Hands-On Workshop: Developing with the **Software Development Kit** for **Kinetis MCUs**

EUF-IND-T1475

Antonio Concio | Field Application Engineer

Lorenzo Daniele | Field Application Engineer

M A Y . 2 0 1 5

freescale™

External Use

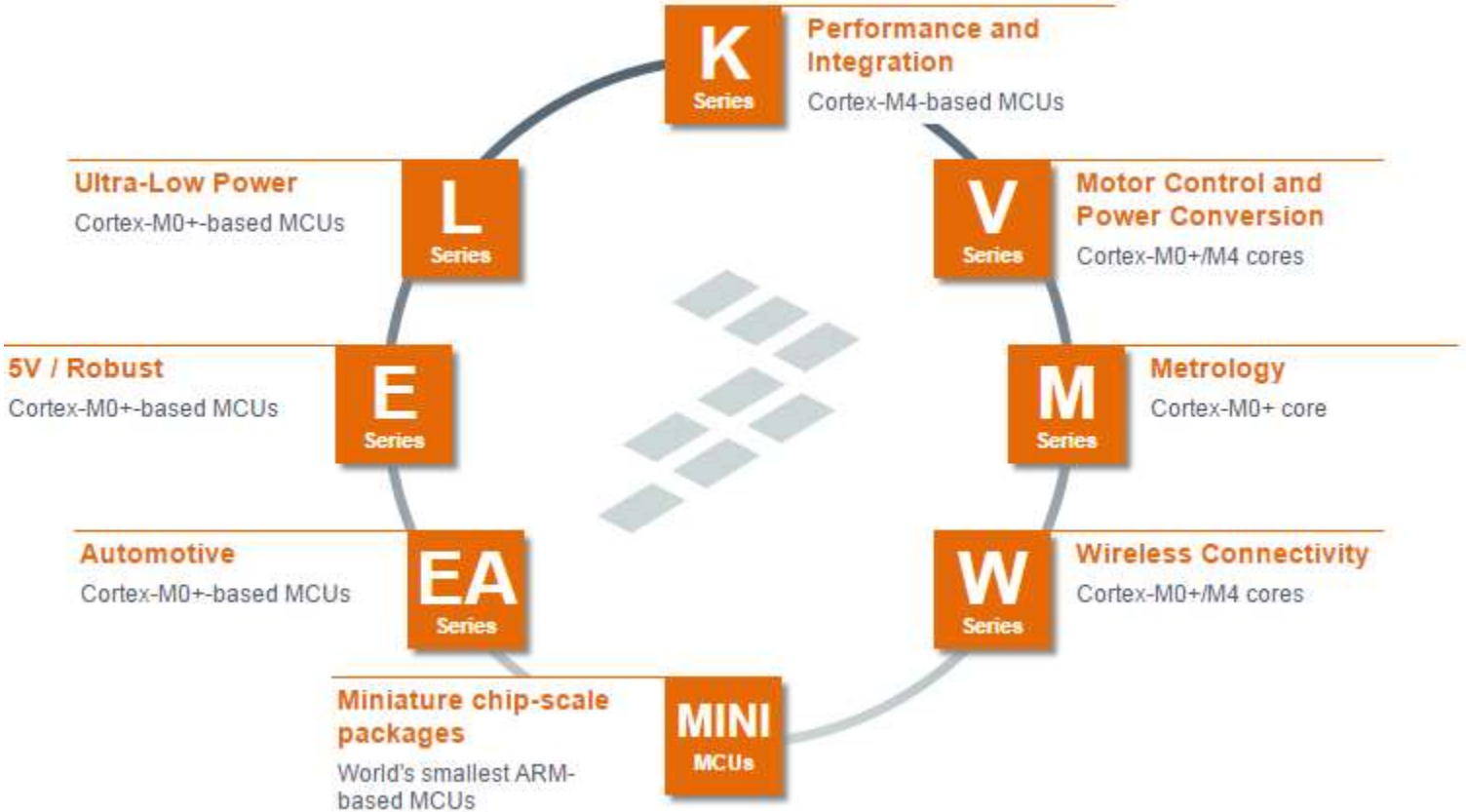
Agenda

- KSDK In-Depth
  - Lab
- KSDK + RTOS
- KSDK + USB
- KSDK + Processor Expert
  - Lab
- Conclusion

Kinetis Software Development Kit (KSDK)

Freescale Kinetis

- Freescale's ARM® Cortex®-M0+, M4, and M7 microcontroller family
- Hardware and software compatibly across hundreds of devices
- Exceptional low-power performance and feature integration



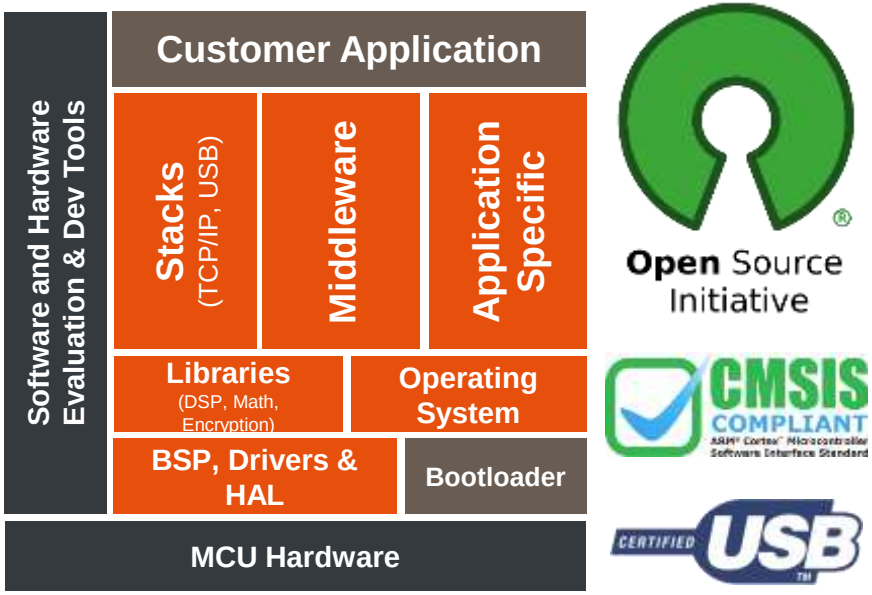
What Is an SDK For and Why Is It Needed ?

- ✓In general, an SDK is a package of pre-written code that developers can re-use in order to minimize the amount of unique code that they need to develop themselves.
- ✓It can help to prevent unnecessary duplication of effort in a development team or community.
- ✓It has a common application programming interface (API) for different platforms or peripherals, what shortens the application developing time.
- ✓Thanks to use of abstraction layers it’s more intuitive and concise for programmers.

Kinetis Software Development Kit (SDK)

The software framework and reference for Kinetis MCU application development

Hardware abstraction, peripheral drivers, stacks, RTOS's, utilities, and usage examples; delivered in C source



Open Source Initiative

CMSIS COMPLIANT

CERTIFIED USB

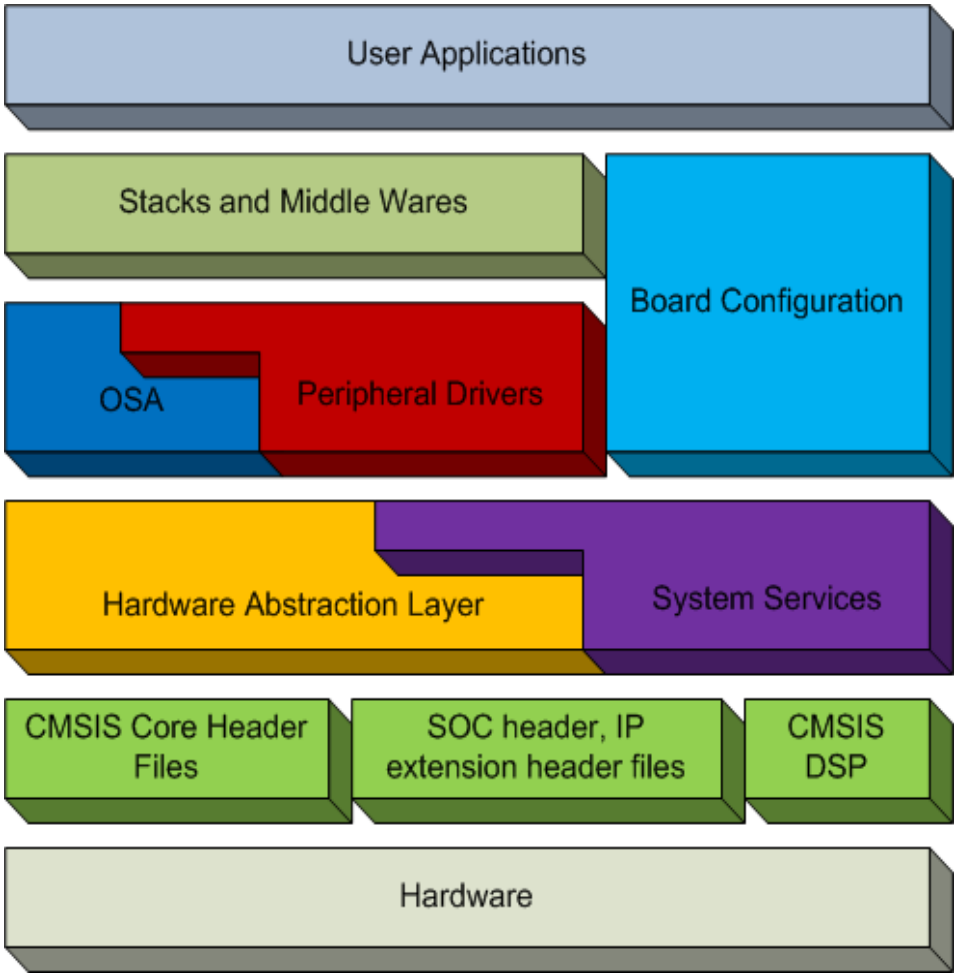
Product Features

- Open source **hardware abstraction layer** (HAL) provides APIs for all Kinetis hardware resources
- BSD-licensed set of **peripheral drivers** with easy-to-use C-language APIs
- Comprehensive HAL and driver **usage examples** and **sample applications** for RTOS and bare-metal
- GUI configurable projects and peripheral drivers using **Processor Expert**
- **CMSIS-CORE** compatible startup plus **CMSIS-DSP** library and examples
- RTOS Abstraction Layer (OSA) with support for Freescale **MQX**, **FreeRTOS**, Micrium **uC/OS**, and **bare-metal**
- Integrates new Freescale unified **USB stack**, open source **TCP/IP stack** (lwIP), open source **FAT file system**, **encryption math/DSP libraries**, and more.
- Support for **multiple toolchains**: GNU GCC, IAR, Keil, Atollic, and Kinetis Design Studio

KSDK Key Components

- Two major components of the KSDK
  - Hardware Abstraction layer (HAL)
  - Peripheral Drivers
- Supporting Components
  - CMSIS-compliant header files
  - System services (clock manager, interrupt manager, low power manager)
  - Operating System Abstraction (OSA) layer
  - Board Support Packages (BSP)
  - Stacks and Middleware

Kinetis SDK Overview



HAL

- Abstracted IP level Basic operations.
- Useable low level drivers.

System Services

- Clock Manager, Interrupt manager, Low power manager, HW timer…
- Can be used with HAL, PD and Application

FSL Peripheral Drivers

- Use case driven high level drivers.

OS Abstraction Layer (OSA)

- Adapt to different OS (MQX, FreeRTOS and uCos) through corresponding OSA

BSP & Configuration

- Board Configuration, Pin Muxing, GPIO Configuration

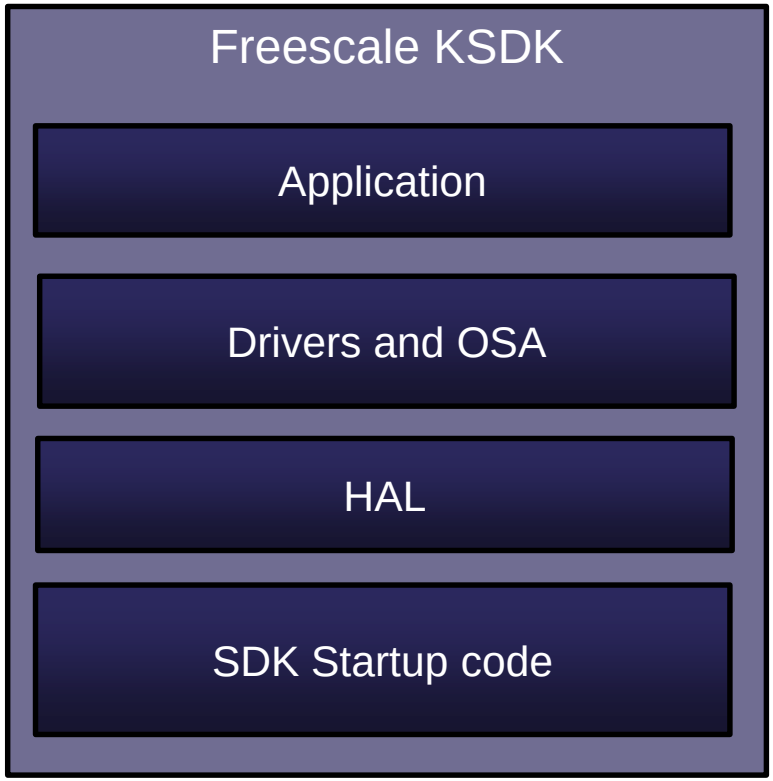
Stacks & Middle Wares

- USB stack, TCP/IP stack, BTLE…
- Audio, Graphics, Boot Loader…

Note: The IP extension header files could be merged with the SoC header in later on KSDK releases…

HAL and Drivers

- HAL is at a lower level than the KSDK drivers
  - No state awareness
  - Mostly macros to provide user-friendly naming to access MCU registers
- KSDK Drivers make use of HAL API to implement their functionality.



HAL Overview

- Create the basic abstraction layer over MCU internal peripherals
  - Each individual peripheral has own dedicated HAL
- Full coverage of all peripherals features
  - Also implements the function for module initialization (reset)
- Possible configurability
  - In compilation time via feature header files
  - In run-time by taking user defined configuration data through “init” function call
- Does not implement the interrupt driven logic (ISR)
  - It’s implemented by Peripheral Drivers or User Application
  - User Application based only on HAL need to define own ISR entries
- HAL Source at C:\Freescale\KSDK_1.1.0\platform\hal
- HAL Library at C:\Freescale\KSDK_1.1.0\lib\ksdk_hal_lib

Example of HAL for SPI

```
void      SPI_HAL_Init                (uint32_t baseAddr)

uint32_t SPI_HAL_SetBaud          (uint32_t baseAddr,  uint32_t bitsPerSec, uint32_t sourceClockInHz)
void      SPI_HAL_SetDataFormat(uint32_t baseAddr, spi_clock_polarity_t polarity,
                                                   spi_clock_phase_t phase,
                                                   spi_shift_direction_t direction)

static inline void SPI_HAL_Enable (uint32_t baseAddr)
static inline void SPI_HAL_Disable(uint32_t baseAddr)

static inline void SPI_HAL_SetMasterSlave(uint32_t baseAddr, spi_master_slave_mode_t mode)
static inline bool SPI_HAL_IsMaster(uint32_t baseAddr)
…
static inline void SPI_HAL_SetMatchIntCmd(uint32_t baseAddr, bool enable)
static inline boolSPI_HAL_IsMatchPending(uint32_t baseAddr)
…
static inline uint8_t SPI_HAL_ReadData(uint32_t baseAddr)
static inline void    SPI_HAL_WriteData(uint32_t baseAddr, uint8_t data)
```

Drivers Overview

- KSDK implements complex high level logic over SoC peripherals
- Are based on one or multiple HAL, other drivers and/or system services
- Support run-time configuration through “init” function call
  - Configuration data are passed by pointer to driver’s specific configuration structure
- Defines needed ISR entries for the interrupt driven driver
  - All actions needed to be taken in ISR entries cover a public function general for all instances of drivers *xxx_DRV_IRQHandler(uint32_t instance)*
  - The fsl_xxx_irq.c file inside drivers directory contains the default implementation of handlers used in vector table
  - User can update the ISR entries by adding user actions, the C file with ISR entries will not be built into the driver library
- Same driver API is used when accessing same function across HAL with similar functionality
- For some of these drivers, MQX brings POSIX compliant API wrappers

- Driver Source at C:\Freescale\KSDK_1.1.0\platform\drivers
- Driver+HAL library at C:\Freescale\KSDK_1.1.0\lib\ksdk_platform_lib

Example of PD for SPI (MASTER)

```
void SPI_DRV_MasterInit(uint32_t instance, spi_master_state_t * spiState);

void SPI_DRV_MasterConfigureBus(spi_master_state_t * spiState,
                                const spi_master_user_config_t * device,
                                uint32_t * calculatedBaudRate);

spi_status_t SPI_DRV_MasterTransferBlocking(spi_master_state_t * spiState,
                const spi_master_user_config_t * restrict device,
                const uint8_t * restrict sendBuffer, uint8_t * restrict receiveBuffer,
                size_t transferByteCount, uint32_t timeout);

spi_status_t SPI_DRV_MasterTransfer(spi_master_state_t * spiState,
                const spi_master_user_config_t * restrict device,
                const uint8_t * restrict sendBuffer,
                uint8_t * restrict receiveBuffer,
                size_t transferByteCount);

spi_status_t SPI_DRV_MasterGetTransferStatus(spi_master_state_t * spiState, uint32_t * bytesTransferred);
spi_status_t SPI_DRV_MasterAbortTransfer(spi_master_state_t * spiState);
```

CMSIS, SoC and IP extensions headers

- Cortex Microcontroller Software Interface Standard (CMSIS)
  - Core specific macros and inline functions
  - Compliance startup codes
  - DSP lib and source files included for GCC (other tool chains such as IAR and KEIL has CMSIS DSP lib built in)

- SoC header files
  - Mapped memory and register's addresses over SoC (similar to CMSIS headers)
  - Are generated by using API factory tool owned by Processor Expert team.

- IP extension header files
  - Each IP has own extension header file
  - Create easy access to IP registers via bit-field macros (SET, CLR, GET, …).
  - Are using BME where possible.

System services

- Common used services
  - System Timer (can be running on any of the hw-timers in SoC)
  - Centralized Clock Manager (for peripherals driven)
  - Centralized Interrupt Manager
  - Low Power Manager
- Are built over SoC header files and some HAL components
- Are used by Peripheral Drivers or User Application
  - User can just use HAL and System Services to build applications.
  - If user would only use Peripheral Drivers, then do not need to use system services.
- Are used by OSA

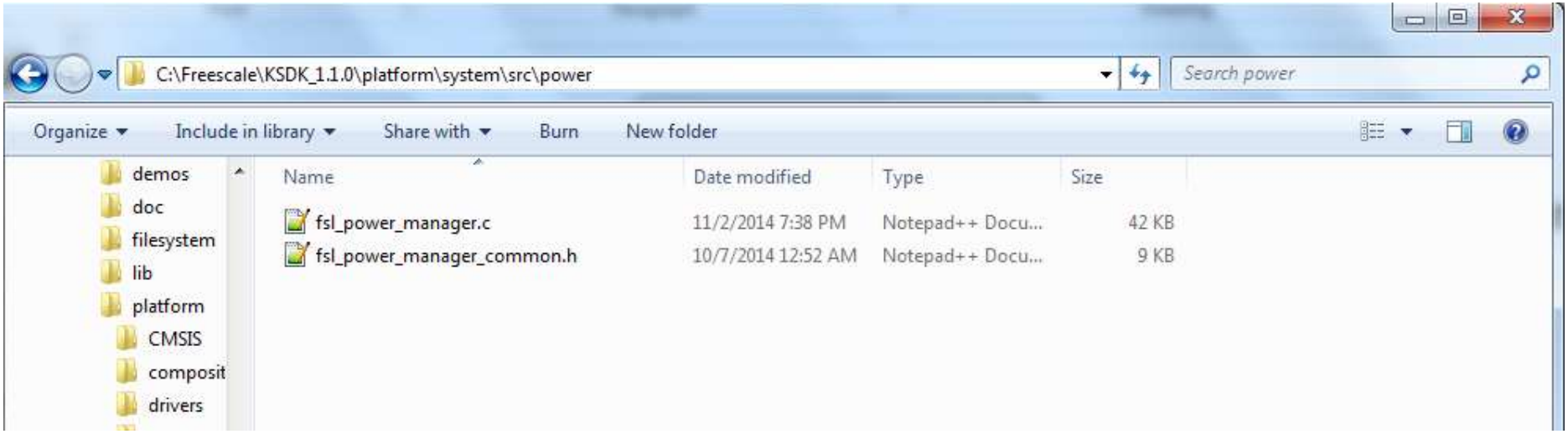
KSDK Power Manager

What is the SDK Power Manager?

- A high-level API that allows an application to easily manage and utilize its supported power modes.
- Provides the ability to execute application-defined callbacks before and/or after power mode transitions.
- Enables agreeable or forcible transition between power modes, allowing peripherals to hold-off transition requests or the application to force transition.

Where can you find the Power Manager?

- The Power Manager is part of the Kinetis SDK. Specifically, it is a component of the platform library's system services.



Power Manager Overview – Initialization

- The application defines the supported power modes.
  - This will typically be a subset of what the specific MCU supports since it's application-specific.
  - Supported modes are defined as structures and passed into **POWER_SYS_Init()**.

- Callbacks are defined during device initialization and also passed into **POWER_SYS_Init()**.

Power Manager Interaction With Other Components

- The Power Manager only touches the **SMC, PMC and RCM** registers, which are the main blocks needed to transition into a low power state.

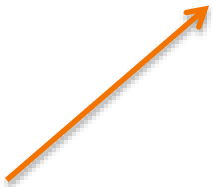
- It *does not* configure wake-up sources or adjust clock frequencies. The application is responsible for enabling and configuring wake-up and clock adjustments.

- It relies on user-defined callback functions to interact with other application components.
  - For example, if clocks need to be adjusted prior to changing power mode, a “before” callback should be used.
  - Allows for user-defined data to be passed into the callback functions. This data can then be used by the application to determine state or perform necessary tasks.

Changing Power Modes

- Changing power modes is very easy with the Power Manager.
- Based on the policy of the selected power configuration, the Power Manager can either force entry or abort if the user callback signals it is not ready.

```
power_manager_error_code_t POWER_SYS_SetMode(uint8_t powerModeIndex)
```



Index of desired user-defined power mode

```
// Create list of supported modes to pass in to
// POWER_SYS_Init().
power_manager_user_config_t const *powerConfigs[] =
{
    &waitConfig
};
```

This is what the index refers to

Future Improvements

- The Kinetis SDK is still in its infancy and there are plans to improve the power manager in future releases.

- Future improvements being discussed:
  - Automatic clock adjustment/checking based on a tightly coupled relationship with the SDK clock manager.
  - Power Manager awareness built into SDK reference drivers.

