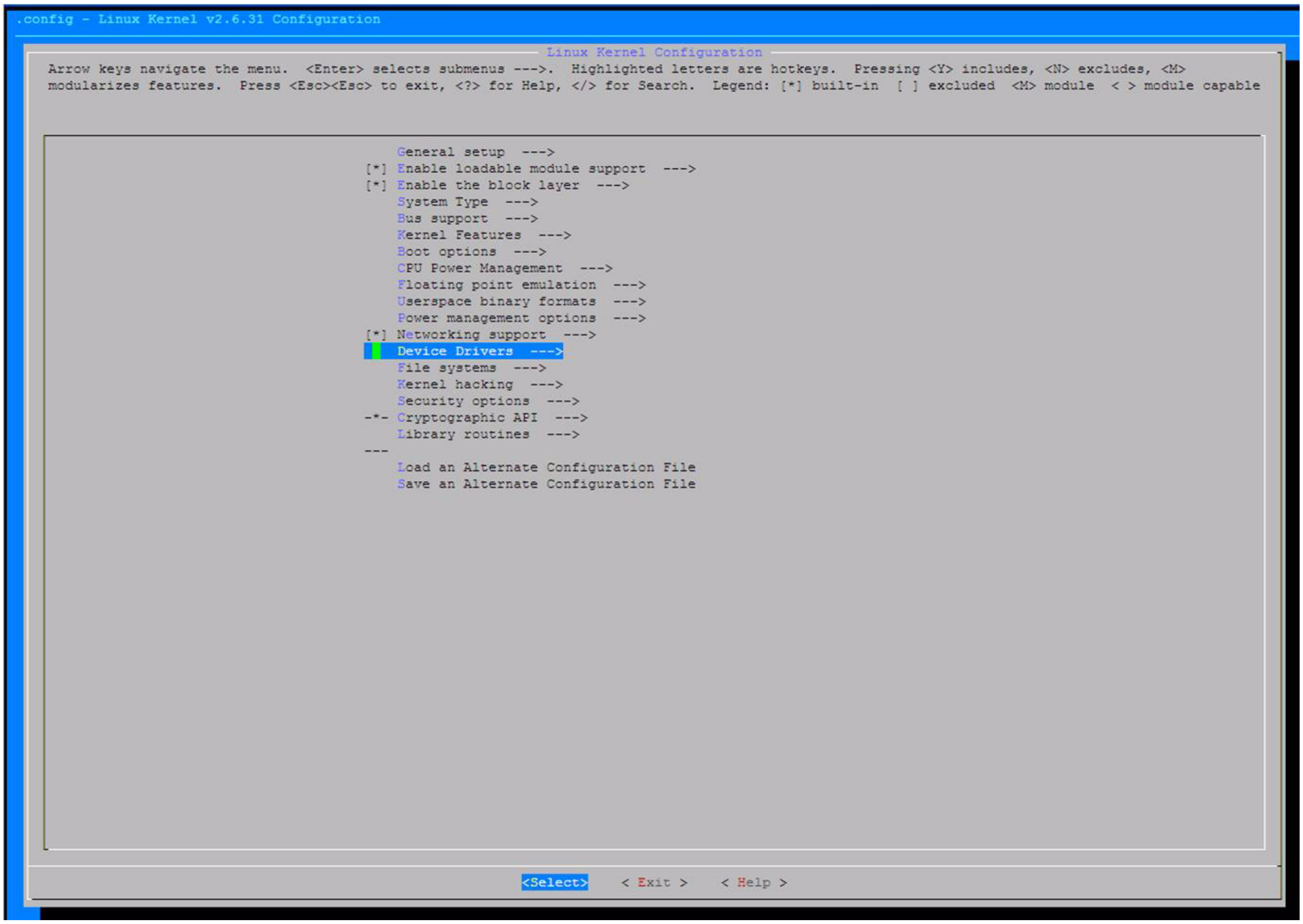



**Figure 12-1. Linux Kernel Configuration Menu**

This menu allows users to enable or disable drivers that are part of the Android framework’s included Linux image. Make your selections and exit the menu.

After you exit, the system creates the .config file, which contains the variables used to configure different interfaces and peripherals on the chip. It also contains variables for libraries and tools that are part of a Linux image.

### 12.2.2 Changing the Configuration File

After the system has created the .config file, users can change the configuration file to enable the environment variables required by the Android image. Configuration files for different platforms are located at: `myandroid/kernel-imx/arch/arm/config/`

Choose the appropriate configuration file for your platform and double check the .config file for the following variables:

- CONFIG_PANIC_TIMEOUT=0
- CONFIG_BINDER=y
- CONFIG_LOW_MEMORY_KILLER=y
- CONFIG_ANDROID_PARANOID_NETWORK=y
- CONFIG_ANDROID_LOGGER=y
- CONFIG_ANDROID_PMEM=y
- CONFIG_PMEM_SIZE=24
- CONFIG_ANDROID_RAM_CONSOLE=y
- CONFIG_ANDROID_RAM_CONSOLE_ENABLE_VERBOSE=y
- CONFIG_ANDROID_BINDER_IPC=y
- CONFIG_CRYPTO_DEFLATE=y
- CONFIG_CRYPTO_LZO=y
- CONFIG_DEVMEM=y
- CONFIG_LZO_COMPRESS=y
- CONFIG_LZO_DECOMPRESS=y
- CONFIG_ASHMEM=y

### 12.2.3 Android's Memory Map

Android's memory map is divided into four main blocks:

- GPU
- PMEM for GPU
- PMEM
- System memory

The total amount of memory is passed through a parameter called mem. This parameter usually contains all the memory available on the platform, and it is passed on the bootloader as the following configuration line.

```
setenv bootargs_android 'setenv bootargs $bootargs init=/init androidboot.console=ttymxc0
di0_primary calibration ip=dhcp mem=512M'
```

**NOTE**

By default the i.MX53 EVK board is set with 512 Mbytes.

Android's memory map hardcodes three of its four main blocks to a specific value. The final block uses whatever memory remains after the other three blocks have defined their boundaries. This remaining block of memory is used by the system memory as standard RAM memory for loading the kernel and apps execution.

Figure 12-2 shows how the Android's memory map is organized on a 512 Mbyte system.

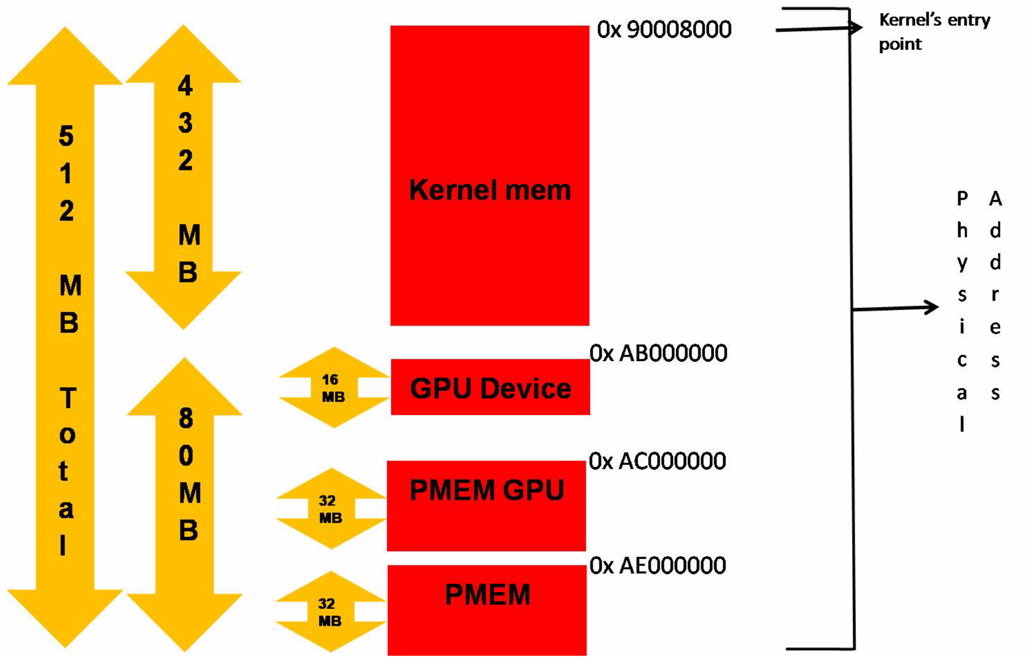



**Figure 12-2. Android Memory Map (512 Mbyte System)**

This memory map is defined under `/myandroid/kernel_imx/arch/arm/mach-mx5/mx53_evk.c` on the function `init fixup_mxc_board`.

## 12.3 Initializing Android

After the kernel boots, the init application is the first program executed on the system. The init program directly mounts all file systems and devices, using either hard-coded file names or device names generated by probing the sysfs file system. This eliminates the need for a /etc/fstab file in Android.

After the device/system files are mounted, init reads `/etc/init.rc`, which is a text file that contains parameters and commands executed by the init program. These commands are executed sequentially and load some of the main services of Android. The file can also create and mount directories where the system, cache, and data partitions reside.

Init and init.rc load the following services:

- app_process application—launches Zygote
- rild daemon application—manages all radio GSM support
- mediaserver—handles all media, including audio and video
- ts_calibrator—provides the touch screen calibration app

## 12.4 Modifying the init.rc Partition Locations

The init.rc file mounts the three main partitions—system, cache, and data—on the image. By default, these partitions are mounted from the SD/MMC controller.

If you have these partitions stored on another Flash source, modify the following lines to choose from the specific NVM.

- To mount the /system directory:

```
mount ext3 /dev/block/mmcblk0p2 /system
mount ext3 /dev/block/mmcblk0p2 /system ro remount
```

- To mount the /data directory:

```
mount ext3 /dev/block/mmcblk0p5 /data nosuid nodev
```

- To mounts the /recovery directory:

```
mount ext3 /dev/block/mmcblk0p6 /cache nosuid nodev
```

You also can modify the partition number where the directories and files are stored.

## 12.5 Adding Android Enhancements

Most Android porting is performed on the kernel side, as shown in Figure 12-3.


**Figure 12-3. Linux Kernel**

Android adds enhancements to the Linux kernel in order to give upper layers services like interprocess communication and power management policies. Table 12-1 shows the enhancements.

**Table 12-1. Android Enhancements**

| Enhancement | Purpose |
|---|---|
| Alarm | Provide timers functionality to wake up and sleep the device |
| Ashmem | Asynchronous shared memory share memory across process. |
| Binder | Ipc binder driver for interprocess communication |
| Power Management | New stack power management to increase performance |
| Low Memory Killer | Provides the functionality for android memory management |
| Kernel Debugger | Debug purposes |
| Logger | Debug purposes |

Most enhancement implementations are located at `kernel/drivers/staging/android`.

**NOTE**

Android also handles the hardware abstraction layer (HAL) between the Linux kernel and the android library stack. These drivers are related to specific hardware modules such as GPS, Bluetooth, or radio.

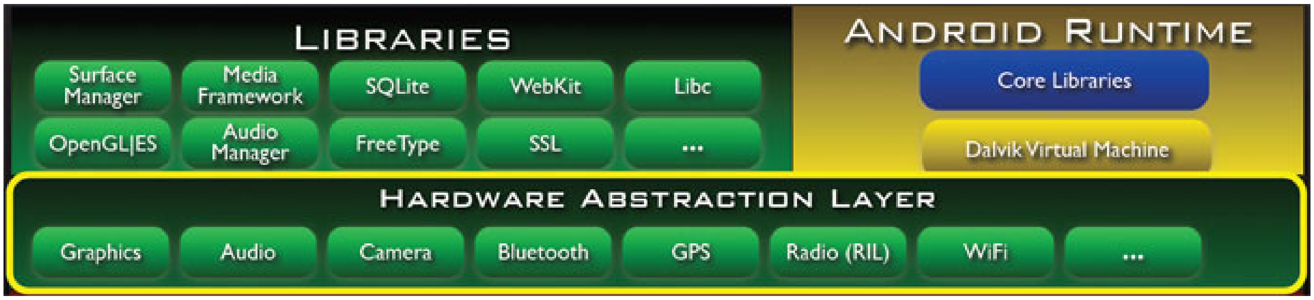



**Figure 12-4. Hardware Abstraction Layer**

This chapter does not cover these implementations. For information about HAL porting, please refer to the Android developer website at http://source.android.com.

# Chapter 13
# Configuring the IOMUX Controller (IOMUXC)

Before using the i.MX53 pins (or pads), users must select the desired function and correct values for characteristics such as voltage level, drive strength, and hysteresis. They do this by configuring a set of registers from the IOMUXC.

For detailed information about each pin, see the “External Signals and Pin Multiplexing” chapter in the *i.MX53 Applications Processor Reference Manual*. For additional information about the IOMUXC block, see the “IOMUX Controller (IOMUXC)” chapter in the *i.MX53 Applications Processor Reference Manual*.

## 13.1 Information for Setting IOMUX Controller Registers

The IOMUX controller contains four sets of registers that affect the i.MX53 registers, as follows:

- General-purpose registers (IOMUXC_GPR*x*)—consist of three registers that control PLL frequency, voltage, and other general purpose sets.
- “Daisy Chain” control registers (IOMUXC_<Instance_port>_SELECT_INPUT)—control the input path to a module when more than one pad may drive the module’s input
- MUX control registers (changing pad modes):
  - — Select which of the pad’s 8 different functions (also called ALT modes) is used.
  - — Can set pad’s functions individually or by group using one of the following registers:
    - – IOMUXC_SW_MUX_CTL_PAD_<PAD NAME>
    - – IOMUXC_SW_MUX_CTL_GRP_<GROUP NAME>
- Pad control registers (changing pad characteristics):
  - — Set pad characteristics individually or by group using one of the following registers:
    - – IOMUXC_SW_PAD_CTL_PAD_<PAD_NAME>
    - – IOMUXC_SW_PAD_CTL_GRP_<GROUP NAME>
  - — Pad characteristics are:
    - – SRE (1 bit slew rate control)—Slew rate control bit; selects between FAST/SLOW slew rate output. Fast slew rate is used for high frequency designs.
    - – DSE (2 bits drive strength control)—Drive strength control bits; select the drive strength (low, medium, high, or max).
    - – ODE (1 bit open drain control)—Open drain enable bit; selects open drain or CMOS output.
    - – HYS (1 bit hysteresis control)—Selects between CMOS or Schmitt Trigger when pad is an input.

- PUS (2 bits pull up/down configuration value)—Selects between pull up or down and its value.
- PUE (1 bit pull/keep select)—Selects between pull up or keeper. A keeper circuit help assure that a pin stays in the last logic state when the pin is no longer being driven.
- PKE (1 bit enable/disable pull up, pull down or keeper capability)—Enable or disable pull up, pull down, or keeper.
- DDR_MODE_SEL (1 bit ddr_mode control)—Needed when interfacing DDR memories.
- DDR_INPUT (1 bit ddr_input control)—Needed when interfacing DDR memories.

## 13.2 Setting Up the IOMUXC and U-Boot

To setup the IOMUXC and configure the pads on U-Boot, use the four files described in Table 13-1:

**Table 13-1. Configuration Files**

| Path | Filename | Description |
|---|---|---|
| cpu/arm_cortexa8/mx53/ | iomux.c | Iomux functions (no need to change) |
| include/asm-arm/arch-mx53/ | iomux.h | Iomux definitions (no need to change) |
| include/asm-arm/arch-mx53/ | mx53_pins.h | Definition of all processor's pads |
| board/freescale/mx53_<reference board name>/ | mx53<reference board name>.c | Board initialization file |

### 13.2.1 Defining the Pads

The iomux.c file contains each pad's IOMUXC definitions. Use the following code to see the default definitions:

```
enum iomux_pins {
...
...
...
MX53_PIN_GPIO_19 = _MXC_BUILD_GPIO_PIN(3, 5, 1, 0x20, 0x348),
MX53_PIN_KEY_COL0 = _MXC_BUILD_GPIO_PIN(3, 6, 1, 0x24, 0x34C),
MX53_PIN_KEY_ROW0 = _MXC_BUILD_GPIO_PIN(3, 7, 1, 0x28, 0x350),
...
...
...
}
```

To change the values for each pad according to your hardware configuration, use the following:

```
MX53_PIN_<PIN NAME> = _MXC_BUILD_GPIO_PIN(gp, gi, ga, mi, pi)
```

Where:

- **gp**—IO Pin
- **gi**—IO Instance
- **ga**—MUX Mode
- **mi**—MUX Control Offset

- **pi**—PAD Control Offset

### 13.2.2 Configuring IOMUX Pins for Initialization Function

The mx53<reference board name>.c file contains the initialization functions for all peripherals (such as UART, $I^2C$, and Ethernet). Configure the relevant pins for each initializing function, using the following:

```
mxc_request_iomux(<pin name>, <iomux config>);
mxc_iomux_set_input(<mux input select>, <mux input config>);
mxc_iomux_set_pad(<pin name>, <iomux pad config>);
```

Where the following applies:

| | |
|---|---|
| **<pin name>** | See all pins definitions on file mx53_pins.h |
| **<iomux config>** | See parameters defined at iomux_config enumeration on file iomux.h |
| **<iomux input select>** | See parameters defined at iomux_input_select enumeration on file iomux.h |
| **<iomux input config>** | See parameters defined at iomux_input_config enumeration on file iomux.h |
| **<iomux pad config>** | See parameters defined at iomux_pad_config enumeration on file iomux.h |

### 13.2.3 Example—Setting a GPIO

For an example, configure and use pin PATA_DA_1 (PIN L3) as a general GPIO and toggle its signal.

Add the following code to the file mx53_<reference board name>.c, function board_init:

```
// Request ownership for an IO pin.
mxc_request_iomux(MX53_PIN_ATA_DA_1, IOMUX_CONFIG_ALT1);


// Set pin as 0
reg = readl(GPIO7_BASE_ADDR + 0x0);
reg &= ~0x80;
writel(reg, GPIO7_BASE_ADDR + 0x0);


// Set pin direction as output
reg = readl(GPIO7_BASE_ADDR + 0x4);
reg |= 0x80;
writel(reg, GPIO7_BASE_ADDR + 0x4);


// Delay 0.5 seconds
udelay(500000);


// Set pin as 1
reg = readl(GPIO7_BASE_ADDR + 0x0);
reg |= 0x80;
writel(reg, GPIO7_BASE_ADDR + 0x0);


// Delay 0.5 seconds
udelay(500000);
```

```
// Set pin as 0
reg = readl(GPIO7_BASE_ADDR + 0x0);
reg &= ~0x80;
writel(reg, GPIO7_BASE_ADDR + 0x0);
```

If done correctly, the pin PATA_DA_1 on the i.MX53 toggles when booting.

## 13.3 Setting Up the IOMUXC in Linux

The folder `linux/arch/arm/mach-<platform name>` contains the specific machine layer file for your custom board. For example, the machine layer file used on the i.MX53 <reference> boards are `linux/arch/arm/mach-mx5/mx53_<reference board name>.c`. This platform is used in the examples in this section. The machine layer files include the IOMUX configuration information for peripherals used on a specific board.

