FTF | FREESCALE TECHNOLOGY FORUM 2015

Hands-On Workshop: Getting Started with Kinetis SDK — Basic

FTF-DES-F1146

Martyn Hunt | Freescale Applications Engineer

JUNE.2015

freescale™

External Use

Agenda

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

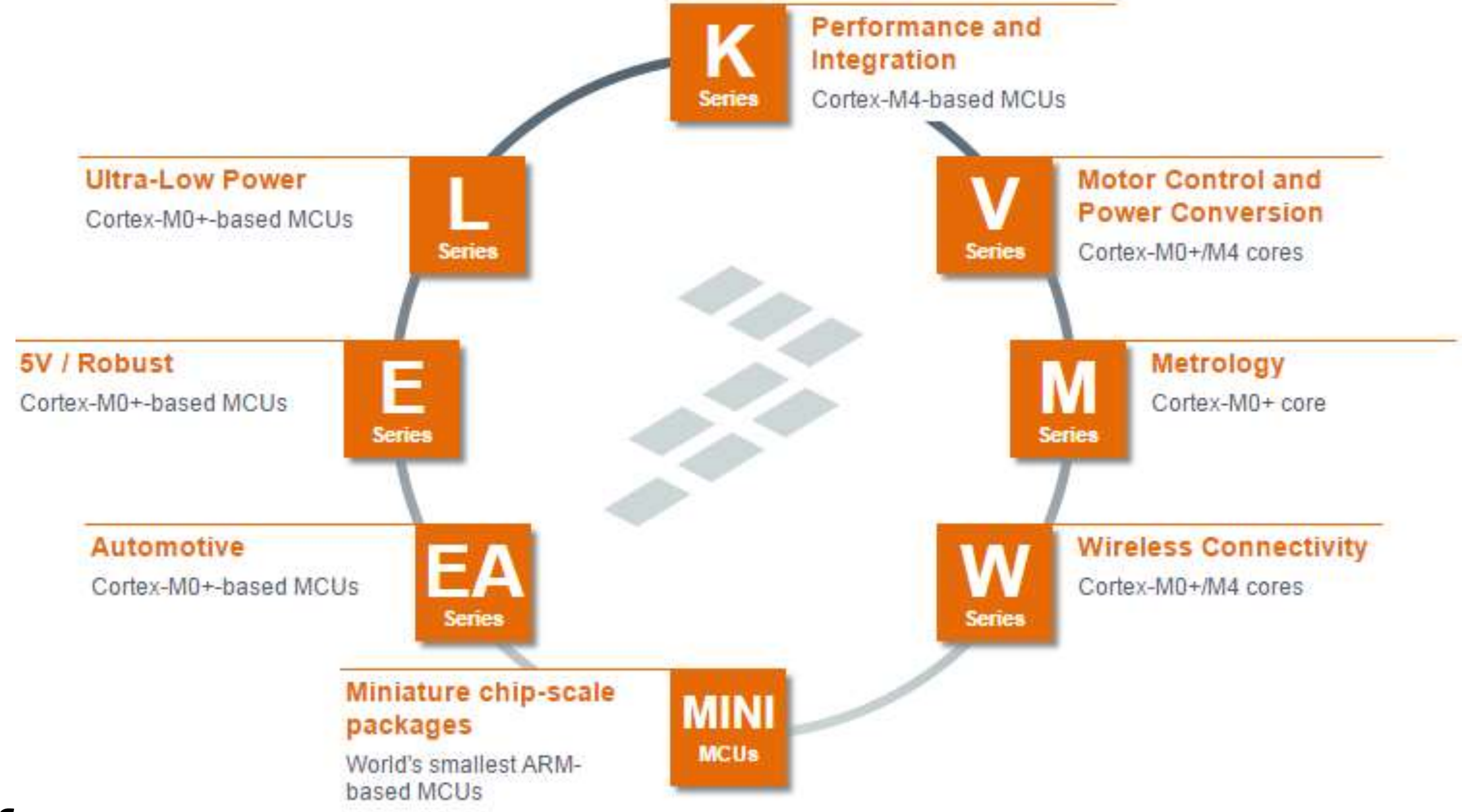
Class Notes

- Presentation and Lab Guides are on Desktop under “Getting started with Kinetis SDK” folder
- Presentation can be found at www.freescale.com/ftf
- Please leave boards on table after class as they will be used at other events
- Computer Password: CodeWarrior1

Kinetis Software Development Kit (KSDK)

Freescale Kinetis MCUs

- Based on ARM® Cortex®-M0+, M4, and M7 cores
- Hardware and software compatibly across hundreds of devices
- Exceptional low-power performance and feature integration



K Series
Performance and Integration
Cortex-M4-based MCUs

V Series
Motor Control and Power Conversion
Cortex-M0+/M4 cores

M Series
Metrology
Cortex-M0+ core

W Series
Wireless Connectivity
Cortex-M0+/M4 cores

MINI MCUs
Miniature chip-scale packages
World's smallest ARM-based MCUs

EA Series
Automotive
Cortex-M0+-based MCUs

E Series
5V / Robust
Cortex-M0+-based MCUs

L Series
Ultra-Low Power
Cortex-M0+-based MCUs

What is an SDK for and why it’s 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

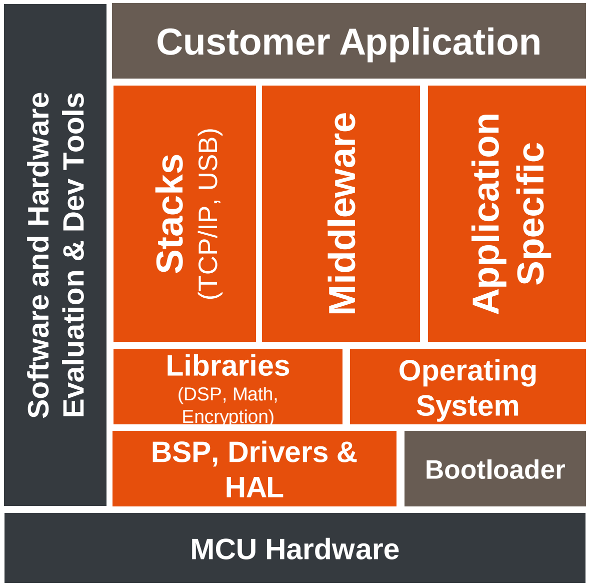
Learn more at: www.freescale.com/KSDK

Kinetis Software Development Kit (SDK)

SDK

A complete software framework for developing applications across all Kinetis MCUs

HAL, peripheral drivers, libraries, middleware, utilities, and usage examples; delivered in C source



Open Source Initiative

CMSIS COMPLIANT

Product Features

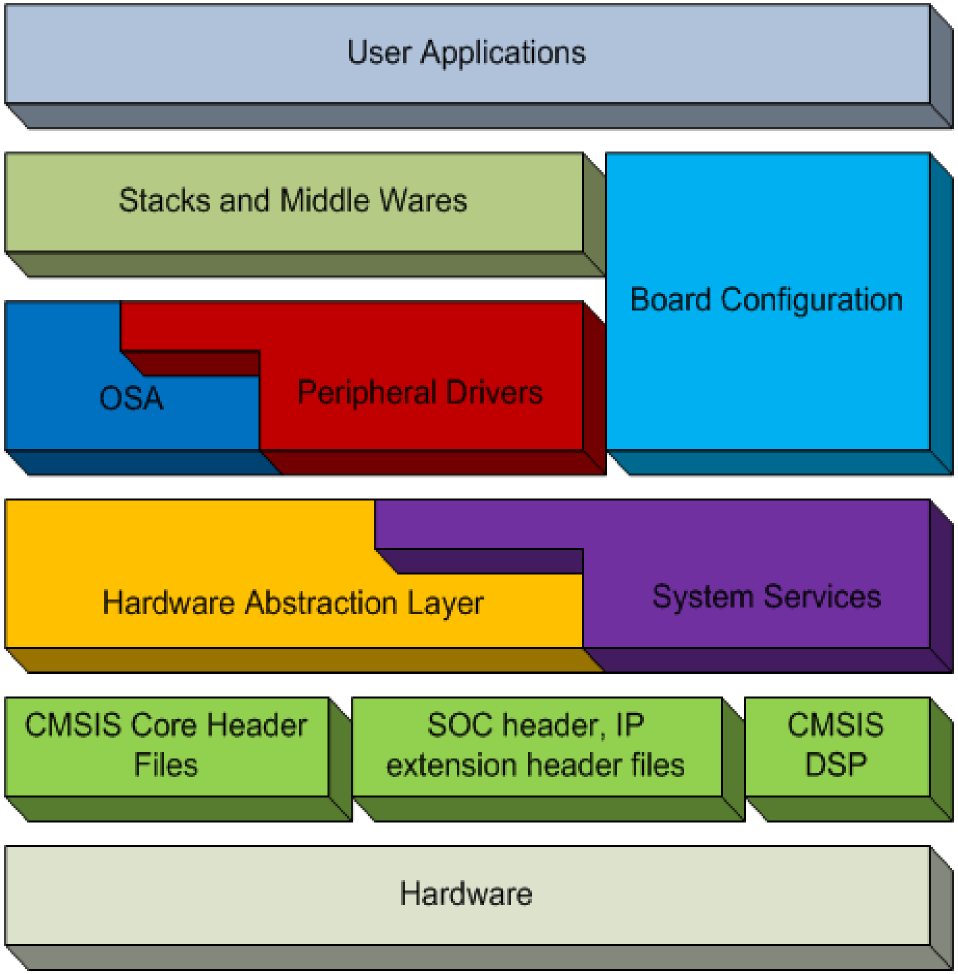
- Open source Hardware Abstraction Layer (HAL) provides APIs for all Kinetis hardware resources
- BSD-licensed set of peripheral drivers with easy-to-use C-language APIs
- Comprehensive HAL and driver usage examples and sample applications for RTOS and bare-metal.
- CMSIS-CORE compatible startup and drivers plus CMSIS-DSP library and examples
- RTOS Abstraction Layer (OSA) with support for Freescale MQX, FreeRTOS, Micrium uC/OS, bare-metal and more
- Integrates USB and TCP/IP stacks, touch sensing software, encryption and math/DSP libraries, and more
- Support for multiple toolchains including GNU GCC, IAR, Keil, and Kinetis Design Studio
- Integrated with Processor Expert

The OSI logo trademark is the trademark of Open Source Initiative.

Kinetis SDK Key Components

- Two major components of the Kinetis SDK
  - 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

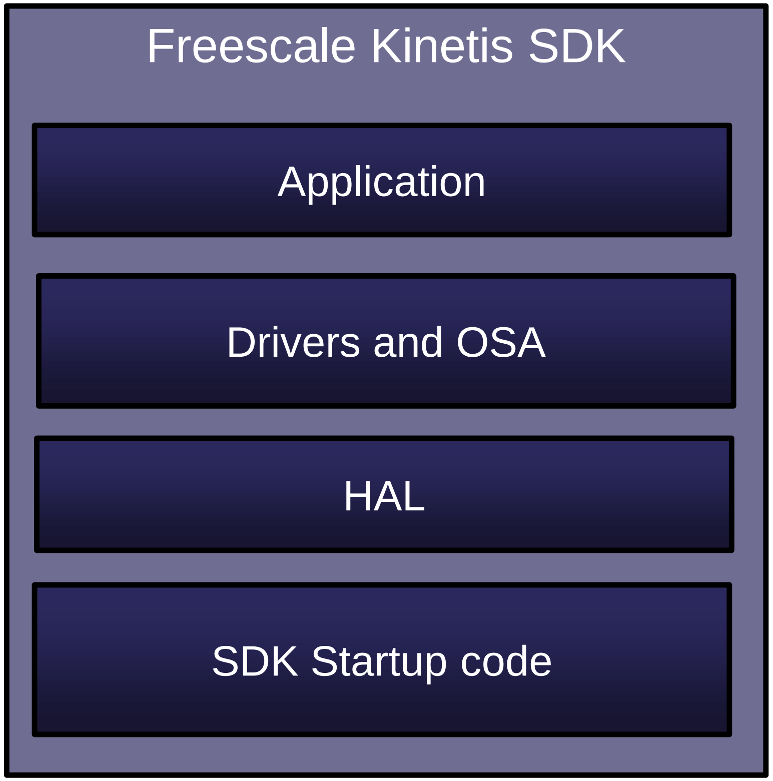
Stacks & Middle Wares

- 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 Kinetis SDK drivers
  - No state awareness
  - Mostly macros to provide user-friendly naming to access MCU registers
- Kinetis SDK 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_1.2.0\platform\hal
- HAL Library at C:\Freescale\KSDK_1.2.0\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

- Kinetis SDK 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_1.2.0\platform\drivers
- Driver+HAL library at C:\Freescale\KSDK_1.2.0\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.

Stacks and Other Middleware

- This layer completes the Kinetis SDK 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 Kinetis SDK drivers
  - Freescale USB stack not adhere to this rule, because SDK HAL is not implementing USB IP now.

Board Configuration and Support

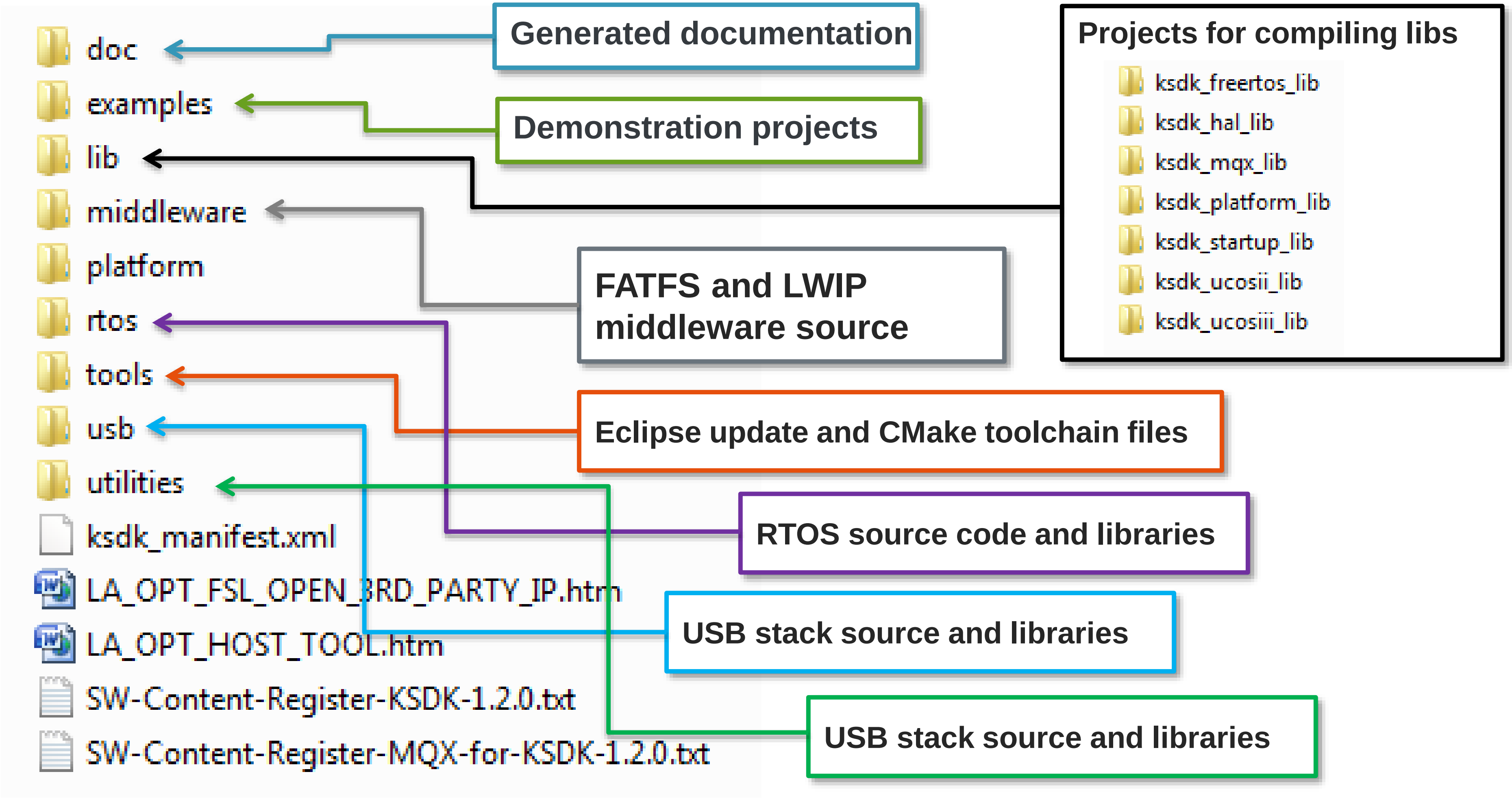
- Pin Muxing
  - Kinetis SDK driver layer will not handle pin muxing. It is handled in the board configuration part, where pin muxing functions are generated using “Pin Muxing” tool in KDS via PEx