  Example application in \demos\power_management_demo

OS Abstraction (OSA)

OS Abstraction Layer Overview

- Enables KSDK to work with different RTOSes
- Support key RTOS services
  - Semaphores, Mutex, Memory Management, Events, more…
- Implementation for different RTOSes
  - Bare Metal
  - MQX™, FreeRTOS, uCOS-II, and uCOS-III
- Does not abstract ISRs
  - ISRs must be set up slightly different depending on the RTOS used
  - Some RTOS require prologue and epilogue for ISR enter and exit
  - Some RTOS require ISR entries be registered with RTOS-specific ISR registration function

OS Abstraction Layer Example: OSA_TimeDelay()

Translation code found in \platform\osa

- For MQX maps to:

```
void OSA_TimeDelay(uint32_t delay)
{
    _time_delay(delay);
}
```

- For FreeRTOS maps to:

```
void OSA_TimeDelay(uint32_t delay)
{
    vTaskDelay(delay/portTICK_RATE_MS);
}
```

- For Baremetal maps to:

```
void OSA_TimeDelay(uint32_t delay)
{
    uint32_t currTime, timeStart;

    timeStart = OSA_TimeGetMsec();

    do {
        currTime = OSA_TimeGetMsec(); /* Get current time stamp */
    } while (delay >= time_diff(timeStart, currTime));
}
```

OS Abstraction Layer

- The OSA layer allows the same user code to be compatible with multiple RTOSes
  - See I2C_rtos example in KSDK
  - Same software works with bare-metal, MQX, FreeRTOS, uCOS

- Still have option of using direct RTOS function calls
  - Use either OSA_TimeDelay(500) or _time_delay(500)

Stacks and Other Middleware

- This layer completes the KSDK source and made it easy to use

- Includes
  - All Freescale stacks like Host and Device USB stacks, …
  - Third party enablement software stacks like lwip, FatFs, …
  - RTOS source codes like MQX, FreeRTOS, uCOSII, uCOSIII, …

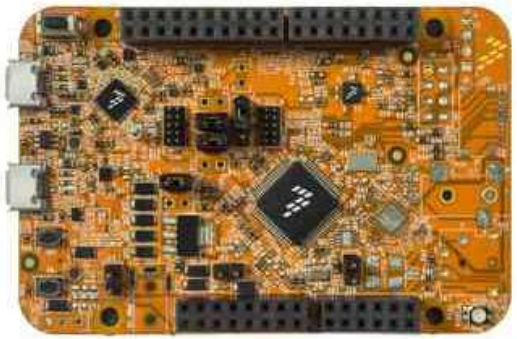
- All middle wares are run on top of the KSDK drivers
  - Freescale USB stack not adhere to this rule, because SDK HAL is not implementing USB IP now.

Board Configuration and Support

- Pin Muxing
  - KSDK driver layer will not handle pin muxing. It is handled in the board configuration part, where pin muxing functions are generated using “*Pin Muxing*” tool in **KDS** via **PEx**.
- Board Specific configuration
  - GPIO configuration
  - Hardware Initialization code
  - Function to initialize serial console for debug purposes.
- Drivers for common devices included in our evaluation boards.
  - ENET PHY
  - Accelerometer
  - Codec

KSDK 1.1 Supported Devices

- **K22F**
  - FRDM-K22F
  - FRDM-K22FK02
  - FRDM-K22FK0264
  - TWR-K22F120M
  - TWR-K22F120MK02
- **K24F**
  - TWR-K24F120M
- **K60D**
  - TWR-K60D100M
- **K64F**
  - TWR-K64F120M
  - FRDM-K64F
- **KL46**
  - FRDM-KL46Z
- **KV10**
  - TWR-KV10Z75M
- **KV31**
  - TWR-KV31F120M
  - TWR-KV31F120MKV30



KSDK 1.1

- Released December 2014
- Changes Include:
  - Reorganized directory structure
  - Support for Atollic IDE
  - New Devices Supported:

Device Families	Boards	Tool Chains
MK24F12	TWR-K64F120M FRDM-K64F	IAR 7.20.2 KDS 2.0.0 KEIL 5.11 Atollic 5.2 ARM GCC 4.8.3
MK63F12		
MK64F12		
MK22F12810	TWRK-22F120M FRDM-K22F	
MK22F25612		
MK22F51212		
MKV31F12810	TWR-KV31F120M	
MKV31F25612		
MKV31F51212		
MK24F25612	TWR-K24F120M	
MK60D10	TWR-K60D100M	
MKL03Z4	FRDM-KL03Z	
MKL46Z4	FRDM-KL46Z	
MKV10Z7	TWR-KV10Z75M	
MKV30F12810	TWR-KV31F120MKV30	
MK02F12810	FRDM-K22FK02 FRDM-K22FK0264 TWR-K22FK02	

KSDK 1.2 – Released on April 28

NEW

MCU Support for all v1.1 Mainline and Standalones, plus :

- K10D 100MHz
- K20D 100MHz
- K30D 100MHz
- K40D 100MHz
- K5xD 100MHz
- K60D 100MHz
- K21F Rev. A 120MHz
- KL02Z
- KL14Z
- KL15Z
- KL24Z
- KL25Z
- K26F 180MHz
- K65F 180MHz
- K66F 180MHz

New Peripheral Support:

- AOI
- ENC
- FLEXBUS
- FLEXIO
- LMEM
- VREF
- XBAR
- PWM

All Supported Eval Platforms:

Kinetis L:
- FRDM-KL02Z
- FRDM-KL03Z
- FRDM-KL25Z
- FRDM-KL26Z
- FRDM-KL27Z
- FRDM-KL43Z
- FRDM-KL46Z
- TWR-KL43Z48M

Kinetis K:
- FRDM-K22F
- FRDM-K64F
- TWR-K21D50M
- TWR-K21F120M
- TWR-K22F120M
- TWR-K24F120M
- TWR-K60D100M
- TWR-K64F120M
- TWR-K65F180M

Kinetis V:
- TWR-KV10Z32M
- TWR-KV31F120M
- TWR-KV46F150M

Kinetis W:
- FRDM-KW24D
- TWR-KW24D512
- USB-KW24D512
- MRB-KW019032xx

Kinetis SDK 1.2 Improvements:

- Organizes SDK examples by board rather than demo
- Adds functionality to clock manager
- Adds functionality to power mode manager
- Simplifies the clock configuration
- Optimizes demo pin mux configurations for low-power

KSDK Tools

Kinetis SDK IDE Options

IAR Embedded Workbench

- A powerful and reliable IDE designed for ease of use with outstanding compiler optimizations for size and speed
- The broadest Freescale ARM/Cortex MCU offering with dedicated versions available with functional safety certification
- Support for multi-core, low power debugging, trace, ...

Keil Microcontroller Development Kit

- Specifically designed for microcontroller applications, easy to learn and use, yet powerful enough for the most demanding embedded applications
- ARM C/C++ build toolchain and Execution Profiler and Performance Analyzer enable highly optimized programs
- Complete Code Coverage information about your program's execution

Kinetis Design Studio

- Complimentary basic capability integrated development environment (IDE) for Kinetis MCUs
- Eclipse and GCC-based IDE for C/C++ editing, compiling and debugging

GNU ARM GCC

- Open Source GNU GCC compiler for ARM devices
- ARM C/C++ build toolchain

Atollic TrueSTUDIO

- Professional ECLIPSE/GNU based IDE with a MISRA-C checker, code complexity analysis and source code review features.
- Advanced RTOS-aware debugger with ETM/ETB/SWV/ITM tracing, live variable watch view and fault analyzer. Dual-core and multi-processor debugging.
- Strong support for software engineering, workflow management, team collaboration and improved software quality.

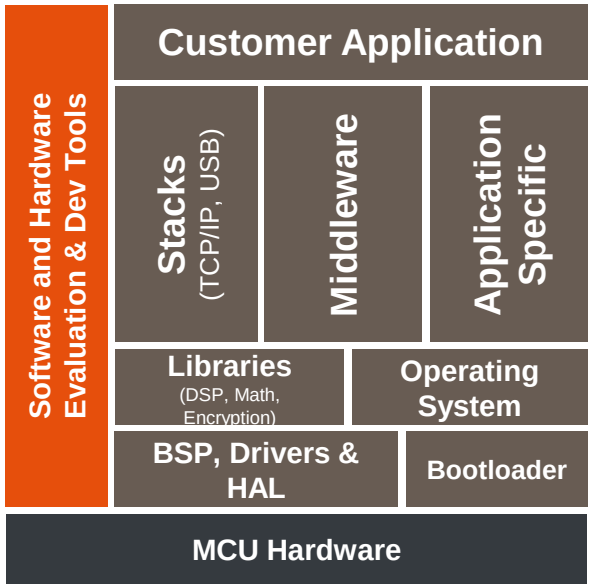
Additional Ecosystem Partners:

Kinetis Design Studio (KDS)

Kinetis Design Studio

No-cost integrated development environment (IDE) for Kinetis MCUs

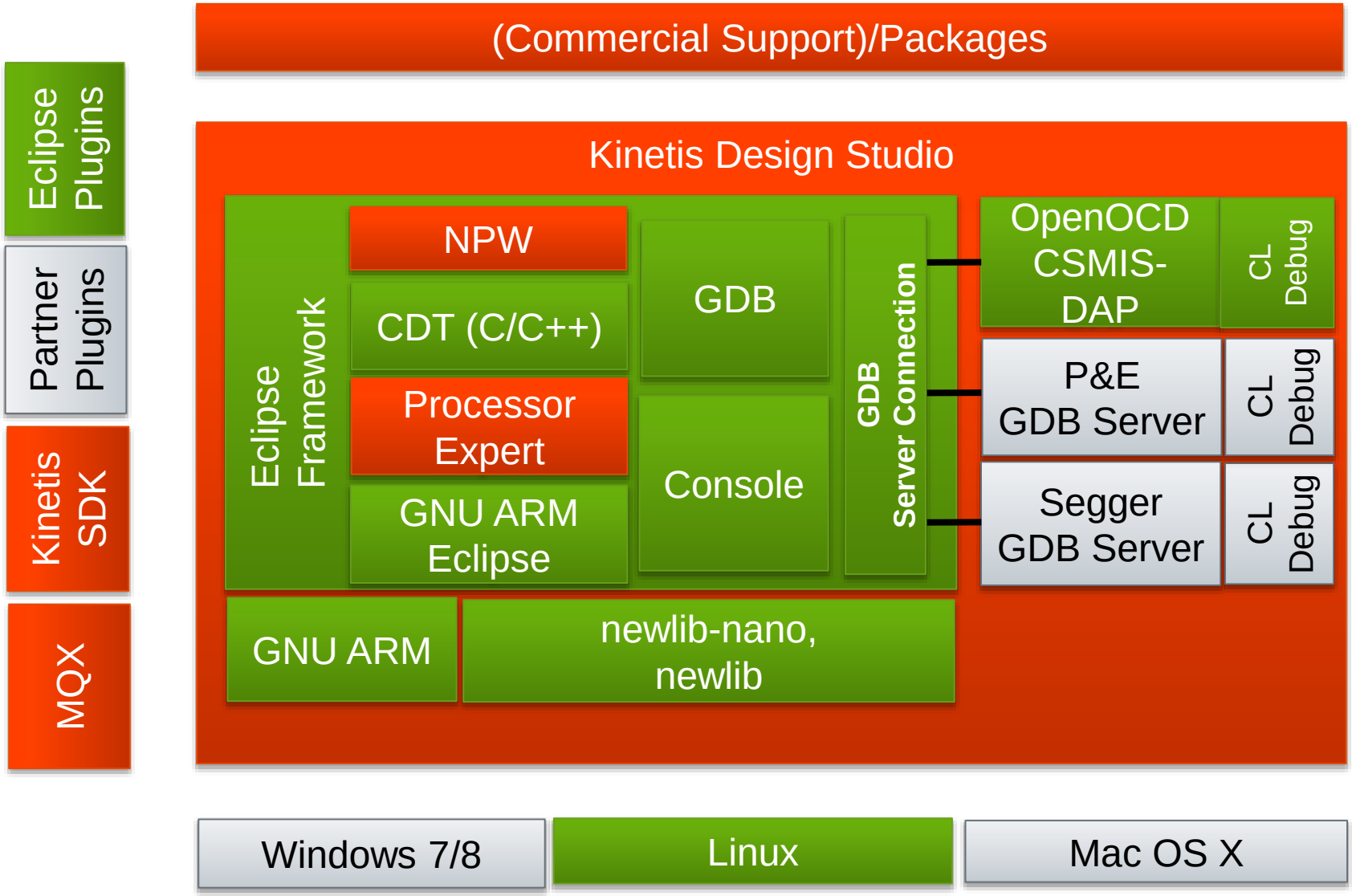
Eclipse and GCC-based IDE for C/C++ editing, compiling and debugging



Product Features

- A free of charge and unlimited IDE for Kinetis MCUs
- A basic IDE that offers robust editing, compiling and debugging
- Based on Eclipse, GCC, GDB and other open-source technologies
- Includes Processor Expert (PEx) with Kinetis SDK integration
  - Supports all existing Kinetis devices via Processor Expert and new project wizard
  - All new Kinetis devices will also feature the Kinetis SDK with Processor Expert configurability
- Host operating systems:
  - Windows® 7/8 (32 and 64-bit)
  - Linux™ (Ubuntu, Redhat, Centos) (64bit)
  - Mac® OS X (with v3.0) and Segger
- Support for SEGGER, P&E and Open SDA/CMSIS-DAP debugger targets
- Support for Eclipse plug-ins including RTOS-awareness (i.e. MQX, FreeRTOS)

Kinetis Design Studio – Block Diagram



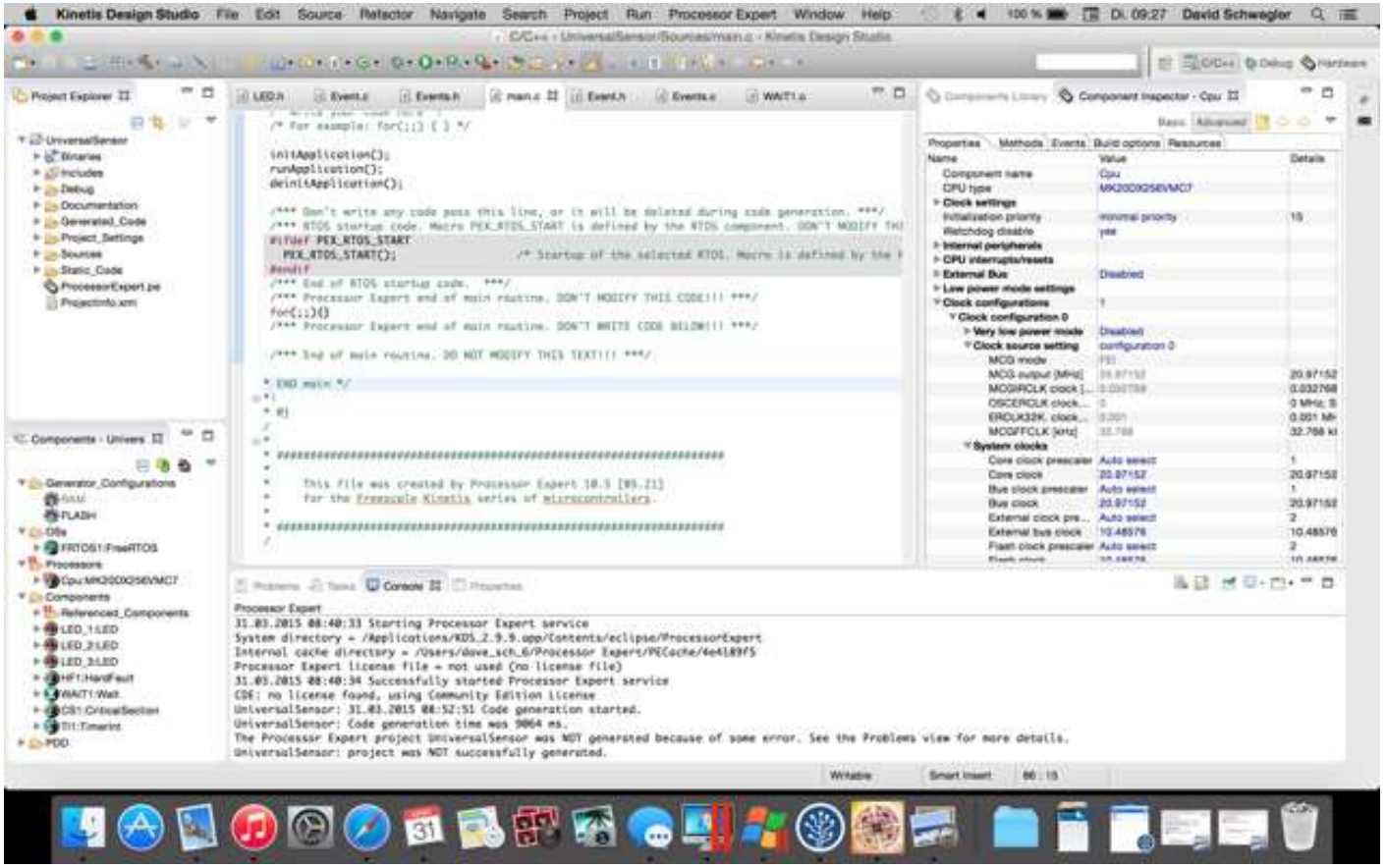
Proprietary | Open Source | Freescale

KDS V3.0.0

- NEW Release: Launched 30-Apr-2015
- Hosts
  - Windows 7/8 (32bit and 64bit)
  - Linux 64bit (Ubuntu 14.04, RedHat/Centos 7) (32bit host libs for Launchpad)
  - Mac OS X ("Yosemite") and Segger Run Control
- Eclipse Luna 4.4 Framework
  - Welcome view, Split view, Black Theme, …
- Processor Expert for Kinetis V3.0
  - Simpler and Easier SDK usage (NPW)
  - Component Repositories
  - Keil and IAR external IDE/build support
- Launchpad GNU Tools for ARM Embedded (4.8)
  - Newlib nano for devices <2 KByte RAM
- Improved Debugging support
  - Updated Segger and P&E (same OpenOCD as in v2.0.0)
  - JTAG Daisy Chaining, Memory Range Protection, Attach to running target
  - Register Details Viewer

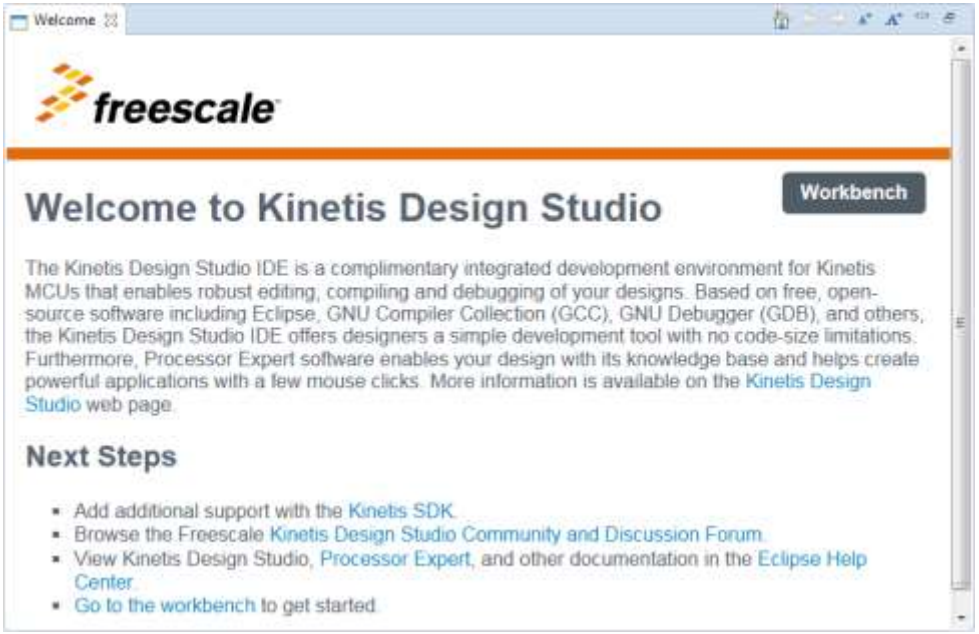
Mac OS X

- Yosemite, 10.10
- Segger J-Link

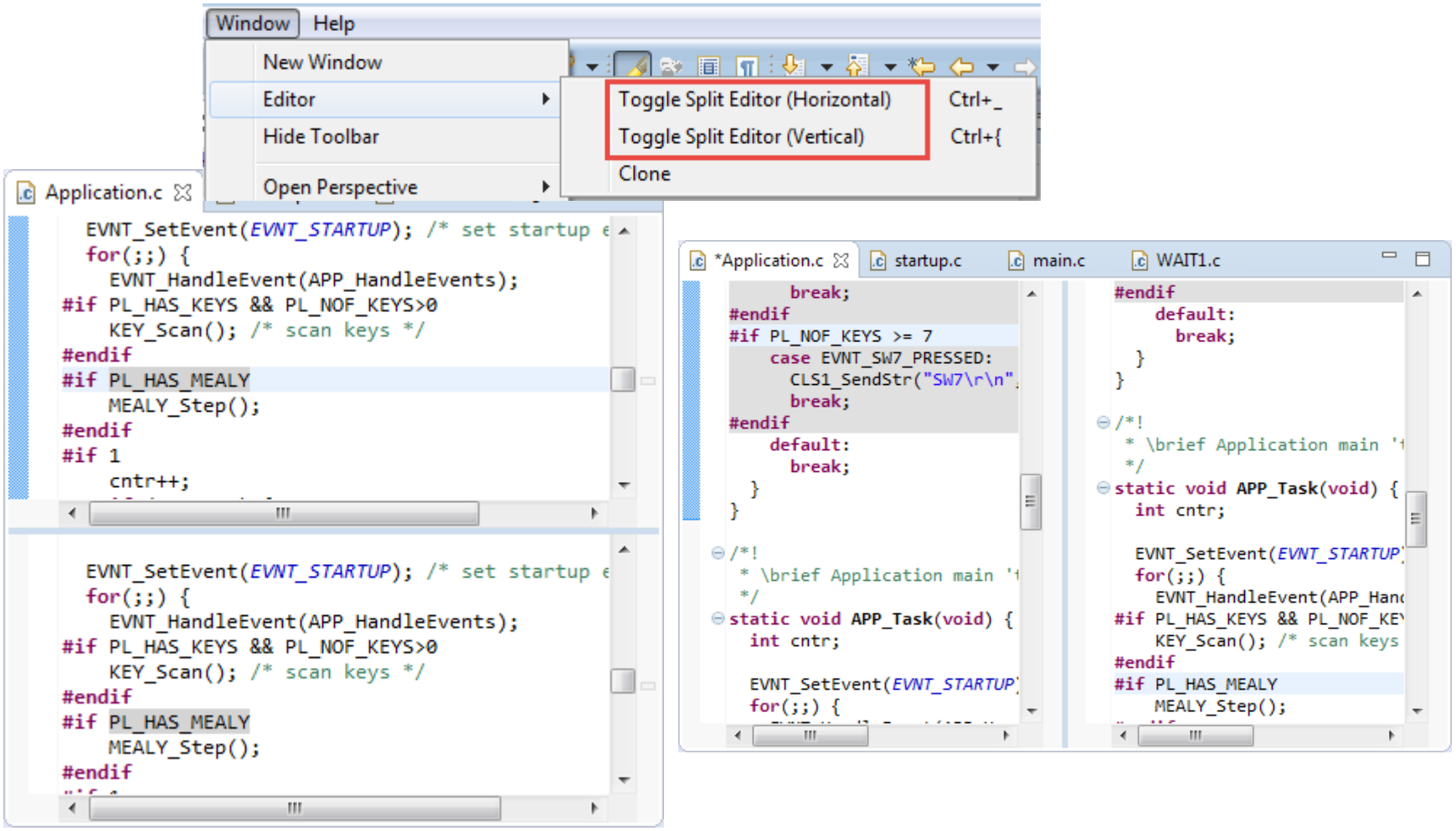


Eclipse, Welcome View

- Eclipse Luna 4.4
  - Same look and feel as Kepler 4.3 (easy migration)
  - Overall performance improvements
- Freescale Welcome view pointing to online resources and help