To set up the IOMUXC and configure the pads, change the two files described in Table 13-2:

**Table 13-2. IOMUX Configuration Files**

| Path | File name | Description |
|---|---|---|
| linux/arch/arm/plat-mxc/include/mach/ | iomux-mx53.h | IOMUX configuration definitions |
| linux/arch/arm/mach-mx5 | `mx53_<reference board name>.c` | Machine Layer File. Contains IOMUX configuration structures |

### 13.3.1 IOMUX Configuration Definition

The iomux-mx53.h file contains definitions for all i.MX53 pins. Pin names are formed according to the formula <SoC>*PAD*<Pad Name>_*GPIO*<Instance name>_<Port name>. Definitions are created with the following line code.

```
IOMUX_PAD(PAD Control Offset, MUX Control Offset, MUX Mode, Select Input Offset, Select Input,
Pad Control)
```

The variables are defined as follows:

**PAD Control Offset** Address offset to pad control register (IOMUXC_SW_PAD_CTL_PAD_<PAD_NAME>)

**MUX Control Offset** Address offset to MUX control register (IOMUXC_SW_MUX_CTL_PAD_<PAD NAME>)

**MUX Mode** MUX mode data, defined on MUX control registers

**Select Input Offset** Address offset to MUX control register (IOMUXC_<Instance_port>_SELECT_INPUT)

**Select Input** Select Input data, defined on select input registers

**Pad Control** Pad Control data, defined on Pad control registers

Definitions can be added or changed, as shown in the following example code:

```
#define MX53_PAD_ATA_CS_1__UART3_RXD        IOMUX_PAD (0x620, 0x2A0, 4, 0x888, 3,
MX53_UART_PAD_CTRL)
```

The variables are as follows:

- **0x620**—PAD Control Offset
- **0x2A0**—MUX Control Offset
- **4**—MUX Mode
- **0x888**—Select Input Offset
- **3**—Select Input
- **MX53_UART_PAD_CTRL**—Pad Control

For all addresses and register values, check the IOMUX chapter in the *i.MX53 Applications Processor Reference Manual*.

### 13.3.2 Machine Layer File

The mx53_<reference board name>.c file contains structures for configuring the pads. They are declared as follows:

```
static struct pad_desc mx53common_pads[] = {
…
…
…
MX53_PAD_ATA_BUFFER_EN__UART2_RXD,
MX53_PAD_ATA_DMARQ__UART2_TXD,
MX53_PAD_ATA_DIOR__UART2_RTS,
MX53_PAD_ATA_INTRQ__UART2_CTS,


MX53_PAD_ATA_CS_0__UART3_TXD,
MX53_PAD_ATA_CS_1__UART3_RXD,
…
…
…
};
```

Add the pad's definitions from iomux-mx53.h to the above code.

On init function (in this example “mx53_<reference board name>_io_init” function), set up the pads using the following function:

```
mxc_iomux_v3_setup_multiple_pads(mx53common_pads, ARRAY_SIZE(mx53common_pads));
```

### 13.3.3 Example—Setting a GPIO

For an example, configure the pin PATA_DA_1 (PIN L3) as a general GPIO and toggle its signal.

On Kernel menuconfig, add sysfs interface support for GPIO with the following code:

```
Device Drivers --->
      [*] GPIO Support --->
           [*] /sys/class/gpio/... (sysfs interface)
```

Define the pad on iomux-mx53.h file as follows:

```
#define MX53_PAD_ATA_DA_1__GPIO_7_7IOMUX_PAD(0x614, 0x294, 1, 0x0, 0, NO_PAD_CTRL)
```

Parameters:

- 0x614—PAD Control Offset
- 0x294—MUX Control Offset
- 1—MUX Mode
- 0x000—Select Input Offset
- 0—Select Input
- NO_PAD_CTRL—Pad Control

To register the pad, add the previously defined pin to the pad description structure in the mx53_<reference board name>.c file, as shown in the following code.

```
static struct pad_desc mx53common_pads[] = {
…
…
…
MX53_PAD_ATA_DA_1__GPIO_7_7,
…
…
…
};
```

To use the pad as GPIO, go to the i.MX53 Linux command line. On this line, it is possible to test the GPIO exporting its number on `/sys/class/gpio/export`.

This number is formed by <GPIO Instance – 1> × 32 + <GPIO Port number>. In this example GPIO7_7 is being used, so its number is (7 – 1) × 32 + 7 = 199.

Export the GPIO7_7:

```
echo 199 > /sys/class/gpio/export
```

Set GPIO199 as output:

```
echo out > /sys/class/gpio/gpio199/direction
```

Set output as 1 or 0:

```
echo 1 > /sys/class/gpio/gpio199/value
echo 0 > /sys/class/gpio/gpio199/value
```

If the steps above were performed correctly, the pin PATA_DA_1 toggles on the i.MX53 reference board when the board is running the system.

# Chapter 14
# Registering a New UART Driver

Because Linux already has a UART driver for the i.MX53, configure the UART pads on the IOMUX registers. This chapter explains how to configure the UART pads, enable the UART driver, and test that the UART was set up correctly.

## 14.1 Configuring UART Pads on IOMUX

The IOMUX register must be set up correctly before the UART function can be used. This section provides example code to show how to do this.

Pads are configured using the file `linux/arch/arm/mach-mx5/<platform>.c`, with <platform> replaced by the appropriate platform file name (see Section 14.4, “File Names and Locations,” for the platform file names). For example, the machine layer file used on the i.MX53 reference boards are `linux/arch/arm/mach-mx5/mx53_<reference board name>.c`.

The iomux-mx53.h file contains the definitions for all i.MX53 pads. Configure the UART pads as follows:

```
/* UART3 */
#define MX53_PAD_ATA_CS_0__UART3_TXD IOMUX_PAD(0x61C, 0x29C, 4, 0x0, 0, MX53_UART_PAD_CTRL)
#define MX53_PAD_ATA_CS_1__UART3_RXD IOMUX_PAD(0x620, 0x2A0, 4, 0x888, 3, MX53_UART_PAD_CTRL)
```

The structures for configuring the pads are contained in the mx53_<reference board name>.c file. Update them so that they match the configured pads’ definition as shown above. The code below shows the non-updated structures:

```
static struct pad_desc mx53common_pads[] = {
…
…
…
        MX53_PAD_ATA_CS_0__UART3_TXD,
        MX53_PAD_ATA_CS_1__UART3_RXD,
…
…
…
};
```

Use the following function to set up the pads on the init function `mx53_<reference board name>_io_init` (found in the `mx53_<reference board name>.c` file).

```
mxc_iomux_v3_setup_multiple_pads(mx53common_pads, ARRAY_SIZE(mx53common_pads));
```

The UART driver is now implemented and needs to be enabled.

## 14.2 Enabling UART on Kernel Menuconfig

Enable the UART driver on Linux menuconfig. This option is located at:

```
-> Device Drivers
     -> Character devices
          -> Serial drivers
               <*> MXC Internal serial port support
               [*] Support for console on a MXC/MX27/MX21 Internal serial port
```

After enabling the UART driver, build the Linux kernel and boot the board.

## 14.3 Testing the UART

By default, the UART is configured as follows:

- Baud Rate: 9600
- Data bits: 8
- Parity: None
- Stop bits: 1
- Flow Control: None

If the user used a different UART configuration for a device that needs to connect to the i.MX53 processor, connection and communication will fail. There is a simple way to test whether the UART is properly configured and enabled.

On the i.MX53 Linux command line, type the following:

```
echo “test” > /dev/ttymxc2
```

UART3 (J2 on the i.MX53 expansion board) sends the string “test”.

## 14.4 File Names and Locations

There are three Linux source code directories that contain relevant UART files.

Table 14-1 lists the UART files that are available on the directory `<linux source code directory>/drivers/serial/`

**Table 14-1. Available Files—First Set**

| File | Description |
|---|---|
| mxc_uart.c | Low level driver |
| serial_core.c | Core driver that is included as part of standard Linux |
| mxc_uart_reg.h | Register values |
| mxc_uart_early.c | Source file to support early serial console for UART |

Table 14-2 lists the UART files that are available on the directory `<linux source code directory>/arch/arm/plat-mxc/include/mach/`

**Table 14-2. Available Files—Second Set**

| File | Description |
|---|---|
| mxc_uart.h | UART header containing UART configuration and data structures |
| iomux-<platform>.h | IOMUX pads definitions |

Table 14-3 lists the UART files that are available on the directory `<linux source code directory>/arch/arm/mach-mx5/`

**Table 14-3. Available Files—Third Set**

| File | Description |
|---|---|
| serial.c | UART configuration data and calls |
| serial.h | Serial header file |
| <platform>.c | Machine Layer file |

# Chapter 15
# Adding Support for the i.MX53 ESDHC

This chapter explains how to add support for the i.MX53 ESDHCV2-1/2/4 and ESDHCV3-3 controller.

The multimedia card (MMC)/secure digital (SD)/secure digital input output (SDIO) host driver implements a standard Linux driver interface for the enhanced MMC/SD host controller (ESDHC). The host driver is part of the Linux kernel MMC framework.

The MMC driver has the following features:

- 1-bit or 4-bit operation for SD and SDIO cards
- Supports card insertion and removal detections
- Supports the standard MMC commands
- PIO and DMA data transfers
- Power management
- Supports 1/4/8-bit operations for MMC cards
- Support eMMC4.4 SDR and DDR mode

## 15.1 Including Support for SD2 and SD4

The following features are required for SD card support in the i.MX53 BSP.

- Card detection.
- Write protection
- Max clock frequency
- Min clock frequency

These settings are configured with the mxc_mmc_platform_data structure defined at `/<ltib>/rpm/BUILD/linux/arch/arm/plat-mxc/include/mach/mmc.h`. The structure is shown below

```
struct mxc_mmc_platform_data {
    unsigned int ocr_mask;    /* available voltages */
    unsigned int vendor_ver;
    unsigned int caps;
    unsigned int min_clk;
    unsigned int max_clk;
    unsigned int clk_flg;    /* 1 clock enable, 0 not */
    unsigned int reserved:16;
    unsigned int card_fixed:1;
    unsigned int card_inserted_state:1;
    unsigned int (*status) (struct device *);
    int (*wp_status) (struct device *);
    char *power_mmc;
    char *clock_mmc;
};
```

**Table 15-1. Structure Descriptions**

| Struct member | Description |
|---|---|
| ocr_mask | Control the voltage on SD pads to be high voltage (around 3.0 V) or low voltage (around 1.8 V). ‘0’ stands for low voltage range Optional output |
| vendor_ver | Vendor version |
| caps | Modes of operation - data transfer modes |
| min_clk | Minimum SD operating frequency in Hz. |
| max_clk | Maximum SD operating frequency in Hz. |
| clk_flg | 0 clock disabled, 1 Clock enabled. |
| reserved | reserved (unused) |
| card_fixed | 0 Read Only Memory (ROM) cards, 1 Read/Write (RW) cards. |
| card_inserted_state | 1 SD card inserted in the slot, 0 there is no SD card attached to the socket. |
| status | Function pointer to the card detection status routine. |
| wp_status | Function pointer to the card write protection routine. |
| power_mmc | power supply for ESDHC |
| clock_mmc | Current MMC clock |

## 15.2 Including Support for SD1/SD2/SD3/SD4

For hardware that includes connectivity for any SD interface, include SD support from the BSP. Make the required changes in the mach-mx5 folder at `<ltib>/linux/arch/arm/mach-mx5` and follow the steps below.

1. Create the `platform_device struct` for all SD cards.
2. Configure the SD card pins.
3. Create `struct mxc_mmc_platform_data`.
4. Set up card detection.

These steps are discussed in detail in the following subsections.

### 15.2.1 Creating Platform Device Structures for all SD Cards

To create required platform device structures, open `<ltib>/linux/arch/arm/mach-mx5/devices.c`. Use the following code to ensure that your BSP include all required SD platform devices.

```
static struct resource mxcsdhcXX_resources[] = {
   {
      .start = MMC_SDHCXX_BASE_ADDR,
      .end = MMC_SDHCXX_BASE_ADDR + SZ_4K - 1,
      .flags = IORESOURCE_MEM,
   },
   {
      .start = MXC_INT_MMC_SDHCXX,
      .end = MXC_INT_MMC_SDHCXX,
      .flags = IORESOURCE_IRQ,
   },
```

```
   {
      .flags = IORESOURCE_IRQ,
   },
};


struct platform_device mxcsdhcXX_device = {
   .name = "mxsdhci",
   .id = YY,
   .num_resources = ARRAY_SIZE(mxcsdhcXX_resources),
   .resource = mxcsdhcXX_resources,
};
```

Variables have values as follows:

- XX can be 1, 2, 3 or 4 depending on the SD port.
- YY can have a value between 0 and 3.
- SD1’s ID is 0; SD2's ID is 1; SD3's ID is 2; and SD4's ID is 3.

Declare the structures as externs in `<ltib>/linux/arch/arm/mach-mx5/devices.h` with the following code.

```
extern struct platform_device mxcsdhc1_device;
extern struct platform_device mxcsdhc2_device;
extern struct platform_device mxcsdhc3_device;
extern struct platform_device mxcsdhc4_device;
```

### 15.2.2 Configuring Pins for SD Function

IOMUX allows several configurations, each with slight variances in the pins. The iomux-mx53.h file contains the definitions for all i.MX53 pads. Add entries in this file to define the configuration for the SD function. See Chapter 13, “Configuring the IOMUX Controller (IOMUXC),” for a description of how to set up the IOMUX and pads for routing signals as desired.

### 15.2.3 Creating the Platform Data Structure

After pin out configuration, SD card characteristics need to be described in an mxc_mmc_platform_data structure. Create one structure per SD in the system: mmc1_data, mmc2_data, mmc3_data, and/or mmc4_data. These structures must be placed in `<ltib>/linux/arch/arm/mach-mx5/mx53_<board name>.c`.

```
static struct mxc_mmc_platform_data mmc4_data = {
   .ocr_mask = MMC_VDD_27_28 | MMC_VDD_28_29 | MMC_VDD_29_30 | MMC_VDD_31_32,
   .caps = MMC_CAP_4_BIT_DATA | MMC_CAP_8_BIT_DATA | MMC_CAP_DATA_DDR,
   .min_clk = 400000,
   .max_clk = 50000000,
   .card_inserted_state = 0,
   .status = sdhc_get_card_det_status,
   .wp_status = sdhc_write_protect,
   .clock_mmc = "esdhc_clk",
   .power_mmc = NULL,
};
```

The preceding example shows the an example of an SD4 structure for a custom board. The SD4 interface supports either 4 bit or 8 bit data transfers (SD4_DAT[7:0]). Clock frequency can be set to a value between

400 KHz and 50 MHz. sdhc_get_card_det_status() and sdhc_write_protect() functions are used for card detection and write protection.

### 15.2.4 Setting Up Card Detection

The SD connector includes an output pin (CD) that changes its state according to the card insertion status. In some cases, CD is not connected to the processor. In those cases, the function should return true to signal that the card is always connected. When CD is connected, the SD card connector triggers the load of the SD into the available devices. After insertion, the system detects the SD and loads the MMC device under `/dev folder (/dev/mmcblk*)`.

To set up card detection, first modify sdhc_get_card_det_status() function by adding an entry for your SD device for detecting when the SD card has been inserted in the slot. This function is located under your platform at `<ltib>/linux/arch/arm/mach-mx5/mx53_<board name>.c`

```
static unsigned int sdhc_get_card_det_status(struct device *dev){
   int ret;
   // SD's Card support for i.MX53 <custom board name>
   if (board_is_mx53_<custom board>()) {     // SD1 Card support for i.MX53 <custom board name>
      if (to_platform_device(dev)->id == 0) {
         ret = gpio_get_value(IOMUX_TO_GPIO(MX53_PIN_GPIO_1));
      }
      // SD2 Card support for i.MX53 <custom board name>
      else if(to_platform_device(dev)->id == 1) {
         ret = gpio_get_value(IOMUX_TO_GPIO(MX53_PIN_GPIO_4));
      }
      // SD3 Card support for i.MX53 <custom board name>
      else if(to_platform_device(dev)->id == 2) {
         ret = 1;
      }
      // SD4 Card support for i.MX53 <custom board name>
      else if(to_platform_device(dev)->id == 3) {
          ret = 1;
      }
      else {
         ret = 1;
      }   // SD's Card support for i.MX53 Default Board
    } else {
      if (to_platform_device(dev)->id == 0) {
         ret = gpio_get_value(IOMUX_TO_GPIO(MX53_PIN_EIM_DA13));
      } else{      /* config the det pin for SDHC3 */
         ret = gpio_get_value(IOMUX_TO_GPIO(MX53_PIN_EIM_DA11));
      }
   }
   return ret;
}
```

Next, configure the pin as a general purpose input in the platform GPIO file located at `<ltib>/linux/arch/arm/mach-mx5/mx53_<board name>_gpio.c`.

```
static struct mxc_iomux_pin_cfg __initdata mx53_<board name>_iomux_pins[] = {
...
   { /* SDHC2 SD_CD */
      MX53_PIN_GPIO_4, IOMUX_CONFIG_GPIO,
   },
```

```
...
};
```

Then link GPIO interrupts with start and end functions in the resource structure of the SD interface in the mx53_<board name>.c file located at `<ltib>/linux/arch/arm/mach-mx5/mx53_<board name>.c`

```
static void __init mxc_board_init(void)
{
...
   /* SD card detect irqs */
   if (board_is_mx53_<board name>()) {
   ...
      // SD2 Card support for i.MX53 custom board
      mxcsdhc2_device.resource[2].start = IOMUX_TO_IRQ(MX53_PIN_GPIO_4);
      mxcsdhc2_device.resource[2].end = IOMUX_TO_IRQ(MX53_PIN_GPIO_4);
      mmc2_data.wp_status = NULL;
   ...
   }
...
}
```

Interfaces without card detection pins do not require any GPIO configuration. However, they need card detection forced to the kernel by setting the card_inserted_state field. An example is shown below:

```
static void __init mxc_board_init(void)
{
...
   /* SD card detect irqs */
   if (board_is_mx53_<custom board name>()) {
   ...
      // SDHC4 Card support for i.MX53 custom board
      mmc4_data.card_inserted_state = 1;
      mmc4_data.status = NULL;
      mmc4_data.wp_status = NULL;
   ...
   }
...
}
```

**NOTE**

SD interfaces without card detection are intended to be used as a soldered device, such as the MovieNAND. For this reason, SD without card_detect is only loaded during driver load (boot up time) if they are present. Be sure that you have inserted the card prior to the ESDHC driver initialization.

## 15.3 Additional Reference Information

This section describes the ESDHC interface features, explains the i.MX53 support for ESDHC, and shows the interface layouts.

### 15.3.1 ESDHC Interface Features

The ESDHC has 15 associate I/O signals with the following functions.

- The SD_CLK is an internally generated clock used to drive the MMC, SD, SDIO cards.
- The CMD I/O is used to send commands and receive responses to/from the card. Eight data lines (DAT7–DAT0) are used to perform data transfers between the ESDHC and the card.
- The SD_CD# and SD_WP are card detection and write protection signals directly routed from the socket. A low on SD_CD# means that a card is inserted and a high on SD_WP means that the write protect switch is active.
- SD_OD is an output signal generated in SoC level outside ESDHC and is used to select the external open drain resistor.
- SD_LCTL is an output signal used to drive an external LED to indicate that the SD interface is busy.

SD_CD#, SD_WP, SD_OD, SD_LCTL are all optional for system implementation. If the ESDHC is configured to support a 4-bit data transfer, DAT7–DAT4 can also be optional and tied to high.

**Table 15-2. ESDHC Pins**

| Pin | Function |
|---|---|
| SD_CLK | Clock for MMC/SD/SDIO card |
| SD_CMD | CMD line connect to card |
| SD_DAT7 | DAT7 line in 8-bit mode - Not used in other modes |
| SD_DAT6 | DAT6 line in 8-bit mode - Not used in other modes |
| SD_DAT5 | DAT5 line in 8-bit mode - Not used in other modes |
| SD_DAT4 | DAT4 line in 8-bit mode - Not used in other modes |
| SD_DAT3 | DAT3 line in 4/8-bit mode or configured as card detection pin. May be configured as card detection pin in 1-bit mode |
| SD_DAT2 | DAT2 line or Read Wait in 4-bit mode. Read Wait in 1-bit mode |
| SD_DAT1 | DAT1 line in 4/8-bit mode. Also used to detect interrupt in 1/4-bit mode |
| SD_DAT0 | DAT0 line in all modes. Also used to detect busy state |
| SD_CD# | Card detection pin. If not used tie high |
| SD_WP | Card write protect detect. If not used tie low |
| SD_OD | Open drain select (not generated within the ESDHC). Optional output |
| SD_LCTL | LED control used to drive an external LED. Active high. Fully controlled by the driver. Optional output |
| SD_VS | Control the voltage on SD pads to be high voltage (around 3.0V) or low voltage (around 1.8 V). 0 stands for low voltage range optional output. |

### 15.3.2 ESDHC Operation Modes Supported by the i.MX53

The ESDHC acts as a bridge, passing host bus transactions to the SD/SDIO/MMC cards by sending commands and performing data accesses to/from the cards. It handles the SD/SDIO/MMC protocols at the transmission level. The i.MX53 ESDHC includes three instances of the Enhanced Secured Digital Host Controller Version 2 (ESDHCv2) within the ports 1, 2 and 4. ESDHC port 3 on the i.MX53 can be configured to work either as ESDHCv3 or ESDHCv2.