- 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

Kinetis SDK Layout

Directory Structure



doc
examples
lib
middleware
platform
rtos
tools
usb
utilities
ksdk_manifest.xml
LA_OPT_FSL_OPEN_3RD_PARTY_IP.htm
LA_OPT_HOST_TOOL.htm
SW-Content-Register-KSDK-1.2.0.txt
SW-Content-Register-MQX-for-KSDK-1.2.0.txt

Generated documentation

Demonstration projects

Projects for compiling libs

ksdk_freertos_lib
ksdk_hal_lib
ksdk_mqx_lib
ksdk_platform_lib
ksdk_startup_lib
ksdk_ucosii_lib
ksdk_ucosiii_lib

FATFS and LWIP middleware source

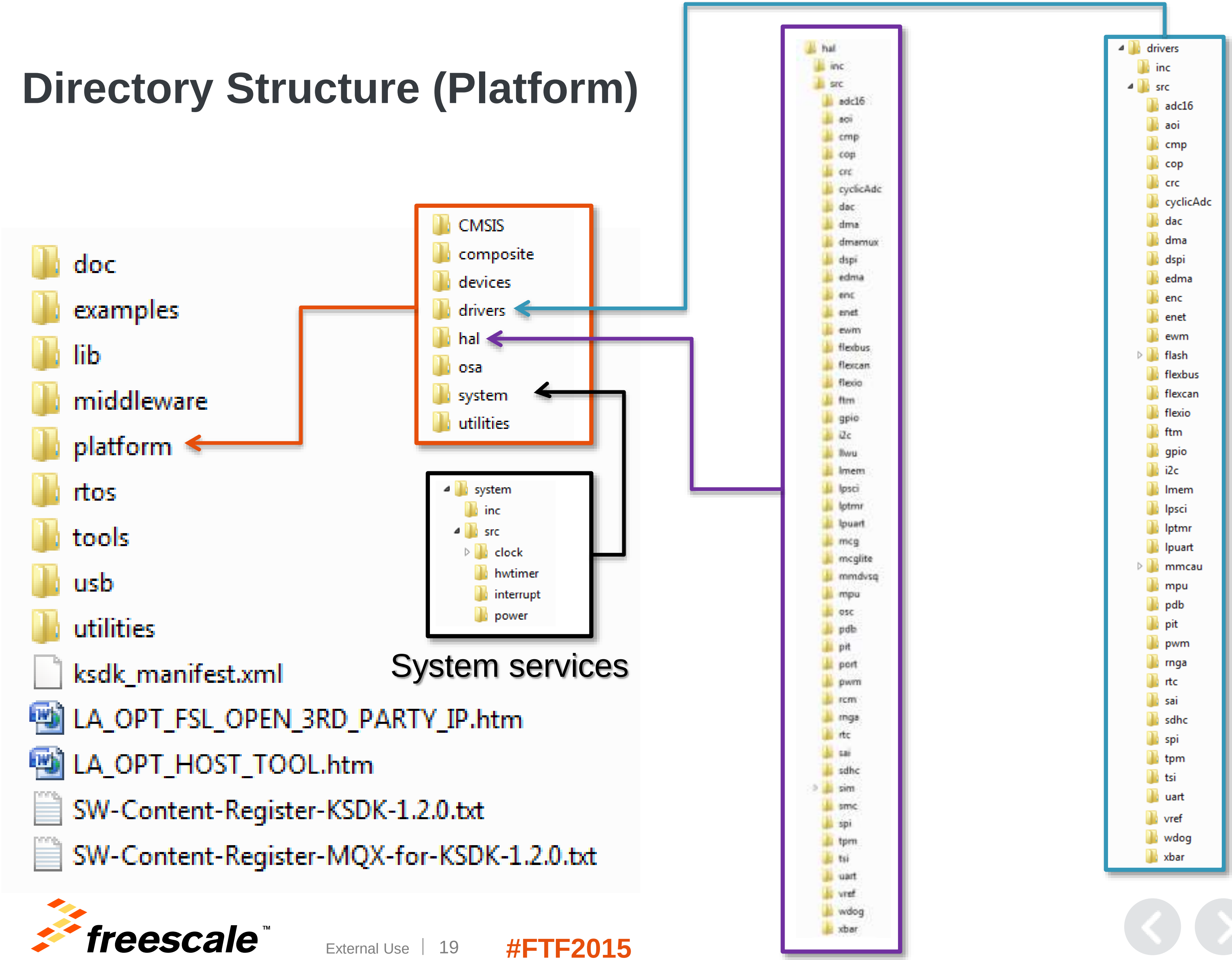
Eclipse update and CMake toolchain files

RTOS source code and libraries

USB stack source and libraries

USB stack source and libraries

Directory Structure (Platform)



- doc
- examples
- lib
- middleware
- platform
- rtos
- tools
- usb
- utilities
- ksdk_manifest.xml
- LA_OPT_FSL_OPEN_3RD_PARTY_IP.htm
- LA_OPT_HOST_TOOL.htm
- SW-Content-Register-KSDK-1.2.0.txt
- SW-Content-Register-MQX-for-KSDK-1.2.0.txt

- CMSIS
- composite
- devices
- drivers
- hal
- osa
- system
- utilities

- system
  - inc
  - src
    - clock
    - hwtimer
    - interrupt
    - power

System services

- hal
  - inc
  - src
    - adc16
    - aoi
    - cmp
    - cop
    - crc
    - cyclicAdc
    - dac
    - dma
    - dmamux
    - dspi
    - edma
    - enc
    - enet
    - ewm
    - flexbus
    - flexcan
    - flexio
    - ftm
    - gpio
    - i2c
    - llwu
    - lmem
    - lpsci
    - lptmr
    - lpuart
    - mcg
    - mcglite
    - mmdvsq
    - mpu
    - osc
    - pdb
    - pit
    - port
    - pwm
    - rcm
    - rnga
    - rtc
    - sai
    - sdhc
    - sim
    - smc
    - spi
    - tpm
    - tsi
    - uart
    - vref
    - wdog
    - xbar

- drivers
  - inc
  - src
    - adc16
    - aoi
    - cmp
    - cop
    - crc
    - cyclicAdc
    - dac
    - dma
    - dspi
    - edma
    - enc
    - enet
    - ewm
    - flash
    - flexbus
    - flexcan
    - flexio
    - ftm
    - gpio
    - i2c
    - lmem
    - lpsci
    - lptmr
    - lpuart
    - mmcau
    - mpu
    - pdb
    - pit
    - pwm
    - rnga
    - rtc
    - sai
    - sdhc
    - spi
    - tpm
    - tsi
    - uart
    - vref
    - wdog
    - xbar

OS Abstraction (OSA)

OS Abstraction Layer Overview

- Enables Kinetis SDK to work with different RTOSes

- Support key RTOS services
  - Semaphores, Mutex, Memory Management, Events, more…

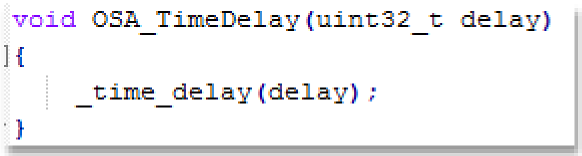
- Implementation for different RTOSes
  - Bare Metal
  - MQX, FreeRTOS, uCOS-II, and uCOS-III

- 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()

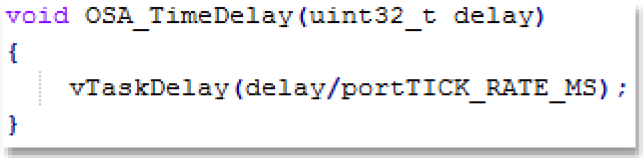
Translation code found in \platform\osa

- For MQX maps to:



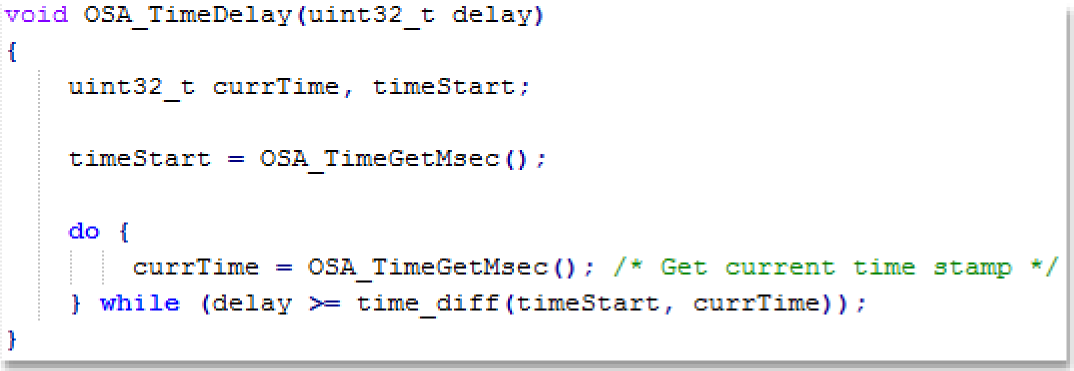
```c
void OSA_TimeDelay(uint32_t delay)
{
    _time_delay(delay);
}
```

- For FreeRTOS maps to:



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

- For Baremetal maps to:



```c
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 Kinetis SDK
  - 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)

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

Clock Manager

Clock Manager Overview

- A high-level API that allows an application to manage and query system and peripheral clocking
- Combines functionality from the Multipurpose Clock Generator (MCG), System Integration Module (SIM), Real-Time Clock (RTC) and Oscillator (OSC) peripherals into a single API set
- Enables forcible or agreeable clock changes with optional application-defined callbacks

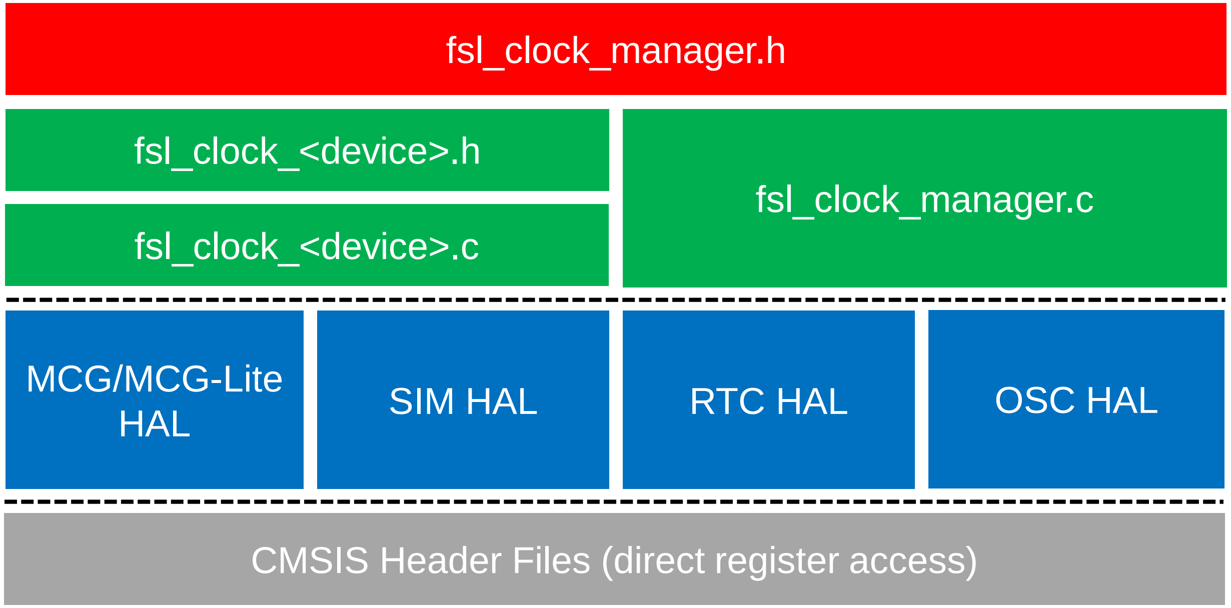
Where Can You Find The Clock Manager?

- System Services are located in the ./platform/system folder of the SDK tree.
  - Header files are in the inc folder
  - In the src folder, each System Service module has its own container folder. For the clock manager, the folder is called clock

- The Clock Manager is layered:
  - Common functions (across all FSL platforms) are located in the top level fsl_clock_manager.c file, which resides in ./platform/system/src/clock
  - Device-specific functions and feature implementations reside in a MCU family sub-folder within ./platform/system/src/clock. For example, the FRDM-K22F's implementation is in ./platform/system/src/clock/MK22F51212

Clock Manager Source Hierarchy

- The user application only includes fsl_clock_manager.h, everything else is automatically pulled in.



Clock Manager Application Usage

- The Clock Manager supports two usage models:
  - Simplified: Application calls CLOCK_SYS_SetConfiguration() and is responsible for notifying or updating peripherals prior to changing the clock.
  - Managed: The Clock Manager is provided with a set of clock configurations to manage and can notify peripherals or application modules of changes via callback functions.
- Managed mode is effectively a wrapper around CLOCK_SYS_SetConfiguration() – with the added intelligence to provide callback functions.

Clock Manager “Simplified” Usage Example

```
// Configure the OSC.
CLOCK_SYS_OscInit(0U, &osc0Config);
```

```
osc_user_config_t osc0Config =
{
    .freq                  = OSC0_XTAL_FREQ,
    .hgo                   = MCG_HGO0,
    .range                 = MCG_RANGE0,
    .erefs                 = MCG_EREFS0,
    .enableCapacitor2p     = OSC0_SC2P_ENABLE_CONFIG,
    .enableCapacitor4p     = OSC0_SC4P_ENABLE_CONFIG,
    .enableCapacitor8p     = OSC0_SC8P_ENABLE_CONFIG,
    .enableCapacitor16p    = OSC0_SC16P_ENABLE_CONFIG,
};
```

```
// Set system, bus, flash and FlexBus clocks.
CLOCK_SYS_SetConfiguration(&g_defaultClockConfigRun);
```

```
const clock_manager_user_config_t g_defaultClockConfigRun =
{
    .mcgConfig =
    {
        .mcg_mode            = kMcgModePEE,    // Work in PEE mode.
        .irclkEnable         = true,   // MCGIRCLK enable.
        .irclkEnableInStop   = false, // MCGIRCLK disable in STOP mode.
        .ircs                = kMcgIrcSlow, // Select IRC32k.
        .fcrdiv              = 0U,     // FCRDIV is 0.

        .frdiv   = 3U,
        .drs     = kMcgDcoRangeSelLow,   // Low frequency range
        .dmx32   = kMcgDmx32Default,     // DCO has a default range of 25%
        .oscsel  = kMcgOscselOsc,        // Select OSC

        .pll0EnableInFllMode        = false,  // PLL0 disable
        .pll0EnableInStop           = false,  // PLL0 disable in STOP mode
        .prdiv0                     = 0x3U,
        .vdiv0                      = 0x10U,
    },
    .simConfig =
    {
        .pllFllSel = kClockPllFllSelPll,    // PLLFLLSEL select PLL.
        .er32kSrc  = kClockEr32kSrcRtc,     // ERCLK32K selection, use RTC.
        .outdiv1   = 0U,
        .outdiv2   = 1U,
        .outdiv3   = 2U,
        .outdiv4   = 3U,
    },
    .oscerConfig =
    {
        .enable       = true,  // OSCERCLK enable.
        .enableInStop = false, // OSCERCLK disable in STOP mode.
        .erclkDiv     = 0U,    // OSCERCLK divider setting.
```

Clock Manager “Managed” Usage Example

```
// Configure the OSC.
CLOCK_SYS_OscInit(0U, &osc0Config);

// Initialize Clock Manager.
CLOCK_SYS_Init(&g_defaultClockConfigurations, NUM_CONFIGS,
               clockCallbackTable, NUM_CALLBACKS);
```

```
#define NUM_CONFIGS   3U

#define VLPR_MODE     1U
#define RUN_MODE      2U

const clock_manager_user_config_t *
g_defaultClockConfigurations[] =
{
    NULL,
    &g_defaultClockConfigVlpr,
    &g_defaultClockConfigRun
};
```

```
#define NUM_CALLBACKS  1U

static clock_manager_callback_user_config_t
*clockCallbackTable[] =
{
    &userAppClockCalback
};
```

```
// Set clock configuration to RUN mode.
CLOCK_SYS_UpdateConfiguration(RUN_MODE,
                              kClockManagerPolicyForcible);

// Set clock configuration to VLPR mode.
CLOCK_SYS_UpdateConfiguration(VLPR_MODE,
                              kClockManagerPolicyForcible);
```

Callback Implementation Example

Notification structure includes notification type, policy and the target clock configuration index

Application-defined data can be passed into or out of a callback.