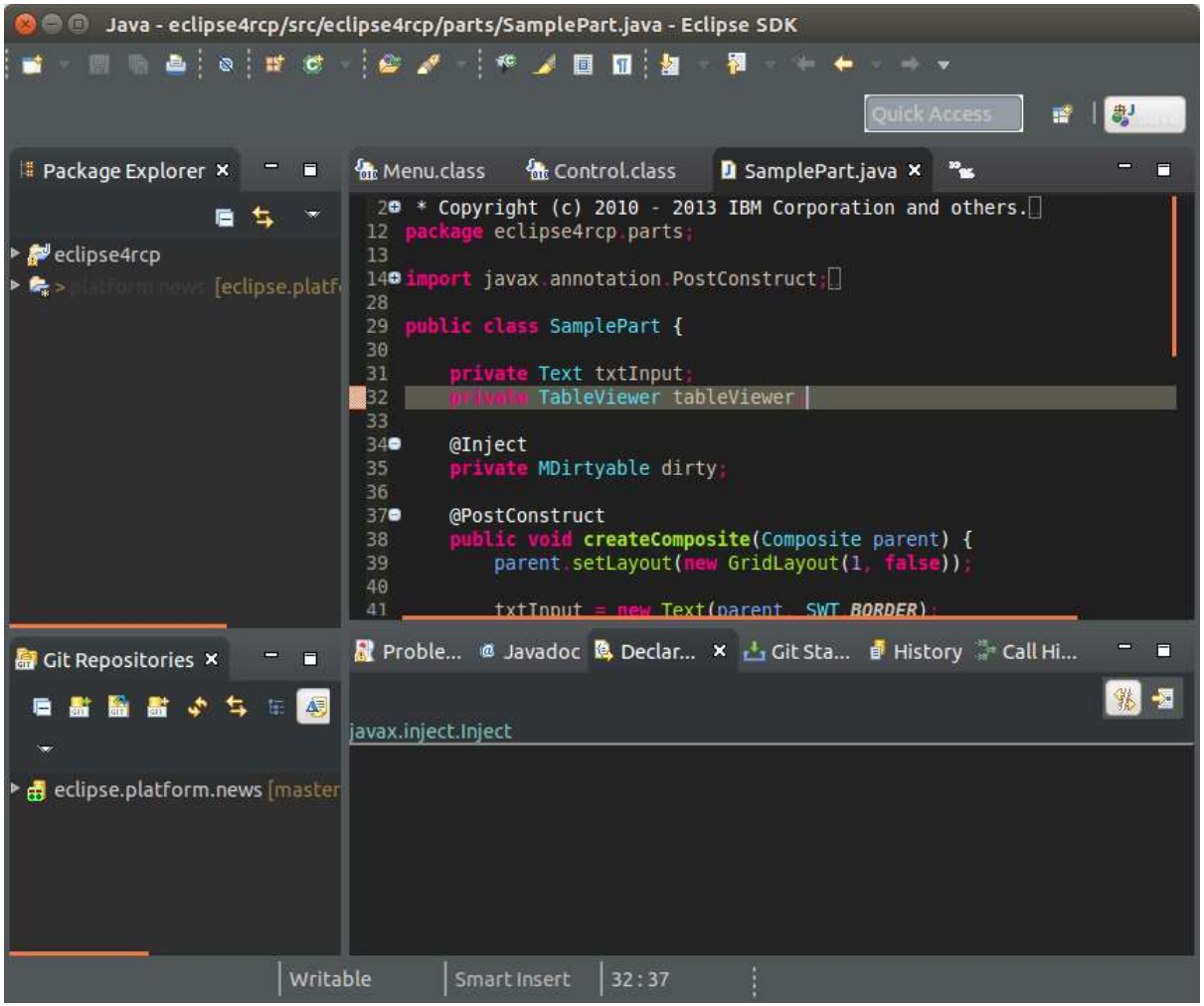


Eclipse Luna (4.4): Split Editor Views



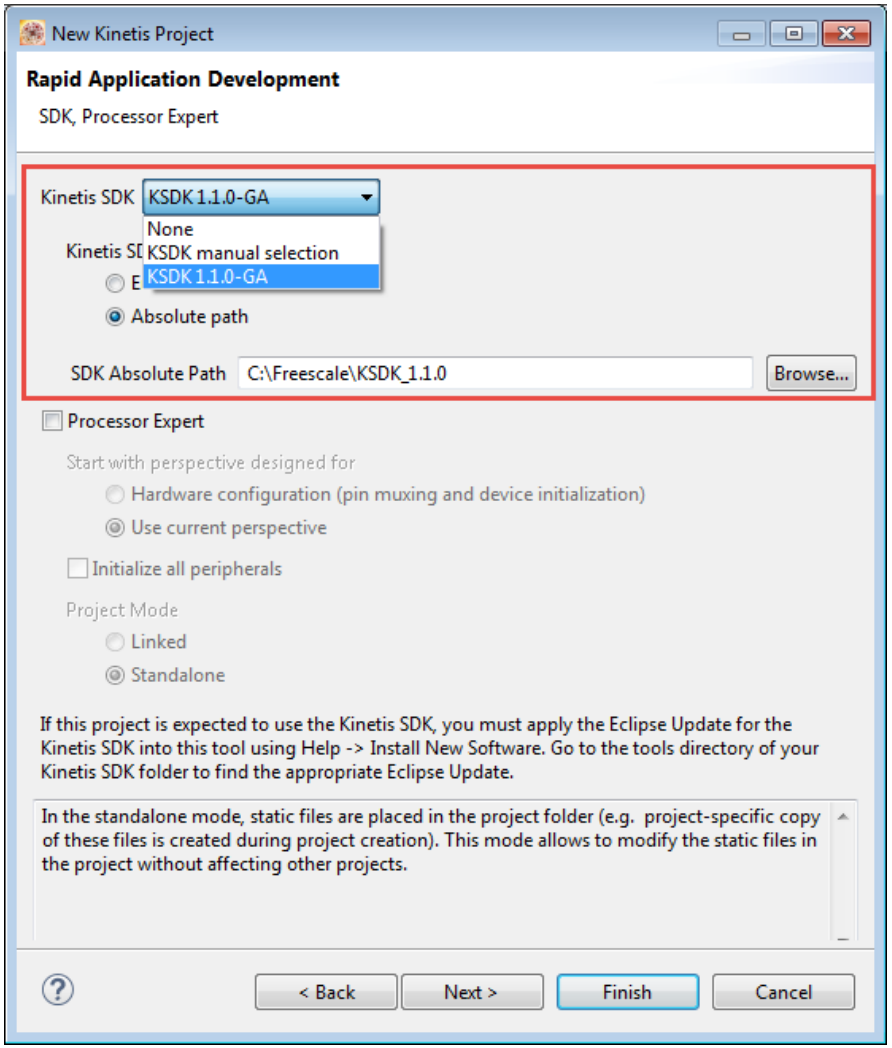
Eclipse Luna: Dark Appearance

- New Styling Engine, General > Appearance



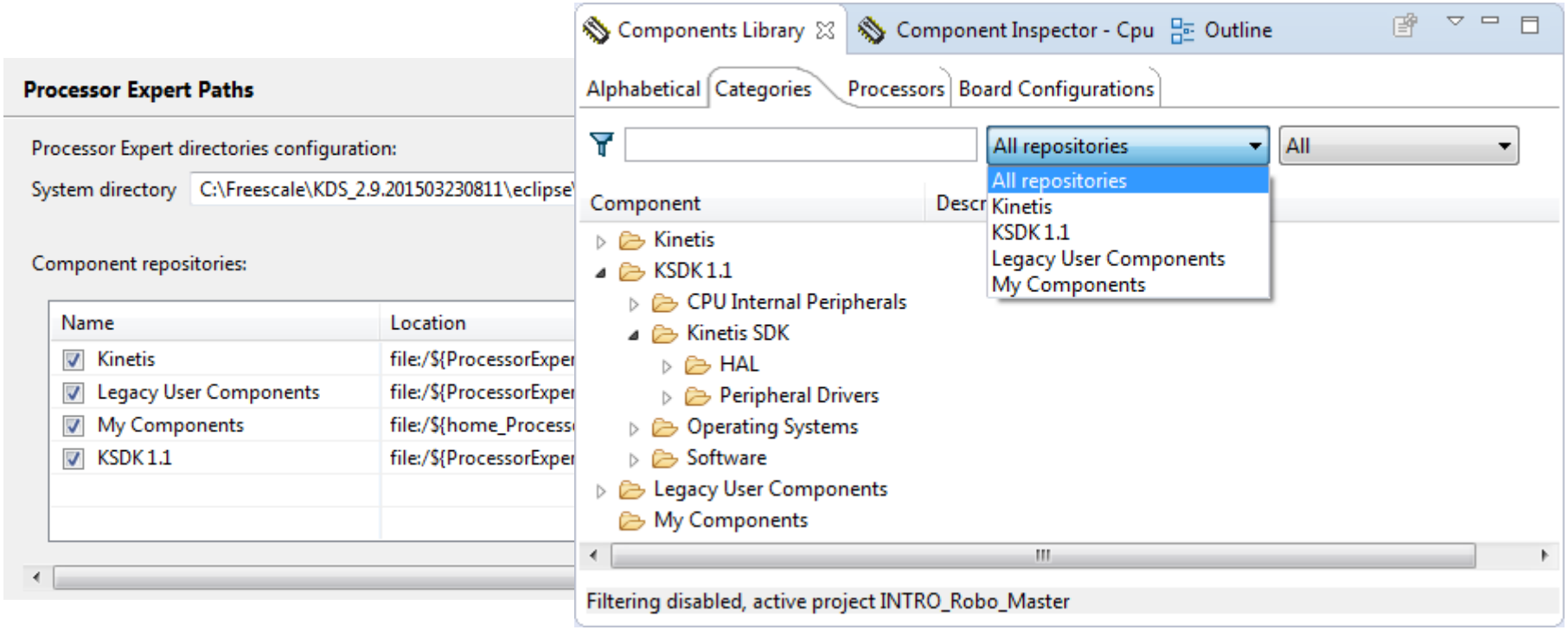
Ease-of-Use: Kinetis SDK Project Creation

- Kinetis SDK Selection
- Path Selection



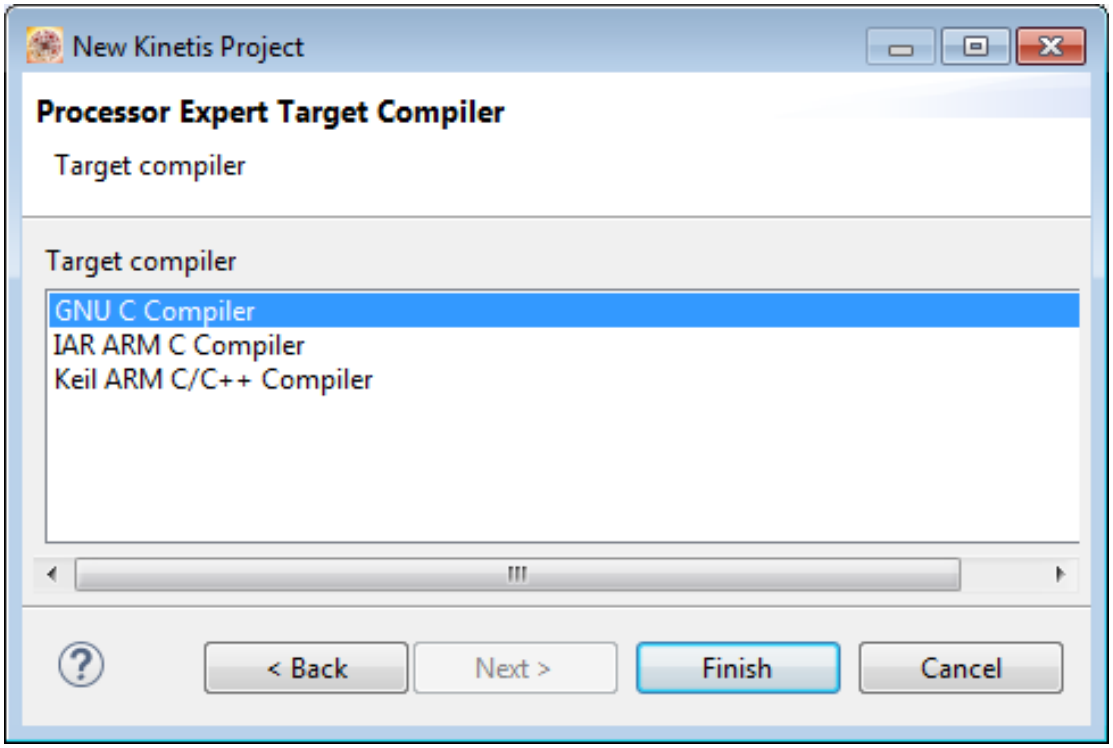
Processor Expert Component Repositories

- Improved Version Control System support
- Multiple configurable repositories
- Filtering



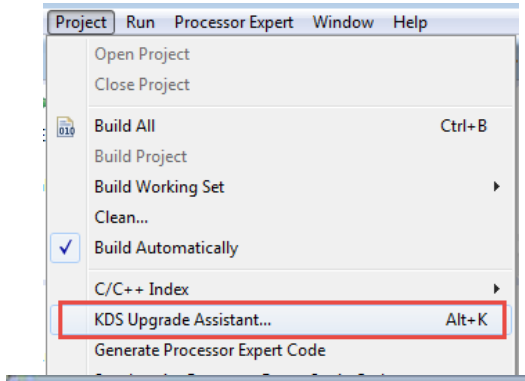
Code Generation for Keil/IAR

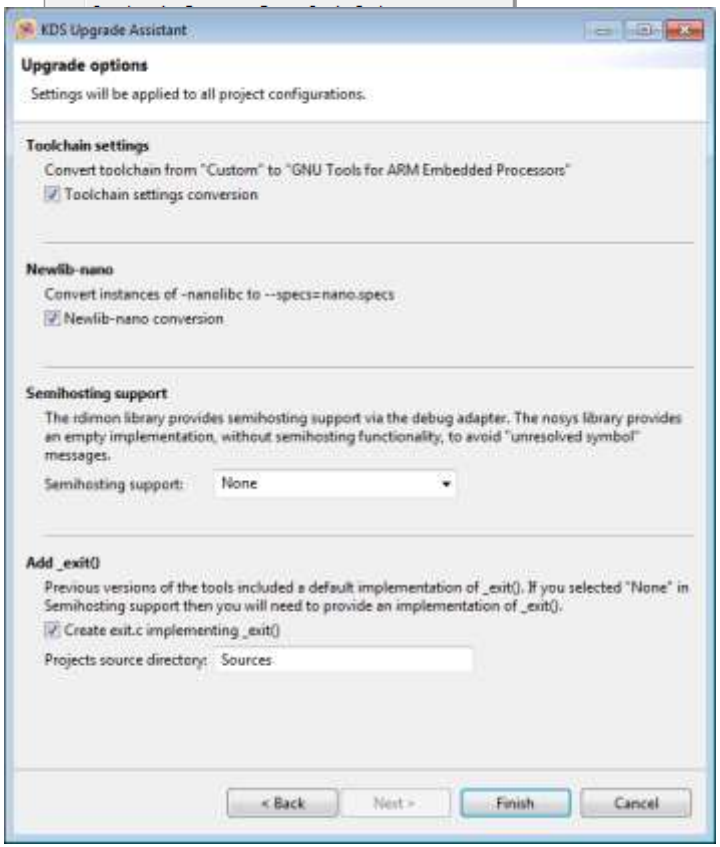
- Processor Expert Multi-Compiler Code Generation
  - GNU: default, for KDS or any other GNU Toolchain, projects within Eclipse
  - External IDE support: IAR & Keil as in DriverSuite



ARM Launchpad Tools

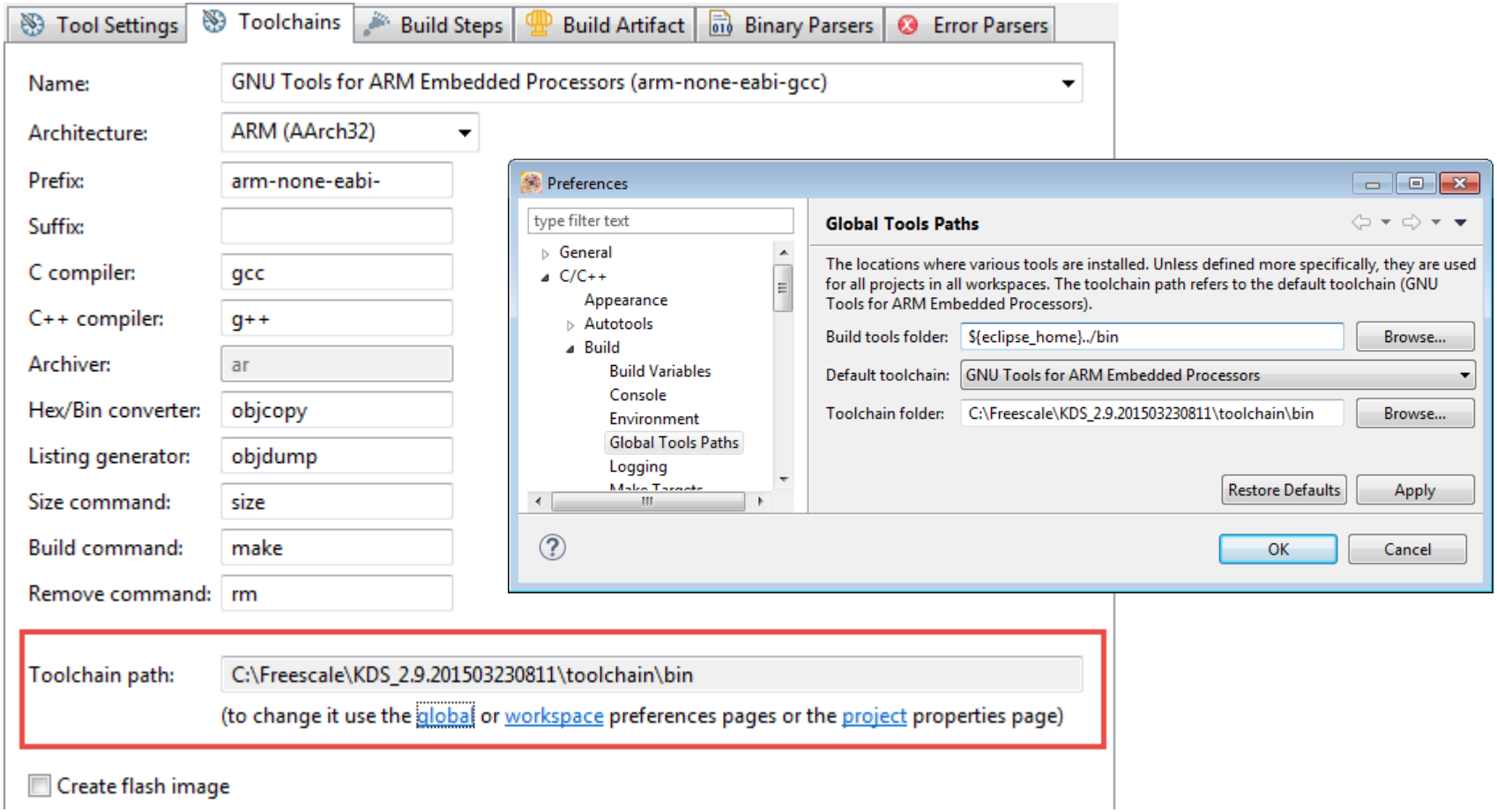
- Using ARM Inc. maintained standard Launchpad GNU Toolchain
  - Removed dependency on external provider, newlib-nano optimized library for devices with < 2 Kbyte RAM
- Project Upgrade wizard
  - Linker options
  - Semihosting
  - _exit() code