Table 15-3 shows the supported operation modes.

**Table 15-3. ESDHC Operation Modes**

| Modes of Operation | Data Transfer Modes | Frequency |
|---|---|---|
| MMC | 1-bit, 4-bits or 8-bits | full-speed (up to 20 MHz) high-speed (up to 52 MHz) |
| SD/SDIO | 1-bit or 4-bit | full-speed (up to 25 MHz ) high-speed (up to 50 MHz) |
| CE-ATA | 1-bit, 4-bit, or 8-bit | — |
| Identification Mode | — | up to 400 kHz |

SD Memory Cards support at least the two bus modes 1-bit or 4-bit width. The SD host sends a command to the SD card to request a bus width change.

## 15.3.3 Interface Layouts

Figure 15-1 shows an example of an i.MX53 SD interface layout.

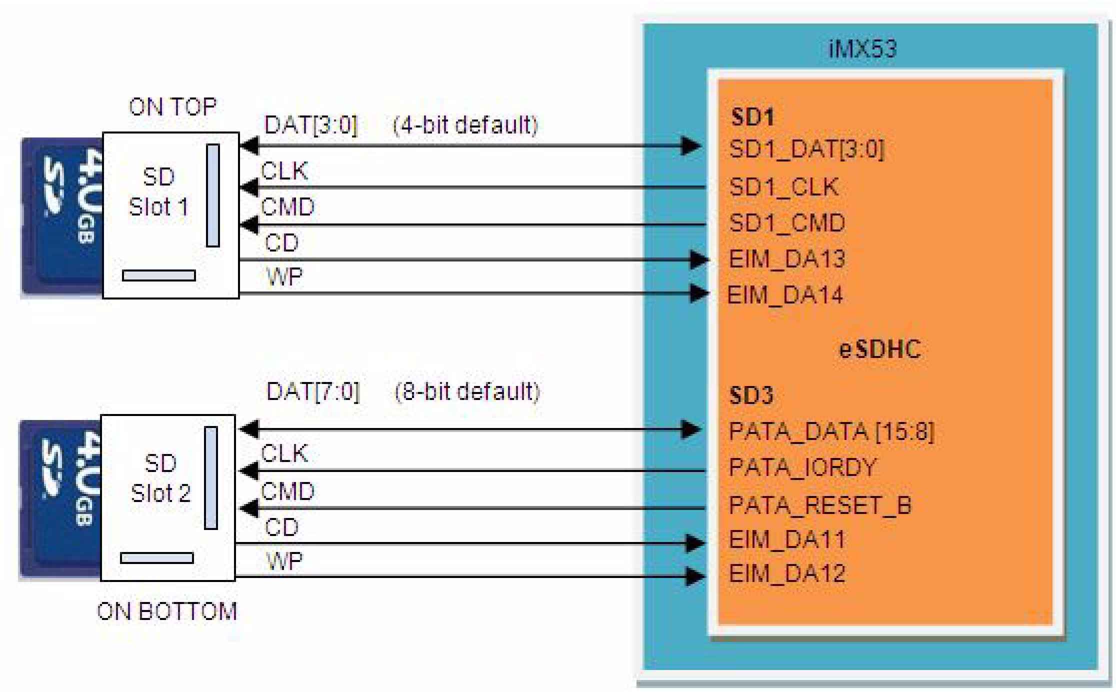



**Figure 15-1. Example i.MX53 Board SD Interface Layout**

Figure 15-2 shows another example i.MX53 SD interface layout.

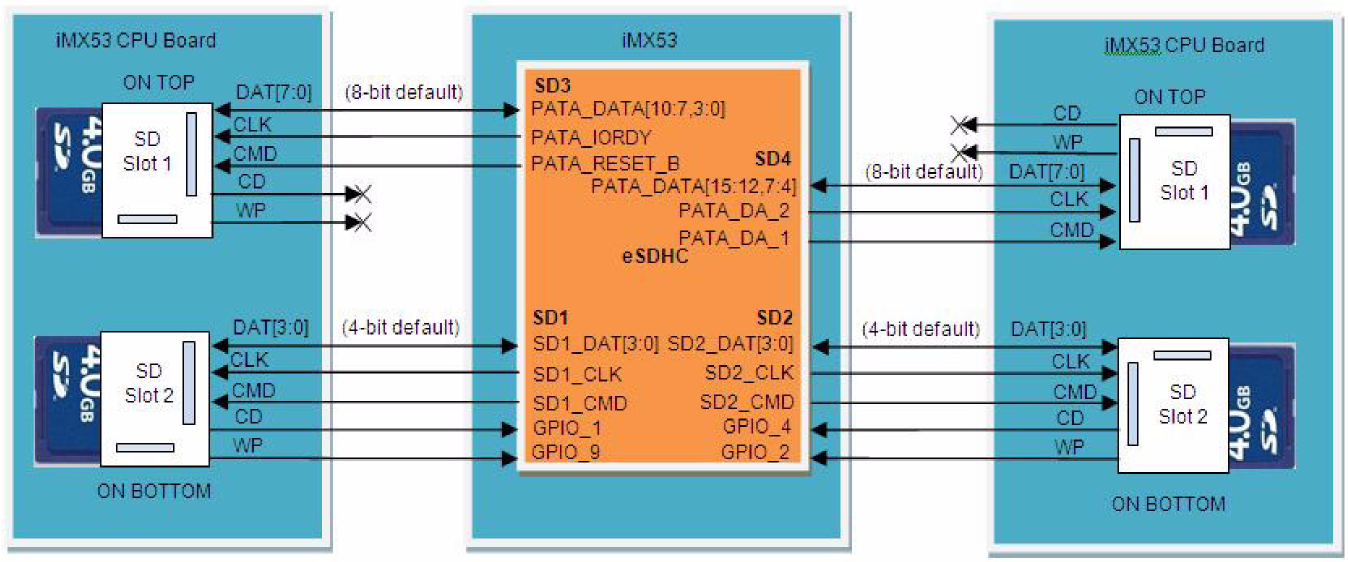



**Figure 15-2. Second Example i.MX53 SD Interface Layout**

Note that some SD interface card detection and write protection pins are not propagated from the SD card to the i.MX53 in all hardware implementations. Also note that SD4 is shared with PATA pins. The second example board provides the connection to the four SD interfaces provided by the ESDHC in the i.MX53.

# Chapter 16
# Configuring the SPI NOR Flash Memory Technology Device (MTD) Driver

This chapter explains how to set up the SPI NOR Flash memory technology device (MTD) driver. This driver uses the SPI interface to support Atmel data Flash. By default, the SPI NOR Flash MTD driver creates static MTD partitions to support Atmel data Flash.

The NOR MTD implementation provides necessary information for the upper layer MTD driver.

## 16.1 Source Code Structure

The SPI NOR MTD driver is implemented in the following directory:

`<ltib_dir>/rpm/BUILD/linux/drivers/mtd/devices/mxc_dataflash.c`

## 16.2 Configuration Options

BSP freescale supports the following ATMEL SPI NOR Flash models:

- "AT45DB011B" "at45db011d"
- "AT45DB021B" "at45db021d"
- "AT45DB041x" "at45db041d"
- "AT45DB081B" "at45db081d"
- "AT45DB161x" "at45db161d"
- "AT45DB321x" "at45db321d"
- "AT45DB642x" "at45db642d"

Those models are defined in the structure `static struct flash_info __devinitdata dataflash_data[]`, located at `<ltib_dir>/rpm/BUILD/linux/drivers/mtd/devices/mxc_dataflash.c`.

The parameters are as follows:

`"at45db011d", 0x1f2200, 512, 256, 8, SUP_POW2PS | IS_POW2PS`

Table 16-1 defines the variables.

**Table 16-1. Parameter Variables**

| Variable | Definition |
|---|---|
| "at45db011d" | Flash Name model |
| 0x1F_2200 | [5:4]Manufacter ID, [3:2]Device ID |
| 512 | Number of pages |

**Table 16-1. Parameter Variables (continued)**

| Variable | Definition |
|---|---|
| 256 | Number of bytes per page |
| 8 | Offset |

**NOTE**

If you want to use another data flash model, add it on the last structure. Be sure the flash models are compatible with the Atmel data flashes.

## 16.3 Selecting SPI NOR on the Linux Image

Table 16-2 provides information for each supported device.

**Table 16-2. Device Information**

| Device | Density | ID Code | #Pages | PageSize | Offset |
|---|---|---|---|---|---|
| AT45DB011B | 1 Mbit (128K) | xx0011xx (0x0C) | 512 | 264 | 9 |
| AT45DB021B | 2 Mbit (256K) | xx0101xx (0x14) | 1024 | 264 | 9 |
| AT45DB041B | 4 Mbit (512K) | xx0111xx (0x1C) | 2048 | 264 | 9 |
| AT45DB081B | 8 Mbit (1M) | xx1001xx (0x24) | 4096 | 264 | 9 |
| AT45DB0161B | 16 Mbit (2M) | xx1011xx (0x2C) | 4096 | 528 | 10 |
| AT45DB0321B | 32 Mbit (4M) | xx1101xx (0x34) | 8192 | 528 | 10 |
| AT45DB0642 | 64 Mbit (8M) | xx111xxx (0x3C) | 8192 | 1056 | 11 |
| AT45DB1282 | 128 Mbit (16M) | xx0100xx (0x10) | 16384 | 1056 | 11 |

Follow these steps to select the desired data flash from Table 16-2.

1. Open the mx53_<board name>.c file (located at `arch/arm/mach-mx5/mx53_<board name>.c`) and modify the structure called `static struct flash_platform_data mxc_spi_flash_data[]`
2. Write the name of the data flash desired on the .type variable of this structure. This name must be exactly the same as it appears on the `dataflash_data[]_` structure.
3. Set the number of partitions you want to use on the SPI NOR Flash. On the mx53_<board name>.c file, go to the structure called `static struct mtd_partition mxc_dataflash_partitions[]`

   Each partition has three elements: the name of the partition, the offset, and the size. By default, these elements are partitioned into a bootloader section and a kernel section, and defined as:

```
.name = "bootloader",
  .offset = 0,
  .size = 0x000100000,

  .name = "kernel",
  .offset = MTDPART_OFS_APPEND,
  .size = MTDPART_SIZ_FULL,
```

Bootloader starts from address 0 and has a size of 1 Mbyte. Kernel starts from address 1 Mbyte and has a size of 3 Mbytes.

**NOTE**

You may create more partitions or modify the size and names of these ones. To add more partitions, define another structure on the `mxc_dataflash_partitions` variable.

4. To get to the SPI NOR MTD driver, use the command `./ltib -c` when located in the `<ltib dir>`.
5. On the screen displayed, select **Configure the kernel** and exit.
6. When the next screen appears, select the following option to enable the SPI NOR MTD driver:

   `CONFIG_MTD_MXC_DATAFLASH`

   This config enables access to the Atmel DataFlash chips, using FSL SPI. In menuconfig, this option is available under `Device Drivers > Memory Technology Device (MTD) support > Self-contained MTD device drivers > Support for AT DataFlash via FSL SPI interface`

## 16.4 Changing the SPI Interface Configuration

The i.MX53 chip has three CSPI interfaces: one CSPI and two ECSPI. By default, the i.MX53 BSP configures ECSPI-1 interface in the master mode to connect to the SPI NOR Flash. It also uses chip select 1 from this ECSPI interface (SS1).

The main difference between CSPI and ECSPI is the supported baud rate. CSPI supports up to 26 Mbps in master mode and ECSPI supports up to 52 Mbps.

### 16.4.1 Connecting SPI NOR Flash to Another CSPI Interface

Before connecting SPI NOR Flash to another CSPI, define the three things listed below:

- CSPI interface (between CSPI, ECSPI-1 or ECSPI-2).
- Chip select (between SS[3:0]).
- External signals

### 16.4.2 Changing the CSPI Interface

To change the CSPI interface used, use the following procedure:

1. Locate the file at `arch/arm/mach-mx5/mx53_<board name>.c`
2. Look for the line `mxc_register_device(&mxcspi1_device, &mxcspi1_data);`
3. Use the function `static void __init mxc_board_init(void)` to register the CSPI-1 interface. To enable the other CSPI interface, replace the first parameter as shown in Table 16-3:

**Table 16-3. CSPI Parameters**

| CSPI | Parameter Name |
|---|---|
| ECSPI-1 | &mxcspi1_device |

**Table 16-3. CSPI Parameters (continued)**

| CSPI | Parameter Name |
|---|---|
| ECSPI-2 | &mxcspi2_device |
| CSPI | &mxcspi3_device |

### 16.4.3 Changing the Chip Select

To change the chip select used, locate the file at `arch/arm/mach-mx5/mx53_<board name>.c` and use the `static struct spi_board_info mxc_dataflash_device[] __initdata structure`.

Replace the value of ".chip_select" variable with the desired chip select value. For example, `.chip_select = 3` sets the chip select to number 3 on the CSPI interface.

### 16.4.4 Changing the External Signals

The iomux-mx53.h file contains the definitions for all i.MX53 pads. Add entries in this file to define the configuration for the CSPI function. See Chapter 13, “Configuring the IOMUX Controller (IOMUXC),” for a description of how to set up the IOMUX and pads for routing signals as desired.

**NOTE**

Check the mxc_iomux_pins structure to ensure that the chosen signal chosen is not used by another interface before configuration.

## 16.5 Hardware Operation

SPI NOR Flash is SPI compatible with frequencies up to 66 MHz. The memory is organized in pages of 512 bytes or 528 bytes. SPI NOR Flash also contains two SRAM buffers of 512/528 bytes each, which allows data reception while a page in the main memory is being reprogrammed as well as the writing of a continuous data stream.

Unlike conventional Flash memories that are accessed randomly, the SPI NOR Flash accesses data sequentially. It operates from a single 2.7–3.6 V power supply for program and read operations.

SPI NOR Flashes are enabled through a chip select pin and accessed through a three-wire interface: serial input, serial output, and serial clock.

## 16.6 Software Operation

In a Flash-based embedded Linux system, a number of Linux technologies work together to implement a file system. Figure 16-1 illustrates the relationships between standard components.

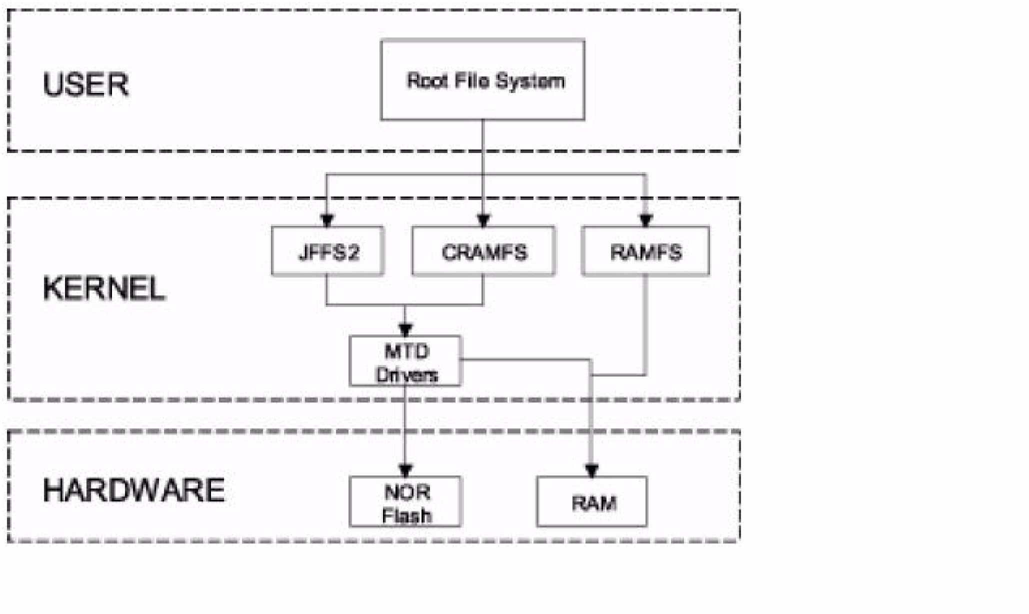



**Figure 16-1. Components of a Flash-Based File System**

The MTD subsystem for Linux is a generic interface to memory devices, such as Flash and RAM, which provides simple read, write, and erase access to physical memory devices. Devices called mtdblock devices can be mounted by JFFS, JFFS2, and CRAMFS file systems. The SPI NOR MTD driver is based on the MTD data Flash driver in the kernel by adding SPI accesses.

In the initialization phase, the SPI NOR MTD driver detects a data Flash by reading the JEDEC ID. The driver then adds the MTD device. The SPI NOR MTD driver also provides the interfaces to read, write, erase NOR Flash.

# Chapter 17
# Setting Up the Keypad Port (KPP)

The KPP is designed to interface with the keypad matrix with 2-point contact or 3-point contact keys. The KPP is designed to simplify the software task of scanning a keypad matrix. With appropriate software support, the KPP is capable of detecting, debouncing, and decoding one or multiple keys pressed simultaneously on the keypad.

Because Linux already contains a driver for the i.MX53 keypad, all users must do to add and configure a new custom keypad is to configure the keypad pins on the IOMUX registers and register the driver in the platform file located at `linux/arch/arm/mach-mx5/<your_platform>.c`

Table 17-1 lists the files used in the setup process:

**Table 17-1. Files for Adding/Configuring a New Keypad**

| File Location | Description |
|---|---|
| linux/drivers/input/keyboard/mxc_keyb.c | Device driver file |
| linux/arch/arm/mach-mx5/devices.c | Implements the driver registries |
| linux/arch/arm/mach-mx5/<platform>.c | Machine Layer file |
| linux/include/usr/include/linux/input.h | Input key codes include file |
| linux/arch/arm/plat-mxc/include/mach/iomux-<platform>.h | IOMUX pads definitions |

## 17.1 Configuring Keypad Pins on IOMUX

To use the keypad function, users must first set up the keypad pins on the IOMUX registers. The pad pins can be configured on file `linux/arch/arm/mach-mx5/<platform>.c`, where <platform> is replaced by the appropriate platform file name. For example, the machine layer file used on the i.MX53 reference boards is `linux/arch/arm/mach-mx5/mx53_<reference board name>.c`. This platform is used in the example procedure in this section.