```
clock_manager_error_code_t
userAppClockCallback(clock_notify_struct_t *notify, void* dataPtr)
{
    clock_manager_error_code_t result = kClockManagerSuccess;

    switch (notify->notifyType)
    {
        case kClockManagerNotifyBefore:
        /* TODO */
        /* Add code here. */
        break;

        case kClockManagerNotifyRecover:
        /* TODO */
        /* Add code here. */
        break;

        case kClockManagerNotifyAfter:
        /* TODO */
        /* Add code here. */
        break;

        default:
            result = kClockManagerError;
        break;
    }
    return result;
}
```

Clock Manager Additional Functionality

```
// Configure the RTC oscillator.
CLOCK_SYS_RtcOscInit(0U, &rtcOscConfig);

// Get values of various clocks.
sys_clk = CLOCK_SYS_GetSystemClockFreq();
bus_clk = CLOCK_SYS_GetBusClockFreq();
flash_clk = CLOCK_SYS_GetFlashClockFreq();

// Update clock divider values.
CLOCK_SYS_SetOutDiv1(2);
CLOCK_SYS_SetOutDiv2(4);
CLOCK_SYS_SetOutDiv3(4);
CLOCK_SYS_SetOutDiv4(4);

// Enable UART0 clock gate.
CLOCK_SYS_EnableUartClock(0);

// Disable UART0 clock gate.
CLOCK_SYS_DisableUartClock(0);

// Get UART0 clock value.
uart_clk = CLOCK_SYS_GetUartFreq(0);

// Set LPUART0’s clock source.
CLOCK_SYS_SetLpuartSrc(0, kClockLpuartSrcIrc48M);
```

```
rtc_osc_user_config_t rtcOscConfig =
{
    .freq                  = RTC_XTAL_FREQ,
    .enableCapacitor2p     = RTC_SC2P_ENABLE_CONFIG,
    .enableCapacitor4p     = RTC_SC4P_ENABLE_CONFIG,
    .enableCapacitor8p     = RTC_SC8P_ENABLE_CONFIG,
    .enableCapacitor16p    = RTC_SC16P_ENABLE_CONFIG,
    .enableOsc             = RTC_OSC_ENABLE_CONFIG,
    .enableClockOutput     = RTC_CLK_OUTPUT_ENABLE_CONFIG,
};
```

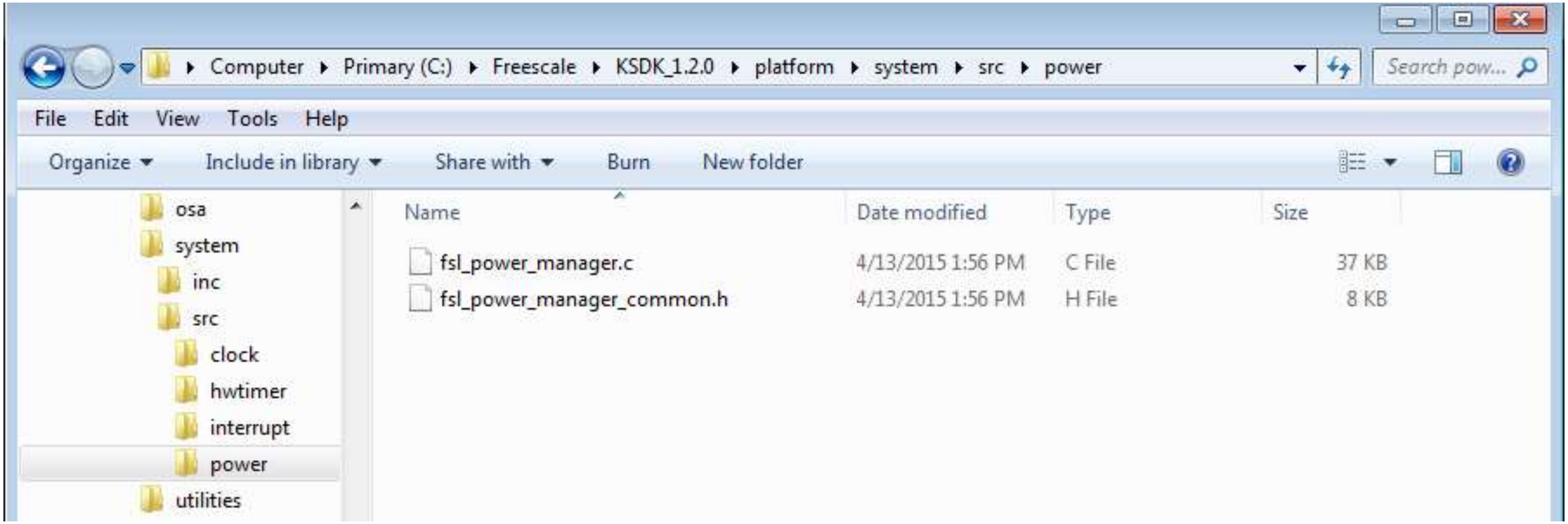
Kinetis SDK Power Manager

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



Name	Date modified	Type	Size
fsl_power_manager.c	4/13/2015 1:56 PM	C File	37 KB
fsl_power_manager_common.h	4/13/2015 1:56 PM	H File	8 KB

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()**

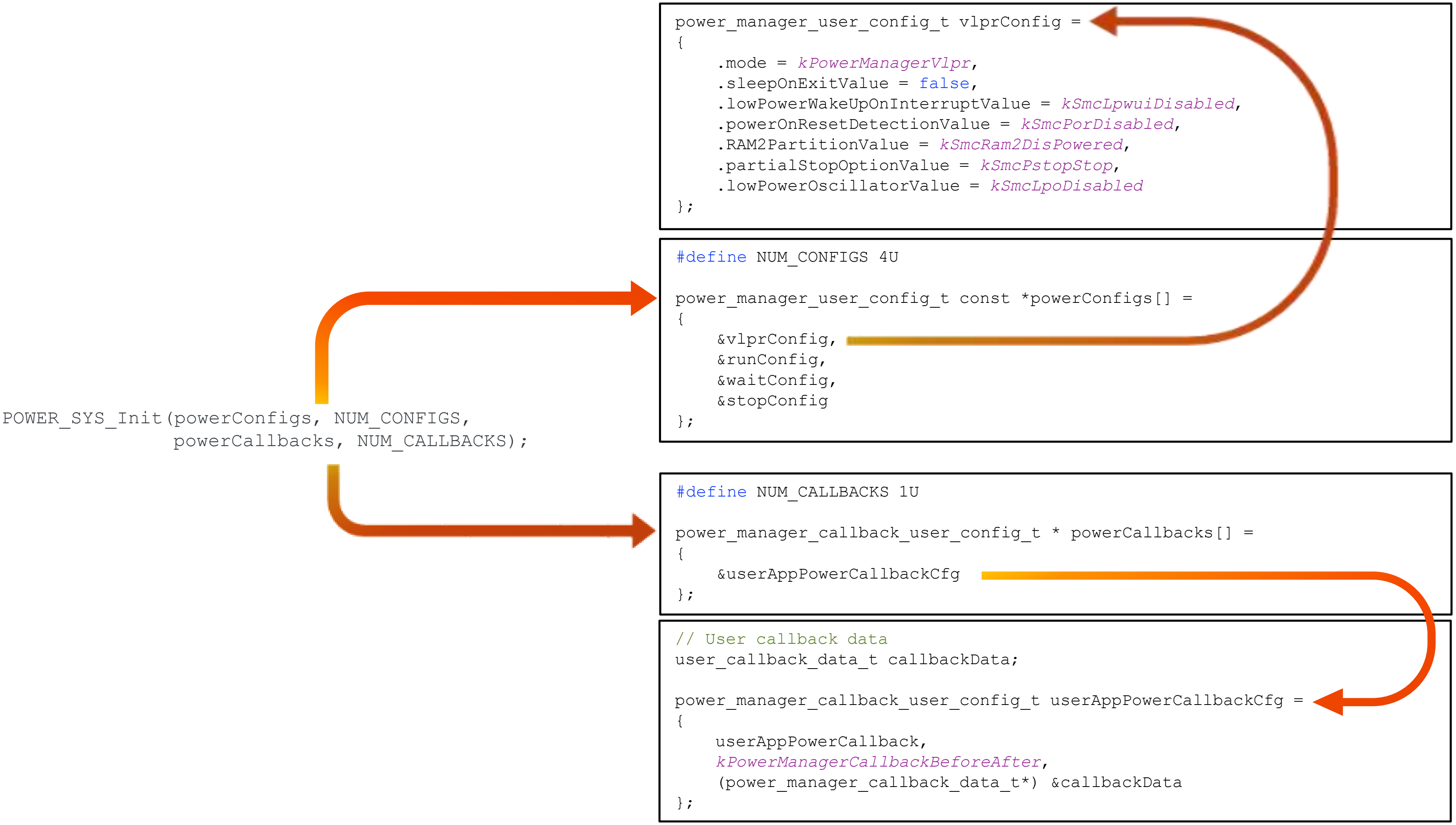
Power Manager Interaction with Other Components

- 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

Power Manager Initialization



```
POWER_SYS_Init(powerConfigs, NUM_CONFIGS,
               powerCallbacks, NUM_CALLBACKS);
```

```
power_manager_user_config_t vlprConfig =
{
    .mode = kPowerManagerVlpr,
    .sleepOnExitValue = false,
    .lowPowerWakeUpOnInterruptValue = kSmcLpwuiDisabled,
    .powerOnResetDetectionValue = kSmcPorDisabled,
    .RAM2PartitionValue = kSmcRam2DisPowered,
    .partialStopOptionValue = kSmcPstopStop,
    .lowPowerOscillatorValue = kSmcLpoDisabled
};
```

```
#define NUM_CONFIGS 4U

power_manager_user_config_t const *powerConfigs[] =
{
    &vlprConfig,
    &runConfig,
    &waitConfig,
    &stopConfig
};
```

```
#define NUM_CALLBACKS 1U

power_manager_callback_user_config_t * powerCallbacks[] =
{
    &userAppPowerCallbackCfg
};
```

```
// User callback data
user_callback_data_t callbackData;

power_manager_callback_user_config_t userAppPowerCallbackCfg =
{
    userAppPowerCallback,
    kPowerManagerCallbackBeforeAfter,
    (power_manager_callback_data_t*) &callbackData
};
```

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 (forcible) or abort if the user callback signals it is not ready (agreeable)

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

Index of desired user-defined power mode