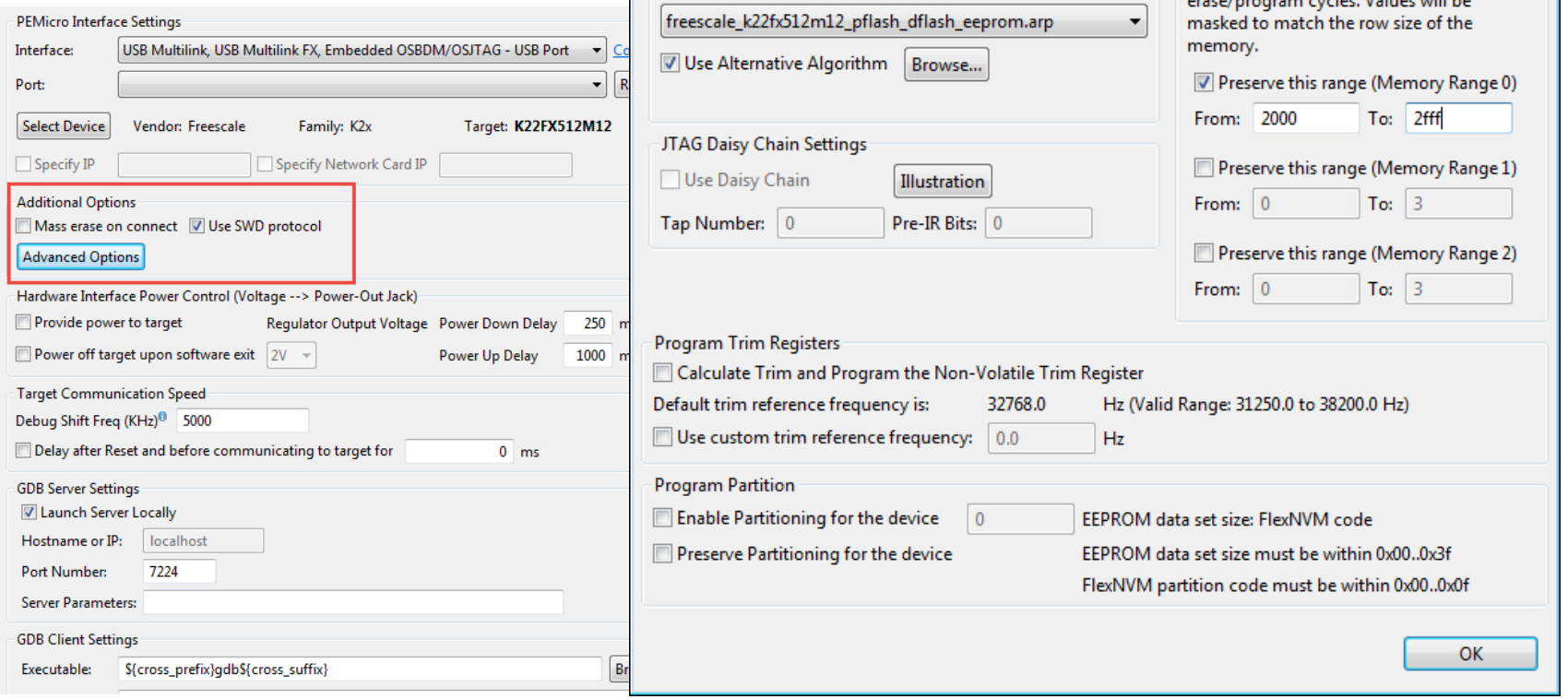
Multi-Toolchain Configuration

- Toolchain configurable globally, on workspace or project



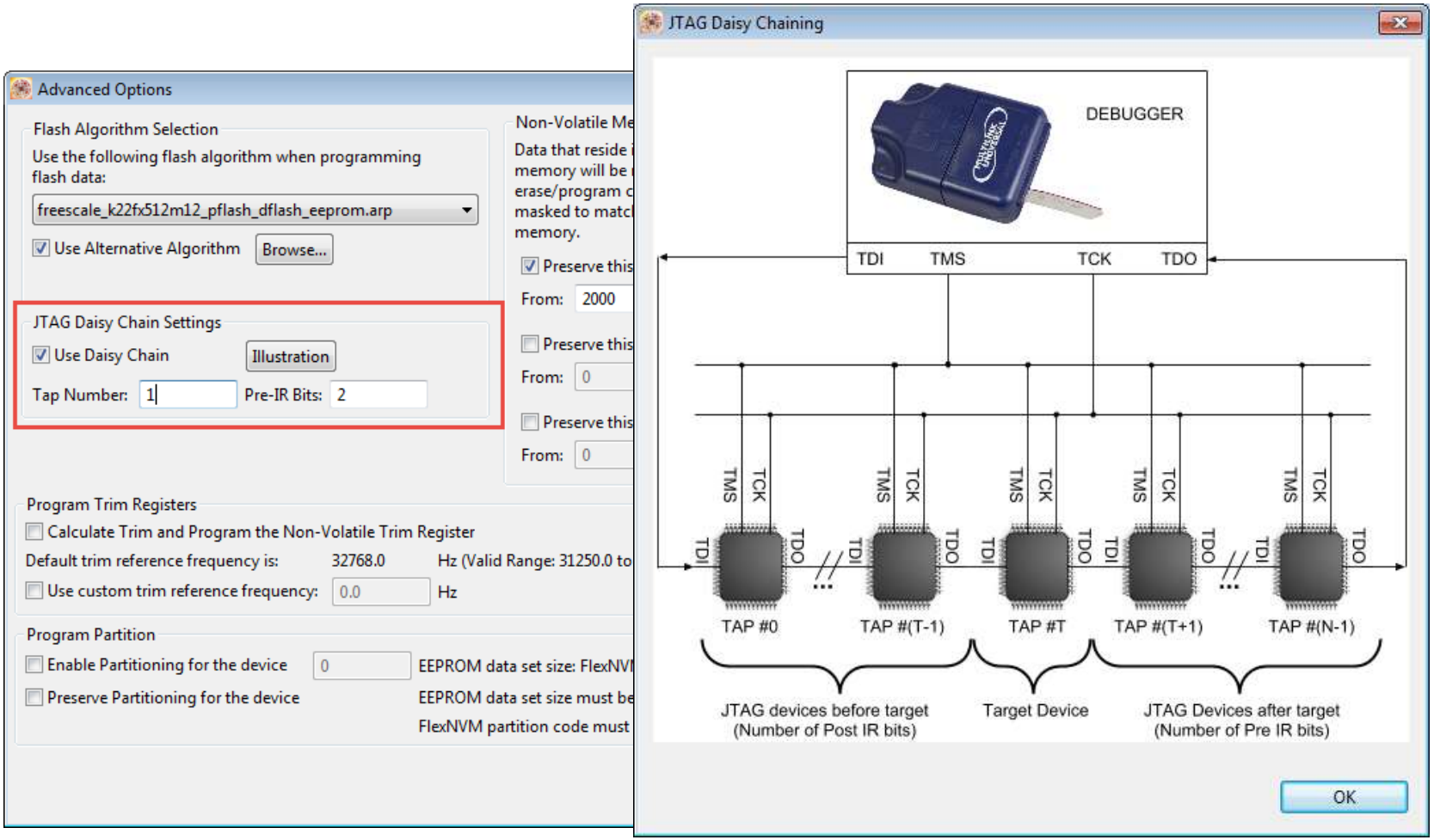
Advanced Debugging Options

- Preserving Memory Ranges
- Using alternative Flash Algorithms
- Trimming



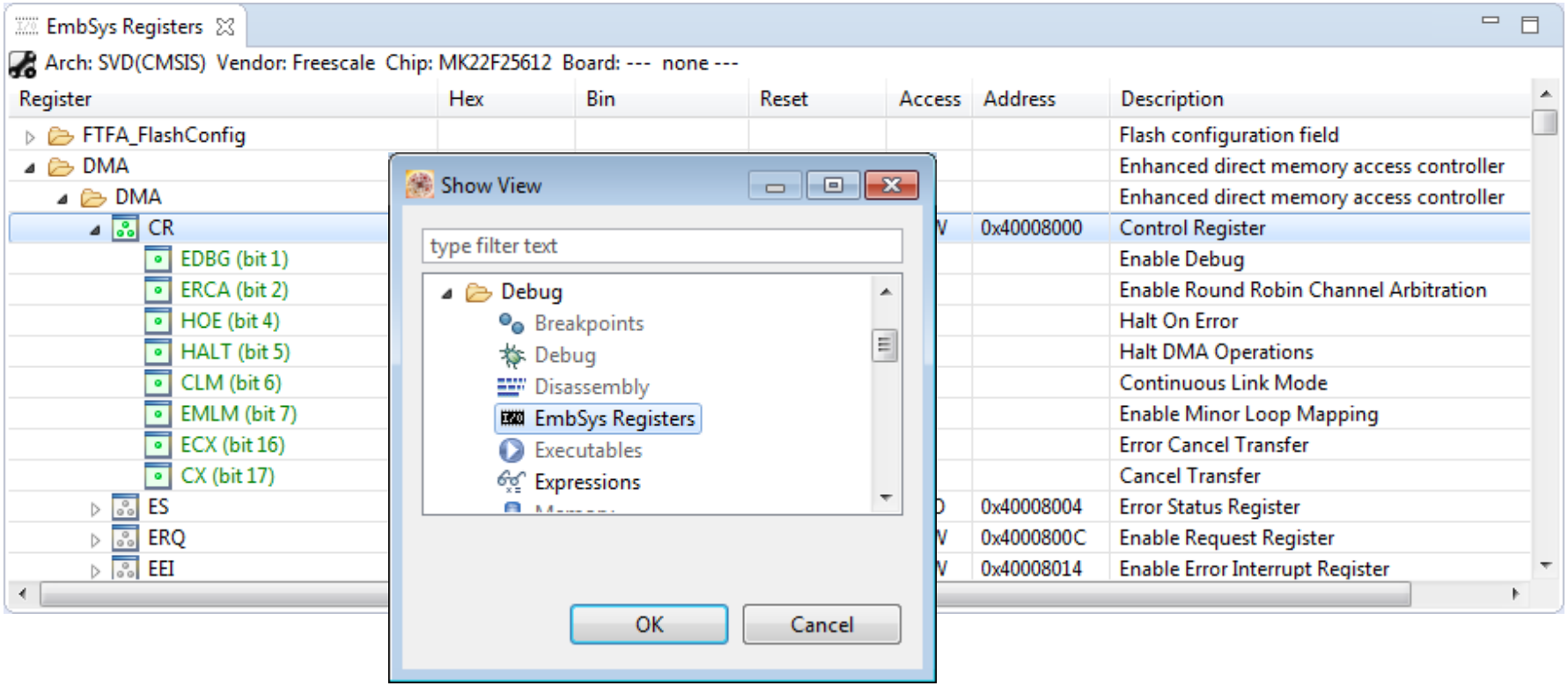
JTAG Multicore Daisy Chaining

- Debug Devices on a JTAG Daisy Chain



Register Detail Viewer

- CMSIS-SVD File support
- Register Bit Level Detail Information
- Includes ARM Core Registers (NVIC, SysTick, …)



Additional Resources

Community
www.freescale.com/community

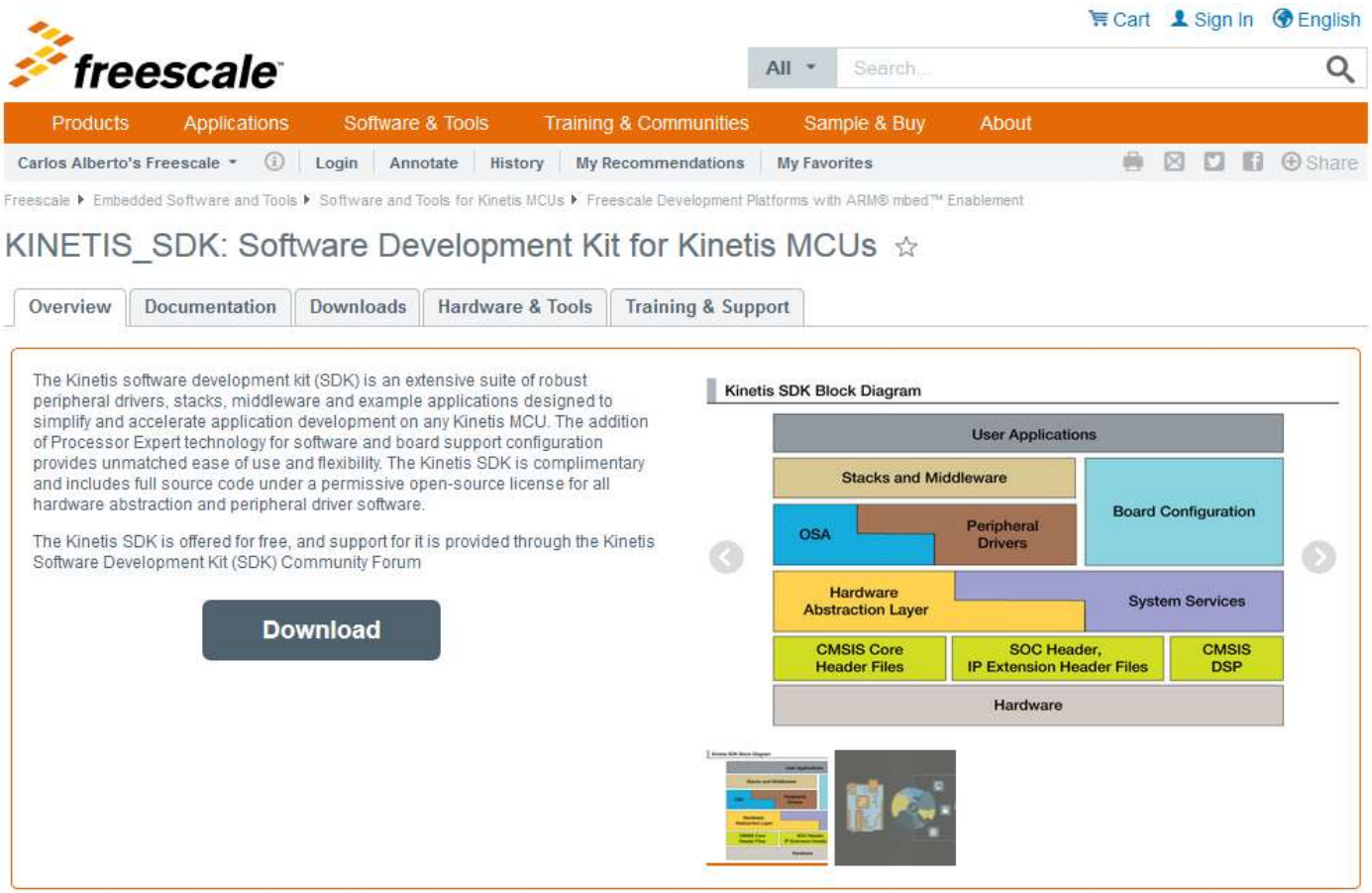
Web
www.freescale.com/kds

Level 2 Support
www.freescale.com/kds/support

KSDK Layout

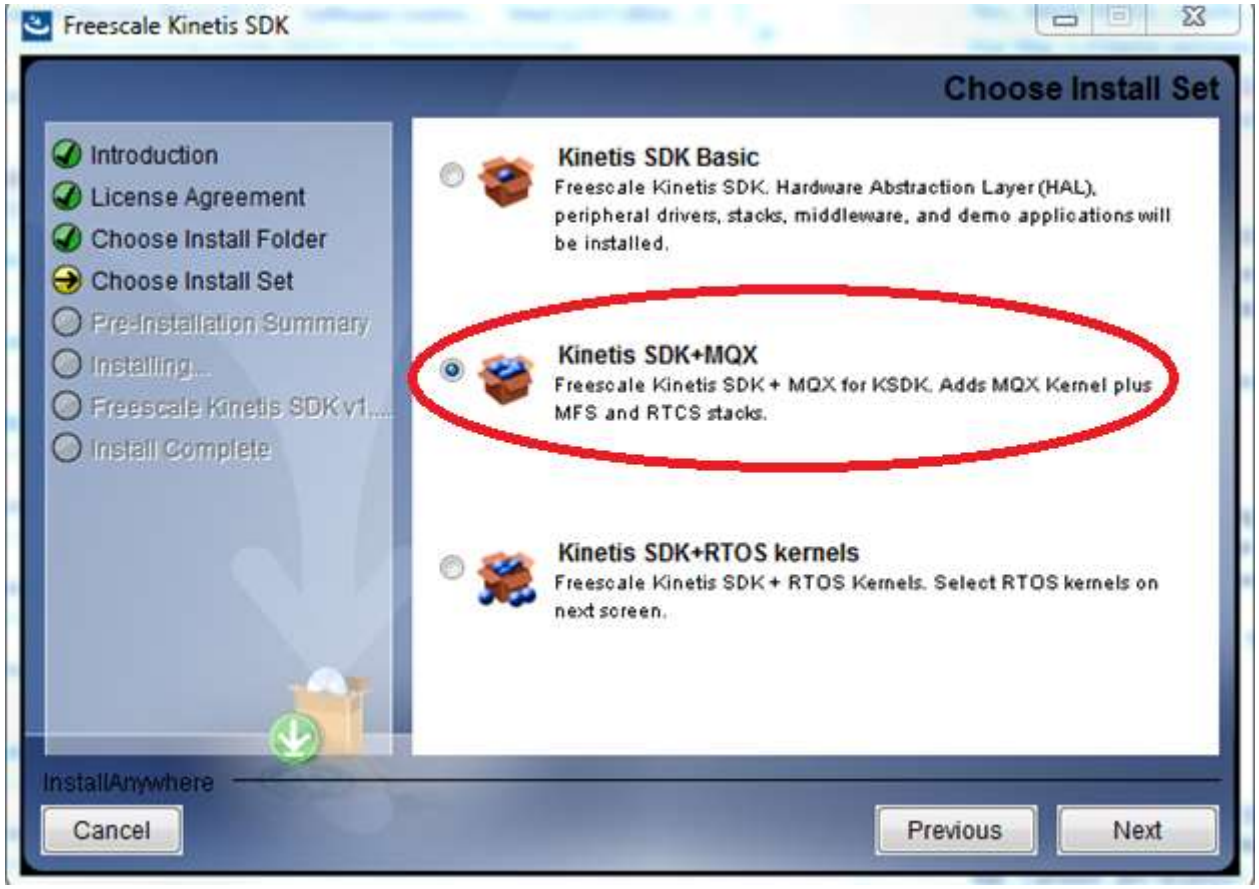
MQX for KSDK 1.1 Installation

- Download Kinetis SDK 1.1 from http://www.freescale.com/ksdk
- Default Install Path: C:\Freescale\KSDK_1.1.0



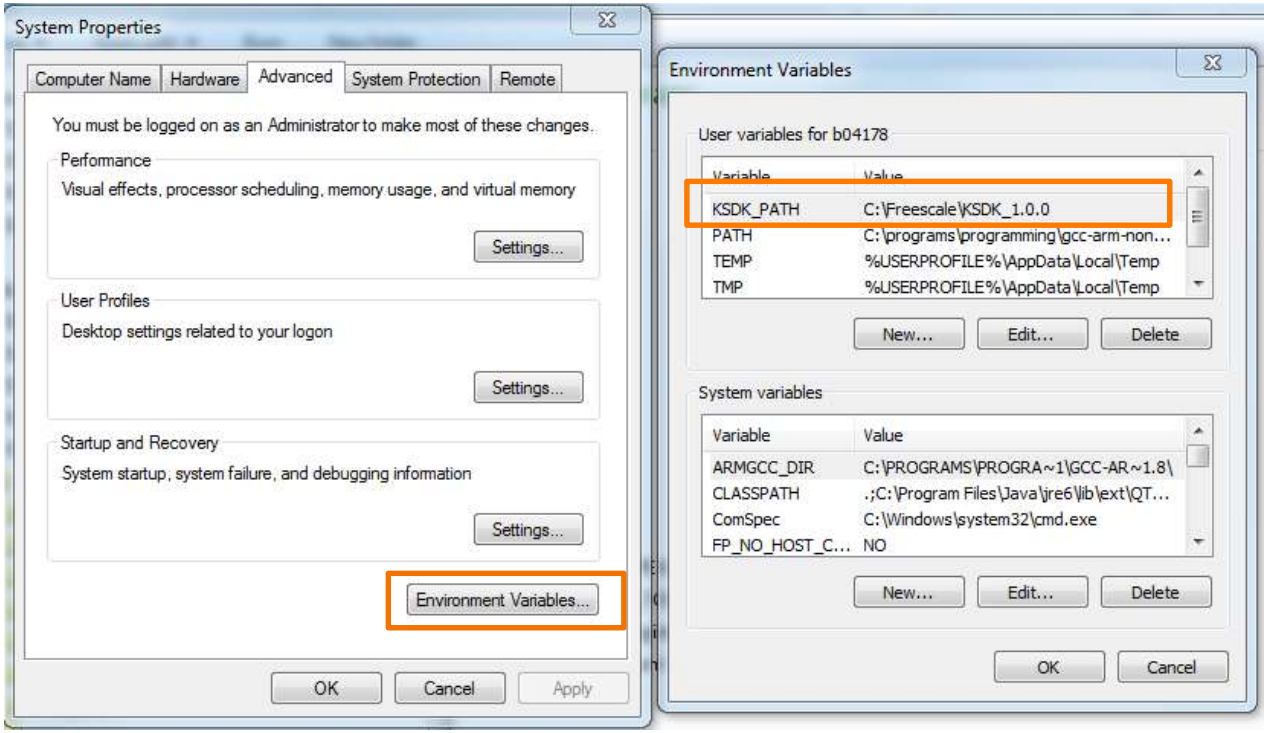
KSDK Installation Package

- During installation, select the RTOSes you're interested in



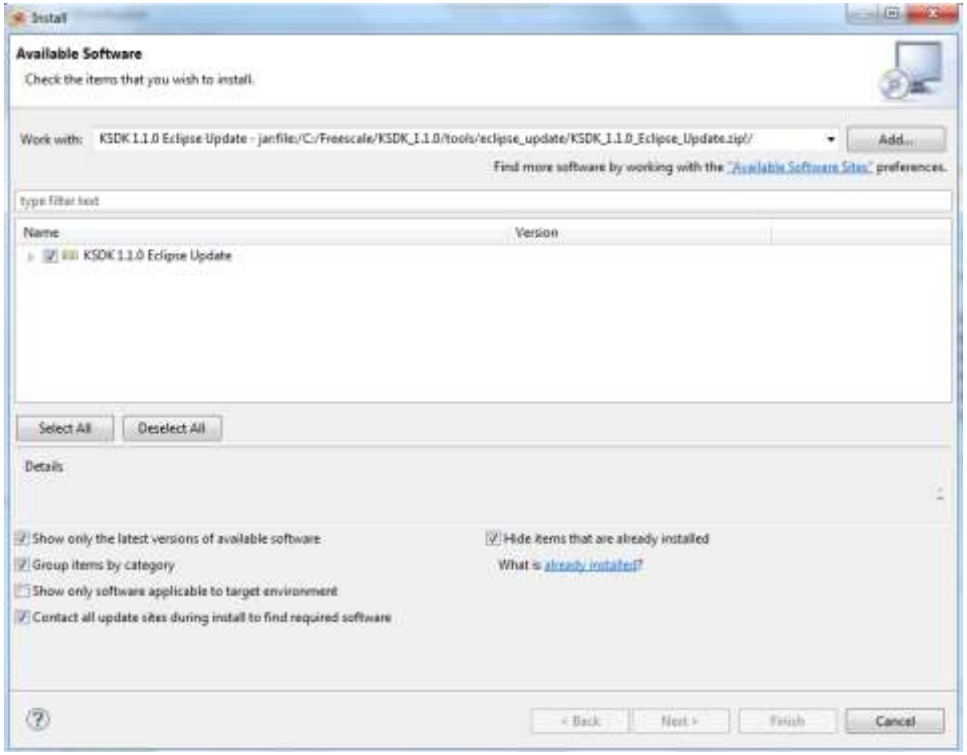
KSDK Environmental Variable

- The KSDK_PATH system variable is used to point to the desired KSDK installation directory in cases were multiple versions of KSDK are installed
- This variable is used by some toolchains
- Control Panel → System Properties → Advanced Tab
  → Environment Variables