The iomux-mx53.h file contains definitions for all i.MX53 pins. Configure the keypad pins as follows:

```
#define MX53_PAD_KEY_COL0__GPIO_4_6IOMUX_PAD(0x34C, 0x24, 1, 0x0, 0, NO_PAD_CTRL)
#define MX53_PAD_KEY_ROW0__GPIO_4_7IOMUX_PAD(0x350, 0x28, 1, 0x0, 0, NO_PAD_CTRL)
#define MX53_PAD_KEY_COL1__GPIO_4_8IOMUX_PAD(0x354, 0x2C, 1, 0x0, 0, NO_PAD_CTRL)
#define MX53_PAD_KEY_ROW1__GPIO_4_9IOMUX_PAD(0x358, 0x30, 1, 0x0, 0, NO_PAD_CTRL)
#define MX53_PAD_KEY_COL2__GPIO_4_10IOMUX_PAD(0x35C, 0x34, 1, 0x0, 0, NO_PAD_CTRL)
#define MX53_PAD_KEY_ROW2__GPIO_4_11IOMUX_PAD(0x360, 0x38, 1, 0x0, 0, NO_PAD_CTRL)
#define MX53_PAD_KEY_COL3__GPIO_4_12IOMUX_PAD(0x364, 0x3C, 1, 0x0, 0, NO_PAD_CTRL)
#define MX53_PAD_KEY_ROW3__GPIO_4_13IOMUX_PAD(0x368, 0x40, 1, 0x0, 0, NO_PAD_CTRL)
#define MX53_PAD_KEY_COL4__GPIO_4_14IOMUX_PAD(0x36C, 0x44, 1, 0x0, 0, NO_PAD_CTRL)
#define MX53_PAD_KEY_ROW4__GPIO_4_15IOMUX_PAD(0x370, 0x48, 1, 0x0, 0, NO_PAD_CTRL)
```

## 17.2 Creating a Custom Keymap

The input.h file defines codes for general keyboards, as follows.

```
...
#define KEY_HOME          102
#define KEY_UP            103
#define KEY_PAGEUP        104
#define KEY_LEFT          105
#define KEY_RIGHT         106
#define KEY_END           107
#define KEY_DOWN          108
#define KEY_PAGEDOWN      109
#define KEY_INSERT        110
#define KEY_DELETE        111
...
```

Use these labels or add new ones to create your custom keymap.

## 17.3 Configuring the Pads with the Machine Layer File

The mx53_<board name>.c file contains the structures to configure the pads. They are as follows:

```
static struct pad_desc mx53common_pads[] = {
…
…
…
/* Keypad */
        MX53_PAD_KEY_COL0__KEY_COL0,
        MX53_PAD_KEY_ROW0__KEY_ROW0,
        MX53_PAD_KEY_COL1__KEY_COL1,
        MX53_PAD_KEY_ROW1__KEY_ROW1,
        MX53_PAD_KEY_COL2__KEY_COL2,
        MX53_PAD_KEY_ROW2__KEY_ROW2,
        MX53_PAD_KEY_COL3__KEY_COL3,
        MX53_PAD_KEY_ROW3__KEY_ROW3,
        MX53_PAD_KEY_COL4__KEY_COL4,
        MX53_PAD_KEY_ROW4__KEY_ROW4,
…
…
…
};
```

Use the following procedure to configure the pads:

1. Add the configured pin's definitions from the iomux-mx53.h files to the structures in the mx53_<board name>.c file.

**NOTE**

Remove any entry that can cause pin conflict. i.e. MX53_PAD_KEY_COL2_*KEY_COL2* conflicts with MX53_PAD_KEY_COL2_TXCAN1.

2. On init function, set up the pads using the function below:

```
mxc_iomux_v3_setup_multiple_pads(mx53common_pads, ARRAY_SIZE(mx53common_pads));
```

3. Add the keymapping matrix as follows:

```
static u16 keymapping[16] = {
        KEY_UP, KEY_DOWN, KEY_MENU, KEY_BACK,
        KEY_RIGHT, KEY_LEFT, KEY_SELECT, KEY_ENTER,
        KEY_F1, KEY_F3, KEY_1, KEY_3,
        KEY_F2, KEY_F4, KEY_2, KEY_4,
};
```

4. Change the KEYS according to input.h labels and your keypad layout.
5. Add the following structure to configure the keypad:

```
static struct keypad_data keypad_plat_data = {
        .rowmax = 4,
        .colmax = 4,
        .learning = 0,
        .delay = 2,
        .matrix = keymapping,
};
```

6. Register the keypad device. On the same machine layer file, add the following line on function mxc_board_init:

```
mxc_register_device(&mxc_keypad_device, &keypad_plat_data);
```

The new keypad is now implemented.

## 17.4 Enabling the Keypad

Select the keypad on Linux menuconfig. This option is located at:

```
---> Device Drivers
       ---> Input device support
              ---> Keyboards
                     ---> MXC Keypad Driver
```

Build the Linux kernel and boot the board.

## 17.5 Testing the Keypad

There are two simple ways to test the keypad: using `cat` and using Evtest.

### 17.5.1 Using cat to Test the Keypad

On the i.MX53 Linux command line, type the following:

```
cat /dev/input/keyboard0
```

ASCII characters are displayed when keys are pressed.

### 17.5.2 Using Evtest to Test the Keypad

Evtest is a simple software to test inputs. Build it by selecting the respective package on the ltib package list.

On the i.MX53 Linux command line, type the following:

```
evtest /dev/input/keyboard0
```

Evtest displays the information of every key event.

```
Event: time 862.980003, type 1 (Key), code 106 (Right), value 1
Event: time 863.110002, type 1 (Key), code 106 (Right), value 0
Event: time 863.620003, type 1 (Key), code 158 (Back), value 1
Event: time 863.750002, type 1 (Key), code 158 (Back), value 0
Event: time 865.560003, type 1 (Key), code 139 (Menu), value 1
Event: time 865.730002, type 1 (Key), code 139 (Menu), value 0
Event: time 866.150003, type 1 (Key), code 28 (Enter), value 1
Event: time 866.350002, type 1 (Key), code 28 (Enter), value 0
```

# Chapter 18
# Supporting the i.MX53 Reference Board DISP0 LCD

This chapter explains how to support a new LCD on an i.MX53-based board, using display port 0. There are two options for adding support for a new LCD panel without modifying the BSP: letting the BSP calculate the timings using VESA defaults or reducing the blanking time. VESA and reduced blanking work for many LCDs but fail for some devices because of timing configuration constraints. For those devices, we need to modify the BSP and set the proper timing values. Modifying the boot arguments also allows us to include support for the new driver from LTIB device driver menu, call initialization routines, and load the driver by using the boot arguments.

This chapter focuses on the synchronous Parallel0 RGB interface. Common display cards can be attached to this interface. It provides connectivity for the Chunghwa CLAA057VA01CT VGA LCD and the Chunghwa CLAA070VC01 WVGA LCD panel.

Be aware that the DI RGB interface is multiplexed with all other asynchronous parallel interfaces. Therefore, users cannot send data to a synchronous display and another asynchronous parallel display device at the same time in the same DI. Instead, the i.MX53 sends data to the asynchronous panel (smart display) while the synchronous interface is inactive (during horizontal and vertical back porch and front porches). For this reason, the smart display’s frame rate can be affected when multiple displays are attached to the i.MX53.

## 18.1 Supported Display Interfaces

The i.MX53 processor supports the display interfaces shown in Figure 18-1.

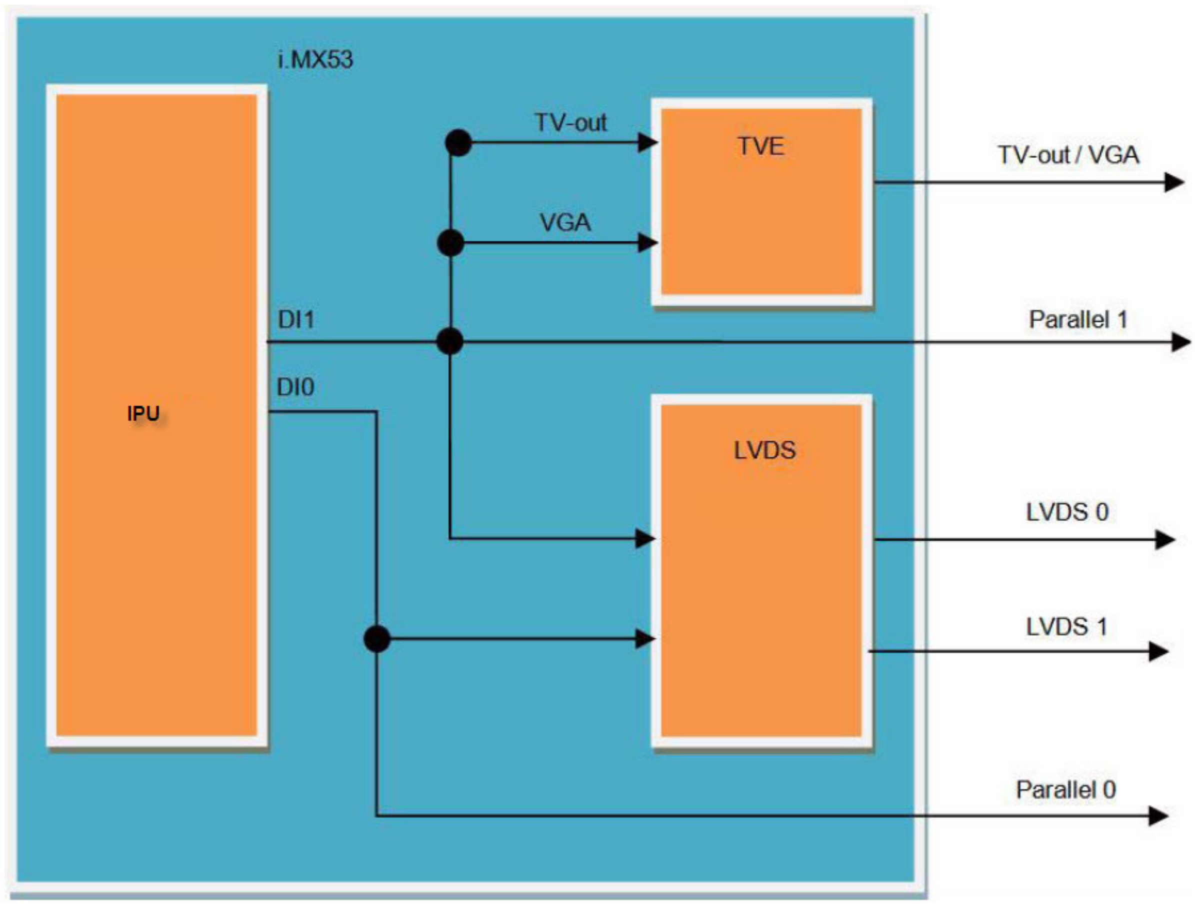



**Figure 18-1. Available Display Interfaces**

Table 18-1 describes the available interfaces.

**Table 18-1. Available Interfaces**

| Feature | IPU (in i.MX53) |
|---|---|
| Number of ports | Two: Full dual-display support |
| Legacy I/F | Parallel and serial.<br>Synchronous (for display refresh) and asynchronous (to memory)<br>Very flexible—glue-less connection to RAM-less displays, display controllers, and TV encoders. |
| MIPI/DSI high-speed I/F | Full Support<br>Up to 2 lanes, 800 Mbps per lane |
| Analog TV-out<br>(composite, S-video, component) | Driven by TVE (Not supported on TO1)<br>Up to 720p at 60 fps or 1080i at 30 fps<br>(720p: 1280x720, 1080i: 1920x1080) |

**Table 18-1. Available Interfaces**

| Feature | IPU (in i.MX53) |
|---|---|
| VGA output | Driven by TVE (Not supported in TO1)<br>Up to WSXGA+ @ 60 Hz, 24 bpp<br>(WSXGA+: 1680x1050) |
| LVDS I/F | Up to UXGA or 2xWXGA @ 60 Hz, 24 bpp<br>(UXGA: 1600x1200, WXGA: 1366x768 |
| **Note:** VGA output is not supported on i.MX53 TO1 processors | |

## 18.2 Adding Support for an LCD Panel

To provide an example for how to add support for an LCD panel, this section shows the code and commands used for adding the support for the CLAA057VA01CT LCD. CLAA057VA01CT is a 5.7" color TFT-LCD (Thin Film Transistor Liquid Crystal Display) module. It is composed of an LCD panel, driver ICs, control circuit, touch screen, and LED backlight. The 5.7" screen produces a high resolution image that is composed of 640 × 480 pixel elements in a stripe arrangement. It uses a 16 bit RGB signal input to display 262K colors.

Figure 18-2 shows the interface between an i.MX53-based board and Chunghwa CLAA057VA01CT 5.7” VGA LCD.

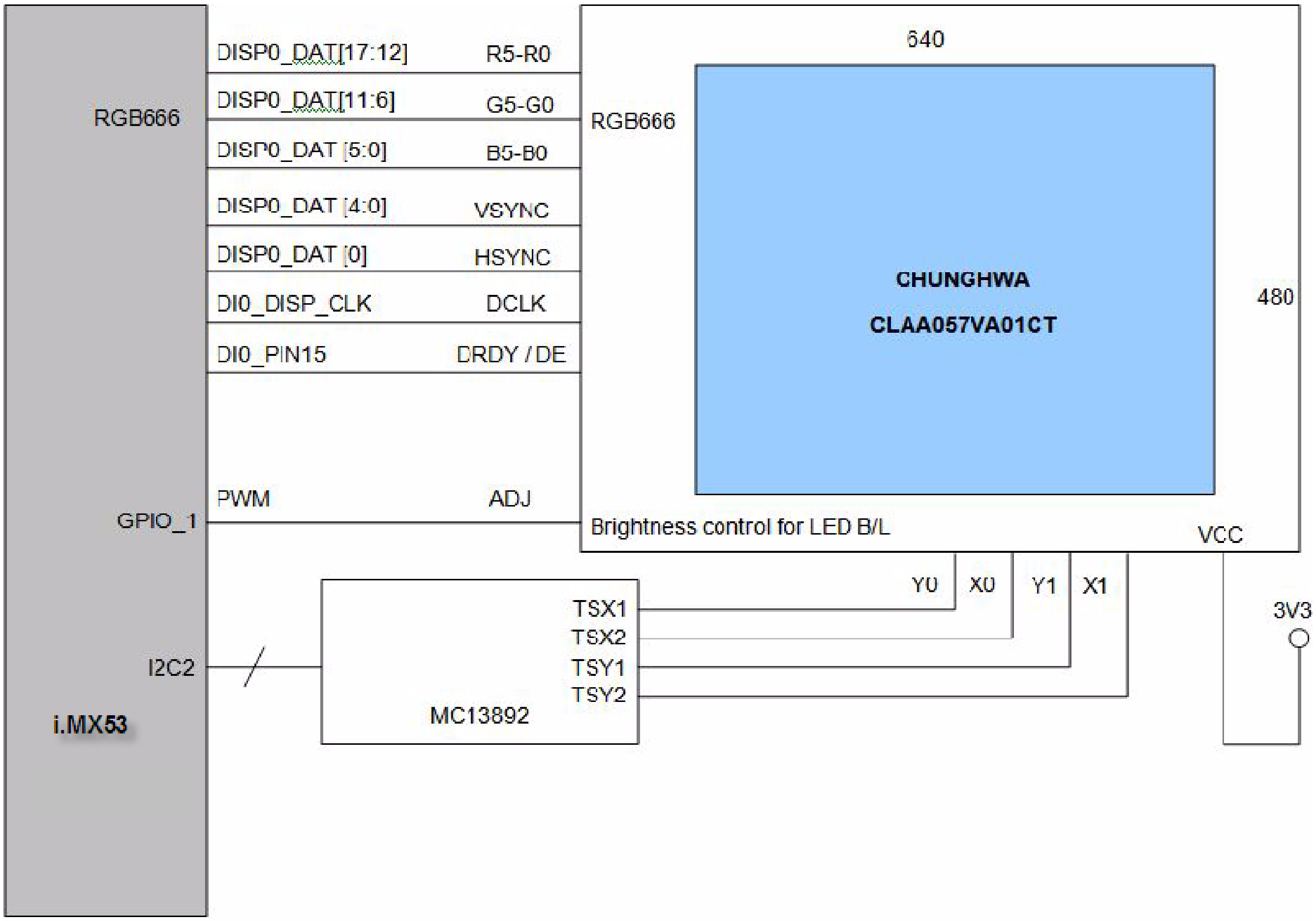



**Figure 18-2. Interface**

The LCD panel requires HSYNC, VSYNC, DE, PIXCLK, and part of the RGB data interface (DISPB_DATA[17:0]). No additional signals, such as a reset signal or serial interface initialization routine commands (SPI or I2C), are required. The backlight unit is controlled by a GPIO signal generated by the i.MX53 (PWM), and the PMIC controls the touch panel interface. The display card includes a connection for this panel.

Table 18-2 shows the timing parameters.

**Table 18-2. Timing Parameters**

| Parameter | Symbol | Min | Typ | Max | Unit |
|---|---|---|---|---|---|
| Screen Height or vertical period | VP | 515 | 525 | 560 | Line |
| VSYNC pulse width | VSW | 1 | 1 | 1 | Line |
| Vertical back porch | VBP | 34 | 34 | 34 | Line |
| Vertical front porch | VFP | 1 | 11 | 46 | Line |

**Table 18-2. Timing Parameters (continued)**

| Parameter | Symbol | Min | Typ | Max | Unit |
|---|---|---|---|---|---|
| Active frame height | VDISP | — | 480 | — | Line |
| Vertical refresh rate | FV | 55 | 60 | 65 | Hz |
| Screen width or horizontal cycle | HP | 750 | 800 | 900 | PIXCLK |
| HSYNC pulse width | HSW | 1 | 1 | 1 | PIXCLK |
| Horizontal back porch | HBP | 46 | 46 | 46 | PIXCLK |
| Horizontal front porch | HFP | 64 | 114 | 214 | PIXCLK |
| Active frame width | HDISP | — | 640 | — | PIXCLK |

## 18.3 Modifying Boot Kernel Parameters to Support a New LCD

Users can use the video and di1_primary kernel parameters to change all timing and interface aspect ratios without writing a single line of code by changing the settings through the default driver.

### 18.3.1 Setting the Video Kernel Parameter

The video kernel parameter is a multipurpose parameter used to configure display features in either display port 0 or display port 1. It controls the following features:

- Display resolution
- Pixel color depth
- Refresh rate
- IPU display output interface format

See the modedb.txt file located at `Documentation/fb/modedb.txt` for specific parameter information.

To set the parameter information for the video argument, use the following format. Variables between square brackets are optional.

```
<interface_id>:<ipu_out_fmt>,<xres>x<yres>[M][R][-<bpp>][@<refresh>][i][m]<name>[-<bpp>][@
<refresh>]
```

Table 18-3 defines the variables.

**Table 18-3. Parameter Information**

| Argument Name | Definition | Units | Values |
|---|---|---|---|
| interface_id | Display interface id (DI0/DI1) | NA | mxcdi0fb, mxcdi1fb |
| interface_pix_fmt | Display Interface output format | NA | RGB565, RGB666, RGB24, YUV444 |
| name | Video mode name | NA | String name |
| xres | Horizontal resolution | pixels | Decimal value |
| yres | Vertical resolution | lines | Decimal value |

**Table 18-3. Parameter Information**

| Argument Name | Definition | Units | Values |
|---|---|---|---|
| M | Timing calculated using VESA(TM) | NA | M |
| R | Timing using reduced blanking | NA | R |
| bpp | Bits per pixel on frame buffer | bits | Decimal value (16 or 24) |
| refresh | LCD refresh rate | Hz | Decimal value |

When <name> is included in the mode_option argument parameters, the timing is not calculated. Instead, it is extracted from BSP code. Valid default modes can be found at `linux/drivers/video/modedb.c` and in files placed at `linux/drivers/video/mxc folder`.

**Example 18-1. DVI Monitor Connected to Display Port 0**

For a DVI monitor connected to display port 0, the kernel command is

```
video=mxcdi0fb:RGB24,1024x768M-16@60
```

Table 18-4 shows how the values in this example correspond to the argument names defined in Table 18-3.