```
power_manager_user_config_t const *powerConfigs[] =
{
    &vlprConfig,
    &runConfig,
    &waitConfig,
    &stopConfig
};
```

This is what the index refers to

Interrupt Manager

Interrupt Manager Overview

- Enable or disable system interrupts at the NVIC level
- Global enable/disable of system interrupts
- Dynamically register/install interrupt service routines (ISRs) into systems that utilize a RAM-based vector table
- To set interrupt priorities, leverage the NVIC_* APIs defined in the CMSIS header file

Agenda

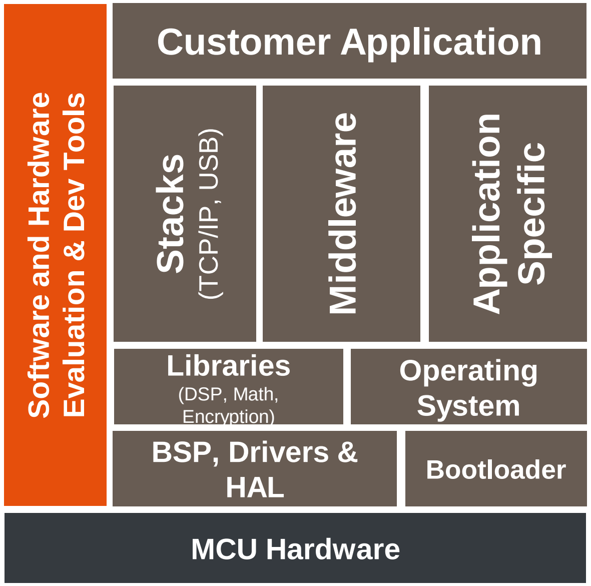
- KSDK In-Depth
  - Lab
- KSDK + RTOS
- KSDK + USB
- KSDK + Processor Expert
  - Lab
- Conclusion

Learn more at: www.freescale.com/freedom
www.freescale.com/mbed

Freedom Development Platforms

Low-cost/low-power development hardware

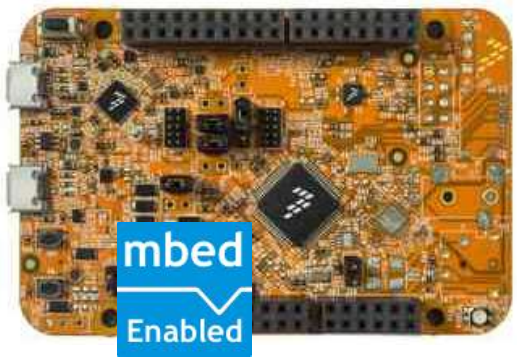
Enables quick application prototyping and demonstration of Kinetis MCU families



Product Features

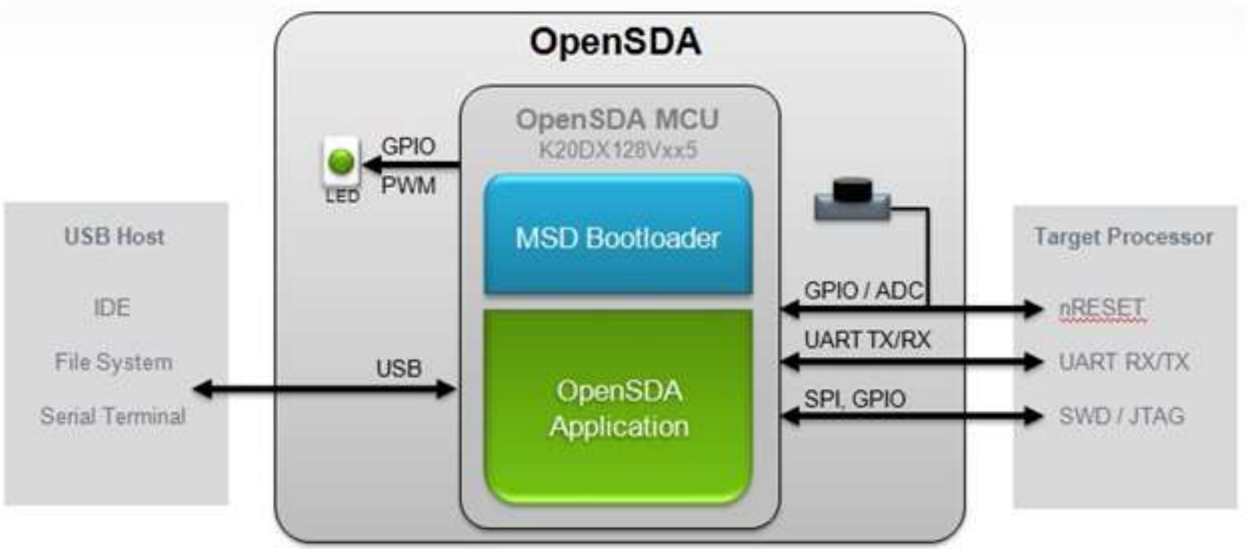
- Low–cost (starting at $12.95 USD)
- Designed in an industry-standard compact form factor (Arduino R3)
- Easy access to the MCU I/O pins
- Integrated open-standard serial and debug interface (OpenSDA)
- Compatible with a rich-set of third-party expansion boards

FRDM-K22F:

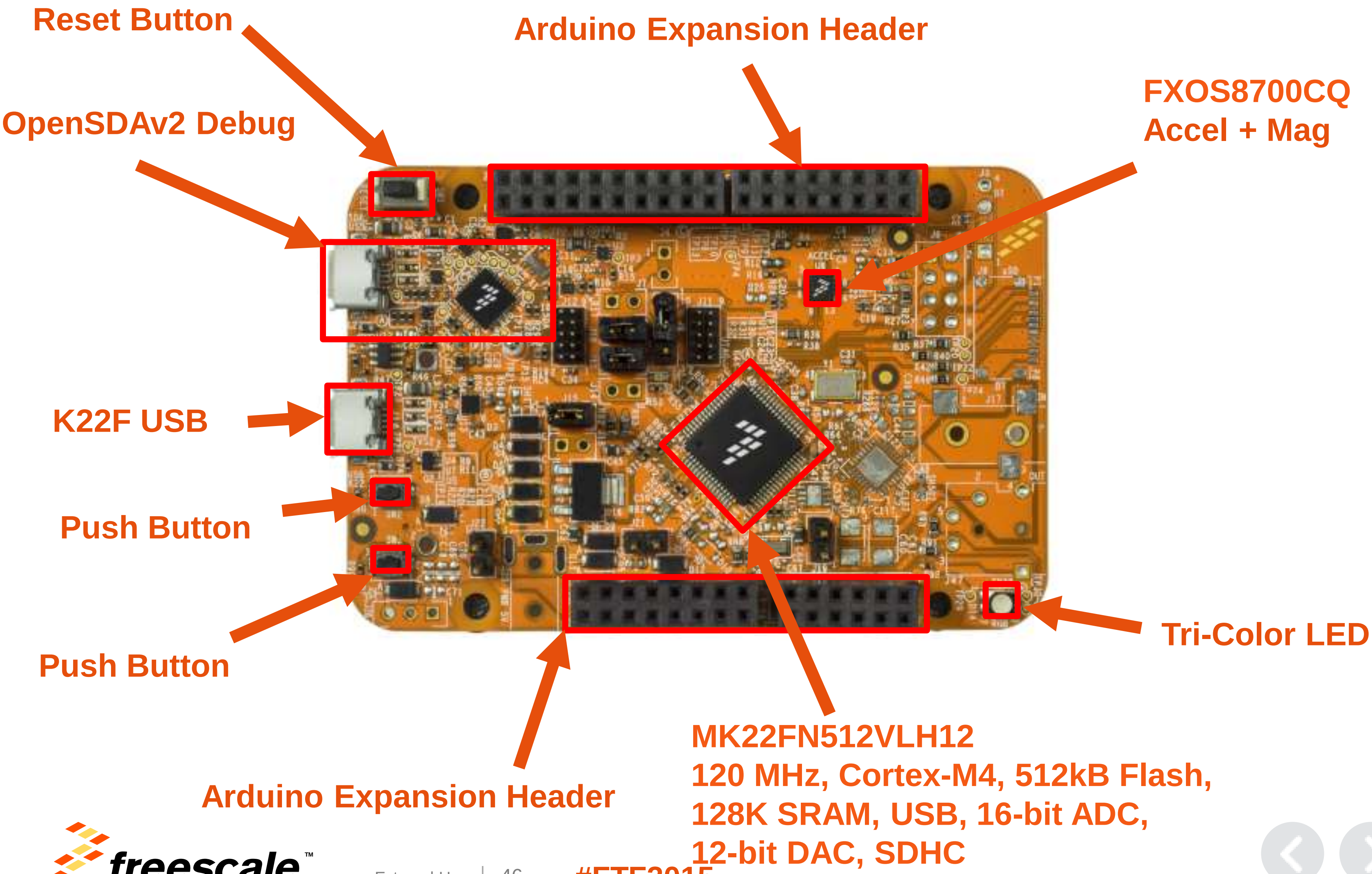


OpenSDA

- 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



Reset Button

Arduino Expansion Header

FXOS8700CQ
Accel + Mag

OpenSDAv2 Debug

K22F USB

Push Button

Push Button

Tri-Color LED

Arduino Expansion Header

MK22FN512VLH12
120 MHz, Cortex-M4, 512kB Flash,
128K SRAM, USB, 16-bit ADC,
12-bit DAC, SDHC

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:

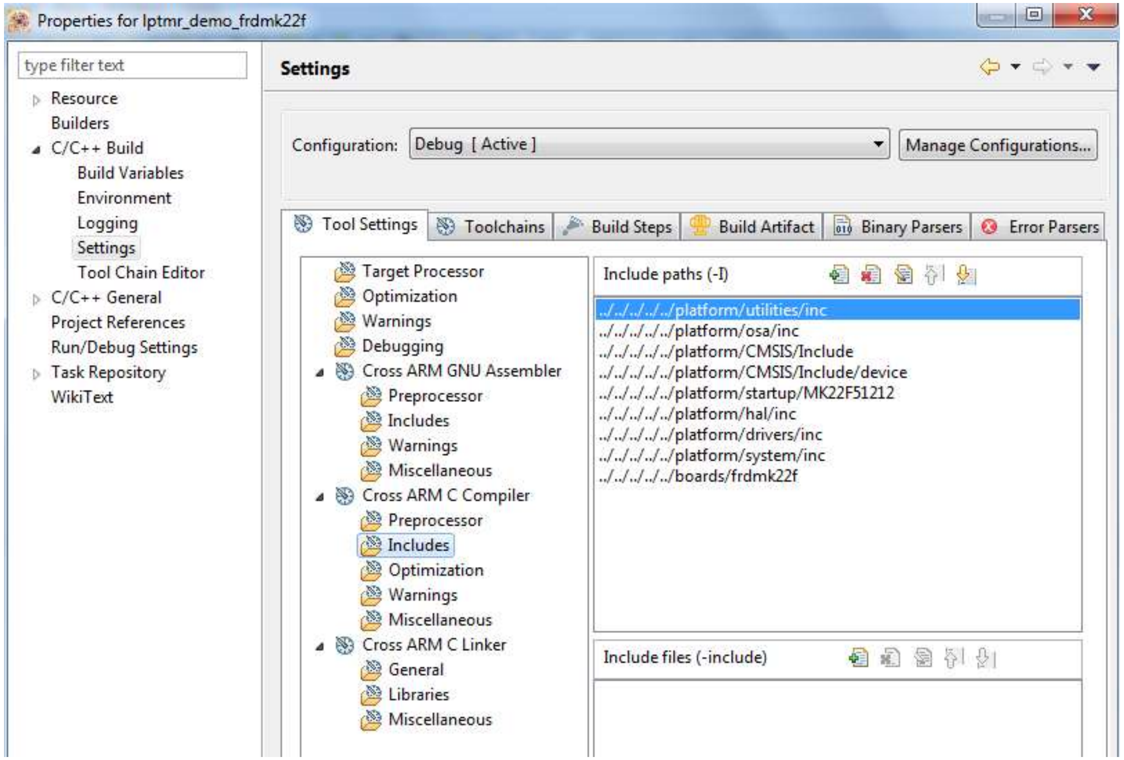
- ❑FRDM-K22F Board configured with CMSIS-DAP Debugger
- ❑Micro USB Cable
- ❑Kinetis Design Studio (v3.0 or newer)
- ❑Kinetis Software Development Kit (v1.2.0)

Lab 1 Summary

- Imported and built KSDK platform library for MK22FN512xxx12.
- Imported and built hwtimer_demo from KSDK_1.2.0.
- Run the demo with KDS.

KSDK Project Information

- Right click on hwtimer 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



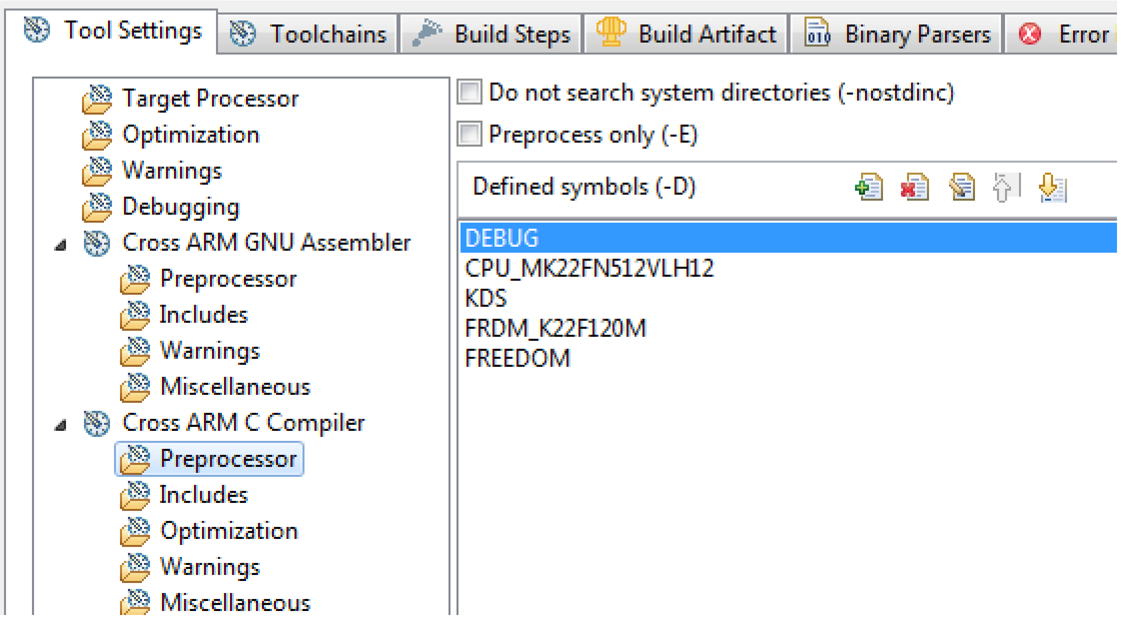
Configuration: Debug [Active]

Include paths (-I)

```
../../../../platform/utilities/inc
../../../../platform/osa/inc
../../../../platform/CMSIS/Include
../../../../platform/CMSIS/Include/device
../../../../platform/startup/MK22F51212
../../../../platform/hal/inc
../../../../platform/drivers/inc
../../../../platform/system/inc
../../../../boards/frdmk22f
```

KSDK Project Information Continued

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



□ Do not search system directories (-nostdinc)

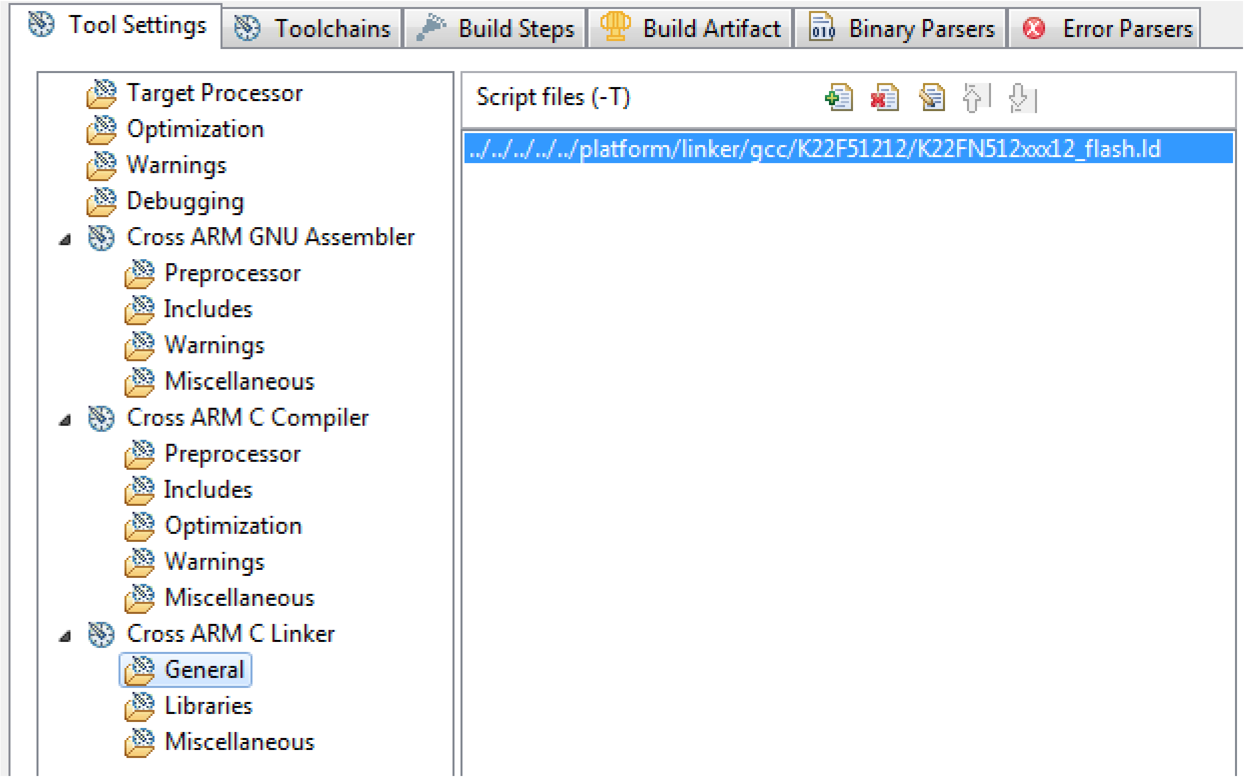
□ Preprocess only (-E)

Defined symbols (-D)

```
DEBUG
CPU_MK22FN512VLH12
KDS
FRDM_K22F120M
FREEDOM
```

KSDK Project Information Continued

- Linker File

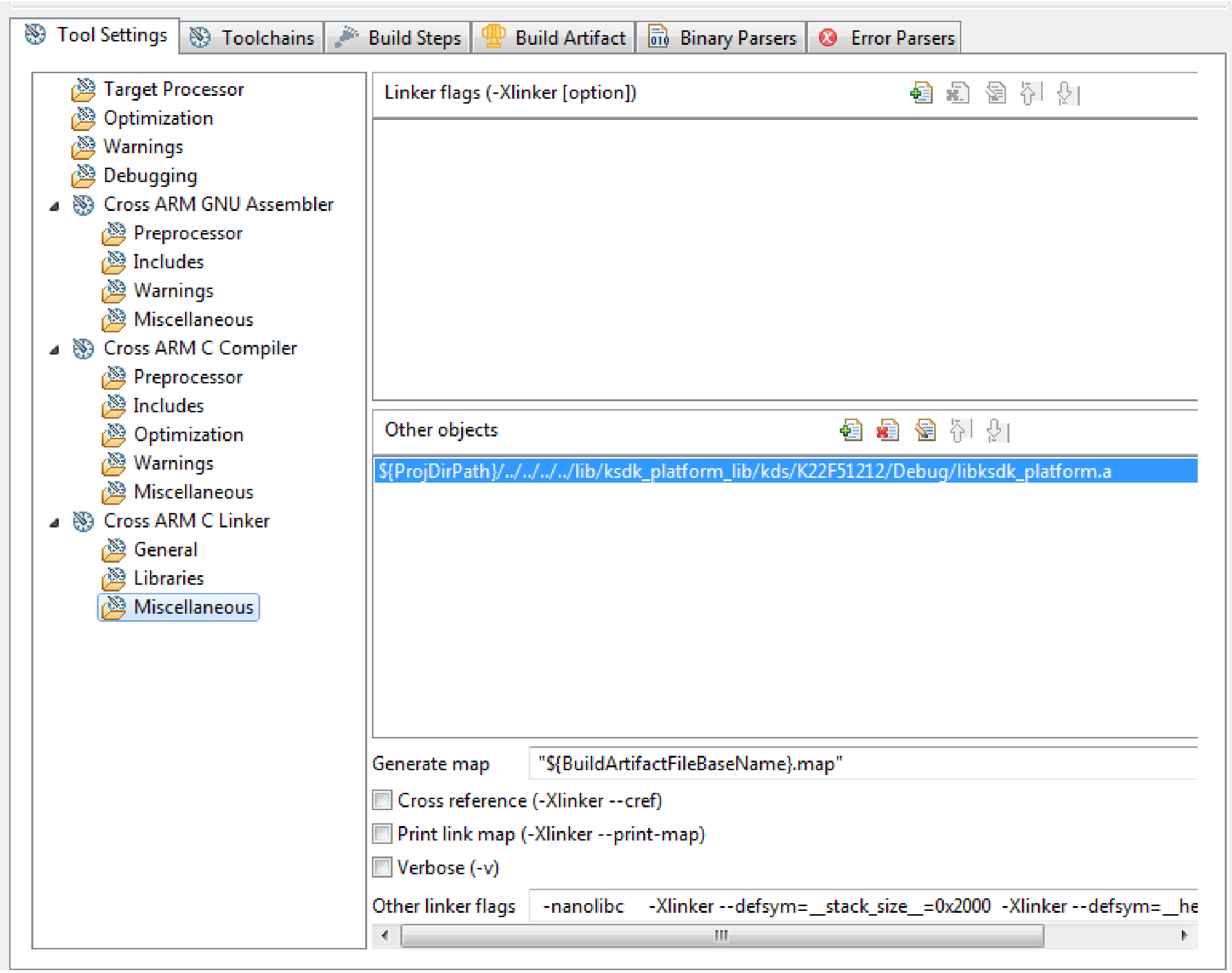


Script files (-T)

../../../../../platform/linker/gcc/K22F51212/K22FN512xxx12_flash.ld

KSDK Project Information Continued

- KSDK Platform Library



Linker flags (-Xlinker [option])

Other objects

${ProjDirPath}/../../../../lib/ksdk_platform_lib/kds/K22F51212/Debug/libksdk_platform.a

Generate map "${BuildArtifactFileBaseName}.map"

□ Cross reference (-Xlinker --cref)

□ Print link map (-Xlinker --print-map)

□ Verbose (-v)

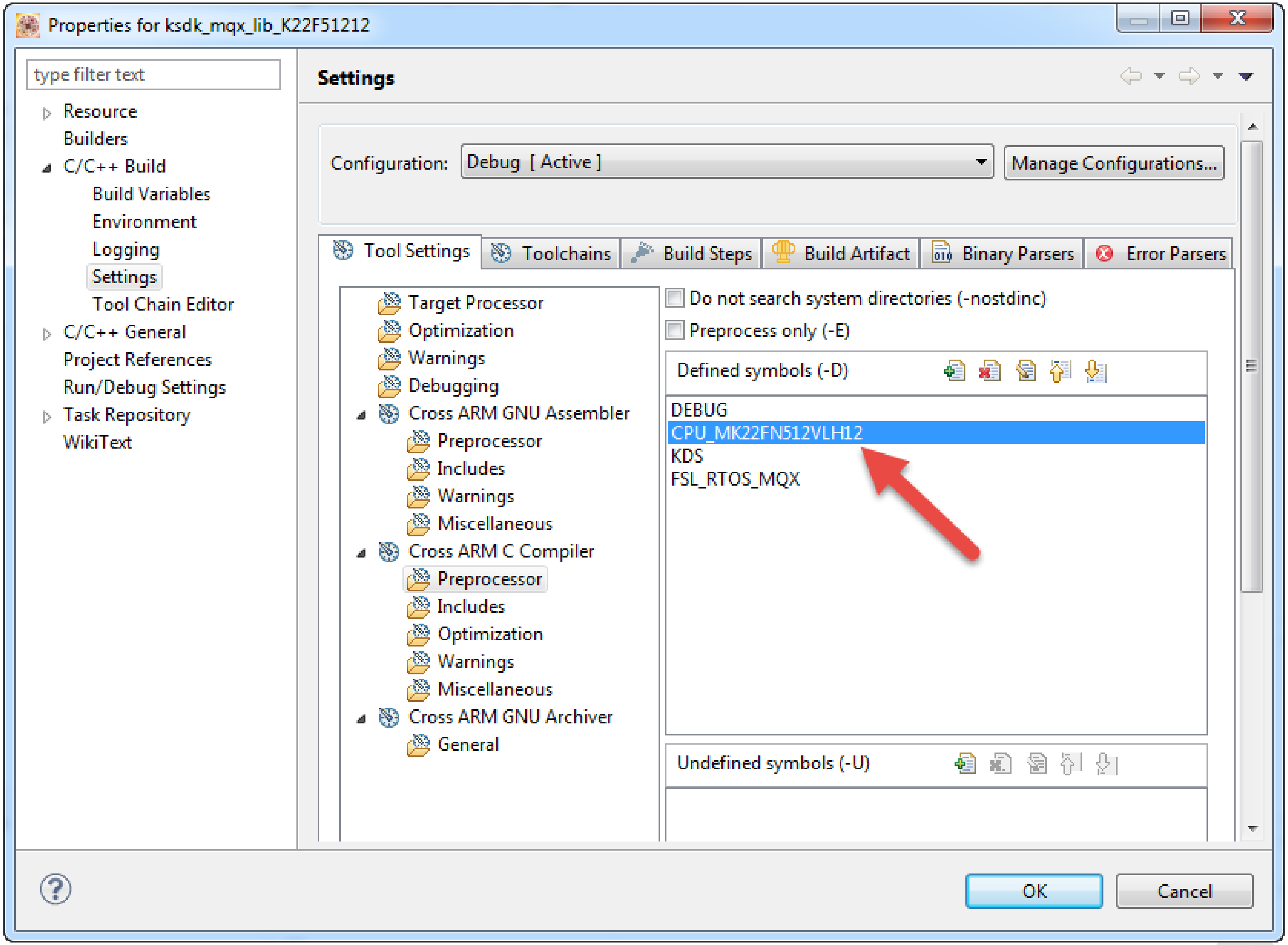
Other linker flags -nanolibc -Xlinker --defsym=__stack_size__=0x2000 -Xlinker --defsym=__he

Porting to subset device

Changing to Kinetis Subset Derivative

- Kinetis SDK makes changing to a subset derivative easy
- Kinetis SDK 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_adc16_features.h
  - Specify which KSDK header files to include in build like <KSDK_PATH> \platform\CMSIS\Include\device\fsl_device_registers.h
- Kinetis SDK uses compiler preprocessor definition to specify derivative.
  - Change in ksdk_platform_lib project and rebuild

KDS Example: Derivative Defined in project



Properties for ksdk_mqx_lib_K22F51212

Settings

Configuration: Debug [Active]

☐ Do not search system directories (-nostdinc)

☐ Preprocess only (-E)

Defined symbols (-D)