KSDK KDS Plugins

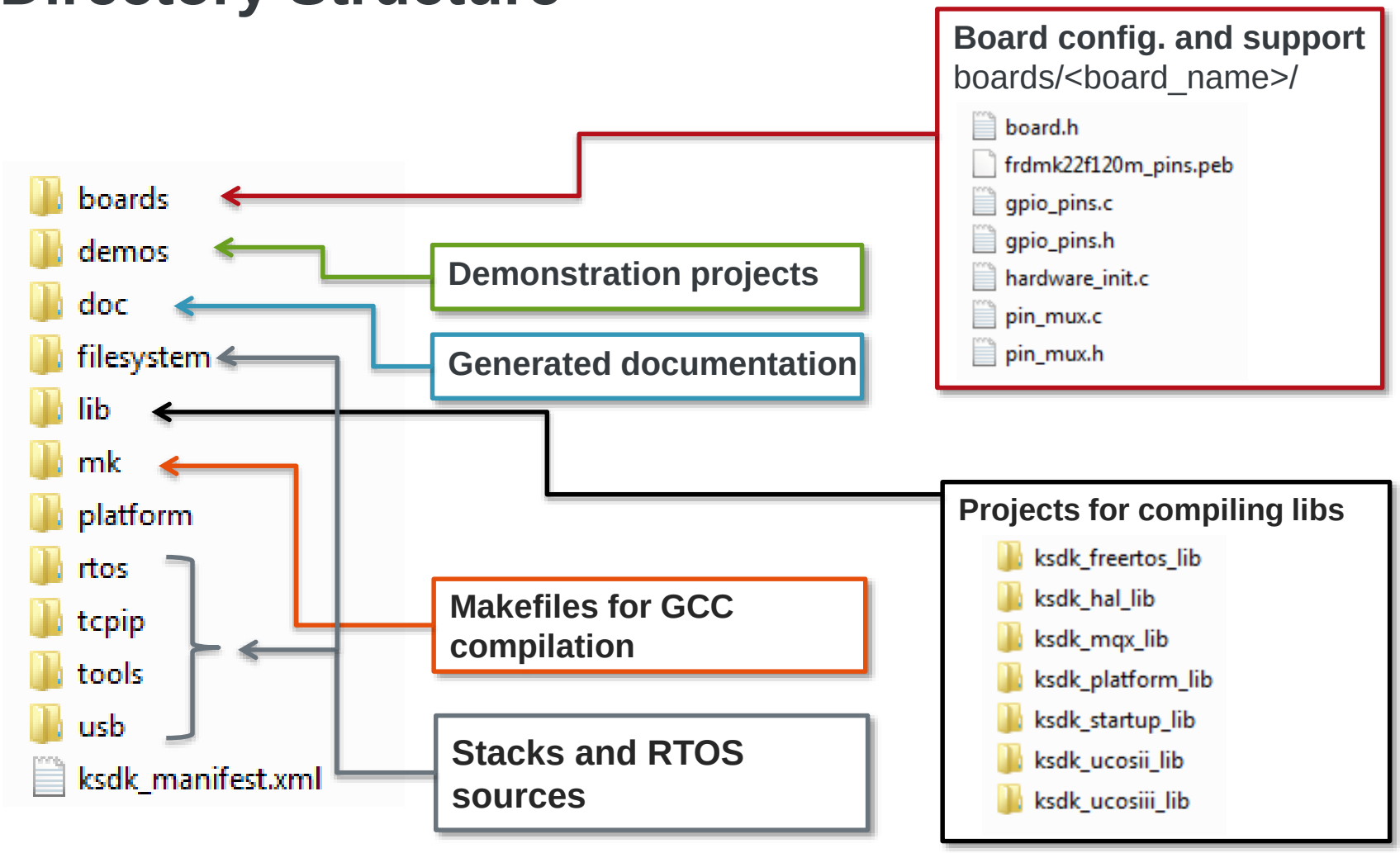
- For Kinetis Design Studio, make sure to install the KSDK Eclipse Update
  - C:\Freescale\KSDK_1.1.0\tools\eclipse_update
- Directions are in the KDS section of the Getting Started with Kinetis SDK (KSDK).pdf



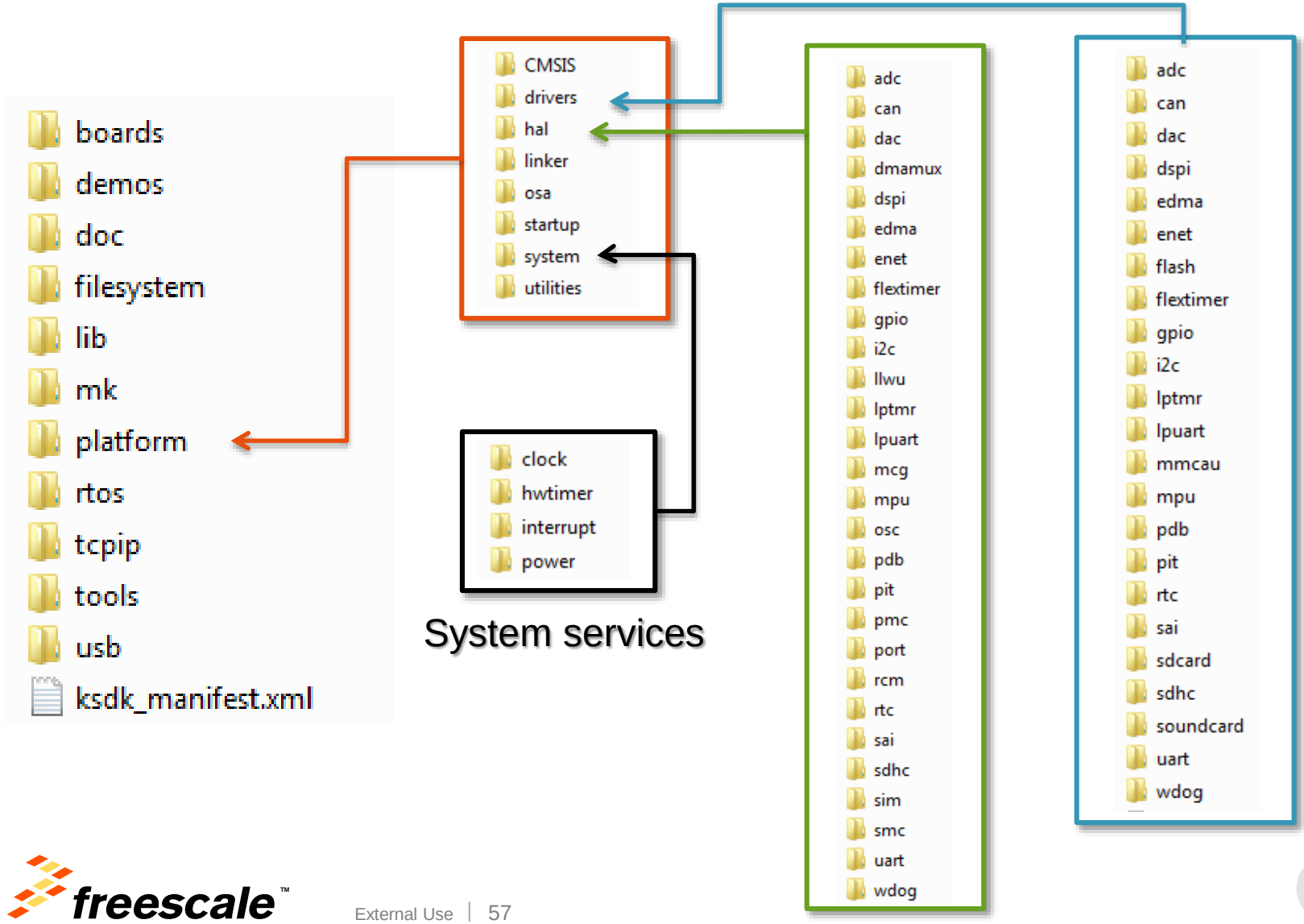
MQX KDS Plugins

- See chapter **5.2 Install Eclipse update** of ***'Getting Started with Kinetis SDK (KSDK)'*** document located in KSDK installation path.
- *C:\Freescale\KSDK_1.1.0\doc\Getting Started with Kinetis SDK (KSDK).pdf*

Directory Structure



Directory Structure (Platform)



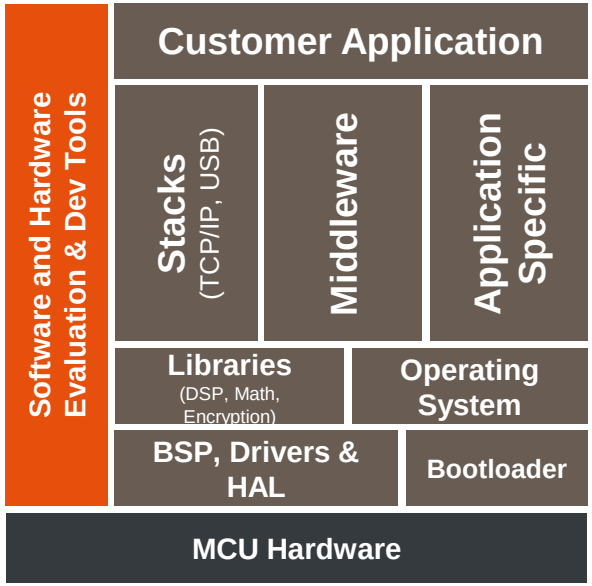
Agenda

- KSDK In-Depth
  - Lab
- KSDK + RTOS
- KSDK + USB
- KSDK + Processor Expert
  - Lab
- Conclusion

Freedom Development Platforms

Low-cost/low-power development hardware

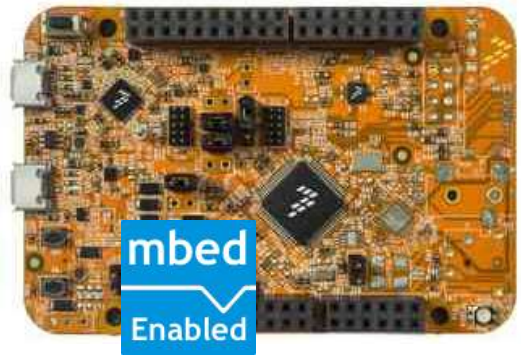
Enables quick application prototyping and demonstration of Kinetis MCU families



Product Features

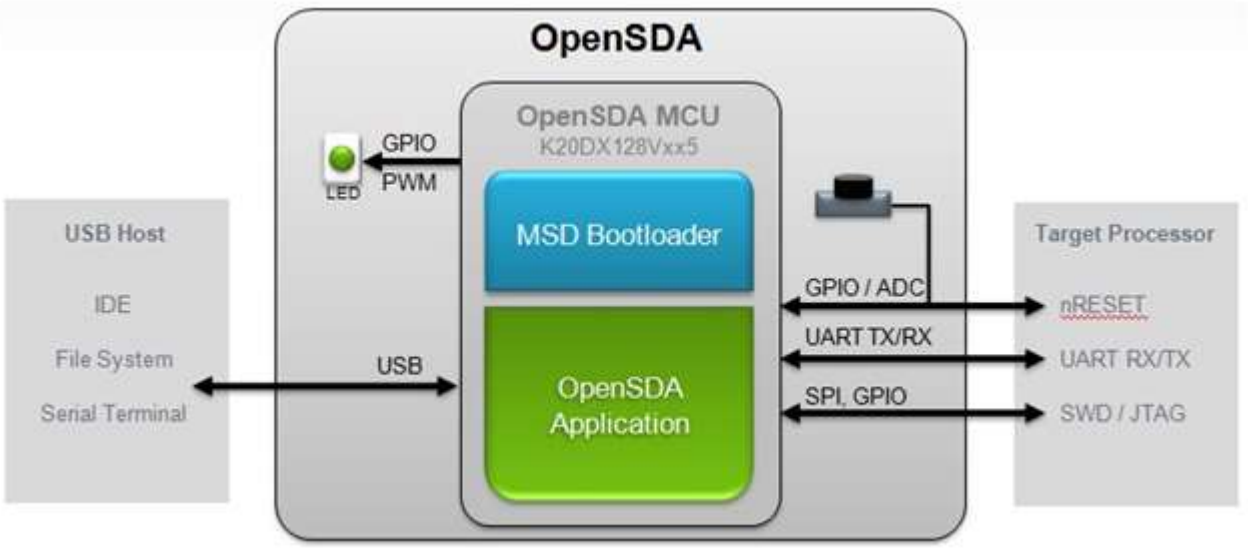
- Low–cost (starting at $12.95 USD)
- Designed in an industry-standard compact form factor (Arduino R3)
- Easy access to the MCU I/O pins
- Integrated open-standard serial and debug interface (OpenSDA)
- Compatible with a rich-set of third-party expansion boards

FRDM-K22F:

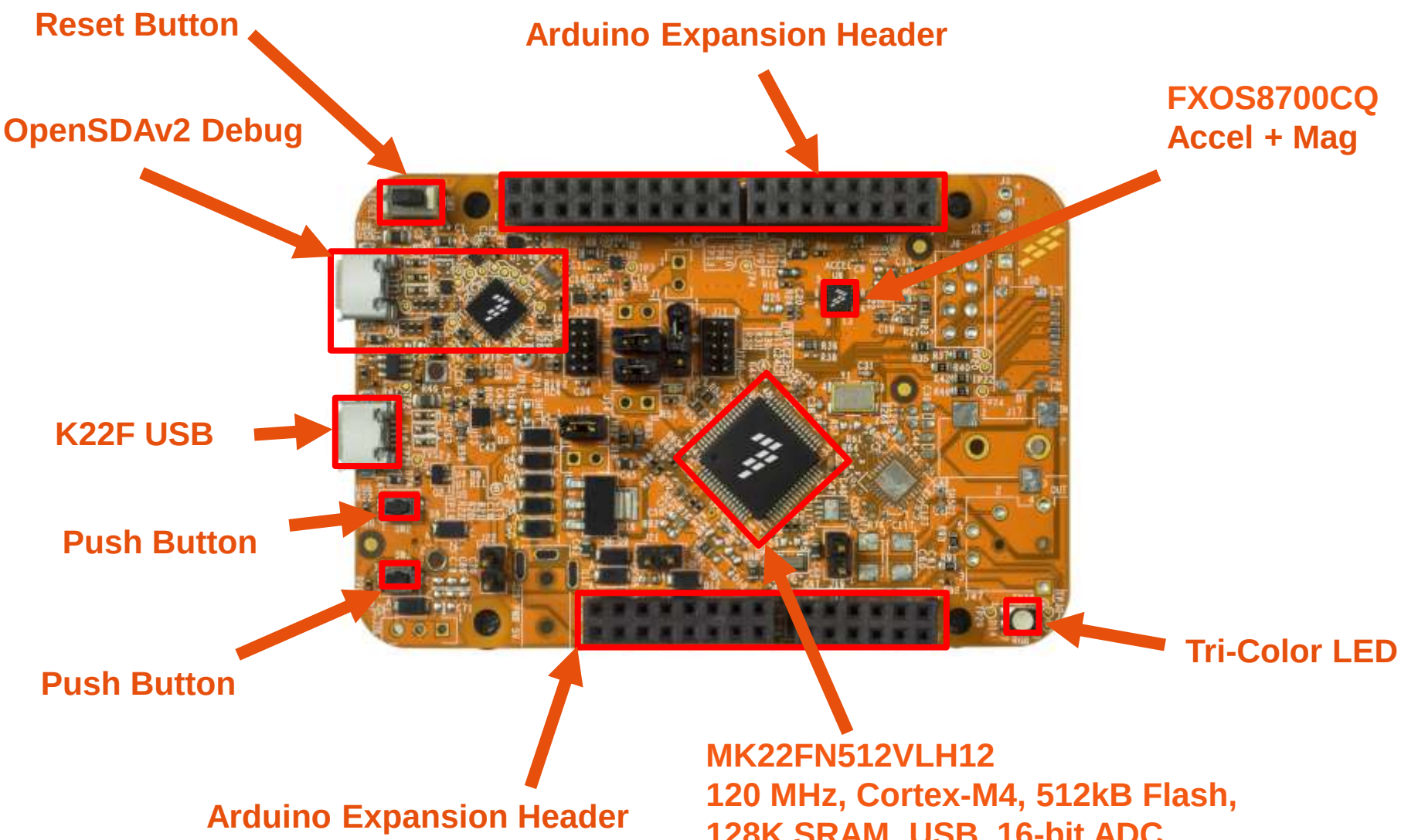


OpenSDA

- OpenSDA is a circuit built into Freescale evaluation boards to provide a bridge between your computer and the embedded target processor
- Purpose is to provide inexpensive debug tool for Freescale evaluation boards
- Different apps can be loaded via a bootloader
- Default CMSIS-DAP app does:
  - Drag-and-drop flashing via a Mass Storage Device
  - Debug via CMSIS-DAP protocol
  - Virtual Serial Port



FRDM-K22F Hardware Overview



Lab 1: Importing KSDK demos

Lab 1 Overview

Objective:

This lab explains how to import and build the demos that are bundled with Kinetis SDK

Lab Flow:

- Importing platform library
- Build Library
- Importing demo project
- Build Demo
- Download and Debug

Required Hardware and Software:

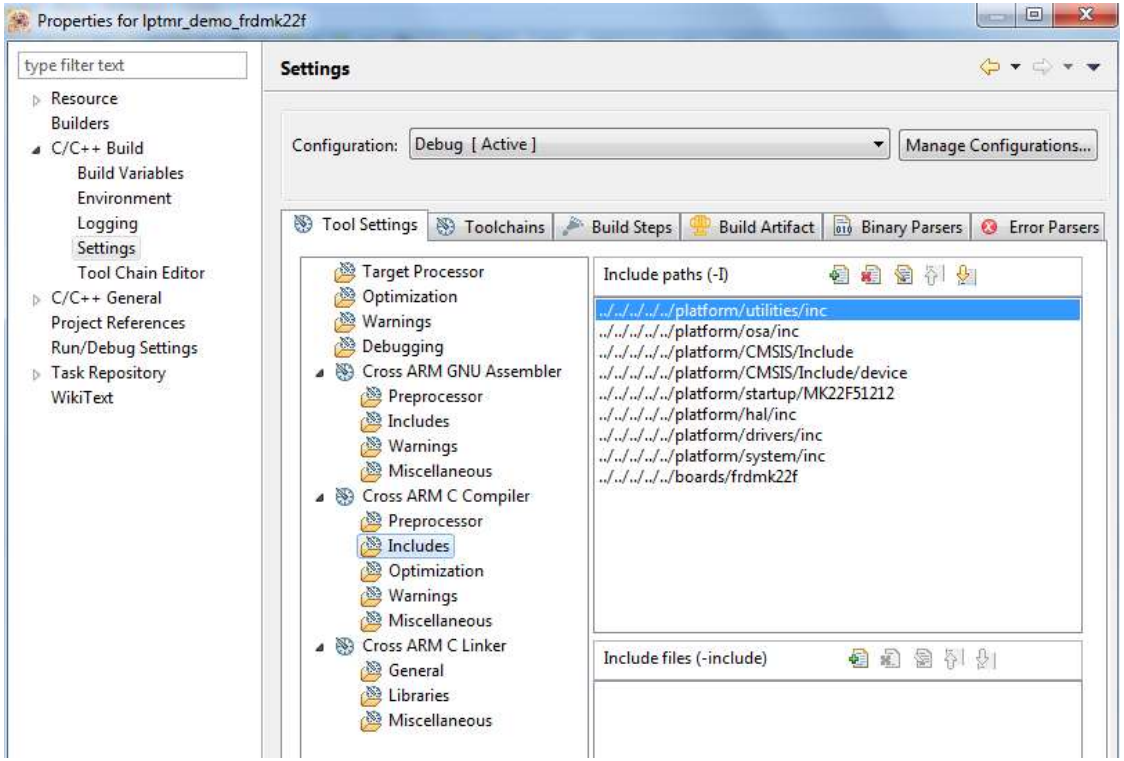
- FRDM-K22F Board configured with CMSIS-DAP Debugger
- Micro USB Cable
- Kinetis Design Studio (v2.0 or newer)
- Kinetis Software Development Kit (v1.1.0)

Lab 1 Summary

- Imported and built KSDK platform library for MK22FN512xxx12.
- Imported and built lptmr_demo from KSDK_1.1.0.
- Ran the demo with KDS.