**Table 18-4. XGA DVI Monitor Example Variables**

| Argument Name | Value | Definition |
|---|---|---|
| interface_id | mxcdi0fb | Display interface 0 settings |
| interface_pix_fmt | RGB24 | RGB bus is 24 bits width DISP0_DAT[23:0]- RGB888 |
| name | Not used in this command | — |
| xres | 1024 | 1024 pixels (Horizontal) |
| yres | 768 | 768 lines (vertical) |
| M | M | Timings calculated using VESA(TM) |
| R | Not used in this command | — |
| bpp | 16 | Frame buffer is 16 bits per pixel |
| refresh | 60 | 60 Hz |

**Example 18-2. VGA LCD Connected to Display Port 0**

For a VGA LCD connected to display port 0, the kernel command is

```
video=mxcdi0fb:RGB565,800x480M-16@55
```

Table 18-5 shows how the values in this example correspond to the argument names defined in Table 18-3.

**Table 18-5. VGA LCD Example Variables**

| Argument Name | Value | Definition |
|---|---|---|
| interface_id | mxcdi0fb | Display interface 0 settings |
| interface_pix_fmt | RGB565 | RGB bus is 16 bits width DISP0_DAT[16:0]- RGB565 |
| name | Not used in this command | — |
| xres | 800 | 800 pixels (Horizontal) |
| yres | 480 | 480 lines (vertical) |
| M | M | Timing calculated using VESA (TM) |
| R | Not used in this command | — |
| bpp | 16 | Frame buffer is 16 bits per pixel |
| refresh | 55 | 55 Hz |

**Example 18-3. 720P TV Connected to Display Port 1**

For a 720P TV connected to display port 1, the kernel command is `video=mxcdi1fb:YUV444,720P60`

Table 18-6 shows how the values in this example correspond to the argument names defined in Table 18-3.

**Table 18-6. 720P TV Example Variables**

| Argument Name | Value | Definition |
|---|---|---|
| interface_id | mxcdi1fb | display interface 1 settings |
| interface_pix_fmt | YUV444 | YUV444 output |
| name | 720P60 | Defined in linux/drivers/video/mxc/tve.c |
| xres | Not used in this command | From tve.c: 1280 |
| yres | Not used in this command | From tve.c: 720 |
| M | Not used in this command | — |
| R | Not used in this command | — |
| bpp | Not used in this command | — |
| refresh | Not used in this command | — |

## 18.3.2 Setting the di1_primary Kernel Parameter

The di1_primary kernel parameter informs the kernel/driver that DI1 is the primary display interface instead of DI0, which is the default setting. Set this kernel parameter by adding the label di1_primary to the boot kernel arguments.

For example, the kernel command for an XGA LVDS connected to LVDS0 using DI1 as the primary display interface is as follows:

```
video=mxcdi0fb:RGB24,LDB-XGA di1_primary
```

### 18.3.3 Modifying the Bits per Pixel Setting

The default bits per pixel setting is 16 bits. To change the default value to another depth, modify the relevant lines in the mxc_ipuv3_fb.c file located at

```
<ltib dir>/rpm/BUILD/linux/drivers/video/mxc/mxc_ipuv3_fb.c
```

**Example 18-4. Changing the Frame Buffer to 32 bpp**

The following example code shows how to change the frame buffer from 16 bpp to 32 bpp.

```
static int mxcfb_probe(struct platform_device *pdev)
{
   ...

   if (!mxcfbi->default_bpp)
      mxcfbi->default_bpp = 16;
   // Change Default BPP to 32 bpp
   printk(KERN_INFO "mxcfb_probe() - default_bpp = 32\r\n");
   mxcfbi->default_bpp = 32;
   ...
   return ret;
}
```

Check the frame buffer bpp and other settings in the /sys/class folder. The output should look like the following:

```
root@freescale ~$ cd /sys/class/graphics/fb0/
root@freescale /sys/devices/platform/mxc_sdc_fb.1/graphics/fb0$ ls
bits_per_pixel     device             pan                subsystem
blank              fsl_disp_property  power              uevent
console            mode               rotate             virtual_size
cursor             modes              state
dev                name               stride
root@freescale /sys/devices/platform/mxc_sdc_fb.1/graphics/fb0$ cat bits_per_pixel
32
```

Note that the final line shows the bits per pixel to be 32, reflecting our change from the default of 16 bpp.

### 18.3.4 Modifying Display Timing for CLAA057VA01CT Using Kernel Parameters

By using the video and di1_primary kernel parameters, the frame buffer driver is able to change all timing and interface aspect ratios. Timing is calculated using the VESA standard.

The video parameter format is <interface_id>:<ipu_out_fmt>,<xres>x<yres>[M][R][-<bpp>][@<refresh>][i][m]<name>[-<bpp>][@<refresh>] with variables between square brackets optional.

Table 18-7 provides sample values for an interface.

**Table 18-7. Sample Values**

| Argument Name | Value | Definition |
|---|---|---|
| interface_id | mxcdi0fb | display interface 0 settings |
| interface_pix_fmt | RGB666 | RGB bus is 18 bits width DISP0_DAT[17:0]- RGB565 |
| name | Not used in this command | — |
| xres | 640 | 640 pixels (Horizontal) |
| yres | 480 | 480 lines (vertical) |
| bpp | 16 | Frame buffer is 16 bits per pixel |
| refresh | 55 | 55 Hz |
| M | M | Timing calculated using VESA(TM) |
| R | Not used in this command | — |

For these sample values, the video parameter is `video=mxcdi0fb:RGB666,640x480M-16@55`

**Example 18-5. Kernel Image Stored in the SD; file system from NFS**

The following commands are for a kernel image stored in the SD with a file system loaded from NFS. All commands are executed on U-Boot:

To configure the network, use the following:

```
EVK U-Boot > setenv serverip 10.112.98.65
EVK U-Boot > setenv fec_addr 00:04:9f:00:ea:d3
EVK U-Boot > setenv ethaddr 00:04:9f:00:ea:d3
EVK U-Boot > setenv bootfile uImage
EVK U-Boot > saveenv
EVK U-Boot > reset
```

To load uImage from server, use the following:

```
EVK U-Boot > bootp $ {loadaddr} ernesto/53/uImage
```

To write uImage to SD, use the following:

```
EVK U-Boot > mmc write 0 ${loadaddr} 0x800 0x1800
Set NFS file system path
EVK U-Boot > setenv nfsroot /tftpboot/ernesto/ltib
```

To configure boot arguments to use NFS and CLAA057VA01CT LCD panel, use the following:

```
EVK U-Boot > setenv bootargs_mmc 'setenv bootargs ${bootargs} root=/dev/nfs ip=dhcp
video=mxcdi0fb:RGB666,640x480M-16@55 nfsroot=${serverip}:${nfsroot},v3,tcp'
Launch OS image
EVK U-Boot > run bootcmd_mmc
```

## 18.4 Adding Support for a New LCD

If neither VESA nor reducing the blanking calculation works for your LCD or if you need a special function, add the support for the new LCD in the BSP.

Perform the following steps to modify the i.MX53 BSP to add the support for synchronous panels:

1. Add a display entry in the ltib catalog.
2. Create the madglobal LCD panel file.
3. Add compilation flag for the new display.
4. Configure LCD timings and display interface.
5. Use boot command to select the new LCD.

The following subsections describe these steps in detail.

### 18.4.1 Adding a Display Entry in the ltib Catalog

Select specific display drivers in `Device Drivers > Graphics support`.

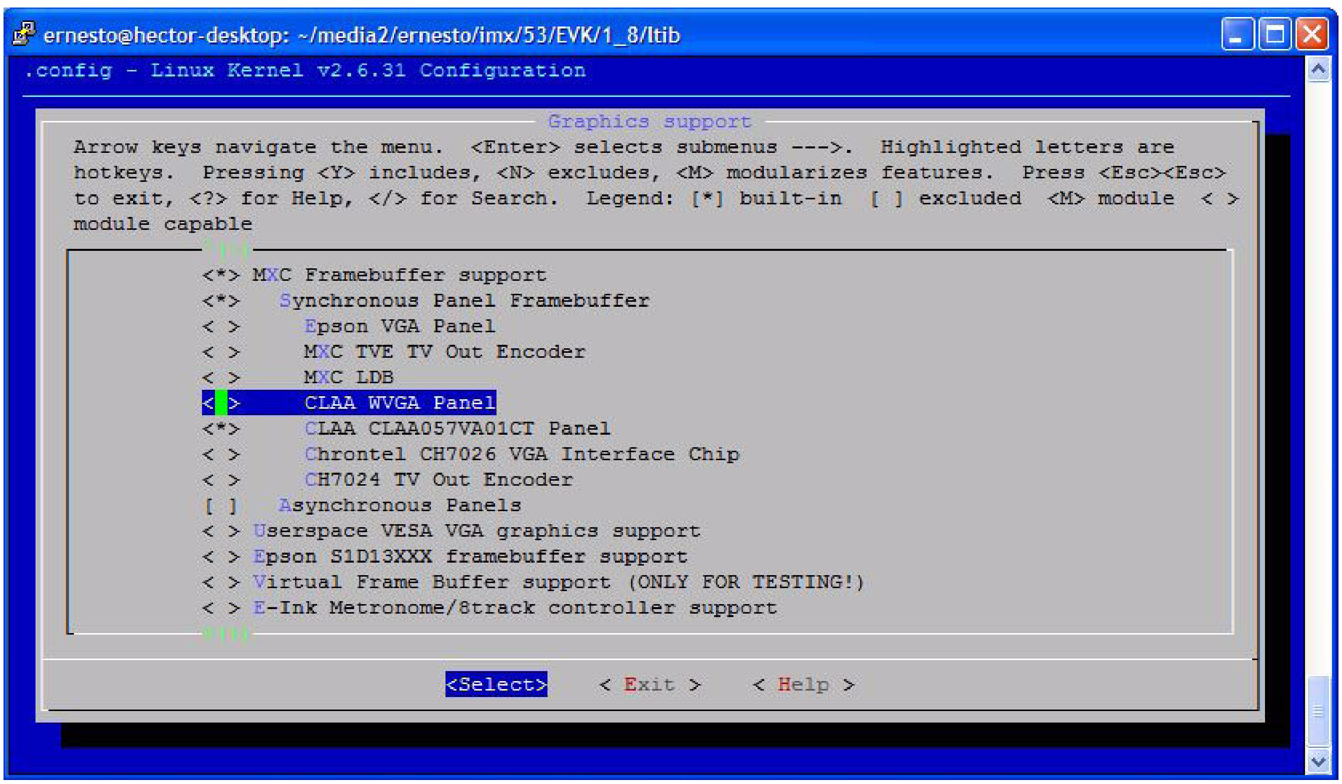



**Figure 18-3. Graphics Support Options Menu**

To add an entry for a new LCD, perform the following steps:

1. Enter the i.MX53 display specific folder as follows.

```
$ cd <ltib dir>/rpm/BUILD/linux/drivers/video/mxc
```

2. Open the Kconfig file with the command `gedit Kconfig &`
3. Use the following code to add the entry where you want it to appear.

```
config FB_MXC_CLAA057VA01CT_SYNC_PANEL
       depends on FB_MXC_SYNC_PANEL
       tristate "CLAA CLAA057VA01CT Panel"
```

### 18.4.2 Creating the LCD Panel File (initialization, reset, power settings, backlight)

Because power settings are handled by the ATLAS APL PMIC and other voltage regulators, the display driver must configure the APL PMIC during initialization to set up the power voltage configuration if this has not already been done. Also, the reset waveform and initialization routine must be included. To do these tasks, create an LCD file with panel-specific functions at the following location:

`<ltib dir>/rpm/BUILD/linux/drivers/video/mxc/mxcfb_CLAA057VA01CT.c`

**WARNING**

> Before connecting an LCD panel to the i.MX53 board, check whether the LCD is powered with the proper supply voltages and whether the display data interface has the correct VIO value. Incorrect voltages and values may harm the device.

The LCD file must include the definition of four basic functions described in Table 18-8 and can include other functions and macros as needed.

**Table 18-8. Required Functions**

| Function Name | Function Declaration | Description |
|---|---|---|
| lcd_probe | static int __devinit lcd_probe(struct platform_device *pdev) | Called when the LCD module is loaded. It should contain, pmic configuration, reset, power on sequence and the initialization routine. |
| lcd_remove | static int __devexit lcd_remove(struct platform_device *pdev) | Called when the LCD module is removed. It should contain power off pmic configuration, power off sequence and the de-initialization routine. |
| lcd_suspend | static int lcd_suspend(struct platform_device *pdev, pm_message_t state) | Not always implemented, but used for enhance low power modes on the device. Usually called when system goes to suspend mode. |
| lcd_resume | static int lcd_resume(struct platform_device *pdev) | Not always implemented, but used for enhance low power modes on the device. Usually called when system returns from suspend mode. |

Next, create a platform device that can be loaded and unloaded. This example declares the new platform device using the devices.h and devices.c files located at:

`<ltib dir>//rpm/BUILD/linux/arch/arm/mach-mx5/`

1. Declare the plaform device on madglobal:devices.h using the following:

   ```
   extern struct platform_device mxc_disp_devices[];
   ```

2. Add a new entry on madglobal:devices.c using the following:

   ```
   struct platform_device mxc_disp_devices[] = {
      {
         .name = "lcd_CHUNGHWA_CLAA057VA01CT",
         .id = 0,
   ```

```
    },
};
```

Be careful to use the same name for the new platform device entry as the name included in madglobal:mxcfb_CLAA057VA01CT.c for the driver.

```
static struct platform_driver lcd_driver = {
 .driver = {
     .name = "lcd_CHUNGHWA_CLAA057VA01CT"},
 .probe = lcd_probe,
 .remove = __devexit_p(lcd_remove),
 .suspend = lcd_suspend,
 .resume = lcd_resume,
};
```

3. Register the device at `<ltib dir>//rpm/BUILD/linux/arch/arm/mach-mx5/madglobal:mxc53_<reference board name>.c` by using the following code:

```
static int __init mxc_init_fb(void)
{
 ....
 mxc_register_device(&mxc_disp_devices[0], NULL);
 ....
 return 0;
}
```

### 18.4.3 Adding the Compilation Flag for the New Display

After the LCD file has been created and the entry has been added to the Kconfig file, modify the makefile to include the LCD file in the compilation by using the code shown below. The makefile is in the same folder as the new LCD file: `<ltib dir>/rpm/BUILD/linux/drivers/video/mxc/makefile`

```
ifeq ($(CONFIG_ARCH_MX21)$(CONFIG_ARCH_MX27)$(CONFIG_ARCH_MX25),y)
        obj-$(CONFIG_FB_MXC_TVOUT)              += fs453.o
        obj-$(CONFIG_FB_MXC_SYNC_PANEL)         += mx2fb.o mxcfb_modedb.o
        obj-$(CONFIG_FB_MXC_EPSON_PANEL)        += mx2fb_epson.o
else
ifeq ($(CONFIG_MXC_IPU_V1),y)
        obj-$(CONFIG_FB_MXC_SYNC_PANEL)         += mxcfb.o mxcfb_modedb.o
else
        obj-$(CONFIG_FB_MXC_SYNC_PANEL)         += mxc_ipuv3_fb.o
endif
        obj-$(CONFIG_FB_MXC_EPSON_PANEL)        += mxcfb_epson.o
        obj-$(CONFIG_FB_MXC_EPSON_QVGA_PANEL)   += mxcfb_epson_qvga.o
        obj-$(CONFIG_FB_MXC_TOSHIBA_QVGA_PANEL) += mxcfb_toshiba_qvga.o
        obj-$(CONFIG_FB_MXC_SHARP_128_PANEL)    += mxcfb_sharp_128x128.o
endif
obj-$(CONFIG_FB_MXC_EPSON_VGA_SYNC_PANEL)   += mxcfb_epson_vga.o
obj-$(CONFIG_FB_MXC_CLAA_WVGA_SYNC_PANEL)   += mxcfb_claa_wvga.o
obj-$(CONFIG_FB_MXC_CLAA057VA01CT_SYNC_PANEL)    += mxcfb_CLAA057VA01CT.o
obj-$(CONFIG_FB_MXC_TVOUT_CH7024)           += ch7024.o
obj-$(CONFIG_FB_MXC_TVOUT_TVE)              += tve.o
obj-$(CONFIG_FB_MXC_LDB)                    += ldb.o
obj-$(CONFIG_FB_MXC_CH7026)        += mxcfb_ch7026.o
#obj-$(CONFIG_FB_MODE_HELPERS)    += mxc_edid.o
```

Note that a new object, mxcfb_CLAA057VA01CT.o, is created when CONFIG_FB_MXC_CLAA057VA01CT_SYNC_PANEL flag is set. The LCD module with the initialization and de-initialization routines is only available to the kernel after this object has been created. If the LCD does not need a particular configuration, you may omit the usage of the LCD file and discard any changes on Kconfig and Makefile.

### 18.4.4 Configuring LCD Timings and the Display Interface

To support the new LCD, include the specification for the following LCD characteristics in the madglobal:mxc53_<reference board name>.c file (located at `<ltib dir>//rpm/BUILD/linux/arch/arm/mach-mx5/madglobal:mxc53_<board name>.c`):

- Display resolution
- Pixel color depth
- Refresh rate
- RGB display waveform description.
- IPU display output interface format

For the display, resolution, refresh rate, and RGB display waveform descriptions, add a new `fb_videomode struct` into the video_modes[] array based on the LCD timing and waveforms. See the CLAA-VGA entry on the following example code.

```
static struct fb_videomode video_modes[] = {
      {
       /* NTSC TV output */
       "TV-NTSC", 60, 720, 480, 74074,
       122, 15,
       18, 26,
       1, 1,
       FB_SYNC_HOR_HIGH_ACT | FB_SYNC_VERT_HIGH_ACT | FB_SYNC_EXT,
       FB_VMODE_INTERLACED,
       0,},

        ......

      {
       /* 640x480 @ 60 Hz */
       "CLAA-VGA", 60, 640, 480, 39683, 45, 114, 33, 11, 1, 1,
       FB_SYNC_CLK_LAT_FALL,
       FB_VMODE_NONINTERLACED,
       0,},
};
```

After including the new entry for the CLAA057VA01CT into the video_modes[] array, point the DISP0 configuration to this new `fb_videomode`. The link can be done by using an `mxc_fb_platform_data struct` when the frame buffer device is registered, as follows.

```
static struct mxc_fb_platform_data CLAA057VA01CT_fb_data =
{
   .interface_pix_fmt = IPU_PIX_FMT_RGB666,
   .mode_str = "CLAA-VGA",
   .mode = video_modes,
   .num_modes = ARRAY_SIZE(video_modes),
```

```
};
```

Use this new `mxc_fb_platform_data struct` to register DISP0 in the mxc_init_fb() function, as follows.

```
static int __init mxc_init_fb(void)
{
   ...
   memcpy(&fb_data[0], &CLAA057VA01CT_fb_data, sizeof(struct mxc_fb_platform_data));
   mxc_register_device(&mxc_disp_devices[0], NULL);
   ...
   return 0;
}
```

This code replaces the default WVGA panel settings with the new LCD entry.

### 18.4.5 Adding BSP Support for a New Boot Command to Select CLAA057VA01CT LCD

To take advantage of the kernel boot process, use the kernel boot arguments to select the new LCD. Do this by using the following steps to add extra code to the mxc53_<board name>.c file located at `<ltib dir>//rpm/BUILD/linux/arch/arm/mach-mx5/mxc53_<board name>.c`

1. Select a new flag; this example code uses CLAA057VA01CT.
2. Create a variable to save if the CLAA057VA01CT flag has been included in the command line. In this case 0 is the initial value and it means that flag is not present.

   ```
   static int __initdata enable_CLAA057VA01CT = { 0 };
   ```
3. Use the following code to set the enable_CLAA057VA01CT variable if “CLAA057VA01CT” is included on the command line.

