Freescale Semiconductor

Document Number: KSDKKL43ZRN
Rev. 1.0.2, 2/2015

Kinetis SDK Release Notes for the KL43Z Freescale Platforms

1 Overview

These are the release notes for the standalone release supporting the TWR-KL43Z Freescale Tower System development board platform and the FRDM-KL43Z Freescale Freedom development board platform, based on the Kinetis SDK (SDK) 1.0.0. The core of the Kinetis SDK is a set of peripheral drivers architected in two layers: the Hardware Abstraction Layer (HAL) and the Peripheral Driver layer.

The HAL abstracts the hardware register access into a set of stateless functional primitives, which provide the building blocks for the high level peripheral drivers or applications. The Peripheral Driver layer utilizes one or more HAL layer components and possibly other Peripheral Drivers.

The Kinetis SDK includes a set of example applications demonstrating the use of the peripheral drivers and other integrated software modules, such as a Real Time Operating System (RTOS) through an RTOS abstraction layer. The Kinetis SDK also integrates middleware such as the Freescale USB stack to provide an easy to use Software Development Kit for Kinetis product families.

Contents

2 Development Tools

This version of the KSDK for the TWR-KL43Z and FDRM-KL43Z Freescale platforms was compiled and tested with these development tools:

- IAR Embedded Workbench version 7.20.2 or later
- Keil MDK 5.11 with Freescale Kinetis KLxx Series Device Support pack 1.1.0
- Makefiles support with GCC 4.8.3
- Kinetis Design Studio IDE (KDS) version 1.1

This table provides a list of default debugger configurations for the platform.

Table 1. List of Default Debugger Configurations

IDE	Default debugger	FRDM platform
IAR Embedded Workbench for ARM version 7.20.2	P&E Micro	FRDM-KL43Z
MDK-ARM Microcontroller Development Kit (Keil) 5.11.1	P&E Micro	FRDM-KL43Z
Kinetis Design Studio v1.1	J-Link	FRDM-KL43Z

3 Supported Development Systems

This release supports boards and devices listed in this table. Boards and devices in boldface were tested in this release.

Table 2. Supported MCU devices and development boards

Development boards	Kinetis MCU devices
TWR-KL43Z	MKL17Z128VFM4, MKL17Z128VFT4, MKL17Z128VMP4, MKL17Z128VLH4, MKL17Z256VFM4, MKL17Z256VFT4, MKL17Z256VMP4, MKL17Z256VLH4, MKL27Z128VFM4, MKL27Z128VFT4, MKL27Z128VMP4, MKL27Z128VLH4, MKL27Z256VFM4, MKL27Z256VFT4, MKL27Z256VMP4, MKL27Z256VLH4, MKL33Z128VMP4, MKL33Z128VLH4, MKL33Z256VMP4, MKL33Z256VLH4, MKL43Z128VMP4, MKL43Z128VLH4, MKL43Z256VMP4, **MKL43Z256VLH4**
FRDM-KL43Z	

4 Release Contents

This table describes the release contents.

Table 3. Release Contents

Deliverable	Location
Specific content for the evaluation boards	<install_dir>/boards/...
Demo applications	<install_dir>/demos/...
Documentation	<install_dir>/doc/...
File System	<install_dir>/filesystem/...
Projects to build libraries	<install_dir>/lib/...
Common Make files	<install_dir>/mk/...
Driver library, startup code and utilities	<install_dir>/platform/...
Cortex Microcontroller Software Interface Standard (CMSIS) ARM Cortex-M header files, DSP library source, and IP extension header files	<install_dir>/platform/CMSIS/…
Peripheral Drivers	<install_dir>/platform/drivers/…
Hardware Abstraction Layer	<install_dir>/platform/hal/…
Linker control files for each supported tool chain	<install_dir>/platform/linker/…
OS Abstraction for Bare Metal and RTOS	<install_dir>/platform/osa/…
CMSIS compliant Startup Code	<install_dir>/platform/startup/…
System Services such as clock manager, interrupt manager, unified hardware timer, and low power manager	<install_dir>/platform/system/…
Utilities such as debug console	<install_dir>/platform/utilities/…
RTOS Kernel Code, RTOS abstraction implementations, and RTOS kernel folders	<install_dir>/rtos/...
A Processor Expert service pack and MQX™ RTOS task-aware plugins for tool chains.	<install_dir>tools
USB stack and demos	<install_dir>/usb/...