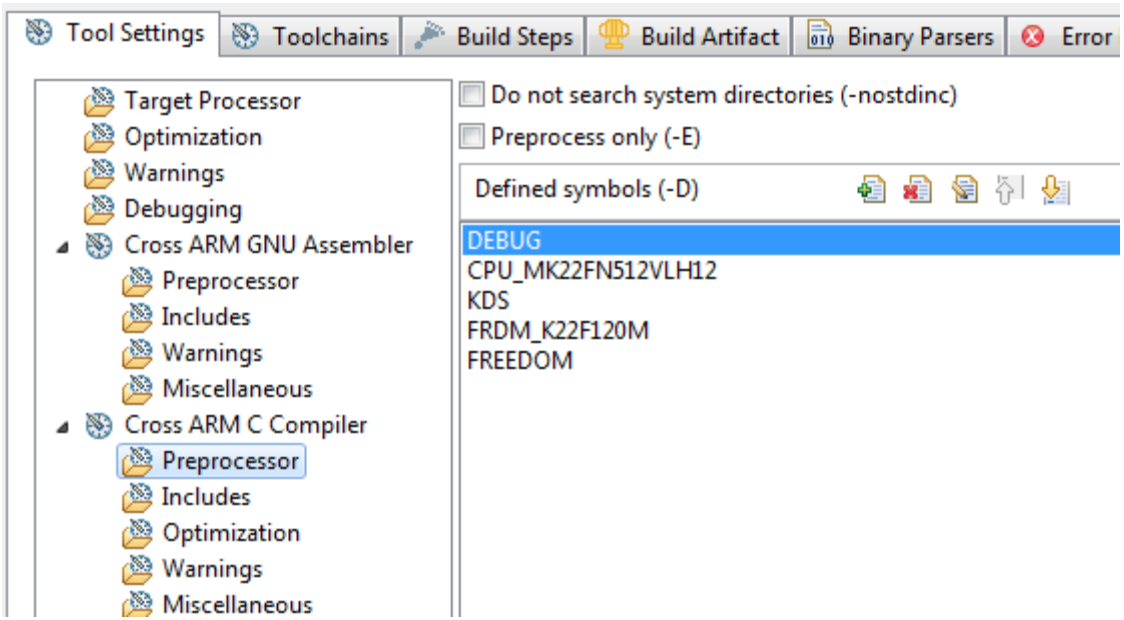
KSDK Project Information

- Right click on lptmr project and select Properties
- Navigate to the C/C++ Build->Settings page
- Look at the Cross ARM C Compiler->Includes screen to see how the KSDK directories are included



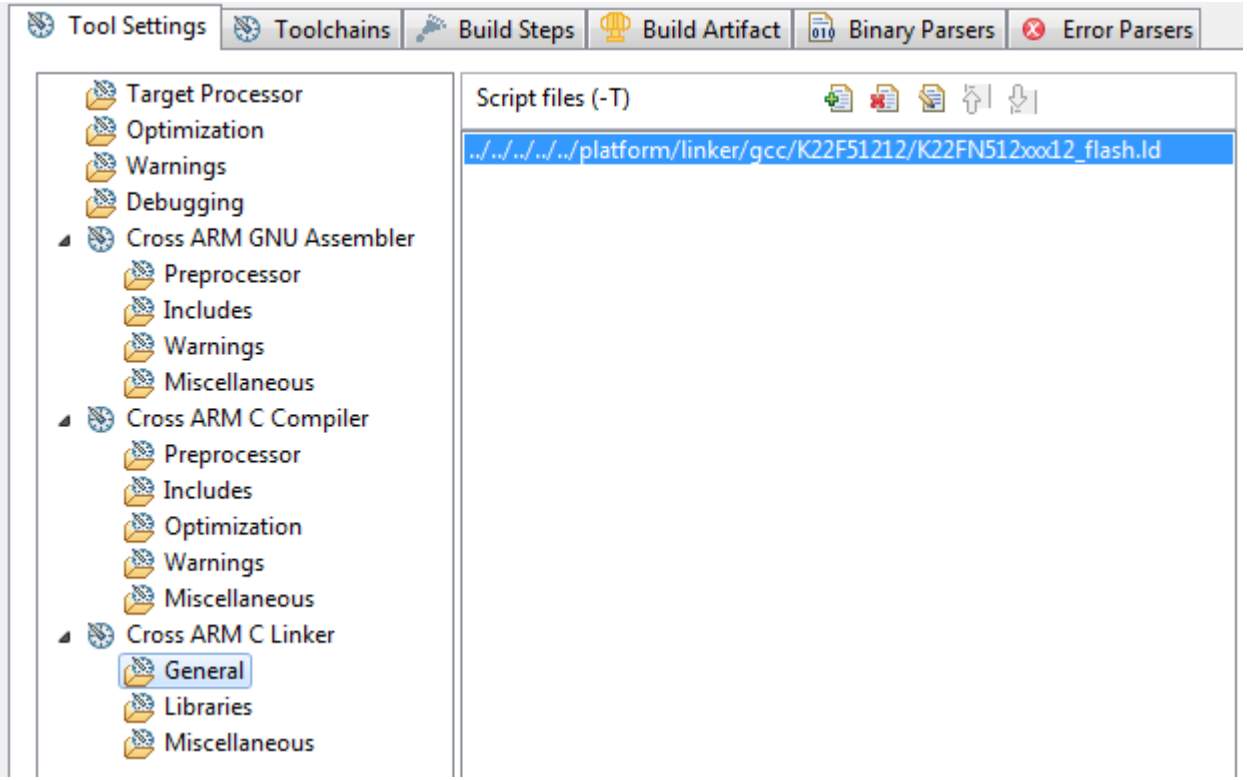
KSDK Project Information Continued

- Look at the Preprocessor screen to see the various KSDK defines



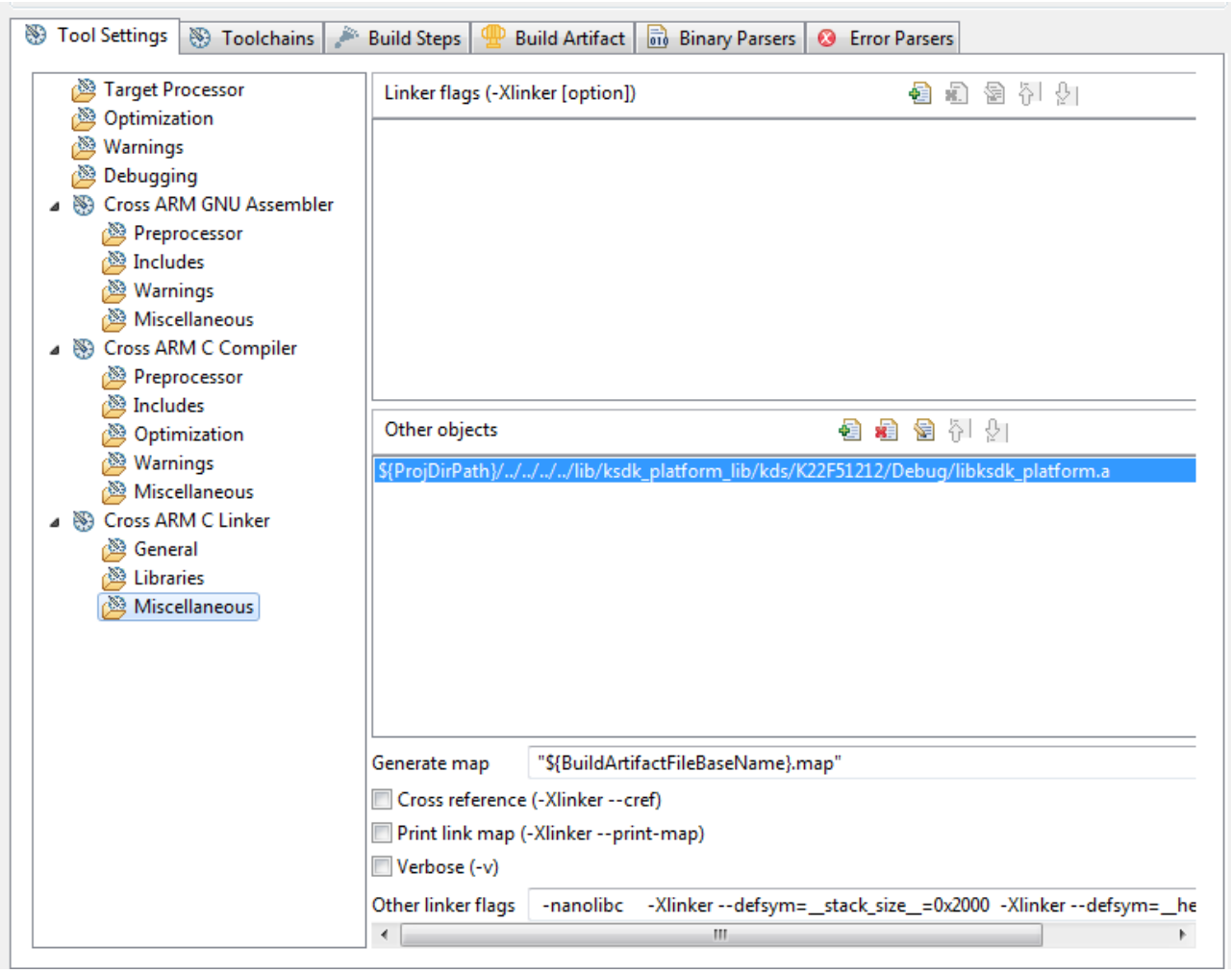
KSDK Project Information Continued

- Linker File



KSDK Project Information Continued

- KSDK Platform Library

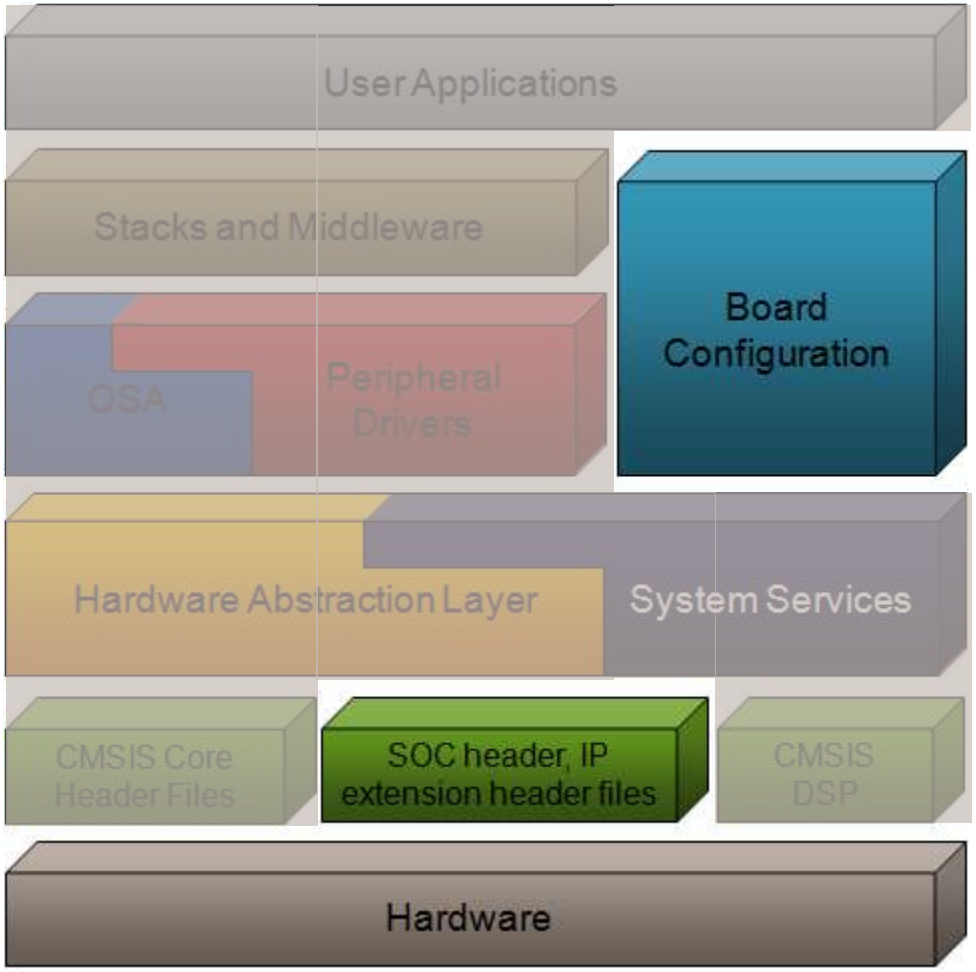


Porting KSDK

Reasons to Port KSDK

- Using custom hardware with differences from Freescale development boards
  - Different peripherals
  - Different pins
- Porting to a different Kinetis derivative
  - Freescale development boards use superset derivatives
  - Custom hardware can use derivative with differences in memory, peripherals, and pins
  - Port Example: from MK64FN1M0VMD12 to MK24FN1M0VLQ12
- Different Clock configurations
  - Internal or external clock sources
  - Different frequencies for core, peripheral bus, and others

Hardware Porting Changes



Minimal Changes Required

- OS, Application, Middleware do not need to change – they reside on top of HAL and peripheral drivers
- HAL and Peripheral drivers do not need to change – they already support different Kinetis derivatives

Kinetis Derivative Differences

- KSDK has derivative information
- Specify derivative when compiling
- KSDK pulls in correct header files

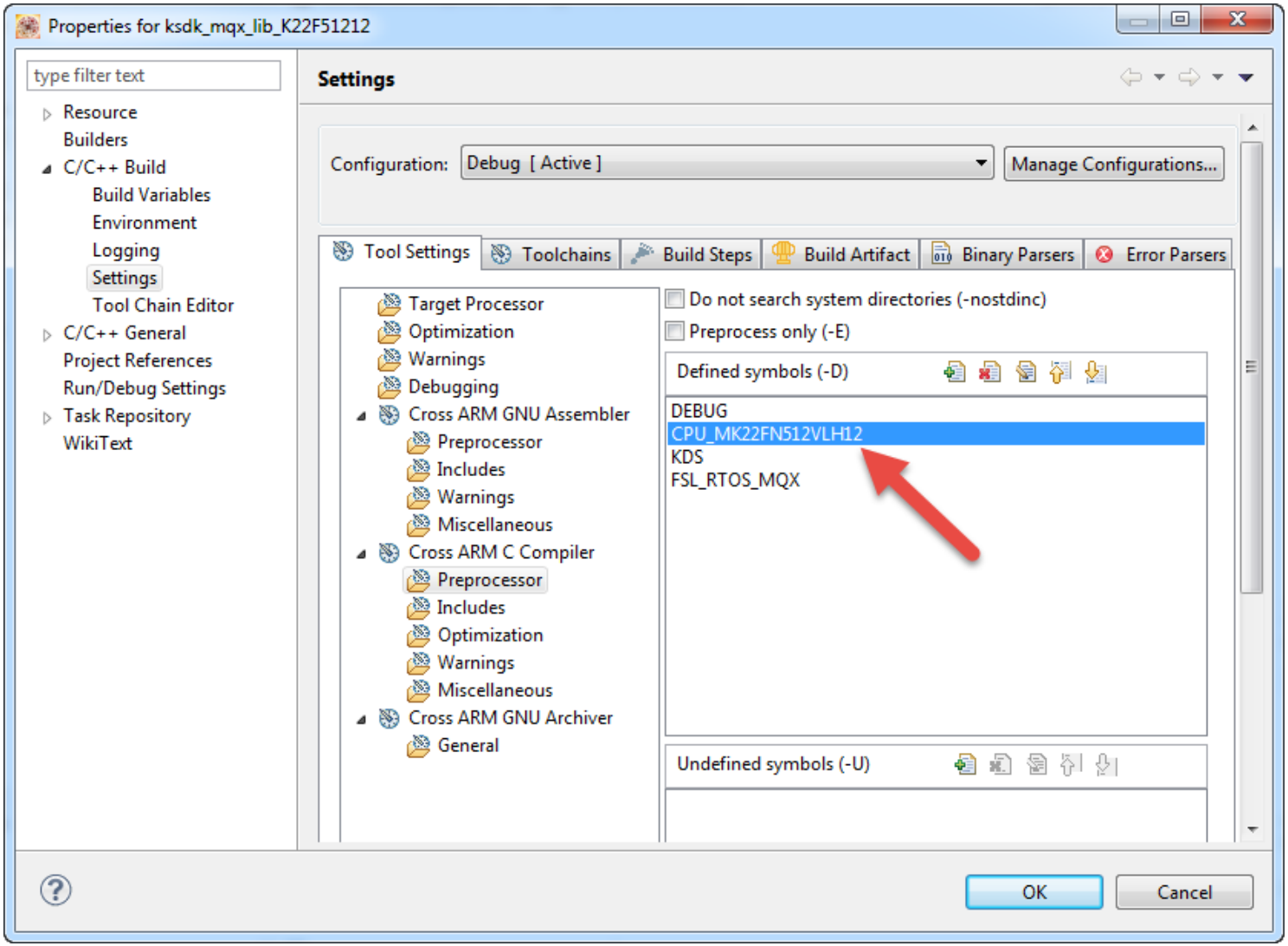
Board Configuration

- Specific to hardware, needs customized
- Pin Muxing
- GPIO Configuration
- Clock startup configuration

Changing Kinetis Derivative

- KSDK makes changing derivative easy
- KSDK already has derivative information in source code
  - Macros used at compile time
  - Specify peripheral differences between Kinetis derivatives like <KSDK_PATH>\platform\hal\adc\fsl_adc_features.h
  - Specify which KSDK header files to include in build like <KSDK_PATH> \platform\CMSIS\Include\device\fsl_device_registers.h
- KSDK uses compiler preprocessor definition to specify derivative.
  - Change in ksdk_mqx_lib project and rebuild.

KDS Example: Derivative Defined in project



Derivative Details

- The symbol to use for derivative based on Kinetis part number, like CPU_MK22FN512VLH12
- Change in the toolchain compiler preprocessor settings for the library project ksdk_platform_lib
- KSDK already includes supported derivatives
  - Can find all derivative options in <KSDK_PATH> \platform\CMSIS\Include\device\fsl_device_registers.h
- Porting to a new family is not supported. Only derivatives.
  - Full list of supported derivatives can be found in the Release Notes.

Porting Board Configuration

- Each development board supported by KSDK has board configuration files
- Found in <KSDK_PATH>/boards
- Contains board-specific details for KSDK
  - Applications easily portable across different boards and devices
- These files should be reviewed and modified for custom hardware:
  - board.h
  - pin_mux.c and pin_mux.h
  - gpio_pins.c and gpio_pins.h
  - Hardware_init.c

board.h file

- Specifies debug UART peripheral and pins
  - For stdin/stdout functions, like printf()
- Mainly used for KSDK examples, specifying:
  - Features available on board, like sensor for demos
  - Peripheral instances for examples, like I2C0
  - Pins for LEDs and buttons
- Custom port can also use for quick test of board using KSDK example projects.

KSDK Porting – Change Default UART

- Modify board.h to select the UART and baud rate to use

```
41  /* The UART to use for debug messages. */
42  #ifndef BOARD_DEBUG_UART_INSTANCE
43      #define BOARD_DEBUG_UART_INSTANCE   1
44      #define BOARD_DEBUG_UART_BASEADDR   UART1_BASE
45  #endif
46  #ifndef BOARD_DEBUG_UART_BAUD
47      #define BOARD_DEBUG_UART_BAUD       115200
48  #endif
```

- Modify pin_mux.c to select the pins to use

```
159  void pin_mux_UART(uint32_t instance)
160  {
161    switch(instance) {
162      case 1:                              /* UART1 BT */
163        /* PORTE_PCR0 */
164        PORT_HAL_SetMuxMode(g_portBaseAddr[4],0u,kPortMuxAlt3);
165        /* PORTE_PCR1 */
166        PORT_HAL_SetMuxMode(g_portBaseAddr[4],1u,kPortMuxAlt3);
167        break;
168      default:
169        break;
170    }
171  }
```

pin_mux.c and pin_mux.h

- Kinetis devices provide great flexibility in muxing signals
  - Each digital port pin has up to 8 signals muxed on pin
  - Some peripherals route same signals to multiple pins
- pin_mux.c:
  - Functions to set pin mux options for all pins used on board
  - Function for each peripheral type, like configure_can_pins()
- Hardware_init.c calls these functions in pin_mux.c during startup

10.3.1 K64 Signal Multiplexing and Pin Assignments

144 LQFP	144 MAP BGA	121 XFBGA	100 LQFP	Pin Name	Default	ALT0	ALT1	ALT2	ALT3	ALT4	ALT5	ALT6	ALT7	EzPort
1	D3	E4	1	PTE0	ADC1_SE4a	ADC1_SE4a	PTE0	SPI1_PCS1	UART1_TX	SDHC0_D1	TRACE_CLKOUT	I2C1_SDA	RTC_CLKOUT	
2	D2	E3	2	PTE1/ LLWU_P0	ADC1_SE5a	ADC1_SE5a	PTE1/ LLWU_P0	SPI1_SOUT	UART1_RX	SDHC0_D0	TRACE_D3	I2C1_SCL	SPI1_SIN	
3	D1	E2	3	PTE2/ LLWU_P1	ADC0_DP2/ ADC1_SE6a	ADC0_DP2/ ADC1_SE6a	PTE2/ LLWU_P1	SPI1_SCK	UART1_CTS_b	SDHC0_DCLK	TRACE_D2			
4	E4	F4	4	PTE3	ADC0_DM2/ ADC1_SE7a	ADC0_DM2/ ADC1_SE7a	PTE3	SPI1_SIN	UART1_RTS_b	SDHC0_CMD	TRACE_D1		SPI1_SOUT	

K64 Sub-Family Reference Manual, Rev. 2, January 2014

gpio_pins.c and gpio_pins.h

- KSDK uses pin configuration structures for each pin
  - Pin configuration structures in gpio_pin.c, configures
    - Input/output
    - Pull-up/pull-down enabled
    - Pin filtering
    - Interrupt enabled/disabled
    - Initial output polarity
    - Slew rate and drive strength setting

- gpio_pins.h declares
  - pin names used in board
  - PORT pin to use, like PTE0

System Startup Files

- KSDK uses startup file per Kinetis derivative
  - Called system_<MCU>.c/h, like system_MK64F12.h
  - Located in <KSDK_PATH>\platform\startup\<MCU>
  - SystemInit() in System_<MCU>.c called at startup
    - Disables watchdog
    - Initializes System Clocks
- System Clock initialization controlled by macros in system_<MCU>.h
  - Generated by Processor Expert, can be edited manually
  - Can easily change clock configuration just using this file

USB Hardware Porting

- USB stacks have hardware-specific file
  - Device stack \usb\usb_core\device\sources\bsp\<Board>\usb_dev_bsp.c
  - Host stack \usb\usb_core\host\sources\bsp\<Board>\usb_host_bsp.c
  - OTG stack \usb\usb_core\otg\sources\bsp\<Board>\usb_otg_bsp.c
- Modify this file if USB clock source or divider need to change

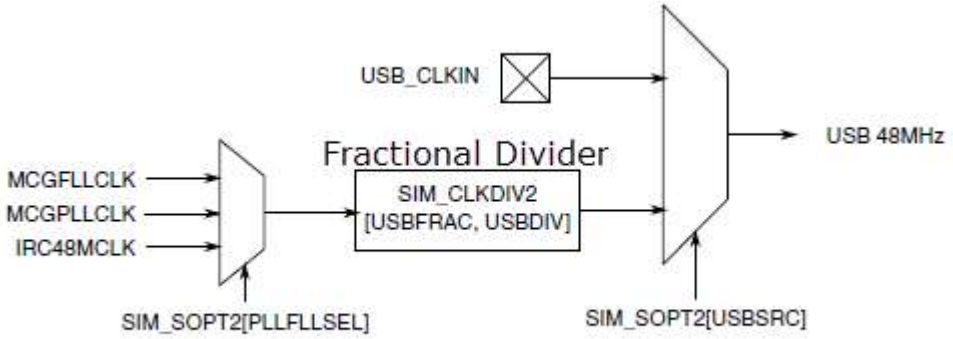


Figure 5-7. USB 48 MHz clock source

KSDK Project Creation

- Two methods for KSDK Project Creation
  - Use Kinetis Design Studio/Processor Expert New Project Wizard
  - Copy existing example project

- KDS/PEx creation covered in next section and online example here: https://community.freescale.com/docs/DOC-102612

- Simple script to copy an example project and give it a new name can be found here: https://community.freescale.com/docs/DOC-102547
- More full-featured project creation application being developed

Agenda

- KSDK In-Depth
  - Lab
- KSDK + RTOS
- KSDK + USB
- KSDK + Processor Expert
  - Lab
- Conclusion

KSDK with RTOS

There are lots of reasons to use an RTOS…..