   ```
   static int __init CLAA057VA01CT_setup(char *__unused)
   {
      printk(KERN_INFO "CLAA057VA01CT flag\r\n");
      enable_CLAA057VA01CT = 1;
      return 1;
   }
   __setup("CLAA057VA01CT", CLAA057VA01CT_setup);
   ```

   Once the enable_CLAA057VA01CT variable is set, it is easy to distinguish which LCD should be used on DISP0 interface. The code looks like the following:

   ```
   static int __init mxc_init_fb(void)
   {
      ...

      if(enable_CLAA057VA01CT)
      {
         memcpy(&fb_data[0], &CLAA057VA01CT_fb_data, sizeof(struct mxc_fb_platform_data));
         mxc_register_device(&mxc_disp_devices[0], NULL);
         printk(KERN_INFO "DI0 driving CLAA057VA01CT\n");
      }
      else
      {
         printk(KERN_INFO "DI0 driving CLAA-WVGA\n");
      }
      ...
   ```

```
    return 0;
}
```

4. Modify the boot command with the following code.

```
EVK U-Boot > setenv bootargs_mmc 'setenv bootargs ${bootargs} root=/dev/nfs ip=dhcp
CLAA057VA01CT nfsroot=${serverip}:${nfsroot},v3,tcp'

EVK U-Boot > run bootcmd_mmc
```

## 18.5 i.MX53 Display Interface Helpful Information

TFT panels are handled by the i.MX53 IPU legacy interface, which enables the display port to use different types of display interfaces that conform to the following features and restrictions.

- The IPU supports four displays in total.
- The display port has two DI interfaces.
- Each interface can handle up to three displays.
- Each DI can handle up to two asynchronous interfaces (e.g. Smart LCD, Graphic accelerator).
- Only one asynchronous interface per DI can be serial.
- Each DI can handle one synchronous interface (e.g. TV, dumb LCD).
- Asynchronous displays that are accessed in synchronous mode are considered a synchronous interface (dual_mode).

Each DI in the i.MX53 can handle one synchronous (dumb display).

Figure 18-4 shows an example i.MX53 board display interface layout.

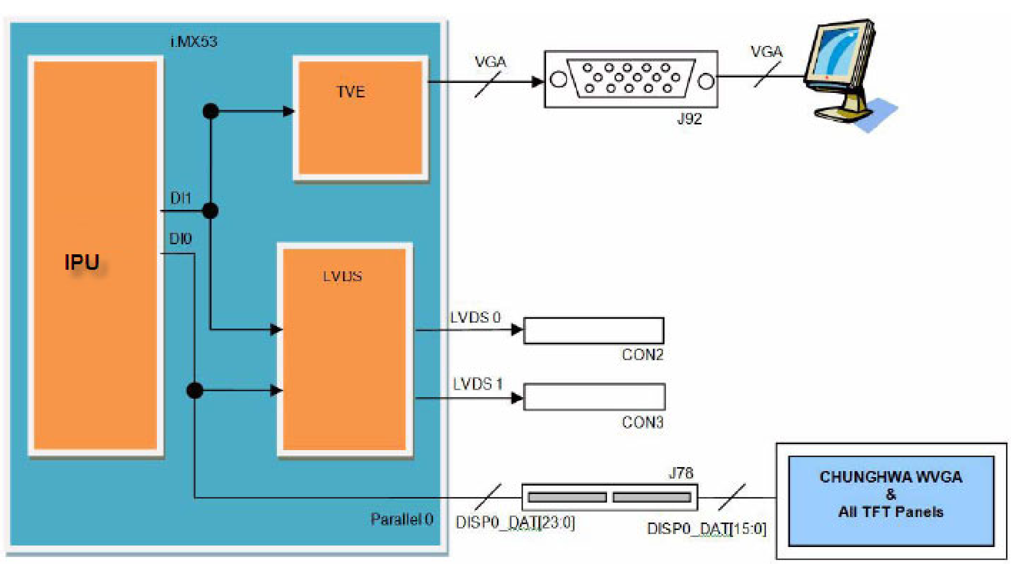



**Figure 18-4. i.MX53 Board Display Interface Layout**

The example board’s two display interface (DI) modules are each configured to handle one or more different kind of panels. The DI module is responsible for the timing waveforms for each signal in its display’s interface. It is composed of the following:

- 8 sets of waveform generators (PIN1–PIN8) that control signals associated with the DI’s clock, such as VSYNC and HSYNC.
- 12 sets of waveform generators (PIN11–PIN17 + 2 CS), controlling signals associated with data, such as DRDY/DE, CS, or RS

The DI also generates the display’s clock based on IPU HSP_CLK or from an external clock (PLL or pin).

The IPU provides the flexibility to select from a range of pins to use as an output for the synchronization signals. Therefore, there is no unique pin for VSYNC, HSYNC and DE. However, the i.MX51 reference boards have been assigned a specific pin for each signal, which is reflected in the schematics and BSP support.

To develop a system with a new LCD panel that does not have a driver already implemented, it is necessary to implement the new driver into the Linux's kernel and do it taking the advantage of all processor's hardware designed to the respective task, like the IPU from the i.MX53 in order to enhance the processor's performance.

For additional details about timing and TFT signals, see AN3977, AN3978, and AN4041 (available on the Freescale website).

# Chapter 19
# Connecting an LVDS Panel to an i.MX53 Reference Board

This chapter explains how to connect the LVDS panel to an i.MX53 reference board. The i.MX53 processor has an LVDS display bridge (LDB) block that drives LVDS panels without external bridges. The LDB supports the flow of synchronous RGB data from the IPU to external display devices through the LVDS interface. This support covers the following activities:

- Connectivity to relevant devices—display with an LVDS receiver.
- Arranging the data as required by the external display receiver and by LVDS display standards.
- Synchronization and control capabilities.

## 19.1 Connecting an LVDS Panel to the i.MX53 EVK Board

The following LVDS panels were tested on the i.MX53 reference boards:

- LG display (model number: LB150X02)
- 150XG01
- 1080p LVDS panel (model number: M216H1-L01)
- Sharp (model number: LQ084S3LG01)

The kernel command line for 24-bit LVDS panels (4 pairs of LVDSdata signals) displays the following lines if the panel is properly connected.

```
DI0 (LDB0 CON2 on top of board): video=mxcdi0fb:RGB24,XGA ldb
DI1 (LDB1 CON3 on bottom of board): video=mxcdi1fb:RGB24,XGA di1_primary ldb
```

The kernel command line for 18-bit LVDS panels (3 pairs of LVDS data signals) displays the following lines if the panel is properly connected.

```
DI0 (LDB0 CON2 on top of board): video=mxcdi0fb:RGB666,XGA ldb
DI1 (LDB1 CON3 on bottom of board): video=mxcdi1fb:RGB666,XGA di1_primary ldb
```

## 19.2 Enabling an LVDS Channel

The LDB driver source code is available at `<ltib_dir>/rpm/BUILD/linux/drivers/video/mxc/ldb.c`. To make a built-in LDB driver functional, add the ‘ldb’ option to the kernel command line. The driver configures the LDB when the device is probed.

When the LDB device is probed properly, the driver uses platform data information to configure the LDB’s reference resistor mode and regulator. The LDB driver probe function also tries to match video modes for external display devices with an LVDS interface. The display signal polarities and LDB control bits are set according to the matched video modes.

The LVDS channel mapping mode and the LDB bit mapping mode of LDB are set according to the boot up LDB option chosen by the user. If the user has not specified an option but the video mode can be found in the local video mode database, the driver chooses an appropriate LDB setting. If no video mode is matched, nothing is done in probe function. Users can set up the LDB later by using ioctrls. The LDB will be fully enabled in probe function if the driver finds that the primary display device is a single display device with an LVDS interface.

The steps the driver takes to enable a LVDS channel are as follows:

1. Set ldb_di_clk's parent clock and the parent clock's rate.
2. Set ldb_di_clk's rate.
3. Enable both ldb_di_clk and its parent clock.
4. Set the LDB in a proper mode, including display signals' polarities, LVDS channel mapping mode, bit mapping mode, reference resistor mode.

## 19.2.1 Locating Menu Configuration Options

Linux kernel configuration options are provided for the build-in status to enable this module. To locate these options, use the following procedure.

1. Go to `<ltib dir>`.
2. Use the ./ltib -c command.
3. Select **Configure the Kernel** on the screen displayed and exit.
4. When the next screen appears, follow this sequence: `Device Drivers > Graphics support > MXC Framebuffer support > Synchronous Panel Framebuffer > MXC LDB`

## 19.2.2 Programming Interface

The APIs in the mxc_ldb_ioctl() function control every other LDB unit setting. The user may call these these APIs to set LDB modes or to enable and disable the LDB.

## 19.3 LDB Ports

Figure 19-1 shows the LDB block.

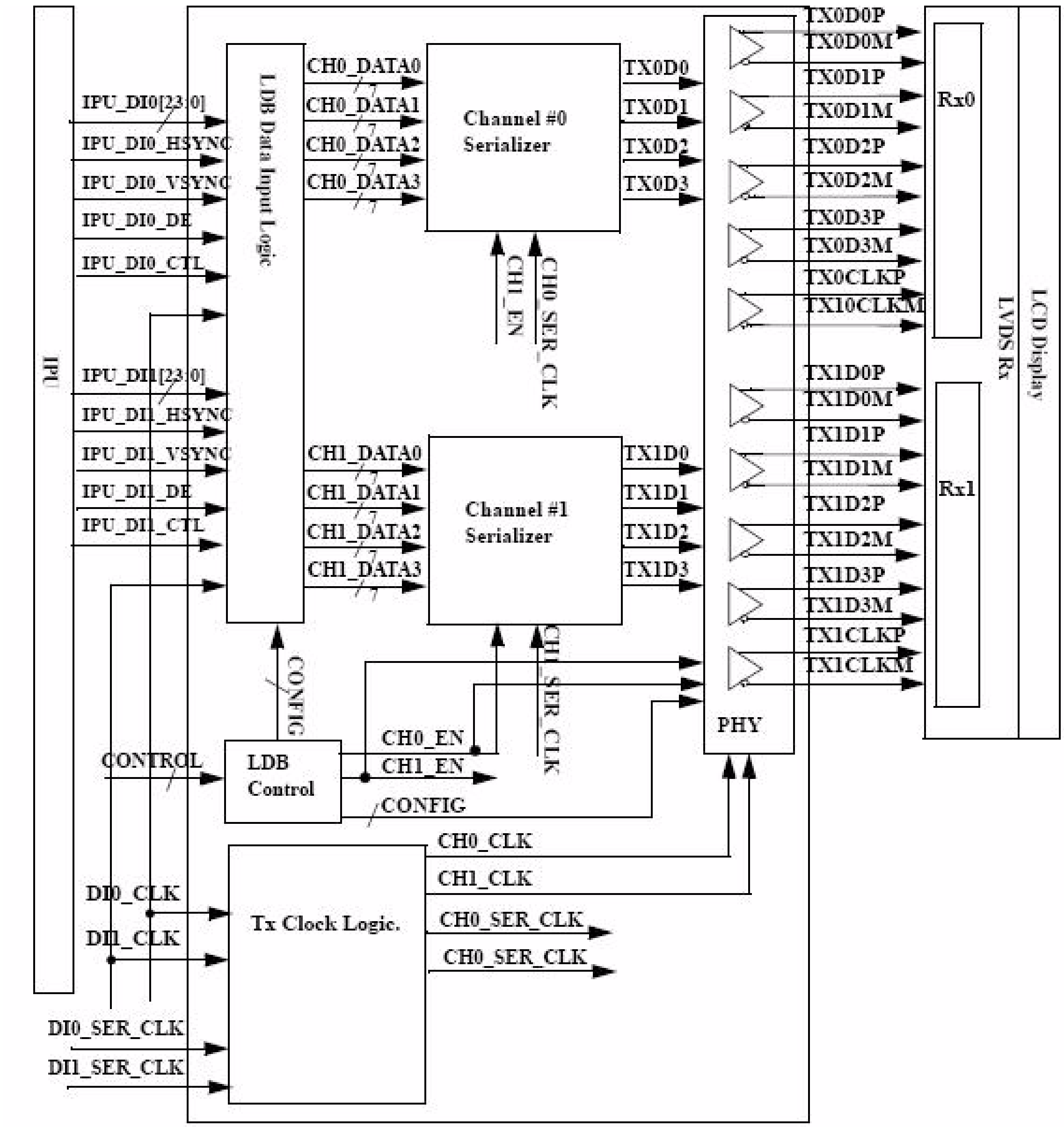



**Figure 19-1. i.MX53 LVDS Display Bridge (LDB) Block**

The LDB has the following ports:

- Two input parallel display ports.
- Two output LVDS channels
- Control signals to configure LVDS parameters and operations.
- Clocks from the SoC PLLs.

### 19.3.1 Input Parallel Display Ports

The LDB is configurable to support either one or two (DI0, DI1) parallel RGB input ports. The LDB only supports synchronous access mode.

Each RGB data interface contains the following:

- RGB data of 18 or 24 bits
- Pixel clock
- Control signals
  - HSYNC, VSYNC, DE, and one additional optional general purpose control
  - Transfers a total of up to 28 bits per data interface per pixel clock cycle

The LVDS supports the following data rates:

- For dual-channel output: up to 170 MHz pixel clock (e.g. UXGA—1600 × 1200 at 60 Hz + 35% blanking)
- For single-channel output: up to 85 MHz per interface. (e.g. WXGA—1366 × 768 at 60 Hz + 35% blanking).

### 19.3.2 Output LVDS Ports

The LDB has two LVDS channels, which are used to communicate RGB data and controls to external LCD displays either through the LVDS interface or through LVDS receivers. Each channel consists of four data pair and one clock pair, with a pair meaning an LVDS pad that contains PadP and PadM.

The LVDS ports may be used as follows:

- One single-channel output
- One dual channel output: single input, split to two output channels
- Two identical outputs: single input sent to both output channels
- Two independent outputs: two inputs sent, each to a different output channel

## 19.4 Further Reading

Please consult the following reference materials for additional information:

- i.MX53 Multimedia Applications Processor Reference Manual
- i.MX53 START Linux Reference Manual, included as part of the Linux BSP

# Chapter 20
# Supporting the i.MX53 Camera Sensor Interface CSI0

This chapter provides information about how to use the expansion connector to include support for a new camera sensor on an i.MX53 reference board. It explains how to do the following:

- Configure the CSI unit in test mode (Section 20.3, "Configuring the CSI Unit in Test Mode")
- Add support for a new CMOS sensor in the i.MX53 BSP (L2.6.31_10.07.11) (Section 20.4, "Adding Support for a New CMOS Camera Sensor")
- Set up and use the $I^2C$ interface to handle your camera bus (Section 20.5, "Using the $I^2C$ Interface)
- Load and test the camera module (Section 20.6, "Loading and Testing the Camera Module")

It also provides reference information about the following:

- Required software and hardware
- i.MX53 reference CSI interfaces layout (Section 20.2, "i.MX53 CSI Interfaces Layout")
- CMOS sensor interfaces (CSI) supported by the i.MX53 (IPU) (Section 20.7.1, "CMOS Interfaces Supported by the i.MX53)
- i.MX53 EVK CSI parallel interface (Section 20.7.2, "i.MX53 CSI Parallel Interface")
- i.MX53 CSI test mode (Section 20.7.3, "Timing Data Mode Protocols")

## 20.1 Required Software

In Freescale BSPs, all capture devices are based on the V4L2 standard. Therefore, only the CMOS-dependent layer needs to be modified to include a new CMOS sensor; all other layers have been developed to work with V4L2.

Required development tools are as follows:

- Linux host with i.MX53 Linux L2.6.31_10.07.11 or newer
- Serial port terminal (such as Hyperterminal, TeraTerm, Minicom).

## 20.2 i.MX53 CSI Interfaces Layout

Figure 20-1 shows the camera interface layout on an i.MX53-based board.

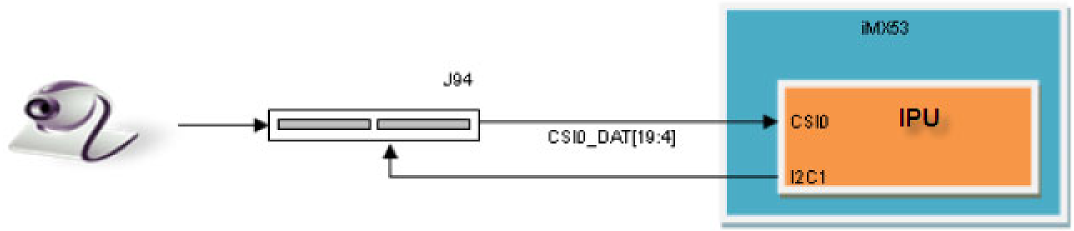



**Figure 20-1. Camera Interface Layout**

Only CSI0 is used for the purpose of this document. The usage of CSI1 depends on its availability on the board being used.

## 20.3 Configuring the CSI Unit in Test Mode

This chapter uses the test mode for its example scenario of a new camera driver that generates a chess board. Setting the TEST_GEN_MODE register places the device in test mode, which is used for debugging. The CSI generates a frame by itself and sends it to one of the destination units. The sent frame is a chess board composed of black and configured color squares. The configured color is set with the registers PG_B_VALUE, PG_G_VALUE and PG_R_VALUE. The data can be sent in different frequencies according to the configuration of DIV_RATIO register.

When CSI is in test mode, configure the CSI unit with a similar configuration to the described settings in Table 20-1. Call ipu_csi_init_interface() to configure the CSI interface protocol, formats and features.

**Table 20-1. Settings for Test Mode**

| Bit Field | Value | Description |
|---|---|---|
| CSI0_DATA_DEST | 0x4 | Destination is IDMAC via SMFC |
| CSI0_DIV_RATIO | 0x0 | SENSB_MCLK rate = HSP_CLK rate |
| CSI0_EXT_VSYNC | 0x1 | External VSYNC mode |
| CSI0_DATA_WIDTH | 0x1 | 8 bits per color |
| CSI0_SENS_DATA_FORMAT | 0x0 | Full RGB or YUV444 |
| CSI0_PACK_TIGHT | 0x0 | Each component is written as a 16 bit word where the MSB is written to bit #15, color extension is done for the remaining least significant bits. |
| CSI0_SENS_PRTCL | 0x1 | Non-gated clock sensor timing/data mode |
| CSI0_SENS_PIX_CLK_POL | 0x1 | Pixel clock is inverted before applied to internal circuitry |
| CSI0_DATA_POL | 0x0 | Data lines are directly applied to internal circuitry. |
| CSI0_HSYNC_POL | 0x0 | HSYNC is directly applied to internal circuitry |
| CSI0_VSYNC_POL | 0x0 | VSYNC is directly applied to internal circuitry |

## 20.4 Adding Support for a New CMOS Camera Sensor

To add support for a new CMOS camera sensor to your BSP, first create a device driver for supporting it. This device driver is the optimal location for implementing initialization routines, the power up sequence, power supply settings, the reset signal, and other desired features for your CMOS sensor. It is also the optimal location to implement the CSI configuration for the parallel protocol used between the camera and the i.MX53.

Perform the following three steps on the i.MX53 BSP to create the device driver:

1. Add a camera sensor entry on the ltib catalog.
2. Create the camera file.
3. Add compilation flag for the new camera sensor.

These steps are discussed in detail in the following subsections.