```
DEBUG
CPU_MK22FN512VLH12
KDS
FSL_RTOS_MQX
```

Undefined symbols (-U)

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

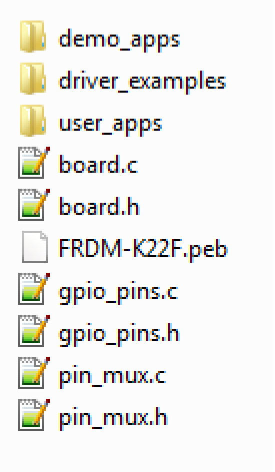
- Kinetis SDK already includes supported derivatives
  - Can find all derivative options in <KSDK_PATH> \platform\CMSIS\Include\device\fsl_device_registers.h

- Porting to a new family is not supported. Only derivatives.
  - Full list of supported derivatives can be found in the Release Notes

Porting to new board layout

Custom Board Configuration

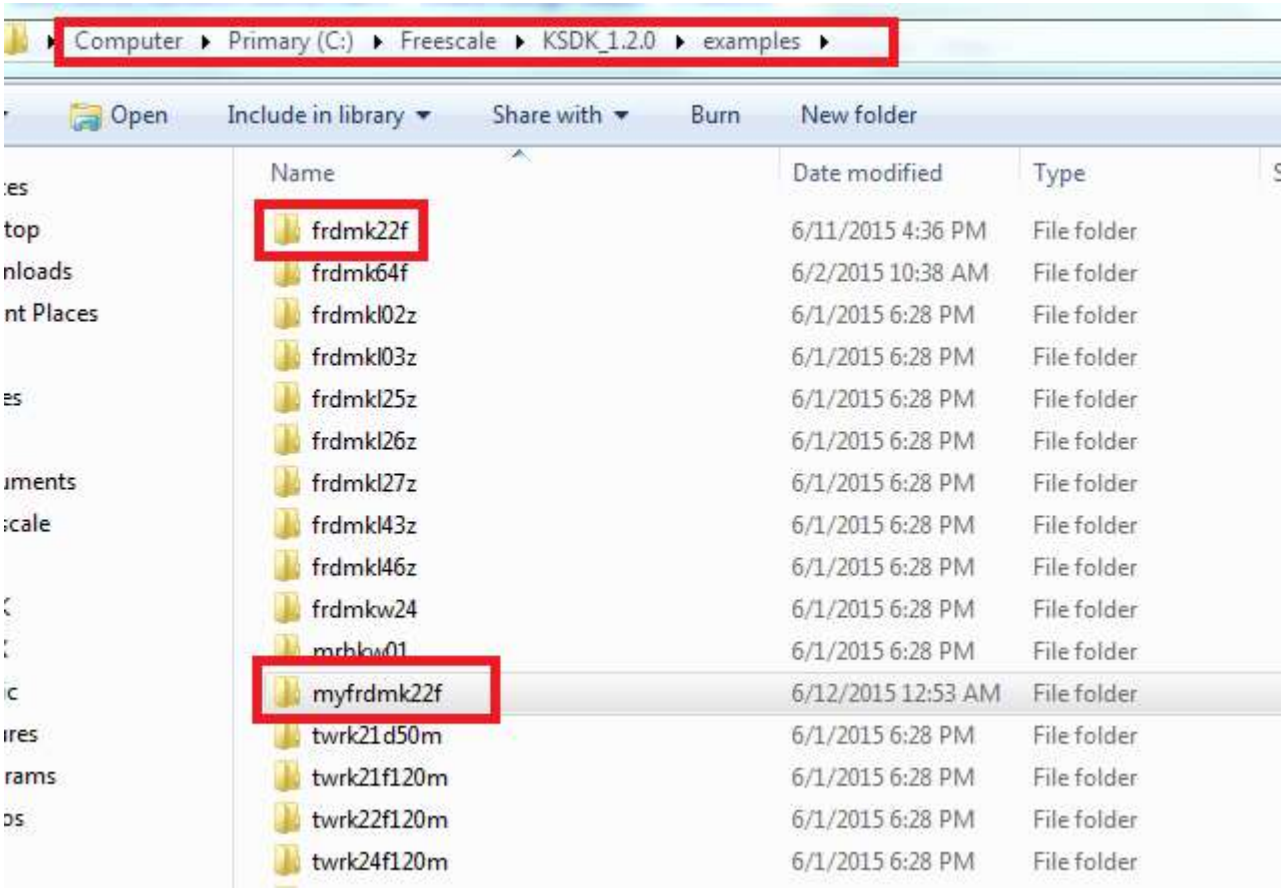
- Each development board supported by Kinetis SDK has board configuration files
- Found in <KSDK_PATH>/examples/<board_name>
- Contains board-specific details for Kinetis SDK
  - Applications easily portable across different boards and devices
- These files should be reviewed and modified for custom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



demo_apps
driver_examples
user_apps
board.c
board.h
FRDM-K22F.peb
gpio_pins.c
gpio_pins.h
pin_mux.c
pin_mux.h

New Board Support

- Copy and rename closest board folder in the examples directory



Name	Date modified	Type
frdmk22f	6/11/2015 4:36 PM	File folder
frdmk64f	6/2/2015 10:38 AM	File folder
frdmkl02z	6/1/2015 6:28 PM	File folder
frdmkl03z	6/1/2015 6:28 PM	File folder
frdmkl25z	6/1/2015 6:28 PM	File folder
frdmkl26z	6/1/2015 6:28 PM	File folder
frdmkl27z	6/1/2015 6:28 PM	File folder
frdmkl43z	6/1/2015 6:28 PM	File folder
frdmkl46z	6/1/2015 6:28 PM	File folder
frdmkw24	6/1/2015 6:28 PM	File folder
mrbkw01	6/1/2015 6:28 PM	File folder
myfrdmk22f	6/12/2015 12:53 AM	File folder
twrk21d50m	6/1/2015 6:28 PM	File folder
twrk21f120m	6/1/2015 6:28 PM	File folder
twrk22f120m	6/1/2015 6:28 PM	File folder
twrk24f120m	6/1/2015 6:28 PM	File folder

board.h file

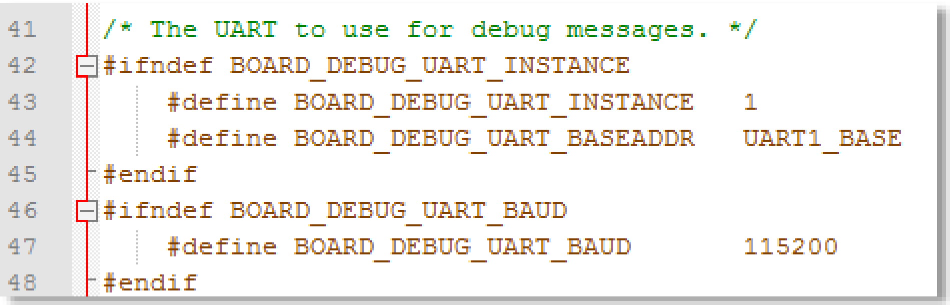
- Defines debug UART peripheral and pins
  - For stdin/stdout functions, like printf()
- Mainly used for Kinetis SDK examples, specifying:
  - Features available on board, like sensor for demos
  - Peripheral instances for examples, like I2C0
  - Pins for LEDs and buttons

board.c file

- Defines clock structures
- BOARD_ClockInit()
  - Uses clock manager to configure the system clocks

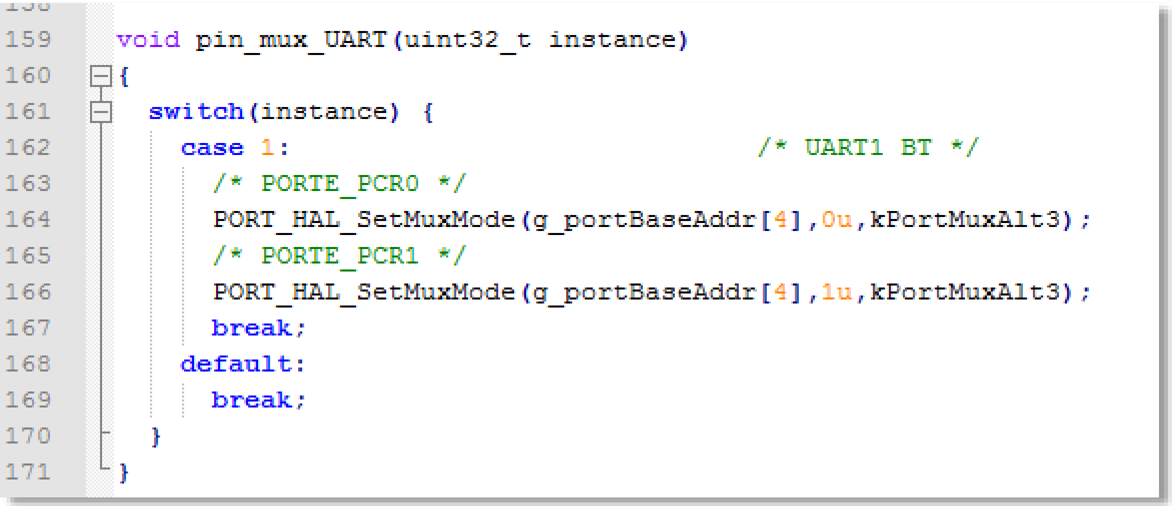
Kinetis SDK Porting — Change Default UART

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



```
/* The UART to use for debug messages. */
#ifndef BOARD_DEBUG_UART_INSTANCE
    #define BOARD_DEBUG_UART_INSTANCE   1
    #define BOARD_DEBUG_UART_BASEADDR   UART1_BASE
#endif
#ifndef BOARD_DEBUG_UART_BAUD
    #define BOARD_DEBUG_UART_BAUD       115200
#endif
```

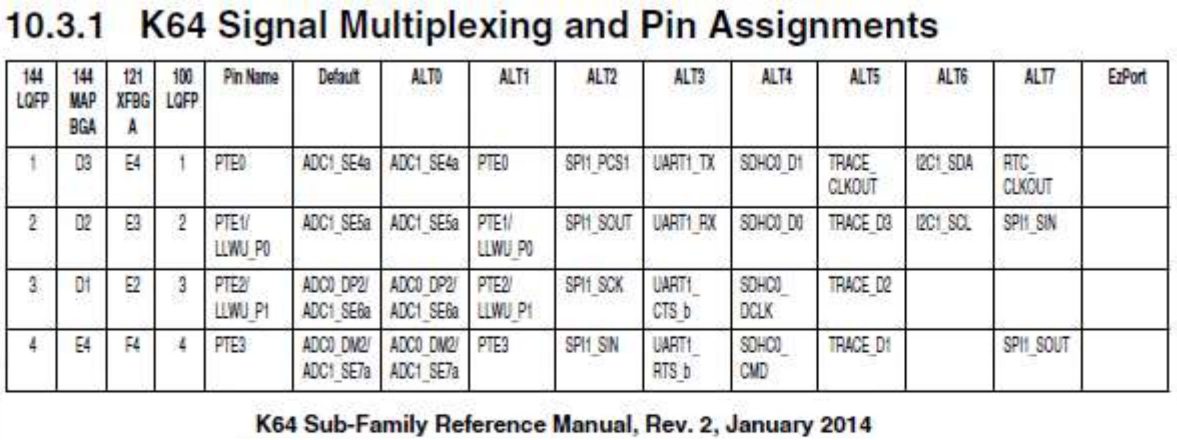
- Modify pin_mux.c to select the pins to use



