NXP Semiconductors
Application Note

Document Number: AN5332
Rev. 0, 09/2016

Kinetis Migration Guide From KE0x to KE1x

By: Ben Wang

1. Introduction

The Kinetis E family provides a highly scalable portfolio of robust 5-V MCUs ranging from the 20-MHz ARM® Cortex®-M0+ MCUs to the 168-MHz ARM Cortex-M4 MCUs. With a power supply of 2.7 V ~ 5.5 V and the focus on exceptional EMC/ESD robustness, the Kinetis E family is well-suited for a wide range of applications in the harsh electrical environments and optimized for the cost-sensitive applications. The Kinetis E family offers a broad range of the memory, peripheral, and package options.

This application note describes the key differences and improvements between the KE0x and KE1x product families, provides the guideline on how to migrate from one to the other, and helps to shorten the learning curve.

Contents

2. Kinetis E Family Device Overview

2.1. Kinetis KE0xZ sub-family overview

The Kinetis KE0xZ MCUs are based on the Cortex-M0+ core running at up to 48 MHz and provide up to 128 KB flash, 16 KB RAM, and 256 B EEPROM. They include a powerful array of analog, communication, timing, and control peripherals. This family consists of low-power, highly robust, and cost-effective MCUs which provide an appropriate 32-bit entry-level solution.

- KE02Z—a broad offering with the mixed-signal integration, ADCs, DAC, ACMPs, and FlexTimers with the addition of the 256-B EEPROM.
- KE04Z—an expansion of the KE02Z family with the addition of the BME module, but with the EEPROM removed.
- KE06Z—an expansion of the KE04Z family with the addition of the MSCAN module.

2.2. Kinetis KE1xZ sub-family overview

The Kinetis KE1xZ MCUs are based on the Cortex-M0+ core running at up to 72 MHz, providing up to 256 KB flash, 32 KB RAM, and a complete set of the analog/digital features. The KE1xZ extends the Kinetis E family to a higher performance and a broader scalability. The robust TSI provides a high level of stability and accuracy for your HMI system. The 1-MSPS ADC and the FlexTimer are well-suited for the BLDC motor-control systems.

- KE14Z—a broad offering with the mixed-signal integration, ADCs, CMPs, and FlexTimers.
- KE15Z—an expansion of the KE14Z family with the addition of the TSI module.

2.3. Kinetis KE1xF sub-family overview

The Kinetis KE1xF MCUs are the high-end MCUs from the Kinetis E family, providing a robust 5-V solution with the high-performance Cortex-M4 core running at up to 168 MHz. The KE1xF offers multiple ADCs and FlexTimers, the CAN 2.0B-compliant FlexCAN module, and a rich suite of communication interfaces including the UARTs, I^{2}Cs, SPIs, and a FlexIO, which provide flexibility for the serial communication emulation. The devices start from 256 KB flash in the 64LQFP package up to 512 KB flash in the 100LQFP package.

- KE14F—a broad offering with the mixed-signal integration, ADCs, DAC, ACMPs, and FlexTimers.
- KE16F—an expansion of the KE14F family with the addition of the FlexCAN module.
- KE18F—an expansion of the KE14F family with the addition of two FlexCAN modules.

3. Software Enablement Comparsion

3.1. KExx_drivers library for KE0x

The KExx_drivers library are the NXP software drivers working on the FRDM boards for the KE0x family. These software drivers are provided in the form of a source code. All the source files are intended to be included directly into the application project or built separately into a statically-linked library.

The KExx_drivers library is regarded as the legacy of the NXP KE0x family peripheral drivers and sample code. It has the advantage of a common coding style and is easy to use. The software structure is also simple to understand and you can easily customize your drivers and application. The disadvantage is that the coding style is not compatible with the SDK and it is difficult to migrate to the other Kinetis families.

3.2. Kinetis SDK v2.0 for KE1x

The Kinetis Software Development Kit (KSDK) v2.0 is a collection of the software enablement for the KE1x devices that includes the peripheral drivers and the integrated RTOS support for the FreeRTOS OS and the µC/OS. In addition to the base enablement, the KSDK is augmented with the demo applications, the driver example projects, and the API documentation to help quickly leverage the support of the Kinetis SDK. This is the architecture diagram:

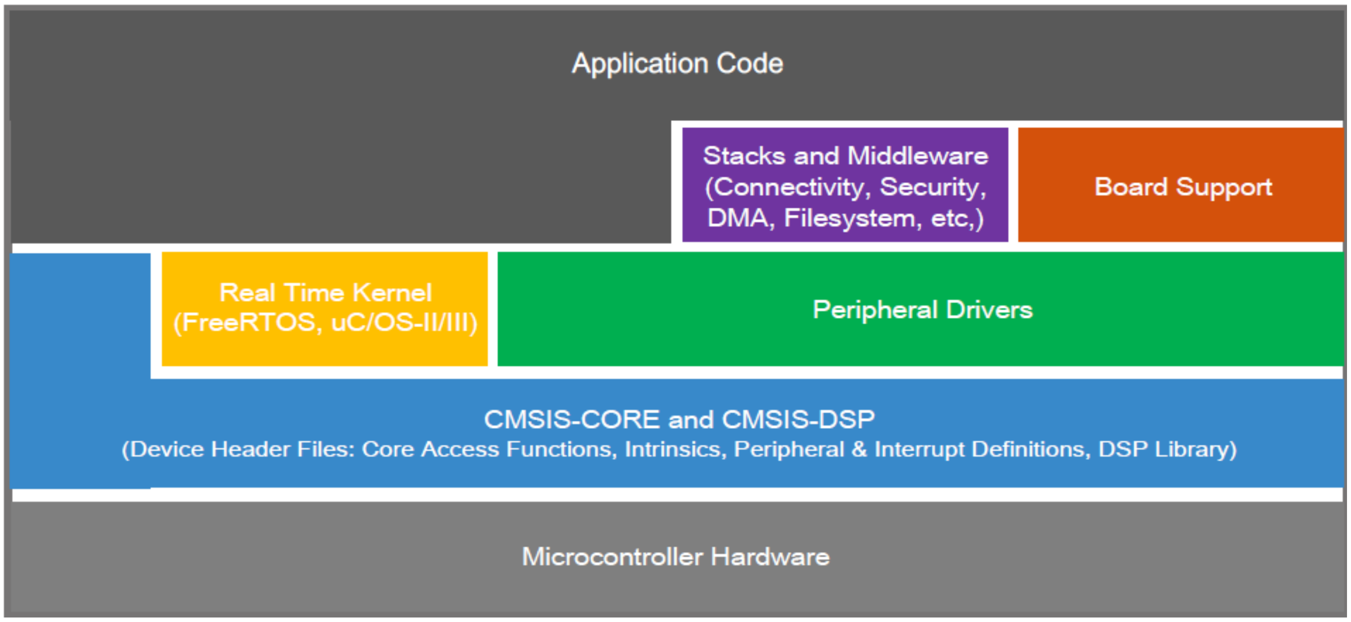
Application Code
Stacks and Middleware (Connectivity, Security, DMA, Filesystem, etc,)
Board Support
Real Time Kernel (FreeRTOS, uC/OS-II/III)
Peripheral Drivers
CMSIS-CORE and CMSIS-DSP
(Device Header Files: Core Access Functions, Intrinsics, Peripheral & Interrupt Definitions, DSP Library)
Microcontroller Hardware

Figure 1. KSDK v2.0 architecture diagram

3.2.1. KSDK board support folders

The KSDK board support provides the example applications for the Kinetis development and evaluation boards (Tower System modular development platform/NXP Freedom and other). The board support packages are located inside the top-level *boards* folder, and each supported board has its own folder (one KSDK package can support multiple boards). The various subfolders within each <*board_name*> folder classify the type of examples they contain. These folders include (but are not limited to):

- *demo_apps*—full-featured applications intended to highlight the key functionality and use cases of the target MCU. These applications typically use multiple MCU peripherals and may leverage the stacks and middleware.
- *driver_examples*—simple applications intended to concisely illustrate how to use the KSDK's peripheral drivers for a single-use case. These applications typically use only one peripheral, but there are cases where more peripherals are used (for example, the ADC conversion using the DMA).
- *rtos_examples*—basic FreeRTOS OS examples showcasing the use of various RTOS objects (semaphores, queues, and other) and interfacing with the KSDK's RTOS drivers.

3.2.2. Example application structure

Each <*board_name*> sub-folder in the *boards* folder contains a comprehensive set of examples that are relevant to the specific piece of hardware. Take the *hello_world* folder as an example (part of the *demo_apps* folder), but the same general rules apply to any type of example in the <*board_name*> folder.