### 20.4.1 Adding a Camera Sensor Entry on the ltib Catalog (Kconfig)

Select specific camera drivers in the following location (as shown in Figure 20-2):

```
Device Drivers  > Multimedia support  > Video capture adapters  >  MXC Camera/V4L2 PRP Features
support
```

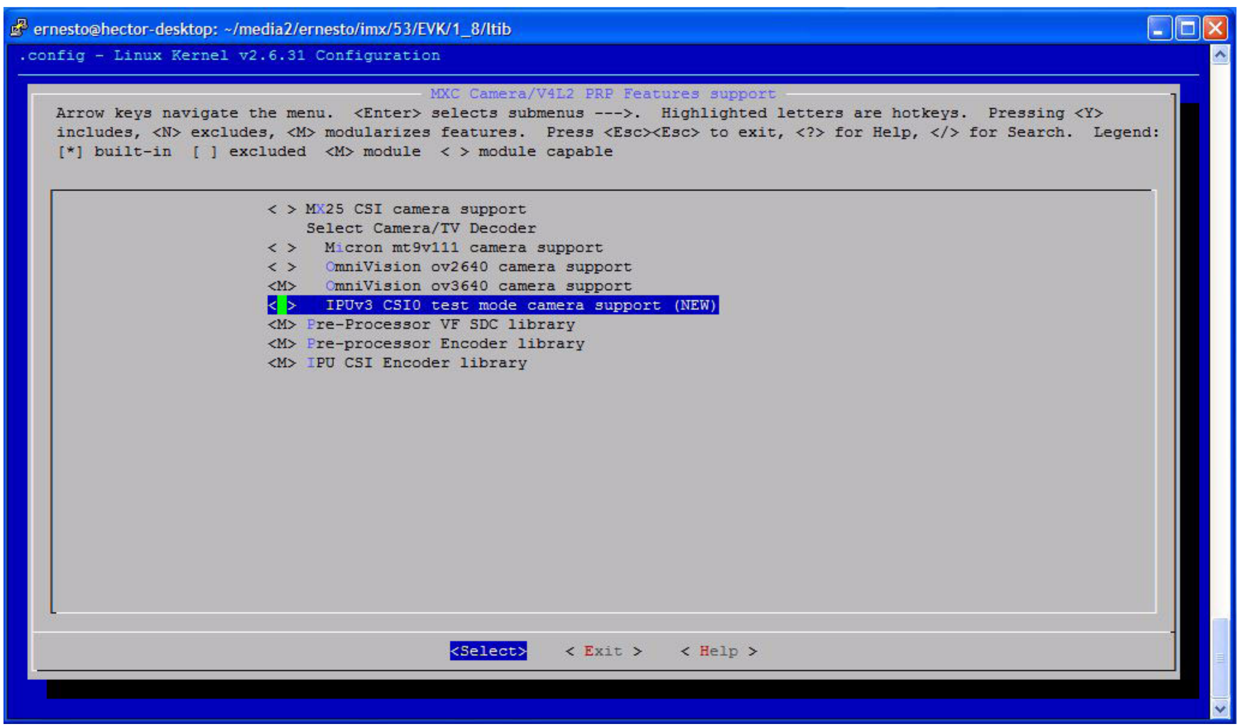



**Figure 20-2. MXC Camera/V4L2 PRP Features Support Window**

To add a new camera sensor entry on the Kconfig camera file, perform the following steps:

1. Enter the following into the i.MX53 display specific folder:

   ```
   $ cd <ltib dir>/rpm/BUILD/linux/drivers/media/video/mxc/capture
   ```

2. Open Kconfig file:

```
$ gedit Kconfig &
```

3. Add the entry where you want it to appear:

```
config MXC_IPUV3_CSI0_TEST_MODE
        tristate "IPUv3 CSI0 test mode camera support"
        depends on !VIDEO_MXC_EMMA_CAMERA
        ---help---
        If you plan to use the IPUv3 CSI0 in test mode with your MXC system, say Y here.
```

## 20.4.2 Creating the Camera Sensor File

The camera sensor file enables camera initialization, reset signal generation, power settings, CSI configuration and all sensor-specific code. Because power settings are handled by the PMIC among other voltage regulators, camera drivers must configure the PMIC during its initialization in order to set up the power voltage configuration unless this has already been done. The reset waveform and the initialization routine must also be included somewhere, usually on the probe function.

**WARNING**

Before connecting a camera sensor to the i.MX53 board, you must check whether the sensor is powered with the proper supply voltages and also whether the sensor data interface has the correct VIO value. Power supply mismatches can damage either the CMOS or the i.MX53.

Create a file with the required panel-specific functions in the following path:

```
<ltib dir>/rpm/BUILD/linux/drivers/media/video/mxc/capture/
```

The camera file—`ipuv3_csi0_chess.c`—must include the functions described in Table 20-2 and may include additional functions and macros required for your driver.

**Table 20-2. Required Functions**

| Function name | Function declaration | Description |
|---|---|---|
| ioctl_g_ifparm | static int ioctl_g_ifparm(struct v4l2_int_device *s, struct v4l2_ifparm *p) | V4L2 sensor interface handler for VIDIOC_G_PARM ioctl |
| ioctl_s_power | static int ioctl_s_power(struct v4l2_int_device *s, int on) | V4L2 sensor interface handler for VIDIOC_S_POWER ioctl. Sets sensor module power mode (on or off) |
| ioctl_g_parm | static int ioctl_g_parm(struct v4l2_int_device *s, struct v4l2_streamparm *a) | V4L2 sensor interface handler for VIDIOC_G_PARM ioctl. Get streaming parameters. |
| ioctl_s_parm | static int ioctl_s_parm(struct v4l2_int_device *s, struct v4l2_streamparm *a) | V4L2 sensor interface handler for VIDIOC_S_PARM ioctl. Set streaming parameters. |
| ioctl_g_fmt_cap | static int ioctl_g_fmt_cap(struct v4l2_int_device *s, struct v4l2_format *f) | Returns the sensor's current pixel format in the v4l2_format parameter. |
| ioctl_g_ctrl | static int ioctl_g_ctrl(struct v4l2_int_device *s, struct v4l2_control *vc) | V4L2 sensor interface handler for VIDIOC_G_CTRL. If the requested control is supported, returns the control's current value from the video_control[] array. Otherwise, returns -EINVAL if the control is not supported. |

Table 20-2. Required Functions (continued)

| Function name | Function declaration | Description |
|---|---|---|
| ioctl_s_ctrl | static int ioctl_s_ctrl(struct v4l2_int_device *s, struct v4l2_control *vc) | V4L2 sensor interface handler for VIDIOC_S_CTRL. If the requested control is supported, sets the control's current value in HW (and updates the video_control[] array). Otherwise, returns -EINVAL if the control is not supported. |
| ioctl_init | static int ioctl_init(struct v4l2_int_device *s) | V4L2 sensor interface handler for VIDIOC_INT_INIT. Initialize sensor interface. |
| ioctl_dev_init | static int ioctl_dev_init(struct v4l2_int_device *s) | Initialize the device when slave attaches to the master. |
| ioctl_dev_exit | static int ioctl_dev_exit(struct v4l2_int_device *s) | Deinitialize the device when slave detaches to the master. |

After the functions have been created, you need to add additional information to `ipuv3_csi0_chess_slave` and `ipuv3_csi0_chess_int_device`. The device uses this information to register as a V4L2 device

The following ioctl function references are included:

```
static struct v4l2_int_slave ipuv3_csi0_chess_slave = {
      .ioctls = ipuv3_csi0_chess_ioctl_desc,
      .num_ioctls = ARRAY_SIZE(ipuv3_csi0_chess_ioctl_desc),
};


static struct v4l2_int_device ipuv3_csi0_chess_int_device = {
   ...
      .type = v4l2_int_type_slave,
   ...
};

static int ipuv3_csi0_chess_probe(struct i2c_client *client,const struct i2c_device_id *id)
{
   ...
      retval = v4l2_int_device_register(&ipuv3_csi0_chess_int_device);
   ...
}
```

It is also necessary to modify other files to prepare the BSP for CSI test mode. Change the sensor pixel format from YUV to RGB565 in the ipu_prp_vf_sdc_bg.c file so that the image converter will not perform color space conversion and the input received from the CSI test mode generator will be sent directly to the memory. Also, modify `mxc_v4l2_capture.c` to preserve CSI test mode settings, which are set by the ipuv3_csi0_chess_init_mode() function in the ipuv3_csi0_chess.c file.

### 20.4.3 Adding a Compilation Flag for the New Camera

After camera files have been created and the Kconfig file has the entry for your new camera, modify the Makefile to create the new camera module during compilation. The Makefile is located in the same folder as your new camera file and Kconfig: `<ltib dir>/rpm/BUILD/linux/drivers/media/video/mxc/capture`.

1. Enter the following into the i.MX53 camera support folder

   ```
   $ cd <ltib dir>/rpm/BUILD/linux/drivers/media/video/mxc/capture
   ```

2. Open the i.MX53 camera support Makefile

```
$ gedit Makefile &
```

3. Add the cmos driver compilation entry to the end of the Makefile.

```
ipuv3_csi0_chess_camera-objs := ipuv3_csi0_chess.o sensor_clock.o
obj-$(CONFIG_MXC_IPUV3_CSI0_TEST_MODE) += ipuv3_csi0_chess_camera.o
```

The kernel object is created using the ipuv3_csi0_chess.c file. You should have the following files as output:

- ipuv3_csi0_chess_camera.mod.c
- ipuv3_csi0_chess.o
- ipuv3_csi0_chess_camera.o
- ipuv3_csi0_chess_camera.mod.o
- ipuv3_csi0_chess_camera.ko

## 20.5 Using the I$^2$C Interface

Many camera sensor modules require a synchronous serial interface for initialization and configuration. This section uses the `<ltib dir>/rpm/BUILD/linux/drivers/media/video/mxc/capture/ov3640.c` file for its example code. This file contains a driver that uses the I$^2$C interface for sensor configuration.

After the I$^2$C interface is running, create a new I$^2$C device to handle your camera bus. If the camera sensor file (called mycamera.c in the following example code) is located in the same folder as ov3640.c, the code is as follows:

```
struct i2c_client * mycamera_i2c_client;

static s32 mycamera_read_reg(u16 reg, u8 *val);
static s32 mycamera_write_reg(u16 reg, u8 val);

static const struct i2c_device_id mycamera_id[] = {
   {"mycamera", 0},
   {},
};

MODULE_DEVICE_TABLE(i2c, mycamera_id);

static struct i2c_driver mycamera_i2c_driver = {
      .driver = {
              .owner = THIS_MODULE,
        .name  = "mycamera",
              },
      .probe  = mycamera_probe,
      .remove = mycamera_remove,
      .id_table = mycamera_id,
};


static s32 ipuv3_csi0_chess_write_reg(u16 reg, u8 val)
{
   u8 au8Buf[3] = {0};
   au8Buf[0] = reg >> 8;
   au8Buf[1] = reg & 0xff;
```

```
   au8Buf[2] = val;
   if (i2c_master_send(my_camera_i2c_client, au8Buf, 3) < 0) {
      pr_err("%s:write reg error:reg=%x,val=%x\n",__func__, reg, val);
      return -1;
   }
   return 0;
}

static s32 my_camera_read_reg(u16 reg, u8 *val)
{
   u8 au8RegBuf[2] = {0};
   u8 u8RdVal = 0;
   au8RegBuf[0] = reg >> 8;
   au8RegBuf[1] = reg & 0xff;

   if (2 != i2c_master_send(my_camera_i2c_client, au8RegBuf, 2)) {
      pr_err("%s:write reg error:reg=%x\n",__func__, reg);
      return -1;
   }

   if (1 != i2c_master_recv(my_camera_i2c_client, &u8RdVal, 1)) {//  @ECA
      pr_err("%s:read reg error:reg=%x,val=%x\n",__func__, reg, u8RdVal);
   return -1;
   }

   *val = u8RdVal;
   return u8RdVal;
}

static int my_camera_probe(struct i2c_client *client, const struct i2c_device_id *id)
{
   ...
   my_camera_i2c_client = client;
   ...
}


static __init int mycamera_init(void)
{
   u8 err;
   err = i2c_add_driver(&mycamera_i2c_driver);
   if (err != 0)
      pr_err("%s:driver registration failed, error=%d \n",__func__, err);
   return err;
}


static void __exit mycamera_clean(void)
{
   i2c_del_driver(&mycamera_i2c_driver);
}

module_init(mycamera_init);
module_exit(mycamera_clean);
```

Check ov3640.c for the complete example code.

After creating the new I$^2$C device, add the following lines to your platform file (located at `<ltib dir>/rpm/BUILD/linux/arch/arm/mach-mx5/mx53_<board name>.c`).

```
static struct mxc_camera_platform_data camera_data = {
   .analog_regulator = "VSD",
   .mclk = 24000000,
   .csi = 0,
   .pwdn = camera_pwdn,
};


static struct i2c_board_info mxc_i2c0_board_info[] __initdata = {
   {
      .type = "ov3640",
      .addr = 0x3C,
      .platform_data = (void *)&camera_data,
   },
   {
     .type = "adv7180",      .addr = 0x21,      .platform_data = (void *)&adv7180_data,   },
{       .type = "cs42888",
      .addr = 0x48,
   },
   {
      .type = "mycamera",
      .addr = 0x2E,
      .platform_data = (void *)&camera_data,
   },
};

static void __init mxc_board_init(void)
{
   ...
  i2c_register_board_info(0, mxc_i2c0_board_info,ARRAY_SIZE(mxc_i2c0_board_info));
   ...
}
```

You may modify the platform file at this point to specify features about your camera such as the CSI interface used (CSI0 or CSI1), the MCLK frequency, and some power supply settings related to the module. Notice that ".addr" is used to specify the I$^2$C address of the camera sensor module.

**NOTE**

It is mandatory that ".type" be equal to the i2c_device_id on your camera sensor file (mycamera.c).

You can now read and write from/to the sensor in the camera sensor file by using the following:

```
retval = mycamera_write_reg(RegAddr, Val);
retval = mycamera_read_reg(RegAddr, &RegVal);
```

## 20.6 Loading and Testing the Camera Module

If your camera driver has been created as a kernel module, as in the example in this chapter, the module must be loaded prior any camera request attempt. According to the Makefile information, the camera module is named `ipuv3_csi0_chess_camera.ko`.

To load the V4L2 camera interface and CSI in test mode, execute the following commands:

```
root@freescale /unit_tests$ modprobe ipuv3_csi0_chess_camera
root@freescale /unit_tests$ modprobe mxc_v4l2_capture
```

To test the video0 input (camera), an mxc_v4l2_overlay test is included in the BSP. If the imx-test package has also been included, open the unit test folder and execute the test.

```
root@freescale ~$ cd /unit_tests/
root@freescale /unit_tests$ ./mxc_v4l2_overlay.out
```

If the imx-test package has not been built, select it from the LTIB package menu:

Package List > imx-test

The chessboard appears in a rectangle located on the left top side of the WVGA panel, as shown in Figure 20-3. The colors of the chessboard toggle between red, green, and blue every time you run the test.

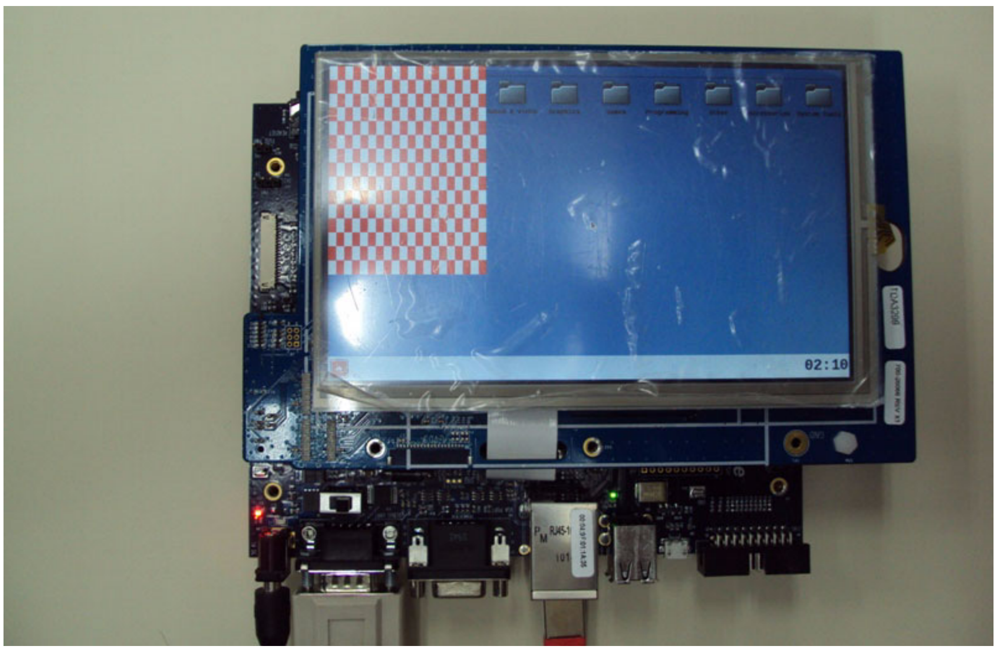



**Figure 20-3. Chessboard Test**

## 20.7 Additional Reference Information

This section provides reference information about the following:

- CMOS interfaces supported by the i.MX53
- i.MX53 CSI parallel interface
- Timing data mode protocols

### 20.7.1 CMOS Interfaces Supported by the i.MX53

The camera sensor interface, which is part of the image processing unit (IPU) module on the i.MX53, handles CMOS sensor interfaces. The i.MX53 is able to handle two camera devices through its CSI ports, one connected to the CSI0 port and the other to the CSI1 port. Both CSI ports are identical and provide glueless connectivity to a wide variety of raw/smart sensors and TV decoders.

Each of the camera ports includes the following features:

- Parallel interface
  - Up to 20-bit input data bus.
  - A single value in each cycle.
  - Programmable polarity.
- Multiple data formats
  - Interleaved color components, up to 16 bits per value (component)
  - Input Bayer RGB, Full RGB, or YUV 4:4:4, YUV 4:2:2 Component order:UY1VY2 or Y1UY2V, grayscale and generic data.
- Scan order: progressive or interlaced
- Frame size: up to 8192 × 4096 pixels
- Synchronization—video mode
  - The sensor is the master of the pixel clock (PIXCLK) and synchronization signals
  - Synchronization signals are received using either of the following methods:
    - Dedicated control signals—VSYNC, HSYNC—with programmable pulse width and polarity
    - Controls embedded in the data stream, following loosely the BT.656 protocol, with flexibility in code values and location.
  - The image capture is triggered by the MCU or by an external signal (e.g. a mechanical shutter)
  - Synchronized strobes are generated for up to 6 outputs—the sensor and camera peripherals (flash, mechanical shutter...)
- Frame rate reduction by periodic skipping of frames
- Window-of-interest selection
- Pre-flash for red-eye reduction and for measurements (e.g. focus) in low-light conditions.

For details, refer to the “Image Processing Unit (IPU)” chapter in the *i.MX53 Applications Processor Reference Manual*. Figure 20-4 shows the block diagram.

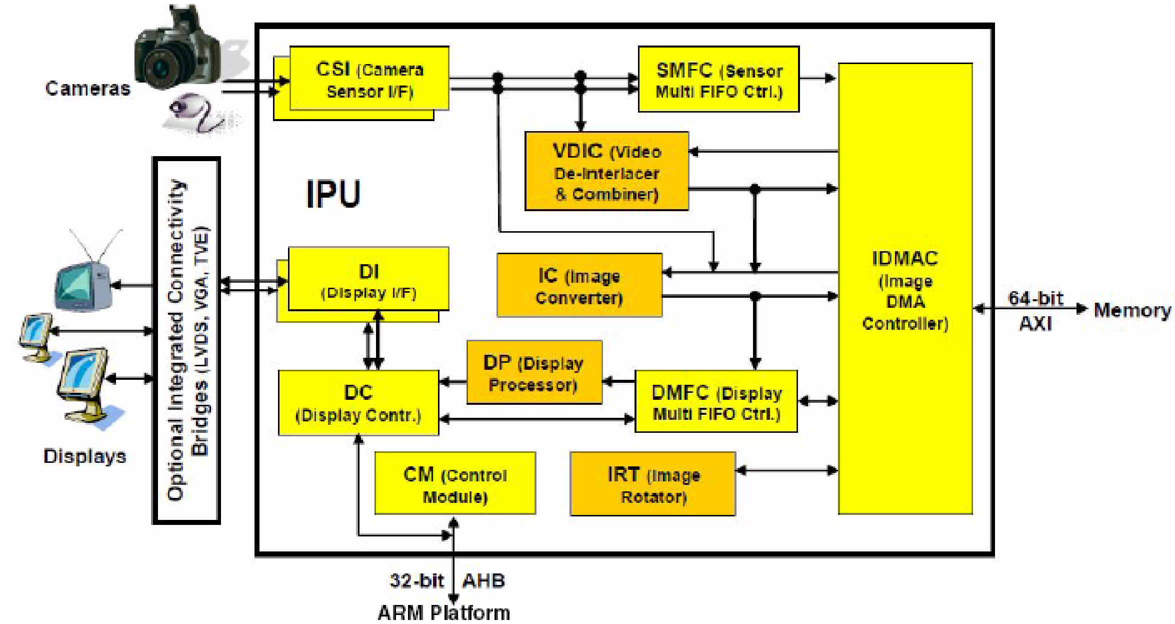



**Figure 20-4. IPU Block Diagram**

Several sensors can be connected to each of the CSIs. Simultaneous functionality (for sending data) is supported as follows:

- Two sensors can send data independently, each through a different port.
- One stream can be transferred to the VDI or IC for on-the-fly processing while the other one is sent directly to system memory.