5 Kinetis SDK Release Overview

The Kinetis SDK is intended for use with the Freescale Kinetis MCU product family based on the ARM® Cortex-M series architectures. The release consists of:

- Kinetis MCU platform support
- Board configuration support
- Demo applications
- The FatFS FAT File System
- USB Host and Device stacks
- RTOS support components
- Documentation (Kinetis SDK reference manual and various user's guides)

5.1 Kinetis MCU platform support

The Kinetis SDK platform directory contains the startup code, operating system abstraction, system services, driver libraries for peripherals, header files, linker files, and utilities such as the debug console implementation.

5.1.1 Startup code

The Kinetis SDK includes simple CMSIS-compliant startup code for the supported Kinetis MCUs, which efficiently deliver the code execution to the `main()` function. An application can either include the startup code directly in the project space, or include a prebuilt startup code library for a cleaner project space.

5.1.2 Operating system abstraction

The drivers are designed to work with or without an operating system through the Operating System Abstraction layer (OSA). The OSA defines a common set of services that abstract most of the OS kernel functionalities. The OSA either maps an OSA service to the target OS function, or implements the service when no OS is used (bare metal), or when the service does not exist in the target OS. The Kinetis SDK implements the OSA for Freescale MQX RTOS, FreeRTOS, µC/OS-II, µC/OS-III, and for OS-less "bare metal" usage. The bare metal OSA implementation is selected as the default option.

5.1.3 Driver library

The Kinetis SDK provides a set of drivers for the Kinetis MCU product family on-chip peripherals. The drivers are designed and implemented around the peripheral hardware blocks rather than for a specific Kinetis MCU, and work with or without an OS through the OS Abstraction layer. The drivers are architected into two layers: the Hardware Abstraction Layer and the Peripheral Driver Layer.

The HAL is designed to abstract hardware register accesses into functional accesses. It is stateless and is intended to cover the entire hardware functionality.

The Peripheral Drivers are built on top of the HAL to provide a set of easy to use interfaces that handle high level data and stateful transactions. They are designed for the most common use cases identified for the underlying hardware. They are written in C language and can be easily ported from product to product, as they are designed to be initialized at runtime based on the driver configuration passed in by the user. In most cases, the Peripheral Drivers can be used as is. However, if the Peripheral Driver does not address a particular target use case, it can either be modified/enhanced, or completely rewritten to meet the target functionality and other requirements. In this case, the existing Peripheral Driver can be used as a reference to build a custom driver based on the HAL. For more details, see the Architectural Overview chapter in the *Kinetis SDK API Reference Manual* (document KSDKKL43ZAPIRM).

Detailed implementation of hardware peripheral functionality is implemented in stages for both the HAL and Peripheral Driver. For example, the current version of the UART driver does not support modem control and smart card features, and the current version of the I2C driver does not support the SMBUS feature. The features which are missing from the current driver versions may be implemented in future releases.

5.1.4 Header files

The Kinetis SDK CMSIS directory contains CMSIS-compliant device-specific header files, which provide direct access to the Kinetis MCU peripheral registers. Each supported Kinetis MCU device in the Kinetis SDK has an overall System-on-Chip (SoC) memory-mapped header file. In addition to the overall SoC memory-mapped header file, the Kinetis SDK includes extension header files for each peripheral instantiated on the Kinetis MCU. Along with the SoC header files and peripheral extension header files, the Kinetis SDK also includes common CMSIS header files for the ARM Cortex-M core and DSP library from the ARM CMSIS version 4.0 release.

5.1.5 Linker files

The Kinetis SDK contains linker control files (or simply linker files) for each supported tool chain and Kinetis MCU device.