This is the contents of the *hello_world* application folder:

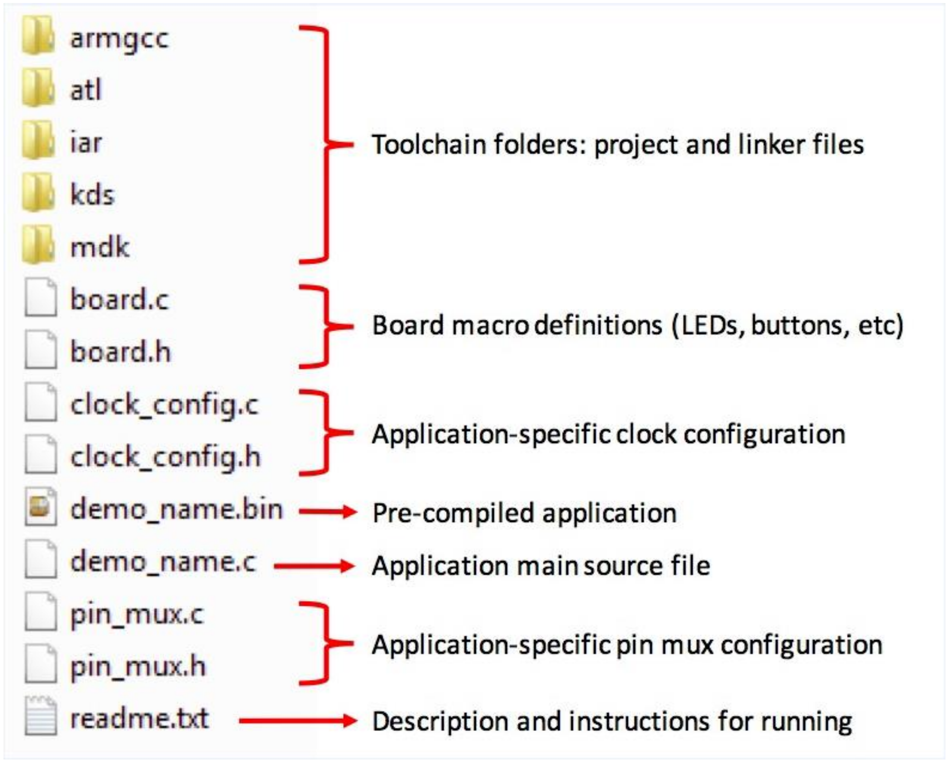
armgcc
atl
iar
kds
mdk
Toolchain folders: project and linker files
board.c
board.h
Board macro definitions (LEDs, buttons, etc)
clock_config.c
clock_config.h
Application-specific clock configuration
demo_name.bin
Pre-compiled application
demo_name.c
Application main source file
pin_mux.c
pin_mux.h
Application-specific pin mux configuration
readme.txt
Description and instructions for running

Figure 2. Application folder structure

All files in the application folder are specific to the example, so it is very easy to copy and paste an existing example to start developing a custom application based on a project provided in the KSDK.

3.2.3. Locating example application source files

When opening an example application in any of the supported IDEs, a variety of source files is referenced. The KSDK *devices* folder is the core component of the application and the central component of all example applications. Because it is a core component, all of the examples reference the same source files and it could potentially impact the behavior of other examples if one of these files is modified.

The main areas of the KSDK tree used in all example applications are:

- *devices/<device_name>*—the device CMSIS header file, the KSDK feature file, and some other things.
- *devices/<device_name>/drivers*—all peripheral drivers for the specific MCU.
- *devices/<device_name>/<tool_name>*—the toolchain-specific startup code. The vector table definitions are located here.
- *devices/<device_name>/utilities*—the items used by many of the example applications (such as the debug console).

For examples that contain the middleware/stacks and/or a RTOS, there are references to the appropriate source code. The middleware source files are in the *middleware* folder and the RTOSes are in the *rtos* folder. Again, the core files of each of these folders are shared, so modifying them can have potential impacts on the other projects that depend on them.

For details, see the Kinetis SDK homepage (www.nxp.com/ksdk). All code snippets in the following sections are based on the KSDK v2.0.

4. Hardware Resources Comparison

4.1. System level differences

Because the Kinetis E product family is built on different processor cores and meant for different purposes and application fields, there is a significant number of differences between the KE0x and KE1x MCUs. This table outlines the system level differences at a high level:

Table 1. System level differences

Feature	KE0xZ	KE1xZ	KE1xF
Processor core	Cortex-M0+	Cortex-M0+	Cortex-M4
Max. CPU frequency	48 MHz	72 MHz	168 MHz
DSP and FPU	—	Yes	Yes
TRGMUX	—	Yes	Yes
Debug	SWD	SWD	JTAG + SWD
Flash size	Up to 128 KB	Up to 256 KB	Up to 512 KB with ECC
SRAM	Up to 16 KB	Up to 32 KB	Up to 64 KB with ECC
EEPROM or FlexRAM	Up to 256 B	Up to 2 KB	Up to 4 KB
FlexNVM	—	Up to 32 KB	Up to 64 KB
BOOT ROM	—	Yes	Yes
GPIO	Up to 71 I/Os	Up to 89 I/Os	Up to 89 I/Os

4.1.1. Clocking strategy

NOTE

The differences in the clocking strategy can significantly affect the clock startup and the configuration of your application.