- Kinetis SDK provides an Operating System Abstraction (OSA) layer to allow RTOS kernels to use KSDK BSP and Drivers

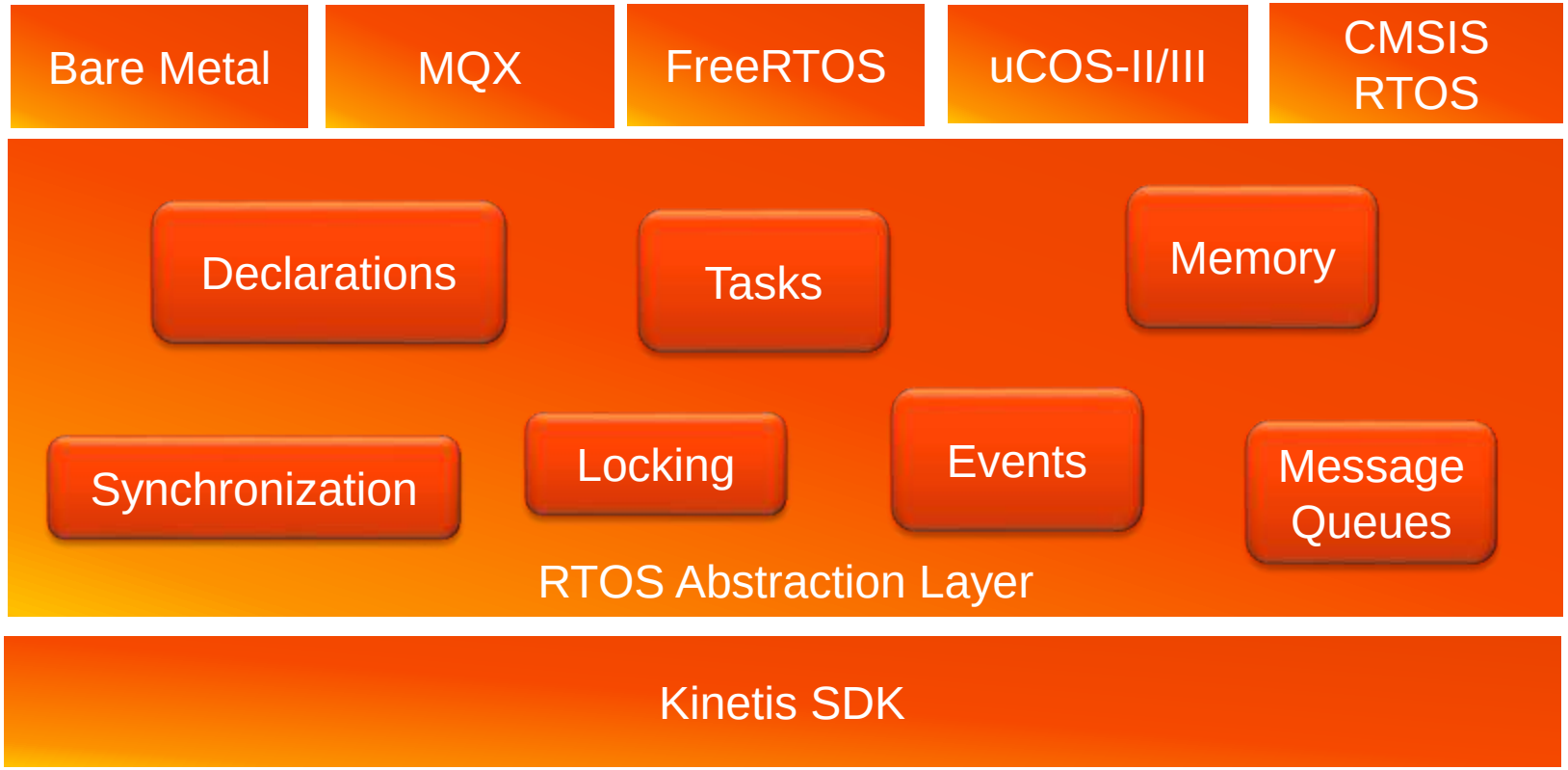
For Embedded Systems that need…

- **Determinism and Low Latency**
  - Systems based on an RTOS verses a super-loop are more stable with lower latency
- **Concurrent Connectivity**
  - Multiple communication interfaces are easier to manage with an RTOS
  - Pre-integrated protocols for TCP/IP, USB, File System, Wi-Fi, etc, enable sophisticated and connected applications
- **Ease of Development**
  - Board Support Packages (BSPs) available with drivers, middleware, and protocols, mean easier and faster development
- **Portability and Scalability**
  - Standard APIs enable high portability of application code across many MCUs
  - Configurable features to scale capabilities to optimize for performance or lower overhead
- **Maintainability and Stability**
  - New features can be added without affecting system timing and higher priority functions

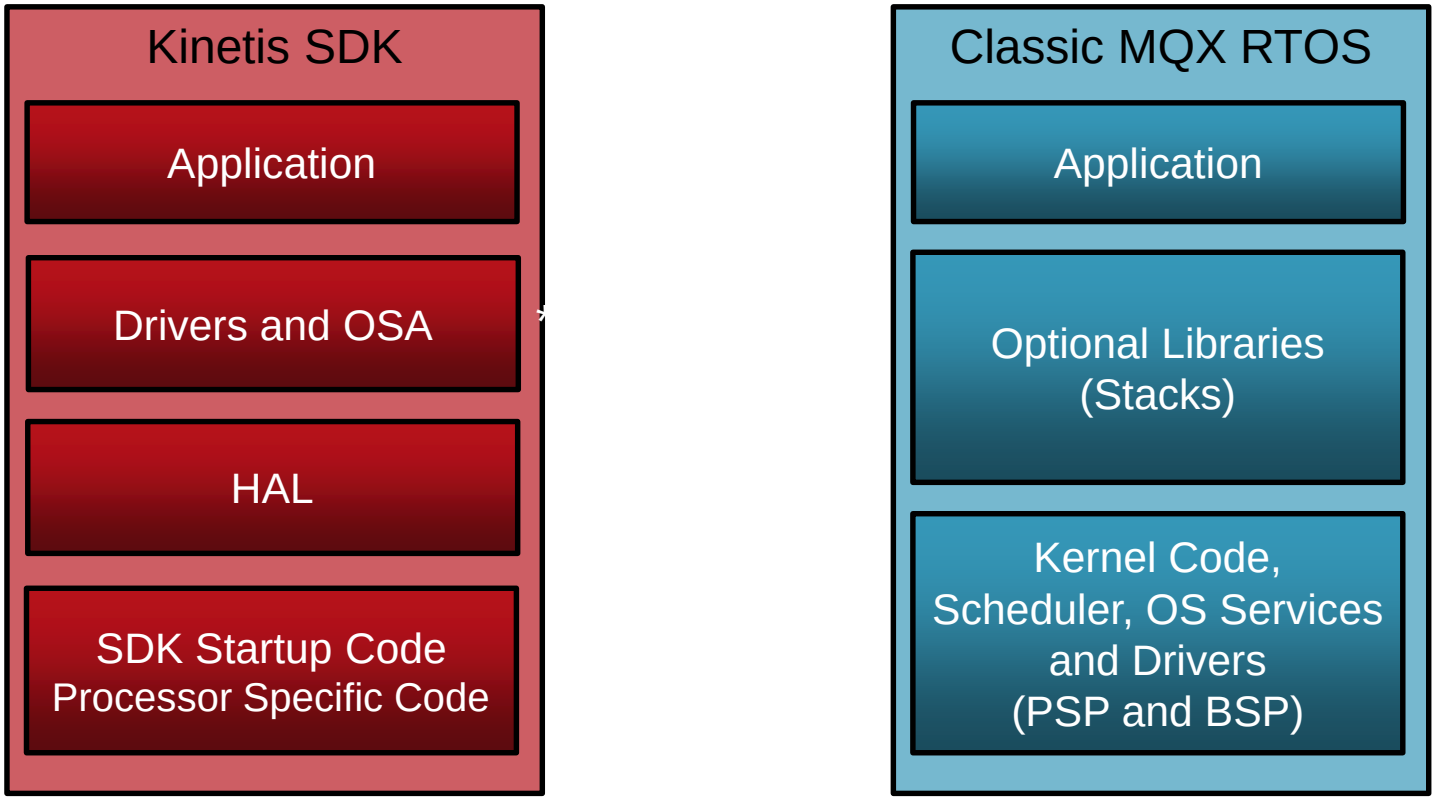
Use an RTOS!

Kinetis SDK RTOS Abstraction

- Common Interface for RTOS/Bare Metal
  - Application
  - Kinetis SDK



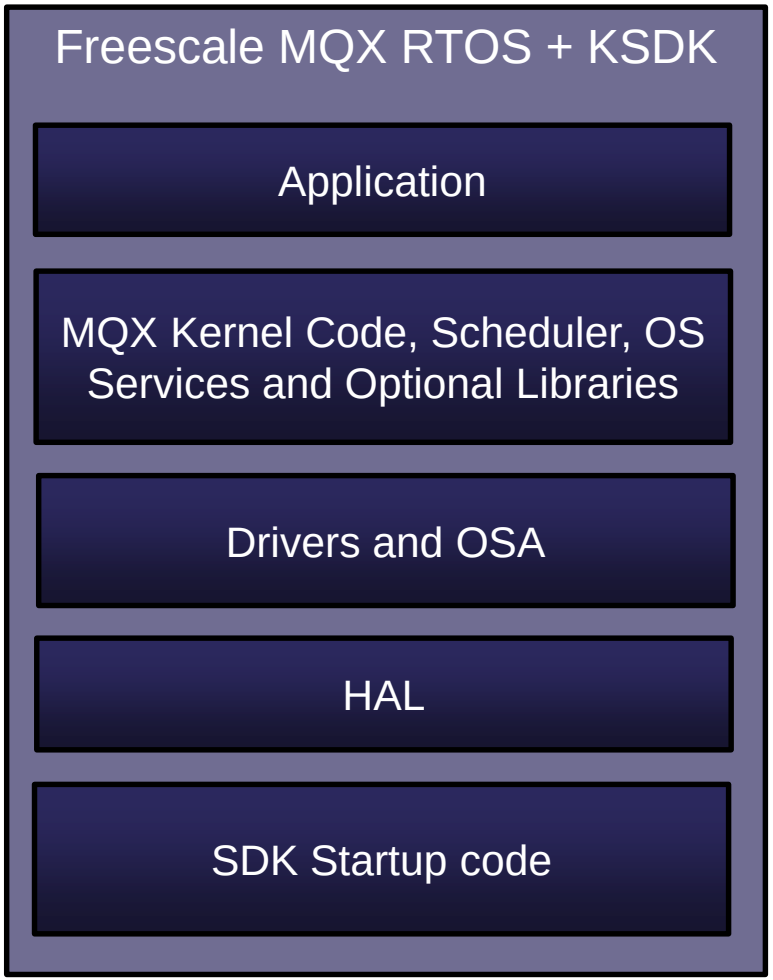
KSDK and RTOS Applications Structure



* Only a few high level drivers provided by MQX RTOS for Kinetis SDK. Applications generally use Kinetis SDK drivers directly.

MQX for Kinetis SDK Application Structure

- A final application project consists of
  - A subset of MQX libraries
    - MQX software scheduler
    - Kernel code
  - KSDK libraries
    - KSDK drivers
    - Hardware Abstraction Layer (HAL)
    - Operating System Abstraction (OSA)



Classic MQX vs MQX for KSDK

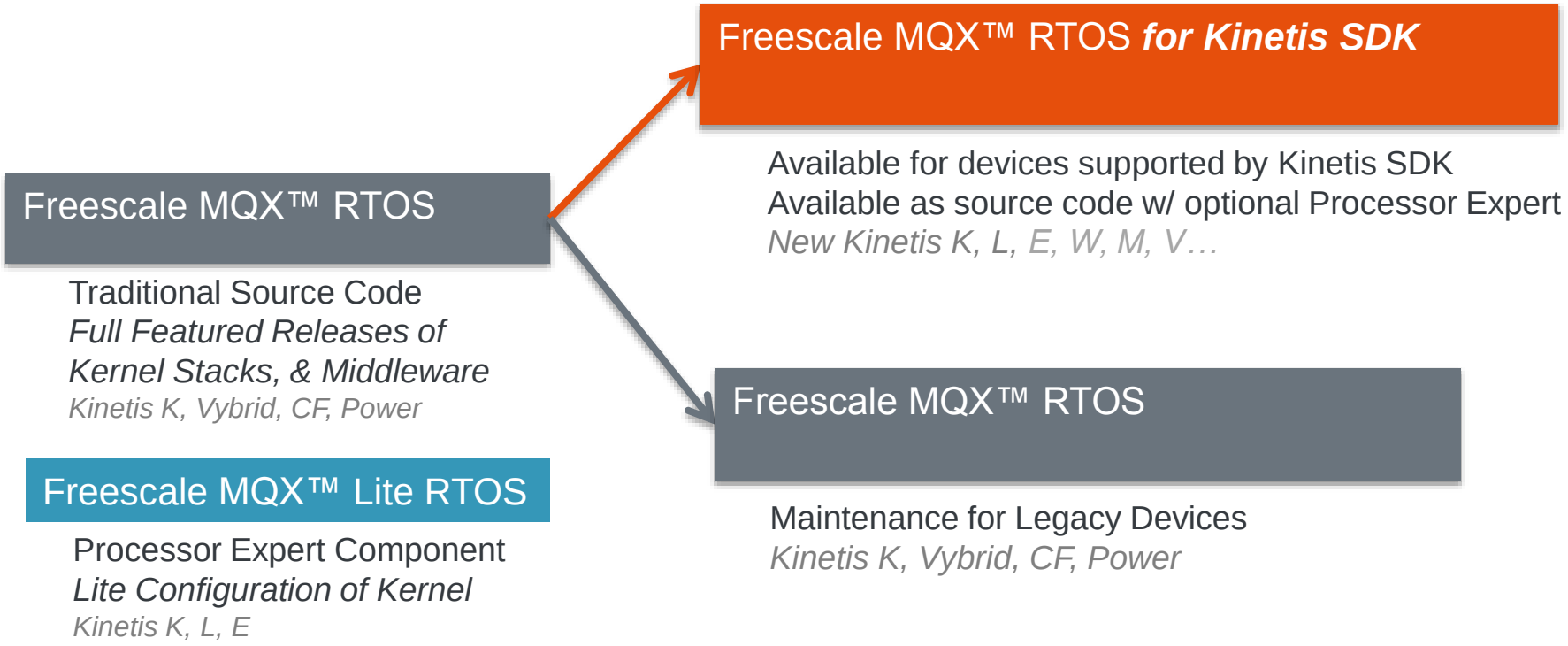
Classic MQX RTOS

- Is a full-featured complimentary Real-Time Operating System
  - Developed by Freescale as a software solution for Freescale devices
  - Provides real-time performance within a small, configurable footprint
- Includes
  - MQX™ Kernel (PSP)
  - Board Support Package (BSP)
  - Implements its own peripheral drivers
  - TCP/IP stack (RTCS)
  - Embedded MS-DOS file system (MFS)
  - USB host/device stack

MQX for KSDK

- Is the latest evolution of the Freescale MQX™ Software Solutions for Kinetis MCUs
- It is built on top of Kinetis SDK
- Leverages the flexible and extendable peripheral drivers found within the KSDK.
- The application developer can use KSDK libraries and device drivers together with Freescale the MQX RTOS core.

Evolution of MQX RTOS



Freescale MQX Version Comparison

	MQX RTOS 4.x *Full-featured, modular and scalable, market proven, widely used*	*MQX Lite RTOS* *Very light MQX kernel for Processor Expert. Easy upward code migration to MQX*	*MQX RTOS for Kinetis SDK* *MQX RTOS in a more flexible and extendible platform for Kinetis MCUs*
Delivery Mechanism	Traditional installer with full source	Processor Expert (PEx) component	Traditional installer with full source
I/O Drivers Included	MQX peripheral drivers; PEx driver optional	PEx drivers	Kinetis SDK HAL & reference drivers
Configurability	User selects needed services from full or lightweight versions	Reduced services only; lightweight options only	User selects needed services from full or lightweight versions
Components	Kernel, TCP/IP stack, USB stack, File System, middleware. Includes own peripheral drivers.	Kernel only. Peripheral drivers provided by PEx.	Kernel, TCP/IP stack, USB stack, File System, middleware. Peripheral drivers provided by Kinetis SDK.
Availability	Select Kinetis K Series, Vybrid, select ColdFire, select Power Architecture	Kinetis L Series, Kinetis K Series, select Kinetis E Series	Kinetis MCUs supported by Kinetis SDK
Cost	Free*	Free*	Free*

** Commercial support and some add-on software packages are extra*

Using KSDK Drivers

- Using KSDK drivers with MQX is the same as using them without an RTOS
- Unlike classic MQX, no driver initialization (beyond pin muxing) occurs during bootup.

- Driver API is in KSDK documentation
  - **C:\Freescale\KSDK_1.1.0\doc\Kinetis SDK API Reference Manual.pdf**

MQX vs KSDK Driver Comparison Example: I2C

- KSDK Drivers are very different than classic MQX Drivers
- Code to initialize I2C and do simple read of accelerometer data

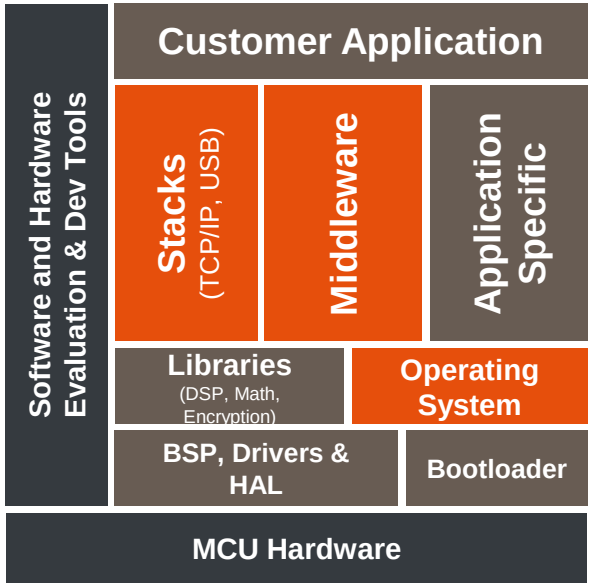
MQX for KSDK	Classic MQX
• I2C_DRV_MasterInit(0, &fxos8700_master); • I2C_DRV_MasterReceiveDataBlocking(0,&slave, &reg, 1,receiveBuff, 1, 200);	• fd = fopen ("i2c1:", NULL); • ioctl (fd, IO_IOCTL_I2C_SET_MASTER_MODE, NULL); • ioctl (fd, IO_IOCTL_I2C_SET_DESTINATION_ADDRESS, &i2c_device_address); • fwrite (&reg, 1, 1, fd); • fflush (fd); • ioctl (fd, IO_IOCTL_I2C_REPEATED_START, NULL); • ioctl (fd, IO_IOCTL_I2C_SET_RX_REQUEST, &n); • fread (&recv_buffer, 1, n, fd); • fflush (fd); • ioctl (fd, IO_IOCTL_I2C_STOP, NULL);

Freescale MQX RTOS for Kinetis SDK

v1.2
April-28

Powerful MQX RTOS, stacks, and middleware built on top of Kinetis SDK

Essential extensions of Kinetis SDK framework for connected and intelligent embedded products

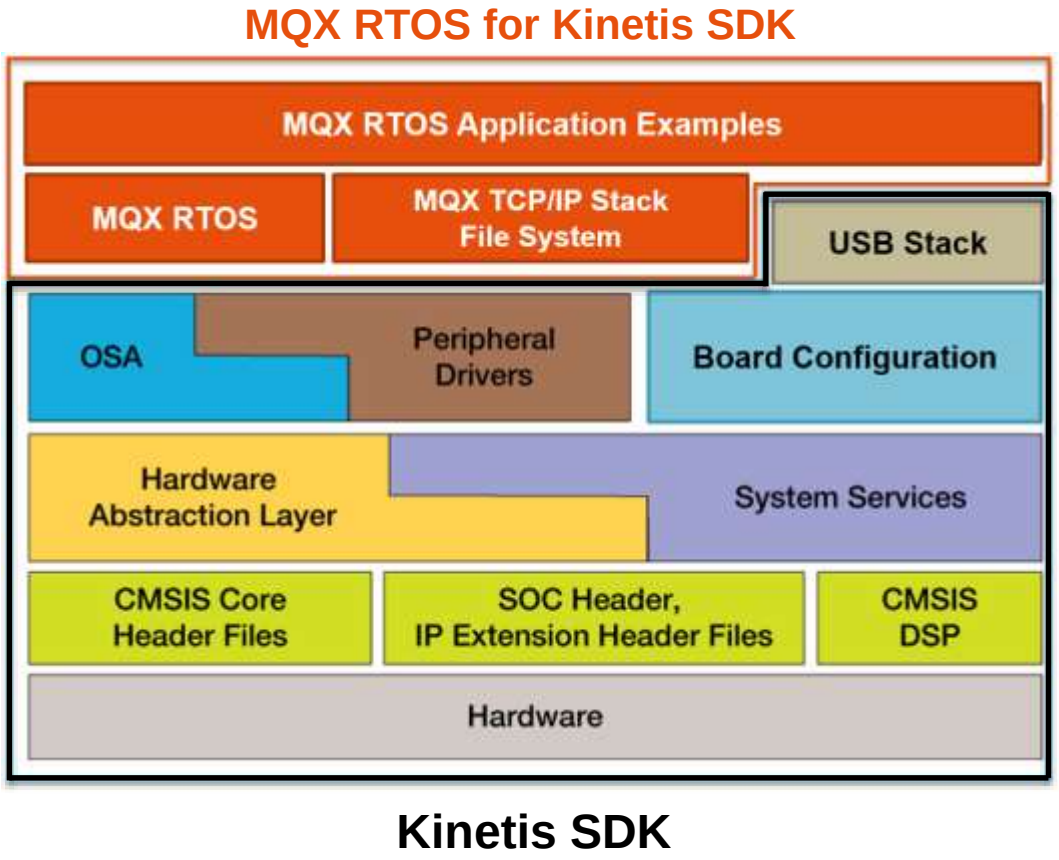


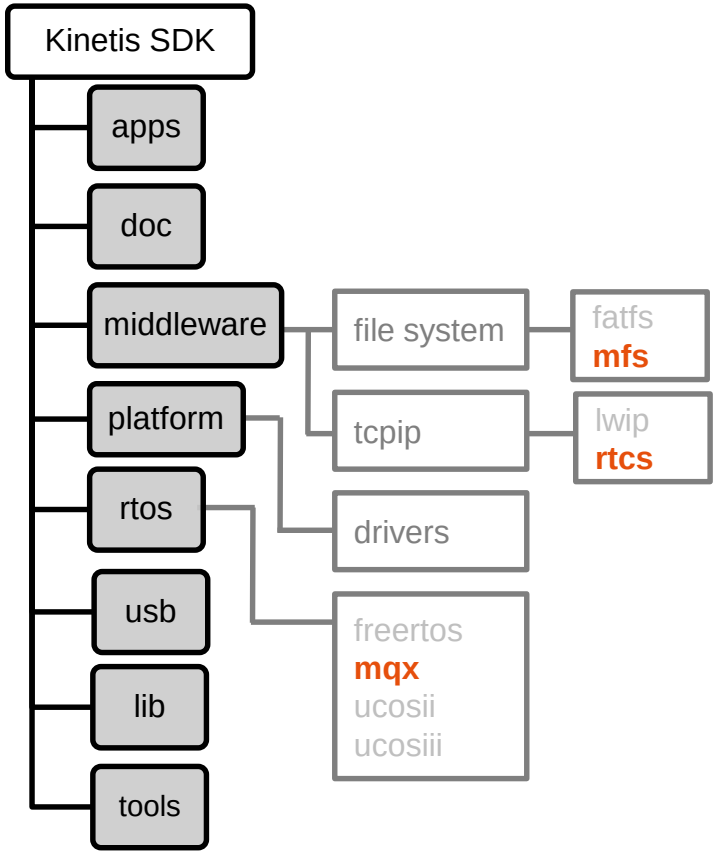
Product Features

- **Now with *MQX Lite* *Configuration***
- All the components of MQX Software Solutions available pre-integrated and tested with Kinetis Software Development Kit (SDK)
  - **MQX RTOS**
  - **MQX Real Time Comm. Suite (TCP/IP)**
  - **MQX File System**
  - **MQX USB Host/Device Stack**
- Leverages **Kinetis SDK *peripheral drivers***
- Builds on common software framework for Kinetis MCUs to enhance flexibility and extendibility