```
void pin_mux_UART(uint32_t instance)
{
  switch(instance) {
    case 1:                  /* UART1 BT */
      /* PORTE_PCR0 */
      PORT_HAL_SetMuxMode(g_portBaseAddr[4],0u,kPortMuxAlt3);
      /* PORTE_PCR1 */
      PORT_HAL_SetMuxMode(g_portBaseAddr[4],1u,kPortMuxAlt3);
      break;
    default:
      break;
  }
}
```

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



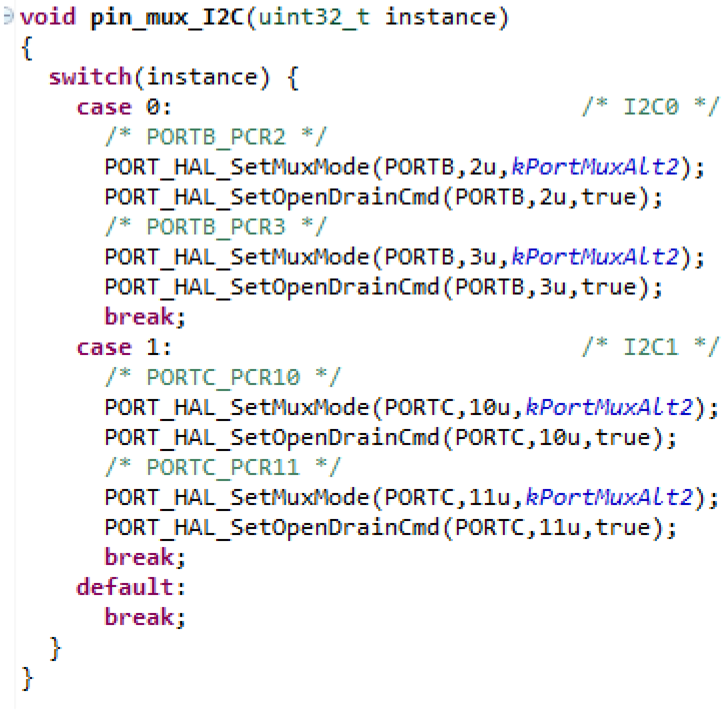
10.3.1 K64 Signal Multiplexing and Pin Assignments

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

pin_mux.c and pin_mux.h

- Not all instances will be populated so may need to add
- Also some modules come out from more than one place, so check that desired pins are being used



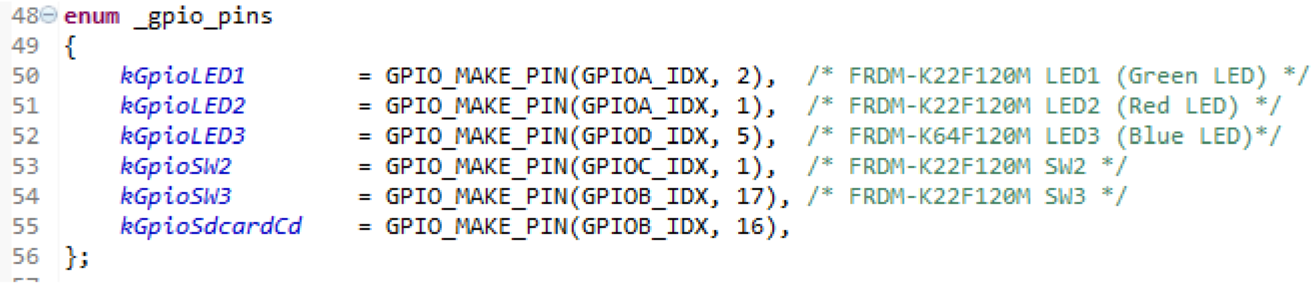
```
void pin_mux_I2C(uint32_t instance)
{
  switch(instance) {
    case 0:                             /* I2C0 */
      /* PORTB_PCR2 */
      PORT_HAL_SetMuxMode(PORTB,2u,kPortMuxAlt2);
      PORT_HAL_SetOpenDrainCmd(PORTB,2u,true);
      /* PORTB_PCR3 */
      PORT_HAL_SetMuxMode(PORTB,3u,kPortMuxAlt2);
      PORT_HAL_SetOpenDrainCmd(PORTB,3u,true);
      break;
    case 1:                             /* I2C1 */
      /* PORTC_PCR10 */
      PORT_HAL_SetMuxMode(PORTC,10u,kPortMuxAlt2);
      PORT_HAL_SetOpenDrainCmd(PORTC,10u,true);
      /* PORTC_PCR11 */
      PORT_HAL_SetMuxMode(PORTC,11u,kPortMuxAlt2);
      PORT_HAL_SetOpenDrainCmd(PORTC,11u,true);
      break;
    default:
      break;
  }
}
```

gpio_pins.c and gpio_pins.h

- Kinetis SDK 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 by board
  - PORT pin to use (ie: PTE3)

gpio_pins.h

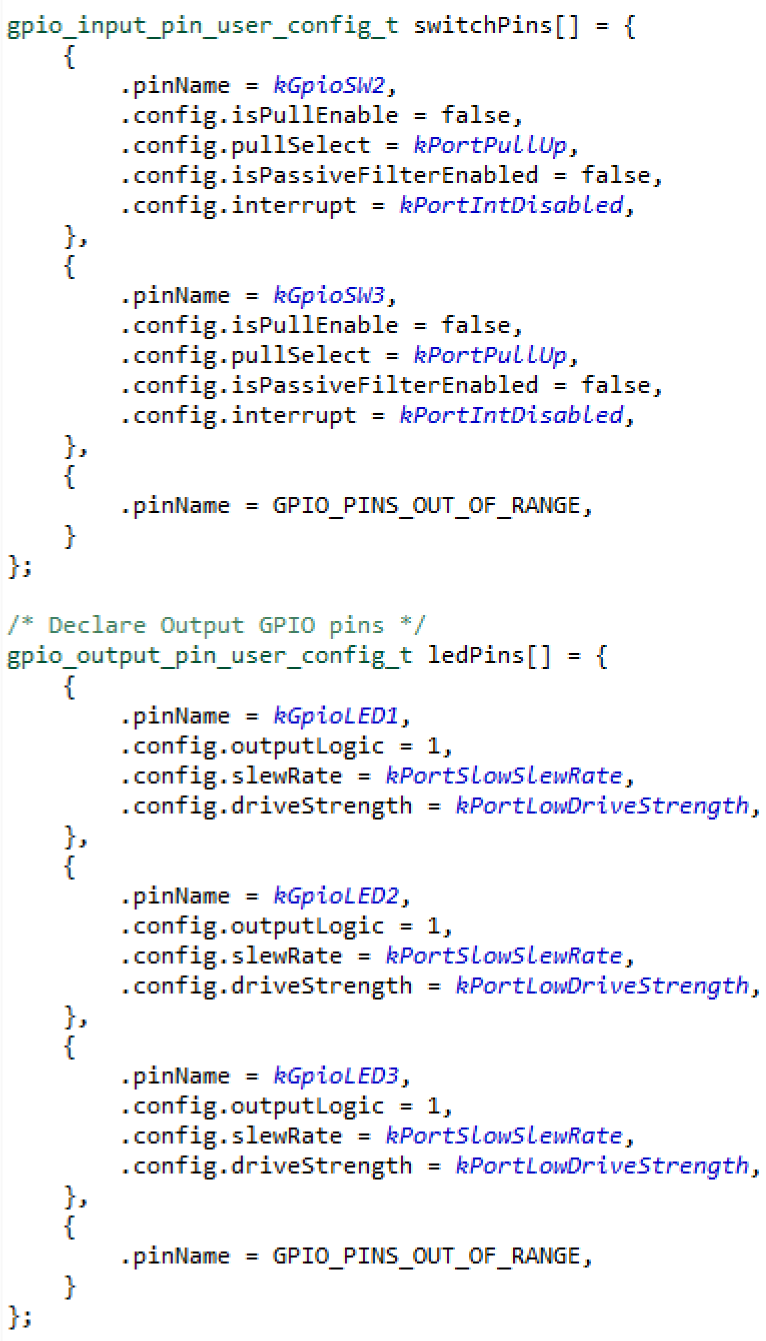
- Contains definitions for LED, Switch, and SD Card chip select



```
enum _gpio_pins
{
    kGpioLED1        = GPIO_MAKE_PIN(GPIOA_IDX, 2),  /* FRDM-K22F120M LED1 (Green LED) */
    kGpioLED2        = GPIO_MAKE_PIN(GPIOA_IDX, 1),  /* FRDM-K22F120M LED2 (Red LED) */
    kGpioLED3        = GPIO_MAKE_PIN(GPIOD_IDX, 5),  /* FRDM-K64F120M LED3 (Blue LED)*/
    kGpioSW2         = GPIO_MAKE_PIN(GPIOC_IDX, 1),  /* FRDM-K22F120M SW2 */
    kGpioSW3         = GPIO_MAKE_PIN(GPIOB_IDX, 17), /* FRDM-K22F120M SW3 */
    kGpioSdcardCd    = GPIO_MAKE_PIN(GPIOB_IDX, 16),
};
```

gpio_pins.c

- Contains GPIO options for each pin



```
gpio_input_pin_user_config_t switchPins[] = {
    {
        .pinName = kGpioSW2,
        .config.isPullEnable = false,
        .config.pullSelect = kPortPullUp,
        .config.isPassiveFilterEnabled = false,
        .config.interrupt = kPortIntDisabled,
    },
    {
        .pinName = kGpioSW3,
        .config.isPullEnable = false,
        .config.pullSelect = kPortPullUp,
        .config.isPassiveFilterEnabled = false,
        .config.interrupt = kPortIntDisabled,
    },
    {
        .pinName = GPIO_PINS_OUT_OF_RANGE,
    }
};