Note

By default, the vector table is rom based, which is read only. If a ram based vector table is needed, it is needed to define the symbol "__ram_vector_table__=1" for linker.

5.1.6 Utilities

The utilities directory contains useful software utilities, such as a debug console.

5.2 Board configuration

The board directory in the Kinetis SDK is mainly used for the board-specific configuration and pin muxing. The board directory also contains software components specific to the boards such as Ethernet PHY, Accelerometer, and SPI Flash implementations.

5.3 Demo applications

The example applications demonstrate the usage of the driver libraries and other integrated software solutions on supported development systems. For details, see the *Kinetis SDK Demo Applications User's Guide* (document KSDKKL43ZDEMOUG).

5.4 Other integrated software solutions

The Kinetis SDK is designed for easy integration with other software solutions such as OS kernels, USB stack, and file systems.

5.4.1 USB stack

A Freescale USB stack is integrated with the Kinetis SDK and was tested both with and without an OS through the OS abstraction layer.

5.4.2 File System

A FAT file system is integrated with Kinetis SDK and can be used to access either the SD card or the USB memory stick when the SD card driver or the USB Mass Storage Device class implementation is used.

5.4.3 RTOS

The Kinetis SDK is pre-integrated with Freescale MQX RTOS, FreeRTOS, µC/OS-II, and µC/OS-III. OS abstraction layers are implemented for these RTOSes.

6 Known Issues

6.1 Maximum file path length in Windows® 7 Operating System

Windows® 7 OS imposes a 260 maximum length for file paths. When installing Kinetis SDK, place it in a directory close to the root to prevent file paths from exceeding the maximum character length specified by the Windows Operating System. The recommended location is the C:\Freescale folder.

6.2 Downloading with P&E Microcomputer Systems in Keil

An issue occurs when downloading with P&E Microcomputer Systems in Keil. If the project size is greater than 16K, use the J-Link to download and debug. See *Kinetis SDK KL43Z Freescale Tower TWR-KL43Z Platform User's Guide* or *Kinetis SDK KL43Z Freescale Freedom FRDM-KL43Z Platform User's Guide* for more information (document KSDKKL43ZUG).

6.3 No spaces in the Kinetis SDK installation

The Freescale MQX RTOS build uses batch files which do not work when there are spaces in the file path.

6.4 USB HUB Power Supply

The USB HUB external power supply must be provided before use.

6.5 Installer Issue

Note that the Linux installer was tested only on a host with Ubuntu 12.0.4.

When uninstalling the Kinetis SDK, the system variable KSDK_PATH will remain set in the Windows Registry until the next PC reboot. If you attempt to install the Kinetis SDK before rebooting the PC, the installer will think that the previous instance is still valid and may not set the KSDK_PATH variable correctly. Reboot your PC after uninstalling the Kinetis SDK to avoid this problem.

6.6 P&E OpenSDA MSD Issue

Certain versions of the P&E Micro OpenSDA software exhibit a problem with dragging and dropping a binary (.bin) file image to the TWR-KL43Z board. The update fails. The P&E Micro has fixed this issue. The updated firmware can be found in this link:
www.pemicro.com/downloads/download_file.cfm?download_id=378.

In this package, there's a firmware for the TWR-KL43Z named "MSD-DEBUG-XTWR-KL43Z48M_Pemicro_v116.SDA". Follow the guide in the package to update the firmware.

6.7 Interrupt vector issue with debuggers

When using a debugger to download and execute the Kinetis SDK code, interrupts may inadvertently be vectored to the boot ROM instead of the Kinetis SDK vector table. This is caused by the fact that at the system boot, the core programs the register RCM_MR bit field BOOTROM to “0x2” which will essentially re-vector interrupts to the ROM. Clearing these RCM_MR[BOOTROM] bits to 0x0 directs the interrupts to the proper Kinetis SDK vector table. These bits are clearable via a write-one-to-clear operation. There are two options that the user may employ to clear these bits:

1. The user adds a simple line of code in one of the startup files that clears the BOOTROM bits. For example, in the file “system_MKL43Z4.c” at the end of the function “SystemInit()” the user can add this line of code: “BW_RCM_MR_BOOTROM(RCM_BASE, 0x3);”. Once added, rebuild the Kinetis SDK project, then download the code and execute.
2. The user manually clears these bits in the debugger after downloading the Kinetis SDK, but before code execution. For example, if using IAR, after downloading the code, click the menu option “View”, then “Register”. In the register view, select “RCM”. In register view of RCM, expand the register RCM_MR, and write 0x2 to BOOTROM. This should clear those bits.