The input rate supported by the camera port is as follows:

- Peak: up to 180 MHz (values/sec)
- Average (assuming 35% blanking overhead), for YUV 4:2:2
  — Pixel in one cycle (BT.1120): up to 135 MP/sec, e.g. 9 Mpixels at 15 fps
  — Pixel on two cycles (BT.656): up to 67 MP/sec, e.g. 4.5 Mpixels at 15 fps.
- On-the-fly processing may be restricted to a lower input rate.

If required, additional cameras can be connected though the USB port.

## 20.7.2 i.MX53 CSI Parallel Interface

The CSI obtains data from the sensor, synchronizes the data and the control signals to the IPU clock (HSP_CLK), and transfers the data to the IC and/or SMFC.

The CSI parallel interface (shown in Figure 20-5) provides a clock output (MCLK), which is used by the sensor as a clock input reference. The i.MX53 requests either video or still images through a different interface between the processor and the camera module. In most cases, the interface is a synchronous serial interface such as the I$^2$C. After the frame has been requested, the camera module takes control of the CSI bus, and uses synchronization signals VSYNC, HSYNC, DATA_EN and PIXCLK to send the image frame to the i.MX53. The camera sensor creates PIXCLK based on MCLK input.

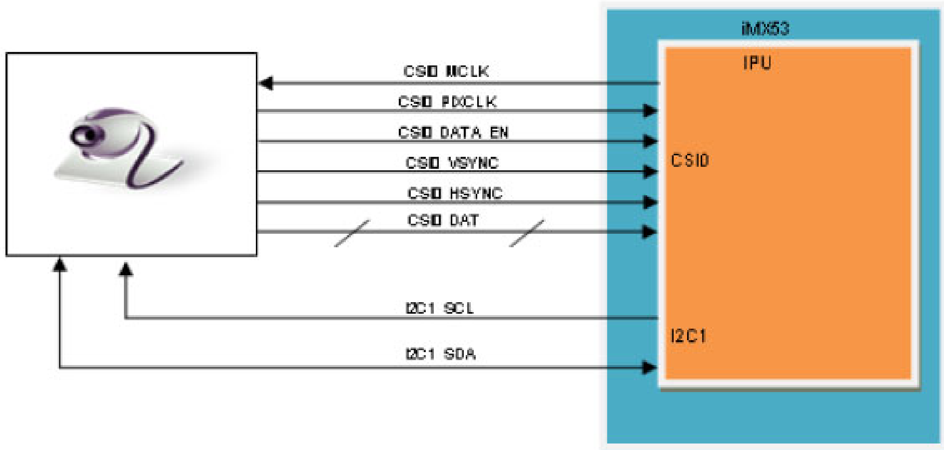



**Figure 20-5. Parallel Interface Layout**

In parallel interface, a single value arrives in each clock—except in BT.1120 mode when two values arrive per cycle. Each value can be 8–16 bits wide according to the configuration of DATA_WIDTH. If DATA_WIDTH is configured to N, then 20-N LSB bits are ignored.

Therefore, you never need CSI0_DAT[3:0], unless you are using BT.1120 mode, because the maximum pixel width is 16 (CSI0_DAT[19:4]). The expansion port 2 includes CSI0_DAT[19:4], but only CSI0_DAT[19:10] are used for the CSI data bus (10-bit wide data). CSI0_DAT[9:4] are shared with other interfaces and are used for audio and I$^2$C.

CSI can supports several data formats according to SENS_DATA_FORMAT configuration. When the data format is YUV, the output of the CSI is always YUV444—even if the data arrives in YUV422 format.

The polarity of the inputs can be configured using the following registers:

- SENS_PIX_CLK_POL
- DATA_POL
- HSYNC_POL
- VSYNC_POL

The camera parallel interface provided by the i.MX53 is a 15 line interface, as described in Table 20-3:

**Table 20-3. CSI0 Parallel Interface Signals**

| Signal | IPU Pin | Description |
|---|---|---|
| MCLK | CSI0_MCLK | Master Clock (Output) |
| PIXCLK | CSI0_PIXCLK | Pixel Clock |
| VSYNC | CSI0_VSYNC | Vertical Synchronization signal |
| HSYNC | CSI0_HSYNC | Horizontal Synchronization signal |

**Table 20-3. CSI0 Parallel Interface Signals (continued)**

| Signal | IPU Pin | Description |
|---|---|---|
| DATA_EN | CSI0_DATA_EN | Data Enable or Data ready |
| DATA[19:10] | CSI0_DAT [19:10] | Pixel data bus, optional to [19:4] |

Section 20.7.3, “Timing Data Mode Protocols,” explains how the timing data mode protocols use these signals. Not all signals are used in each timing data mode protocol.

### 20.7.3 Timing Data Mode Protocols

The CSI interface supports the following four timing/data protocols:

- Gated mode
- Non-gated mode
- BT.656 (Progressive and interlaced)
- BT.1120 (Progressive and interlaced)

In gated mode, VSYNC is used to indicate beginning of a frame, and HSYNC is used to indicate the beginning of a raw. The sensor clock is always ticking.

In non-gated mode, VSYNC is used to indicate beginning of a frame, and HSYNC is not used. The sensor clock only ticks when data is valid.

In BT.656 mode, the CSI works according to recommendation ITU-R BT.656. The timing reference signals (frame start, frame end, line start, line end) are embedded in the data bus input.

In BT1120 mode, the CSI works according to recommendation ITU-R BT.1120. The timing reference signals (frame start, frame end, line start, line end) are embedded in the data bus input.

For details, refer to the *i.MX53 Applications Processor Reference Manual*.

# Chapter 21
# Porting Audio Codecs to a Custom Board

This chapter explains how to port audio drivers from the Freescale reference BSP to a custom board. This procedure varies depending on whether the audio codec on the custom board is the same as or different than the audio codec on the Freescale reference design. This chapter first explains the common porting task and then the different porting tasks.

## 21.1 Common Porting Task

The mxc_audio_platform_data structure must be defined and filled appropriately for the custom board before doing any other porting tasks. An example of a filled structure can be found in the file located at `linux/arch/arm/mach-mx5/mx53_<board name>.c`

```
static struct mxc_audio_platform_data sgtl5000_data = {
        .ssi_num = 1,
        .src_port = 2,
        .ext_port = 5,
        .hp_irq = IOMUX_TO_IRQ(MX53_PIN_ATA_DATA5),
        .hp_status = headphone_det_status,
        .amp_enable = mxc_sgtl5000_amp_enable,
        .init = mxc_sgtl5000_init,
};
```

Customize the structure according to the following definitions:

| | |
|---|---|
| ssi_num | The ssi used for this codec |
| src_port | The digital audio mux (DAM) port used for the internal SSI interface (for details about the internal functionality of the DAM please refer to the AUDMUX chapter of the *i.MX53 Applications Processor Reference Manual*) |
| ext_port | The digital audio mux (DAM) port used for the external device audio interface (for details about the internal functionality of the DAM please refer to the AUDMUX chapter of the *i.MX53 Applications Processor Reference Manual*) |
| hp_irq | The IRQ line used for headphone detection |
| hp_status | A pointer to a function that returns the current headphone detect status. If a different mechanism or GPIO is used for headphone detect in the custom board, this function must be modified to accurately reflect the headphone presence. |
| amp_enable | A pointer to a function that enables/disables the audio codec. For example, this function can be used to turn on or turn off the regulator supplying the audio codec. |
| init | The initialization routine for the audio codec. Any setup necessary for the audio codec should be implemented in this function. |

## 21.2 Porting the Reference BSP to a Custom Board (audio codec is the same as in the reference design)

When the audio codec is the same in the reference design and the custom board, users must ensure that the I/O signals and the power supplies to the codec are properly initialized in order to port the reference BSP to the custom board.

The iomux-mx53.h file contains the definitions for all i.MX53 pads. Add entries in this file to define the configuration for the audio codec signals. See Chapter 13, “Configuring the IOMUX Controller (IOMUXC),” for a description of how to set up the IOMUX and pads for routing signals as desired.

The necessary signals for the sgtl5000 codec, which is used on the i.MX53 reference board, are as follows:

- $I^2C$ interface signals
- $I^2S$ interface signals
- SSI external clock input to i.MX53

Table 21-1 shows the required power supplies for the sgtl5000 codec.

**Table 21-1. Required Power Supplies**

| Power Supply Name | Definition | Value |
|---|---|---|
| VDDD | Digital voltage | 1.98 V |
| VDDIO | Digital IO voltage | 3.6 V |
| VDDA | Analog voltage | 3.6 V |

## 21.3 Porting the Reference BSP to a Custom Board (audio codec is different than the reference design)

When adding support for an audio codec that is different than the one on the Freescale reference design, users must create new ALSA drivers in order to port the reference BSP to a custom board. The ALSA drivers plug into the ALSA sound framework, which allows the standard ALSA interface to be used to control the codec. Details about the ALSA infrastructure and developing ALSA drivers can be found at http://www.alsa-project.org/main/index.php/ASoC.

The source code for the ALSA driver is located in the Linux kernel source tree at `linux/sound/soc`. Table 21-2 shows the files used for the sgtl codec support:

**Table 21-2. Files for sgtl Codec Support**

| File Name | Definition |
|---|---|
| imx-pcm.c | • Shared by the stereo ALSA SoC driver, the 5.1 ALSA SoC driver, and the Bluetooth codec driver.<br>• Responsible for preallocating DMA buffers and managing DMA channels. |
| imx-ssi.c | • Registers the CPU DAI driver for the stereo ALSA SoC<br>• Configures the on-chip SSI interfaces |

**Table 21-2. Files for sgtl Codec Support**

| File Name | Definition |
|---|---|
| sgtl5000.c | • Registers the stereo codec and Hi-Fi DAI drivers.<br>• Responsible for all direct hardware operations on the stereo codec. |
| imx-3stack-sgtl5000.c | • Machine layer code<br>• Creates the driver device<br>• Registers the stereo sound card. |

## NOTE

If using a different codec, adapt the driver architecture shown in Table 21-2 accordingly. The exact adaptation will depend on the codec chosen. Obtain the codec-specific software from the codec vendor.

# Chapter 22
# Porting the Fast Ethernet Controller Driver

This chapter explains how to port the fast Ethernet controller (FEC) driver to the i.MX53 processor. Using Freescale's standard (FEC) driver makes porting to the i.MX53 simple. Porting needs to address the following three areas:

- Pin configuration
- Source code
- Ethernet connection configuration

## 22.1 Pin Configuration

The FEC supports three different standard physical media interfaces: a reduced media independent interface (RMII), a media independent interface (MII), and a 7-wire serial interface.

The Freescale hardware reference platform directly supports RMII, which has a reduced pin-count compared to MII. Therefore, RMII is the recommended interface.

Table 22-1 shows the signals used by the RMII interface.

**Table 22-1. RMII Signals**

| Signal Name | Definition |
|---|---|
| FEC_TX_CLK | (In, Synchronous clock reference) |
| FEC_TX_EN | (Out, Transmit Enable) |
| FEC_TXD[0:1] | (Out, Transmit Data) |
| FEC_RX_DV | (In, Carrier Sense/Receive Data Valid) |
| FEC_RXD[0:1] | (In, Receive Data) |
| FEC_RX_ER | (In, Receive Error) |
| FEC_MDC | (Out, Management Data Clock) |
| FEC_MDIO | (In/Out, Management Data Input/Output) |
| FEC_PHY_RESET_B | (In, PHY reset) |

Because the i.MX53 has more functionality than it has physical I/O pins, it uses I/O pin multiplexing. The general-purpose I/O pins (gpio1 GPIO[22–31]) default to ALT1.

The FEC_PHY_RESET_B signal comes up by default as gpio2 (pin #0), which is ALT function 1. This particular signal/pin is used as a simple GPIO to reset the FEC PHY. To use the pins as FEC signals mentioned above, configure them as the ALT0 function in the I/O multiplexer, except for FEC_PHY_RESET_B.

## 22.2 Source Code

The source code for the Freescale FEC Linux environment is located under the `../ltib/rpm/BUILD/linux/drivers/net directory`. It contains the following files:

**Table 22-2. Source Code Files**

| | File Names |
|---|---|
| FEC low-level Ethernet driver: | • fec.h<br>• fec.c |
| MAC Switch software | • fec_switch.h<br>• fec_switch.c |
| IEEE 1588 PTP (network time sync) | • fec_1588.h<br>• fec_1588.c |
| MPC52xx PowerPC Ethernet Driver | • fec_mpc52xx.h<br>• fec_mpc52xx.c<br>• fec_mpc52xx_phy.c |

Of those files, only the FEC low-level Ethernet driver code (fec.[ch]) constitutes the Linux i.MX53 FEC driver.

The driver uses the following compile definitions:

CONFIG_FEC_1588 Set for IEEE 1588 network time synchronization.

CONFIG_M5272 PowerPC information. Can be safely ignored and should not be set.

CONFIG_MXC IMXxx parts. Should be defined.

CONFIG_MXS Legacy MXS part. Should generally not be defined.

## 22.3 Ethernet Configuration

This section covers aspects such as duplex and speed configurations.

The two most common issues are as follows:

- MAC address is missing or invalid
- Ethernet connection (duplex, speed)

By default, the Ethernet driver reads the burned-in MAC address, which is found in code from the fec.c file located in the function **fec_get_mac()**. If no MAC address exists in the hardware, the MAC is read as all zeros, which creates problems. If this occurs, modify the code to read the MAC address from Flash or elsewhere.

The FEC driver and hardware are designed to comply to the IEEE standards for Ethernet auto-negotiation. See the FEC chapter in the *i.MX53 Applications Processor Reference Manual* for a description of using flow control in full duplex and more.

# Chapter 23
# Porting USB Host1 and USB OTG

The USB Host1 and the USB OTG signals do not multiplex with other pins on the i.MX53. Therefore, it is not necessary to port IOMUX settings for these interfaces when moving to a new platform.

The only required setup is as follows:

- For the USB Host1 PHY
  — Supply USB_H1_VDDA33 with 3.3 V
  — Supply USB_H1_VDDA25 with 2.5 V
- For the USB OTG PHY
  — Supply USB_OTG_VDDA33 with 3.3 V
  — Supply USB_OTG_VDDA25 with 2.5 V

The USB Host1 PHY uses the following signals:

- USB_H1_GPANAIO
- USB_H1_RREFEXT
- USB_H1_DP
- USB_H1_VDDA33
- USB_H1_DN
- USB_H1_VDDA25
- USB_H1_VBUS

The USB OTG PHY uses the following signals:

- USB_OTG_VBUS
- USB_OTG_ID
- USB_OTG_VDDA25
- USB_OTG_DN
- USB_OTG_VDDA33
- USB_OTG_DP
- USB_OTG_RREFEXT
- USB_OTG_GPANAIO

# Appendix A
# Revision History

Table A-1 provides a revision history for this user guide.

**Table A-1. i.MX53 System Development User Guide Document Revision History**

| Rev. Number | Date | Substantive Change(s) |
|---|---|---|
| Rev 1 | 3/2011 | • In Table 1-1, "Design Checklist," changed "CCR CAMPx_EN" to "CCM_CCR[CAMPx_EN]," under the recommendation 11 and EMI to EIM under the recommendations 3, 4 and 5.<br>• In Section 4.2, "IOMUX Tool Walkthrough," the following introductory text was added to the first paragraph: "The i.MX35 example shown is for illustration only. The steps in the process apply to all i.MX products in the preceding bullet list."<br>• In Table 5-1, "i.MX53 Voltage Rails and Associated DA9053 Regulator," the following updates were done:<br>— VDDGP voltages changed from 1.05 to 1.1 and 1.2 to 1.25<br>— NVCC_LCD changed from 1.87 to 1.8 and 2.77 to 2.775<br>— NVCC_NANDF changed from 1.87 to 1.8<br>— TVDAC_DHVDD changed from 2.75 to 2.775, 1.87 to 1.8, and 2.77 to 2.775<br>— NVCC_RESET changed from 1.87 to 1.8 and 2.77 to 2.775<br>— VDD_REG changed from 1.6 to 2.475<br>• In Table 5-2, "i.MX53 Voltage Rails and Associated LTC3589-1 Regulator," the following updates were done:<br>— VDDGP voltages changed from 1.05 to 1.1 and 1.2 to 1.25<br>— NVCC_LCD changed from 1.87 to 1.8 and 2.77 to 2.775<br>— NVCC_NANDF changed from 1.87 to 1.8<br>— TVDAC_DHVDD changed from 1.87 to 1.8 and 2.77 to 2.775<br>— NVCC_RESET changed from 1.87 to 1.8 and 2.77 to 2.775<br>• In Table 5-2, "i.MX53 Voltage Rails and Associated LTC3589-1 Regulator," the Current Capability for VCC, NVCC_LVDS, USB_H1_VDDA25, VDD_REG, and VPH changed from 1000 to 1200.<br>• In Table 5-2, "i.MX53 Voltage Rails and Associated LTC3589-1 Regulator," changed Fusebox to FUSEBOX, S-ATA to SATA, and in the table's introduction, changed i.MX to i.MX53.<br>• In Table 8-1, "Clock Roots," changed all the block mnemonics to upper case in the Description column.<br>• In Table 8-1, "Clock Roots," changed LVDS to LDB.<br>• In Section 10.2.6, "SD/MMC Test," changed SDHC3 to ESDHCV3-3 and SDHC2 to ESDHCV2-2.<br>• In Section 11.3.1, "Changing the DCD Table for i.MX53 DDR3 Initialization," changed ESDCTLv2.5 to ESDCTL.<br>• In Chapter 13, "Configuring the IOMUX Controller (IOMUXC), changed "IOMUX Controller" to IOMUXC.<br>• In Chapter 15, "Adding Support for the i.MX53 ESDHC," changed "eSDHC1/2/3/4" to "ESDHCV2-1/2/4 and ESDHCV3-3."<br>• Changed eCSPI to ECSPI and eSDHC to ESDHC, throughout the document.<br>• In Section 15.3.3, "Interface Layouts," changed i.MX35 to i.MX53 in the title and description of Figure 15-2.<br>• In Figure 18-2, replaced MCIMX53 with i.MX53.<br>• In Section 20.7.1, "CMOS Interfaces Supported by the i.MX53," replaced the term IPUv3M with IPU.<br>• Replaced the term IPUv3M with IPU in Figure 20-1, Figure 20-5, Figure 18-1, Figure 18-4, and Section 18.5, "i.MX53 Display Interface Helpful Information."<br>• In Table 20-3, "CSI0 Parallel Interface Signals," changed the column heading "IPUv3 Pin" to "IPU Pin."<br>• In Section 21.1, "Common Porting Task," replaced AUDIOMUX with AUDMUX. |
| Rev 0 | 02/2011 | Initial release. |