/* Declare Output GPIO pins */
gpio_output_pin_user_config_t ledPins[] = {
    {
        .pinName = kGpioLED1,
        .config.outputLogic = 1,
        .config.slewRate = kPortSlowSlewRate,
        .config.driveStrength = kPortLowDriveStrength,
    },
    {
        .pinName = kGpioLED2,
        .config.outputLogic = 1,
        .config.slewRate = kPortSlowSlewRate,
        .config.driveStrength = kPortLowDriveStrength,
    },
    {
        .pinName = kGpioLED3,
        .config.outputLogic = 1,
        .config.slewRate = kPortSlowSlewRate,
        .config.driveStrength = kPortLowDriveStrength,
    },
    {
        .pinName = GPIO_PINS_OUT_OF_RANGE,
    }
};
```

GPIO Driver Uses Those Defines

- GPIO_DRV_OutputPinInit(&ledPins[0]); //Init
- GPIO_DRV_WritePinOutput(kGpioLED1, 1); //Turn On
- GPIO_DRV_WritePinOutput(kGpioLED1, 0); //Turn Off

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

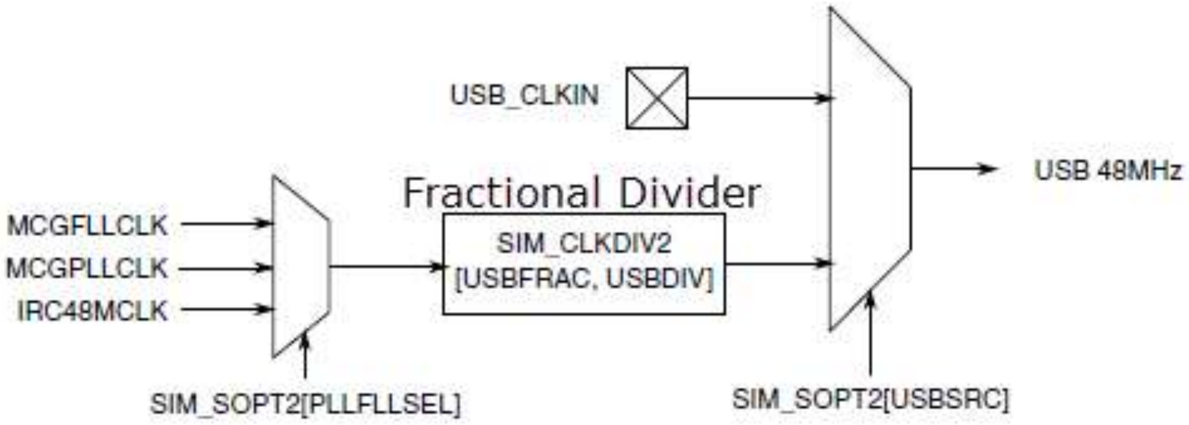


Figure 5-7. USB 48 MHz clock source

PEB

- Processor Expert Configuration Files – PEB
- Found for each board along with the other board files: C:\Freescale\KSDK_1.2.0\examples\<board_name>

Agenda

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

Kinetis SDK 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

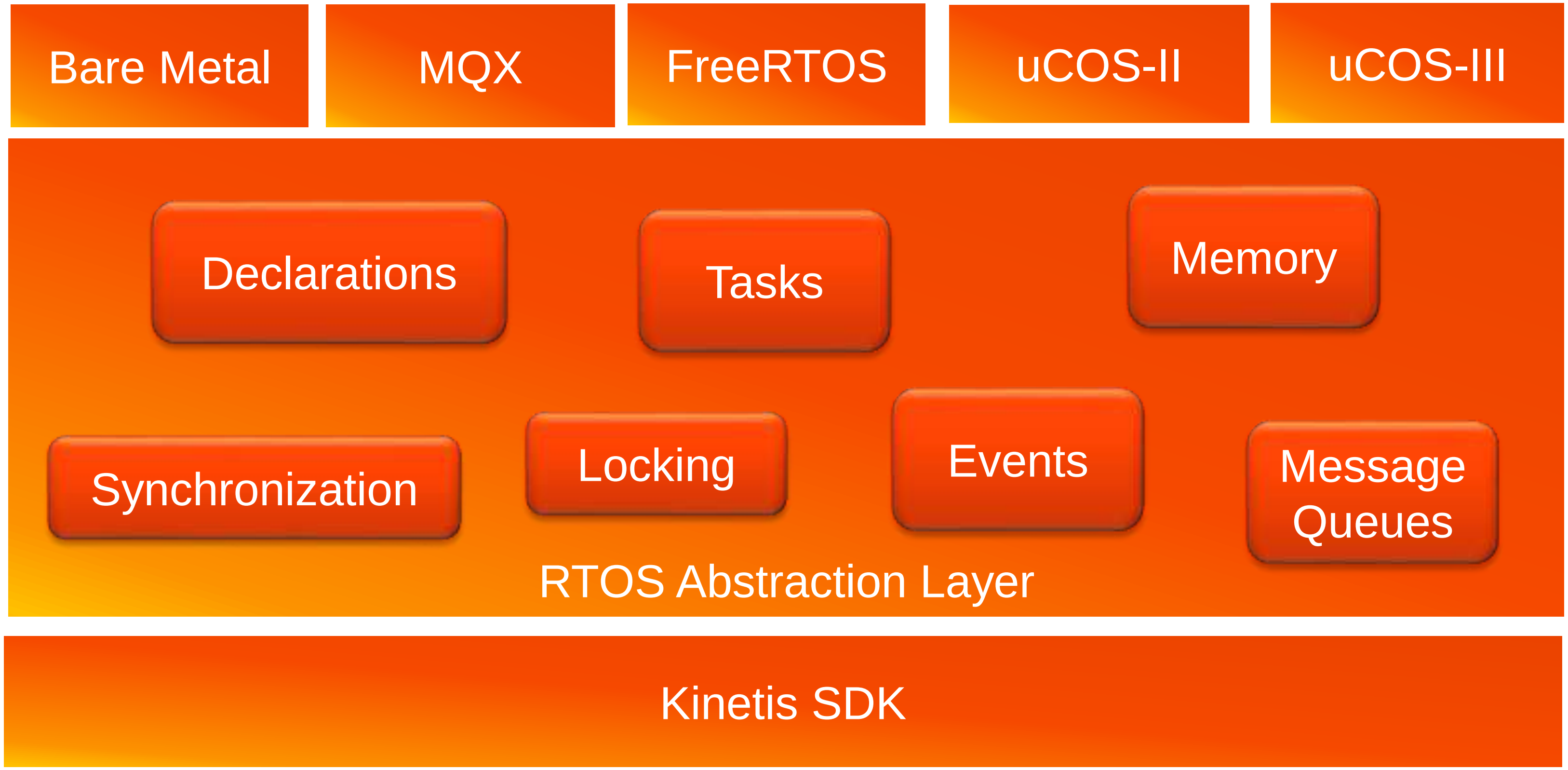
For Embedded Systems that need…

- **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

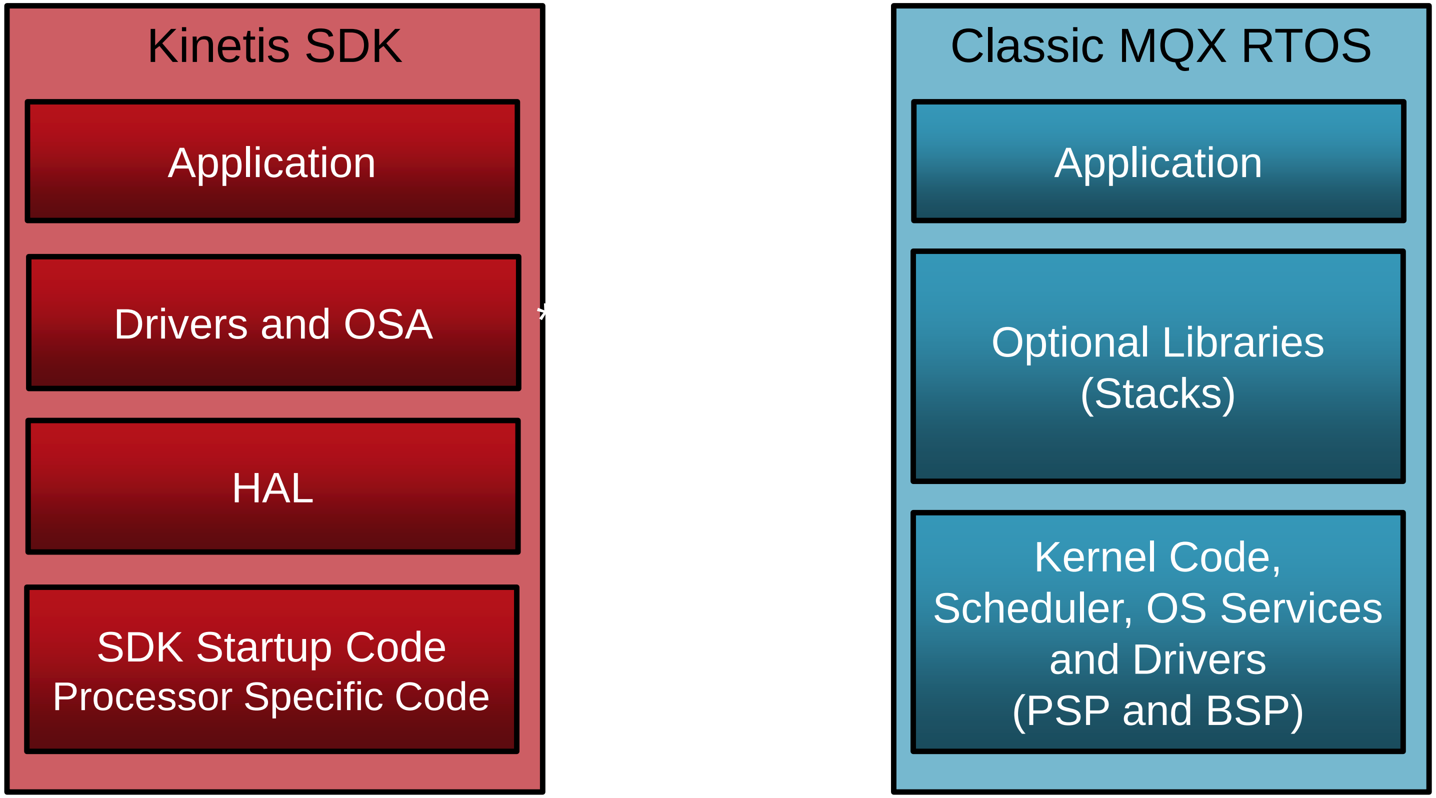
Use an RTOS!

Kinetis SDK RTOS Abstraction

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



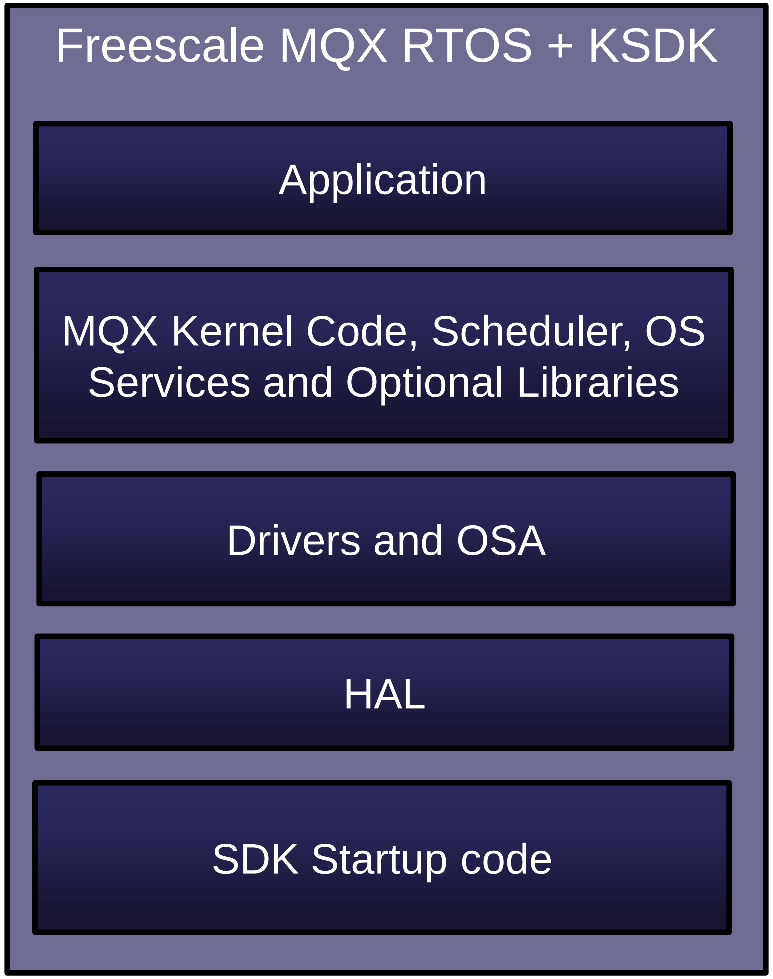
KSDK and RTOS Applications Structure



* 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)



Freescale MQX RTOS + KSDK

Application

MQX Kernel Code, Scheduler, OS Services and Optional Libraries

Drivers and OSA

HAL

SDK Startup code

Classic MQX vs MQX for KSDK

Classic MQX™ RTOS

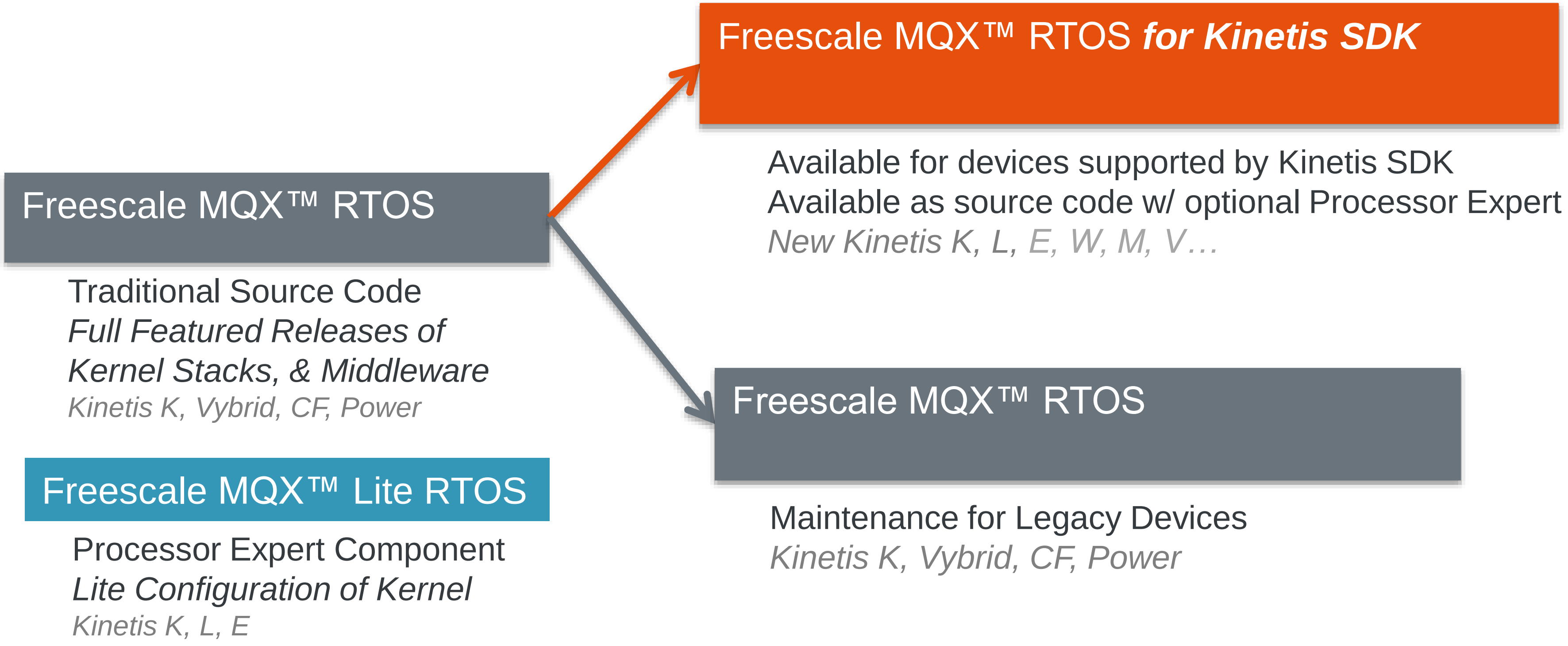
- Is a full-featured complimentary Real-Time Operating System
  - Developed by Freescale as a software solution for Freescale devices
  - Provides real-time performance within a small, configurable footprint
- Includes
  - MQX™ Kernel (PSP)
  - Board Support Package (BSP)
  - Implements its own peripheral 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.

Learn more at: www.freescale.com/mqx

Evolution of MQX RTOS



Freescale MQX™ RTOS

Traditional Source Code
Full Featured Releases of
Kernel Stacks, & Middleware
Kinetis K, Vybrid, CF, Power

Freescale MQX™ Lite RTOS

Processor Expert Component
Lite Configuration of Kernel
Kinetis K, L, E

Freescale MQX™ RTOS ***for Kinetis SDK***

Available for devices supported by Kinetis SDK
Available as source code w/ optional Processor Expert
New Kinetis K, L, E, W, M, V…

Freescale MQX™ RTOS

Maintenance for Legacy Devices
Kinetis K, Vybrid, CF, Power

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_1.2.0\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);

Agenda

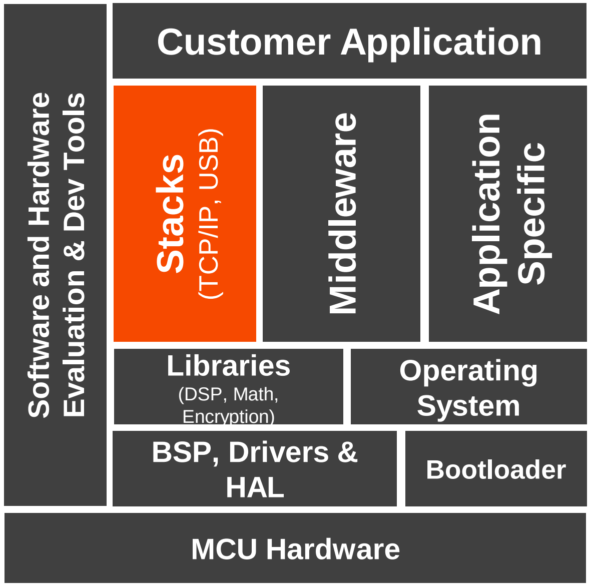
- KSDK In-Depth
  - Lab
- KSDK + RTOS
- KSDK + USB
- KSDK + Processor Expert
  - Lab
- Conclusion

Kinetis MCU 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.



Customer Application

Software and Hardware Evaluation & Dev Tools

Stacks (TCP/IP, USB)

Middleware

Application Specific

Libraries (DSP, Math, Encryption)

Operating System

BSP, Drivers & HAL

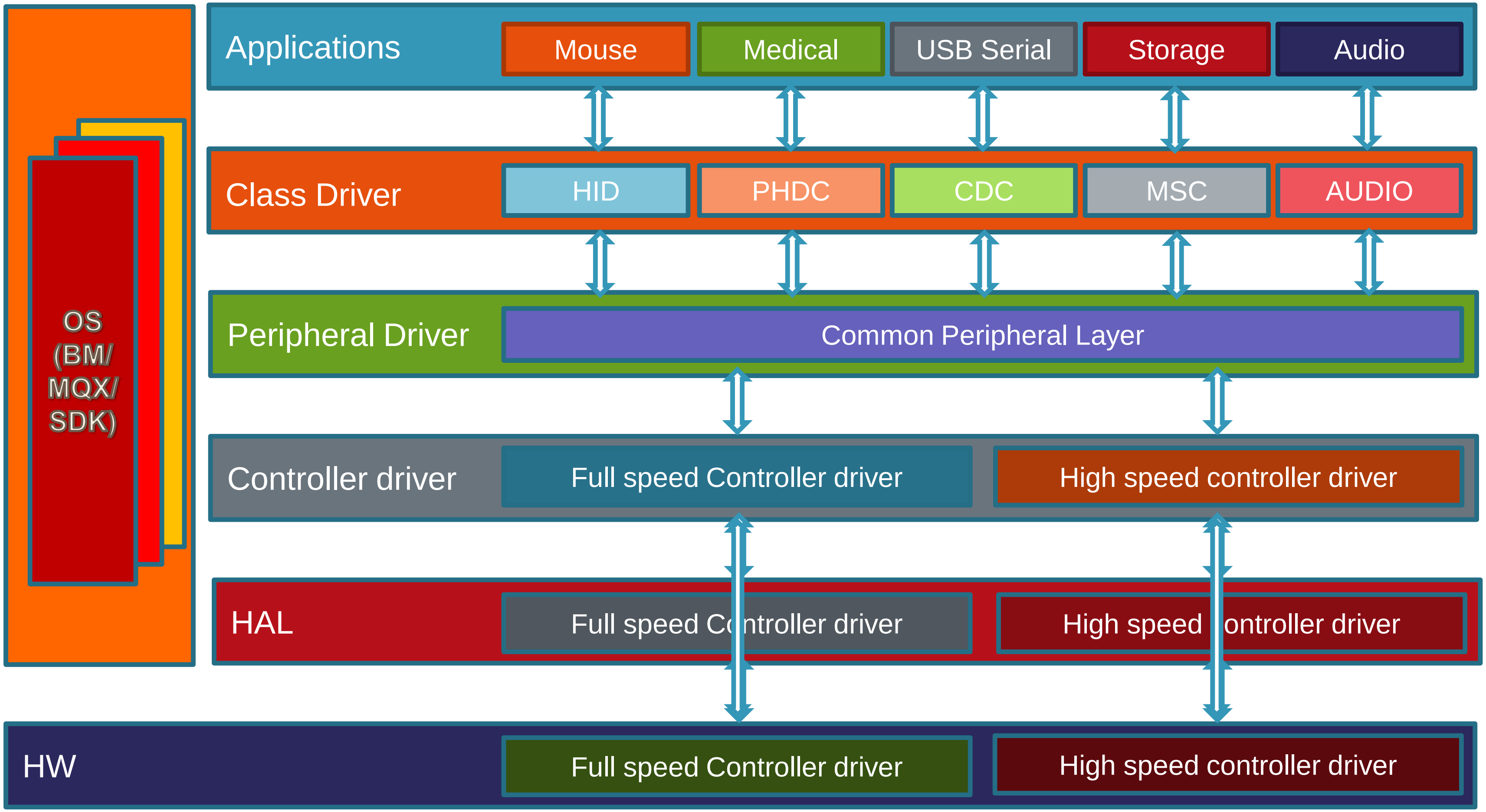
Bootloader

MCU Hardware

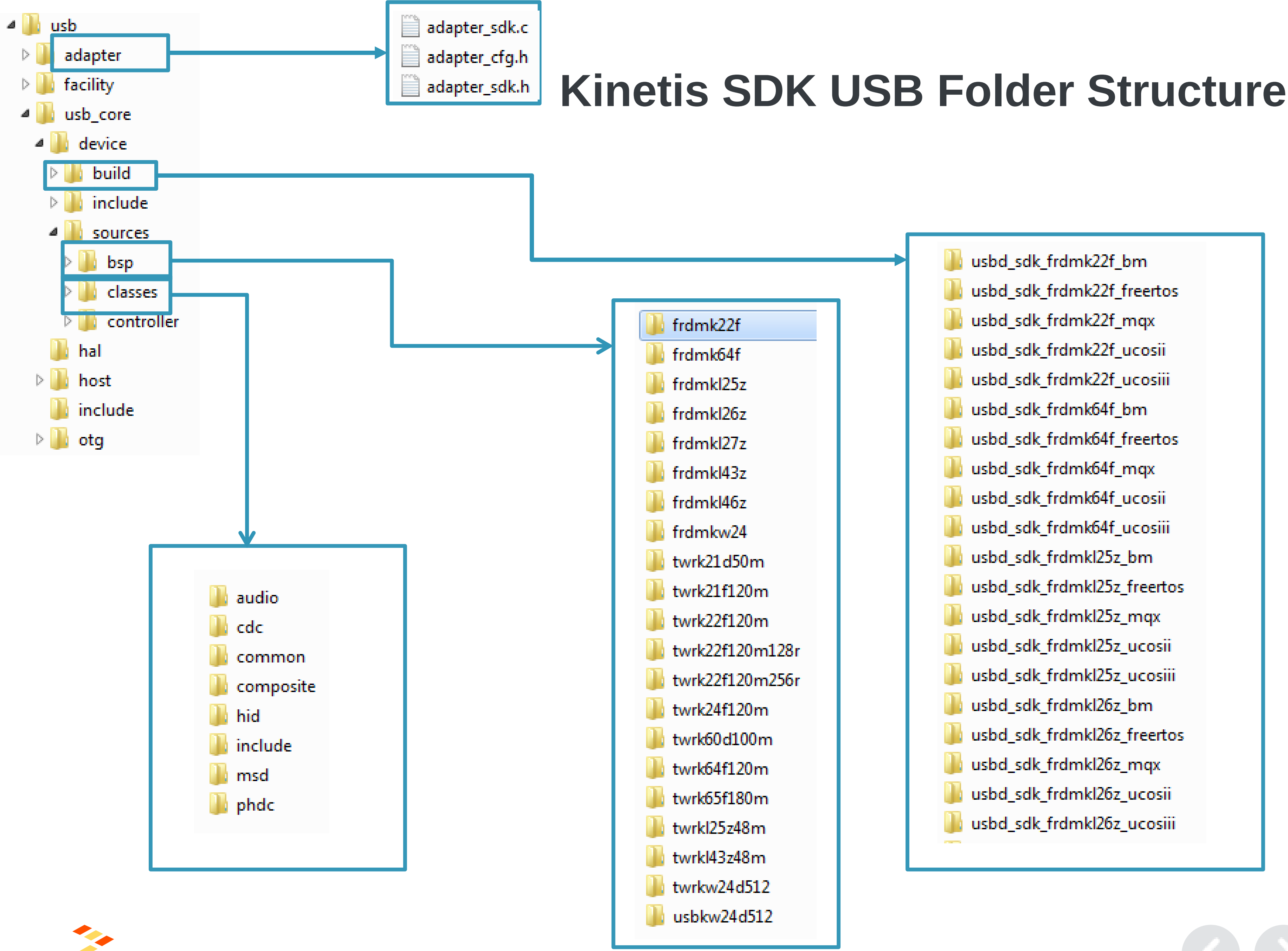
Product Features

- 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


OS
(BM/
MQX/
SDK)
Applications
Mouse
Medical
USB Serial
Storage
Audio
Class Driver
HID
PHDC
CDC
MSC
AUDIO
Peripheral Driver
Common Peripheral Layer
Controller driver
Full speed Controller driver
High speed controller driver
HAL
Full speed Controller driver
High speed controller driver
HW
Full speed Controller driver
High speed controller driver

Kinetis SDK USB Folder Structure