For IAR, it needs to be configured as the following picture shown.

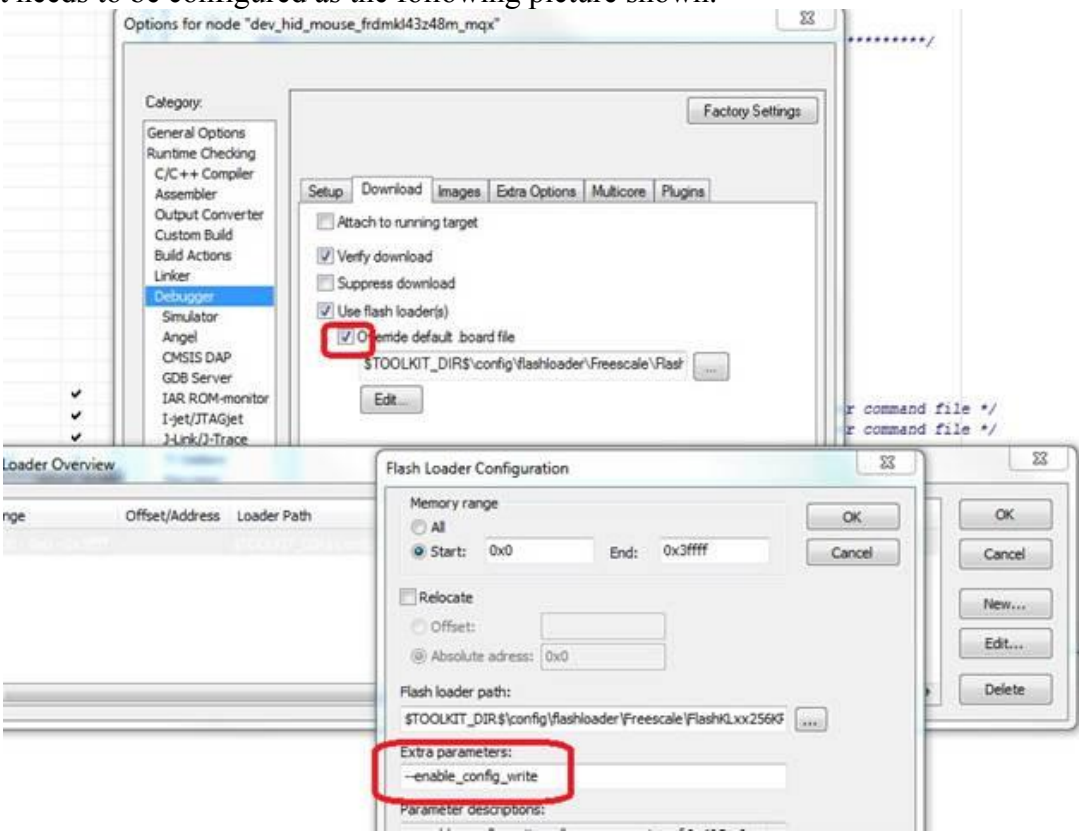

Figure 1: IAR Configurations

In future versions of debugger tools like IAR, Keil, and KDS IDE, the clearing of the BOOTROM bits will be implemented by these tools, rendering the above workaround unnecessary.

7 Revision History

This table summarizes revisions to this document.

Revision History		
Revision number	**Date**	**Substantive changes**
1.0.0	9/2014	Initial release
1.0.1	1/2015	General availability release
1.0.2	2/2015	Added boards in Table 2.

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
www.freescale.com/support

Document Number: KSDKKL43ZRN
Rev. 1.0.2
2/2015