As of Kinetis SDK v1.1, MQX RTOS, stacks, and middleware are available in the Kinetis SDK

Kinetis SDK Block Diagram





MQX RTOS for Kinetis SDK 1.2 Supported Devices

v1.2
April-28

- ***Complimentary* BSPs** covering devices supported by the *Kinetis Software Development Kit (SDK)*

MQX RTOS for KSDK 1.2

KINETIS	
FRDM-KL46Z	✓
FRDM-K22F	✓
FRDM-K64F	✓
FRDM-KL25Z *NEW*	✓
FRDM-KL26Z *NEW*	✓
FRDM-KL27Z *NEW*	✓
FRDM-KL43Z *NEW*	✓
FRDM-KL46Z	✓
FRDM-KW24 *NEW*	✓
MRB-KW01 *NEW*	✓
TWR-K21D50M *NEW*	✓
TWR-K21F120M *NEW*	✓
TWR-K22F120M	✓
TWR-K24F120M	✓
TWR-K60D100M	✓
TWR-K64F120M	✓
TWR-K65F180M *NEW*	✓
TWR-KL43Z48M *NEW*	✓
TWR-KV10Z32 *NEW*	✓
TWR-KV31F120M	✓
TWR-KV46F150M	✓
TWR-KW24D512 *NEW*	✓
USB-KW24D512*NEW*	✓

Agenda

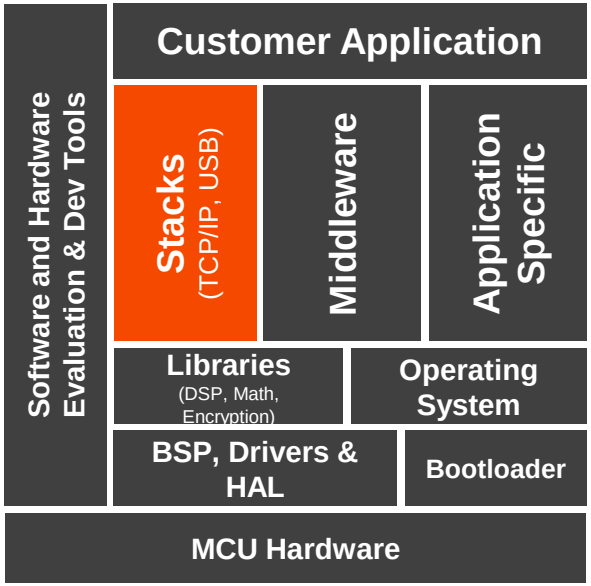
- KSDK In-Depth
  - Lab
- KSDK + RTOS
- KSDK + USB
- KSDK + Processor Expert
  - Lab
- Conclusion

Kinetis Unified USB Stack

Freescale USB Stack

Enable USB applications with Freescale Devices.

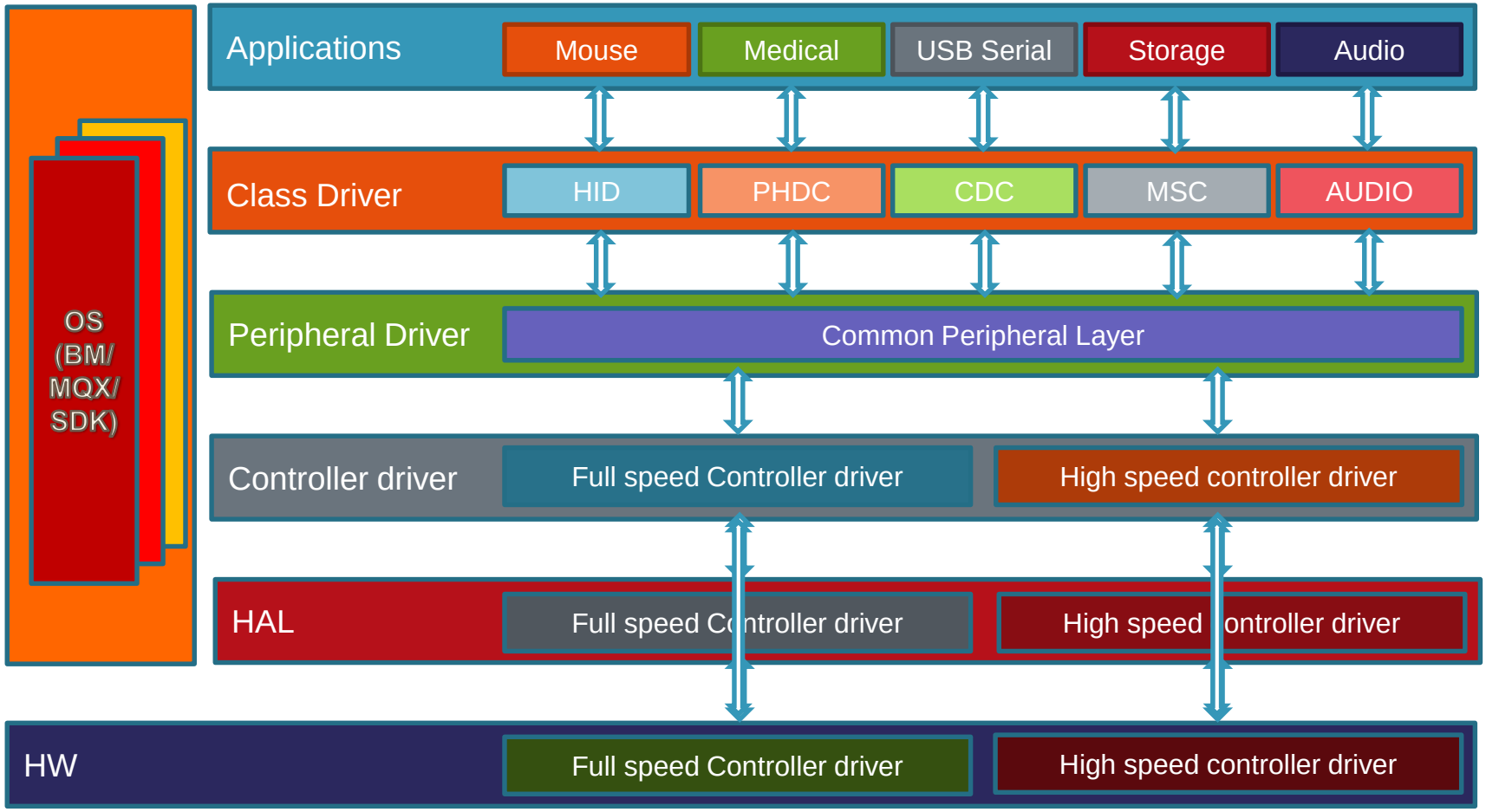
Different USB host and device classes, both bare metal, RTOS and integrated with Kinetis SDK.



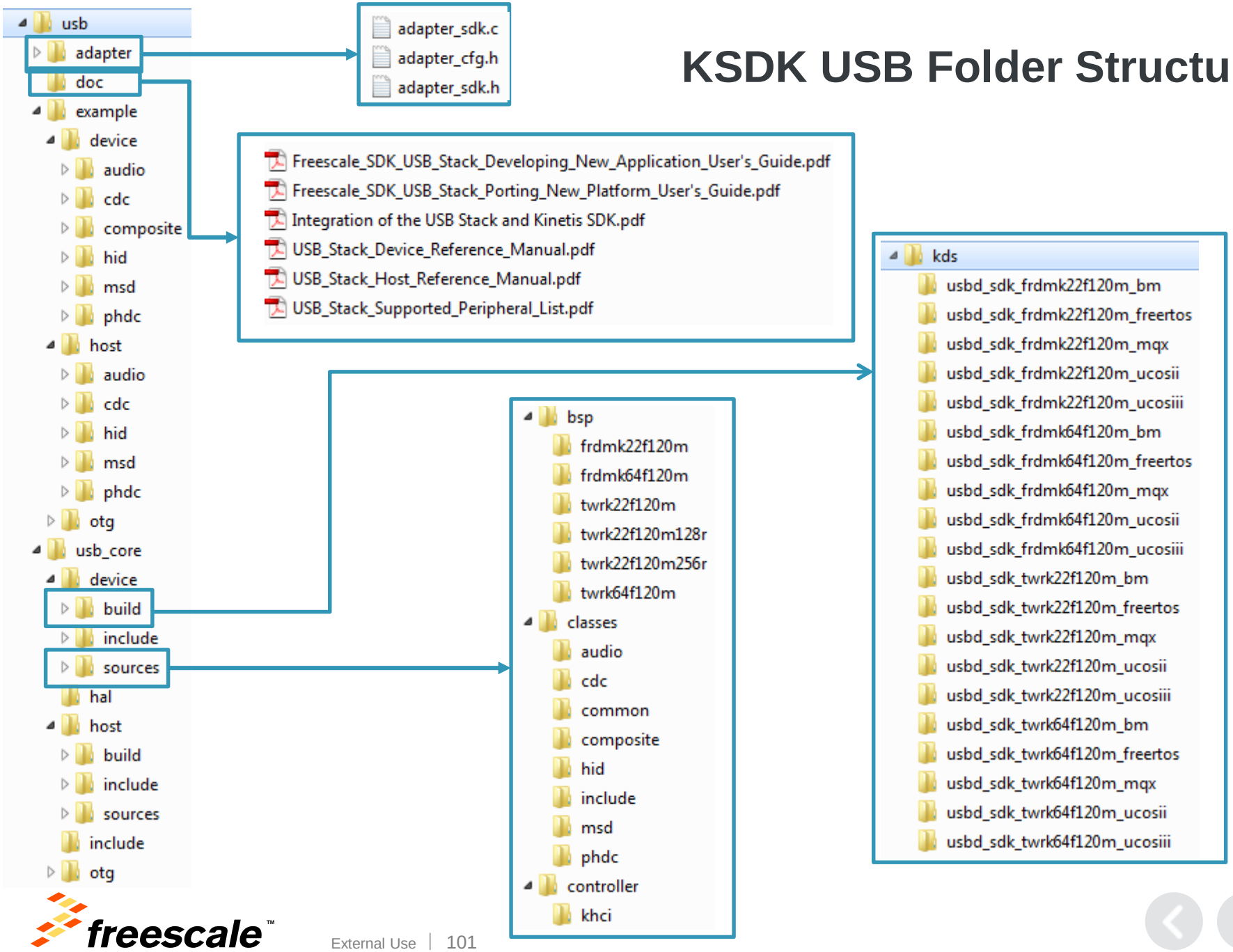
Product Features

- USB stack with all sources provided
- Low footprint: down to 7 KBytes Flash and 2.5 KBytes RAM
- Integrated with Kinetis SDK and MQX 4.2
- Device classes
  - HID, CDC, PHDC, MSC, AUDIO
- Host classes
  - HID, CDC, PHDC, MSC, AUDIO
- USB OTG
  - HNP, SRP
- New 'unified' stack combines MQX and Bare Metal stack
- Support for IAR, Keil, Kinetis Design Studio, and GNU/GCC tool chains.

Architecture



KSDK USB Folder Structure



USB Examples

- The USB examples that come with KSDK require 2 libraries to be built first:
  - KSDK Platform Library
  - USB Host or Device Library (depending on if example is host or device)

- As an example, to run the Device HID Mouse example on FRDM-K22F with KDS would need to import and compile:
  - <ksdk_dir>\lib\ksdk_platform_lib\kds\K22F51212
  - <ksdk_dir>\usb\usb_core\device\build\kds\usbd_sdk_frdmk22f120m_bm
  - <ksdk_dir>\usb\example\device\hid\hid_mouse\sdk\kds\dev_hid_mouse_frdmk22f120m_bm

Agenda

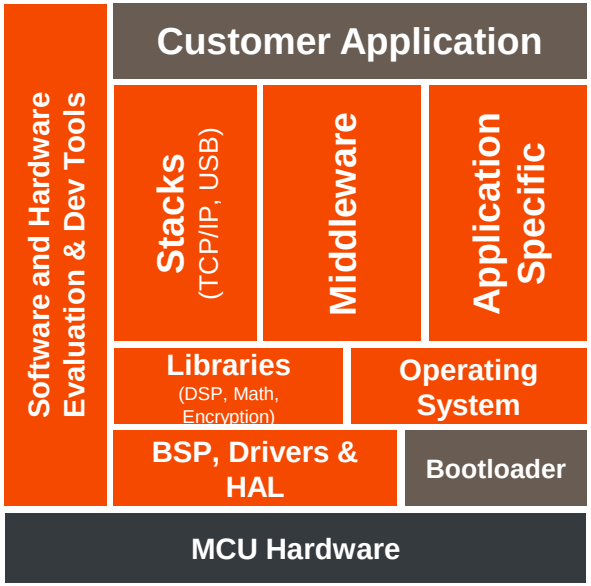
- KSDK In-Depth
  - Lab
- KSDK + RTOS
- KSDK + USB
- KSDK + Processor Expert
  - Lab
- Conclusion

Processor Expert + KSDK

Freescale Processor Expert Software

Create, configure, generate software and drivers for Freescale microcontrollers.

Master complex peripherals with a few mouse clicks, without the need to read thousands of data sheet pages.



Product Features

- Standalone and integrated for
  - Eclipse based IDE's (KDS, Atollic)
  - Freescale CodeWarrior
  - IAR Embedded Workbench
  - Keil MDK
- Easy configuration of Kinetis SDK with Processor Expert Components
- Supports Kinetis, Vybrid, S08, S12, S12Z, ColdFire, DSC and Power Architecture™ processprs with reusable software components
- Knowledge base of pins, registers, muxing, clocks and dependencies
- Initialization and driver code generation with design time consistency checking
- Bare Metal and RTOS drivers
- On-chip and Off-chip Device Drivers
- Middleware and Stacks: RTOS, TSS libraries and communication stacks
- Component Development Environment (CDE) to create and distribute own components

Processor Expert Software

- A development system to create, configure, optimize, migrate, and deliver software and configuration details for Freescale silicon.

Initialization

CMSIS Headers
CMSIS startup code
Reset register values
Vector Table setup
Peripheral Initialization
Pin Muxing initialization

Device Driver Components

RTOS adaptive drivers
Low power capabilities
Configuration integrated
NOW – Kinetis Platform SDK Drivers supported

Processor Knowledge base

>1000 Processors Supported

Configuration

Reset configuration
DDR configure/validate
Pin Muxing
Device Tree Editor
Uboot configuration

API Factory

Script-based build server
CMSIS Header files
3rd Party Tools NPI support
Detailed Register files
Si Validation scripts
[Used by Common Register Repository initiative]

Kinetis SDK and Processor Expert

- Processor Expert is a complimentary PC-hosted software configuration tool (Eclipse plugin)
- Processor Expert (PEx) provides a time-saving option for software configuration through a graphical user interface (GUI)
- Board configuration and driver tuning tasks include:
  - Optional generation of low-level device initialization code for post-reset configuration
  - Pin Muxing tools to generate pin muxing functions
  - Components based on Kinetis SDK drivers
    - Users configure the SoC and Peripherals in a GUI
    - PEx creates the configuration data structures for driver config and init

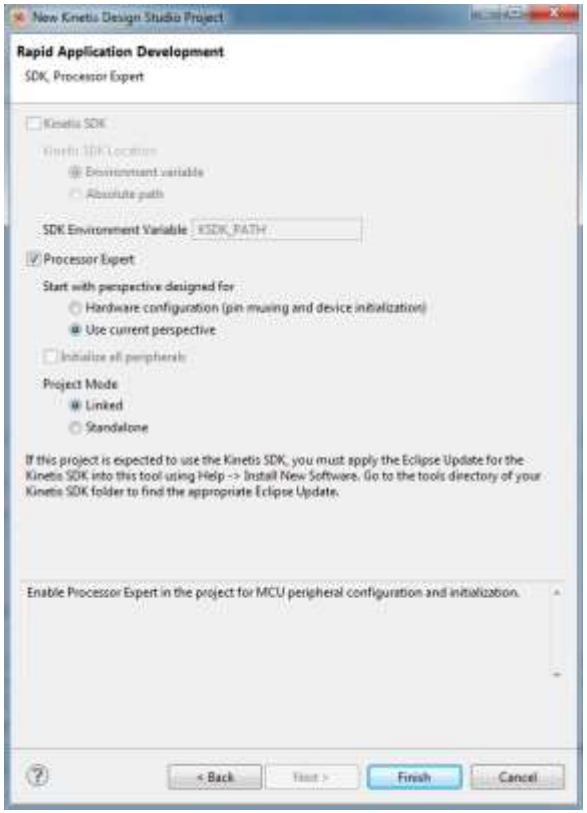
Processor Expert with KSDK

- Processor Expert now uses the KSDK drivers and HAL to implement the automatically generated code
  - Only available for devices supported by KSDK
  - Older devices will still use the classic PEx Logical Device Drivers (LDDs)

- KSDK-based driver code is not compatible with classic PEx LDDs
  - PEx GUI interface will behave similarly
  - Configuration options may change
  - Code generated will be significantly different

Creating a New Processor Expert Project for non-KSDK supported devices

- Devices not supported by Kinetis SDK will use the classic PEx LDDs
- The KSDK checkbox will be grayed out in the New Project wizard.

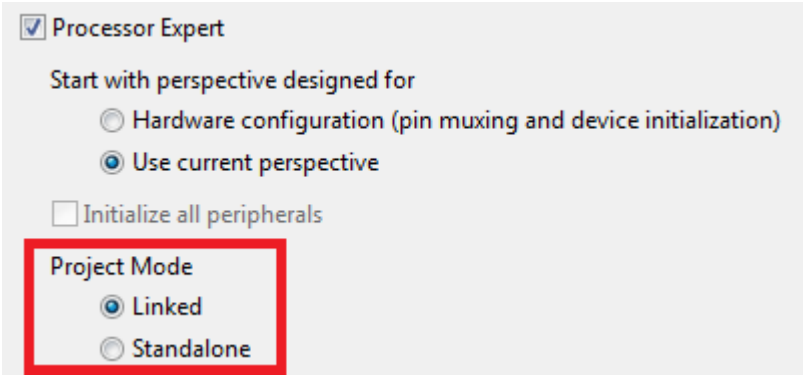


Creating a New Processor Expert Project for KSDK Supported Devices

- Devices supported by KSDK will use the Kinetis SDK drivers.
- The KSDK checkbox will be available for these devices
  - If Kinetis SDK is checked, PEx will use KSDK drivers and HAL.
  - If Kinetis SDK is not checked, PEx will use classic LDDs for drivers (if available)
- Most new devices will be forced to have the KSDK checked in order to use PEx
  - This is because LDD versions have not been created for those new devices. The future is KSDK drivers/HAL option only.

Creating a New Processor Expert Project – Linked vs Standalone

- Under the Processor Expert options when creating a project, you can select Linked or Standalone
- Linked:
  - Project will link to files in the KSDK installation path
  - Any modifications to KSDK source will affect all other projects
  - Good if need to create multiple projects that have same codebase
- Standalone:
  - The PEx wizard will copy necessary KSDK files into the project directory
  - Modifications to KSDK source in that directory won't affect other projects
  - Will take more hard drive space



Lab 2: PEx Device Initialization + SDK Drivers

Lab 2 Overview

Objective:

In this lab we will create a KDS Project with Processor Expert support and use the SDK for peripheral drivers. We will add several components and import a source file with implementation code.

Lab Flow:

- ❑ Create a new Processor Expert + SDK Project in KDS
- ❑ Add and Configure Components
- ❑ Generate Code
- ❑ Add Code to application
- ❑ Build
- ❑ Download Application to Target MCU
- ❑ Debug

Required Hardware and Software:

- ❑ FRDM-K22F Board configured with CMSIS-DAP Debugger
- ❑ Micro USB Cable
- ❑ Kinetis Design Studio (v2.0 or newer)
- ❑ Kinetis Software Development Kit (v1.1.0)

Project Definition

- **Hardware: FRDM-K22F**
- **Clock Configuration**
  Internal PLL: set to 120 MHz
  Bus Clock: 60 MHz
  Flash Clock: 20 MHz
- **Pin Muxing**
  GPIO; UART
- **Blink the Green LED**
  Interrupt timer: set at 10 Hz
- **Turn on Red LED and Disable Timer**
  Switch 2: Press to turn on; Disable Timer
- **Restart Timer; Turn off Red LED**
  Switch 3: Press to restart the Timer

Create a New Project to Blink the LEDs

- This hands-on lab shows you how to…
  - Create a new project with the New Project Wizard
  - Configure Components with the Component Inspector
  - Use Processor Expert Components
  - Add Code
  - Build the project
  - Test the application's functionality

Next up!

- The lab uses the FRDM-K22F board
- The application will blink an LED periodically, and turn on/off blinking LED with push buttons.

Lab 2 Notes

- If you can't find a field, make sure you've scrolled all the way down in the window
- If lose track of a Processor Expert Window and want to reset the view, click on "Processor Expert → Hide Views" and then "Processor Expert → Show Views" from the KDS menu bar
  - Also can use "Windows → Reset Perspective"

Lab 2 Summary

- Using Processor Expert is an easy way to configure a Kinetis MCU
- Adding SDK peripheral drivers with Processor Expert takes care of all of the “under the hood” stuff and properly includes files.

Agenda

- KSDK In-Depth
  - Lab
- KSDK + RTOS
- KSDK + USB
- KSDK + Processor Expert
  - Lab
- Conclusion

Summary

Session Summary

- You should now be able to:
  - Understand how Kinetis SDK works, how to get started writing applications, and how the RTOS and USB additions can make application creation easier
  - Create a new Processor Expert project and understand how it integrates in with KSDK
  - Use the knowledge and hands-on experience you have gained to quickly create applications using Freescale Kinetis MCUs

Additional Resources

Community
https://community.freescale.com/community/kinetis/kinetis-software-development-kit
https://community.freescale.com/community/kinetis

Web
www.freescale.com/ksdk
www.freescale.com/kds
www.freescale.com/freedom
www.freescale.com/mqx
www.freescale.com/usb
www.freescale.com/kboot

www.Freescale.com