```
usb
  adapter
  facility
  usb_core
    device
      build
      include
      sources
        bsp
        classes
        controller
    hal
    host
    include
    otg
```

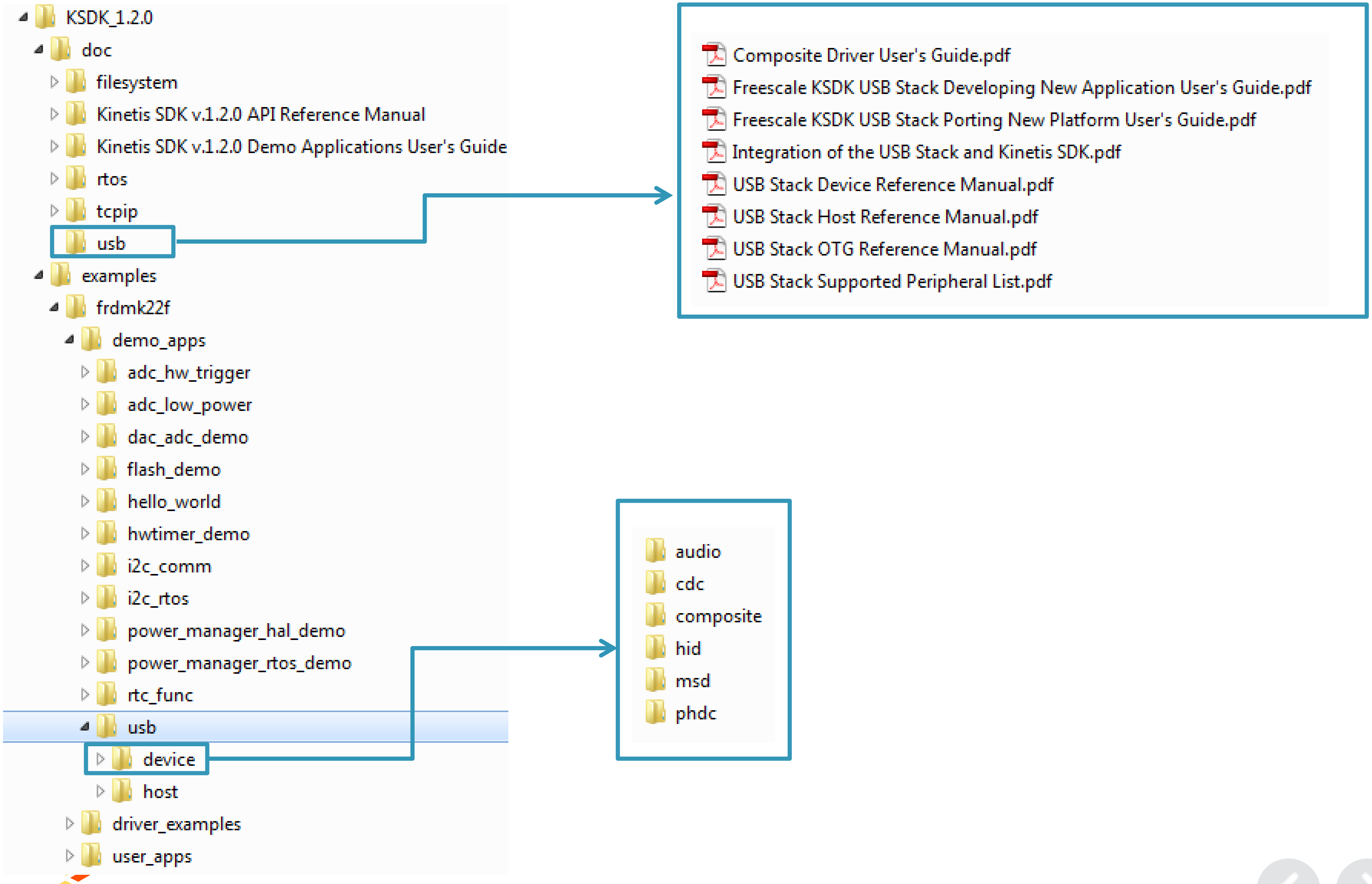
- adapter_sdk.c
- adapter_cfg.h
- adapter_sdk.h

- usbd_sdk_frdmk22f_bm
- usbd_sdk_frdmk22f_freertos
- usbd_sdk_frdmk22f_mqx
- usbd_sdk_frdmk22f_ucosii
- usbd_sdk_frdmk22f_ucosiii
- usbd_sdk_frdmk64f_bm
- usbd_sdk_frdmk64f_freertos
- usbd_sdk_frdmk64f_mqx
- usbd_sdk_frdmk64f_ucosii
- usbd_sdk_frdmk64f_ucosiii
- usbd_sdk_frdmkl25z_bm
- usbd_sdk_frdmkl25z_freertos
- usbd_sdk_frdmkl25z_mqx
- usbd_sdk_frdmkl25z_ucosii
- usbd_sdk_frdmkl25z_ucosiii
- usbd_sdk_frdmkl26z_bm
- usbd_sdk_frdmkl26z_freertos
- usbd_sdk_frdmkl26z_mqx
- usbd_sdk_frdmkl26z_ucosii
- usbd_sdk_frdmkl26z_ucosiii

- frdmk22f
- frdmk64f
- frdmkl25z
- frdmkl26z
- frdmkl27z
- frdmkl43z
- frdmkl46z
- frdmkw24
- twrk21d50m
- twrk21f120m
- twrk22f120m
- twrk22f120m128r
- twrk22f120m256r
- twrk24f120m
- twrk60d100m
- twrk64f120m
- twrk65f180m
- twrkl25z48m
- twrkl43z48m
- twrkw24d512
- usbkw24d512

- audio
- cdc
- common
- composite
- hid
- include
- msd
- phdc

Kinetis SDK USB Examples and Documentation



- KSDK_1.2.0
  - doc
    - filesystem
    - Kinetis SDK v.1.2.0 API Reference Manual
    - Kinetis SDK v.1.2.0 Demo Applications User's Guide
    - rtos
    - tcpip
    - usb
  - examples
    - frdmk22f
      - demo_apps
        - adc_hw_trigger
        - adc_low_power
        - dac_adc_demo
        - flash_demo
        - hello_world
        - hwtimer_demo
        - i2c_comm
        - i2c_rtos
        - power_manager_hal_demo
        - power_manager_rtos_demo
        - rtc_func
        - usb
          - device
          - host
      - driver_examples
      - user_apps

- Composite Driver User's Guide.pdf
- Freescale KSDK USB Stack Developing New Application User's Guide.pdf
- Freescale KSDK USB Stack Porting New Platform User's Guide.pdf
- Integration of the USB Stack and Kinetis SDK.pdf
- USB Stack Device Reference Manual.pdf
- USB Stack Host Reference Manual.pdf
- USB Stack OTG Reference Manual.pdf
- USB Stack Supported Peripheral List.pdf

- audio
- cdc
- composite
- hid
- msd
- phdc

USB Examples

- The USB examples that come with Kinetis SDK require 2 libraries to be built first:
  - Kinetis SDK 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_frdmk22f_bm
  - <ksdk_dir>\examples\frdmk22f\demo_apps\usb\device\msd\bm\kds

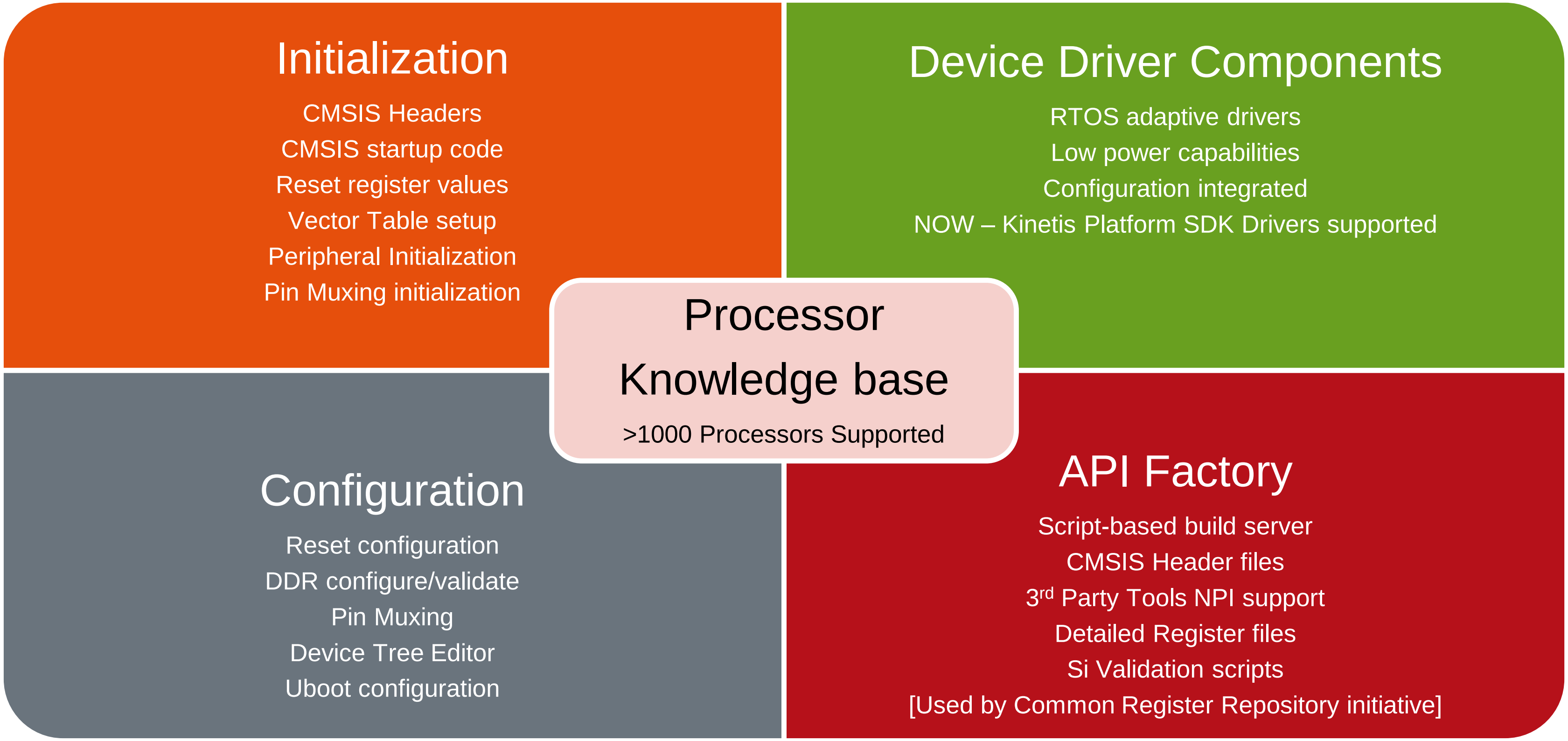
Agenda

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

Processor Expert + KSDK

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

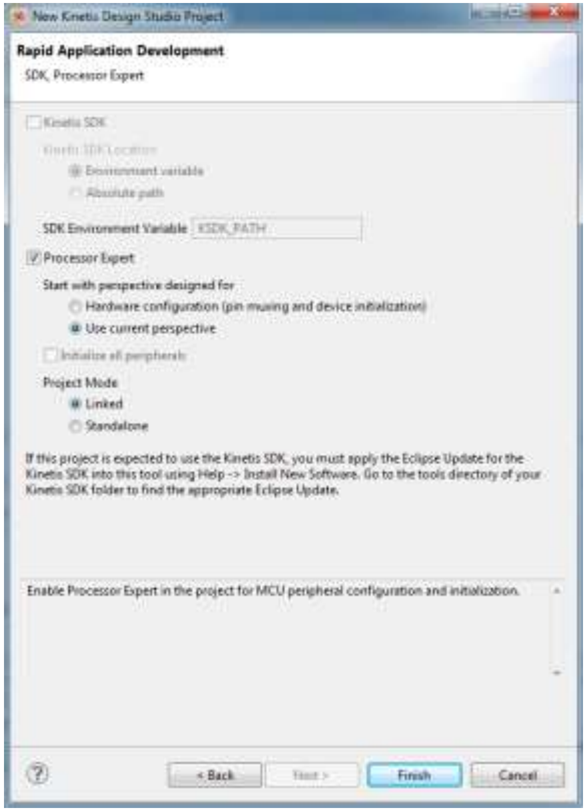
Processor Expert with KSDK

- 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.



Rapid Application Development

SDK, Processor Expert

□ Kinetis SDK

Kinetis SDK Location

● Environment variable

○ Absolute path

SDK Environment Variable: KSDK_PATH

☑ Processor Expert

Start with perspective designed for

○ Hardware configuration (pin muxing and device initialization)

● Use current perspective

□ Initialize all peripherals

Project Mode

● Linked

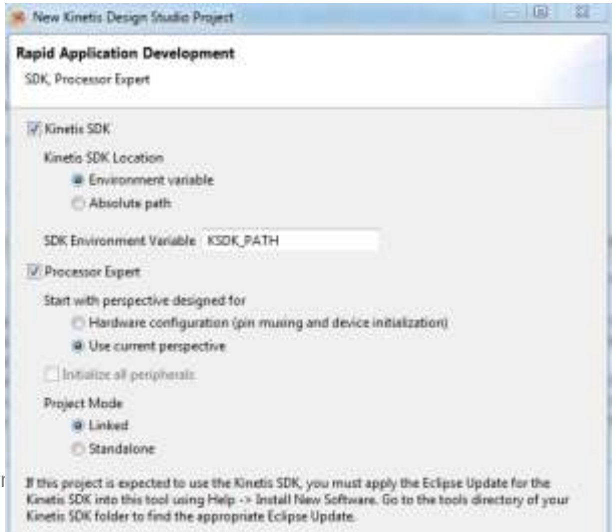
○ Standalone

If this project is expected to use the Kinetis SDK, you must apply the Eclipse Update for the Kinetis SDK into this tool using Help -> Install New Software. Go to the tools directory of your Kinetis SDK folder to find the appropriate Eclipse Update.

Enable Processor Expert in the project for MCU peripheral configuration and initialization.

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.



New Kinetis Design Studio Project

Rapid Application Development

SDK, Processor Expert

☑ Kinetis SDK

Kinetis SDK Location

● Environment variable

○ Absolute path

SDK Environment Variable: KSDK_PATH

☑ Processor Expert

Start with perspective designed for

○ Hardware configuration (pin muxing and device initialization)

● Use current perspective

□ Initialize all peripherals

Project Mode

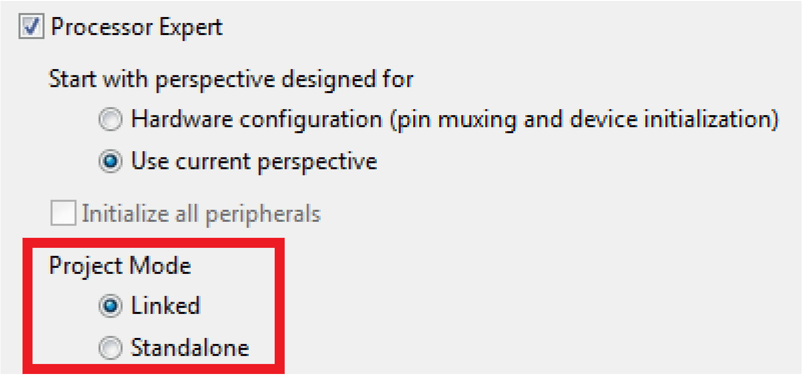
● Linked

○ Standalone

If this project is expected to use the Kinetis SDK, you must apply the Eclipse Update for the Kinetis SDK into this tool using Help -> Install New Software. Go to the tools directory of your Kinetis SDK folder to find the appropriate Eclipse Update.

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



☑ Processor Expert

Start with perspective designed for

○ Hardware configuration (pin muxing and device initialization)

● Use current perspective

□ Initialize all peripherals

Project Mode

● Linked

○ Standalone

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 (v3.0 or newer)
- ❑ Kinetis Software Development Kit (v1.2.0)

Project Definition

- ✓ **Hardware: FRDM-K22F**
- ✓ **Clock Configuration**
  Internal PLL; set to 120MHz
  Bus Clock; 60MHz
  Flash Clock: 20MHz
- ✓ **Pin Muxing**
  GPIO
- ✓ **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 ← **Next up!**
  - Configure Components with the Component Inspector
  - Use Processor Expert Components
  - Add Code
  - Build the project
  - Test the application's functionality
- 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 Kinetis SDK
  - 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