The KE0x contains these on-chip clock sources:

- Internal Clock Source (ICS) module—the main clock-source generator providing the bus clock and other reference clocks to the peripherals.
- System Oscillator (OSC) module—the system oscillator providing the reference clock to the ICS, the Real-Time Clock (RTC) counter clock module, and other MCU sub-systems.
- Low-Power Oscillator (LPO) module—the on-chip low-power oscillator providing the 1-kHz reference clock to the RTC and the Watchdog (WDOG).

The KE0x clock diagram is shown in the following figure. The clock for each module can be individually gated on and off using the SIM_SCGC register. Prior to initializing a module, set the corresponding bit in the SIM_SCGC register to enable the clock. Before turning the clock off, make sure to disable the module first. Any bus access to a peripheral that has its clock disabled generates an error termination.

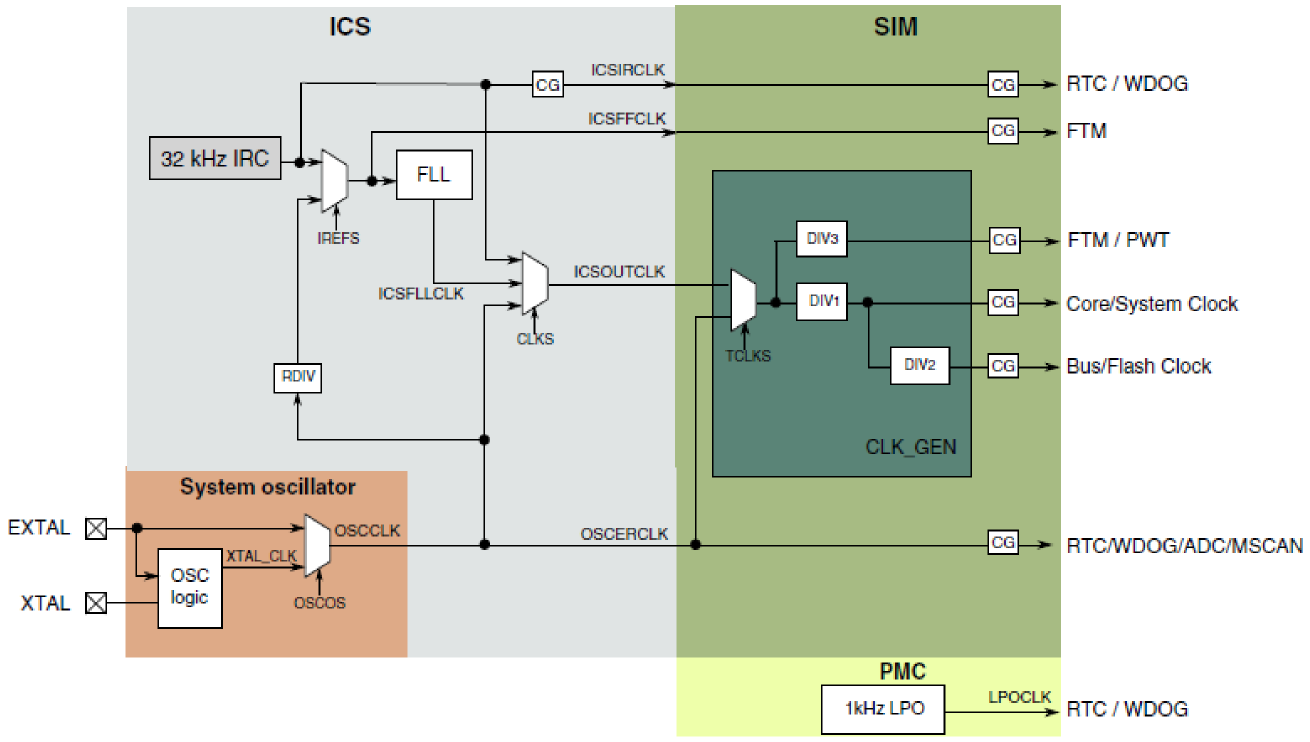

Figure 3. KE0x clock diagram

In the KE1x family, a new revised System Clock Generation (SCG) module is deployed to create various clock trees. This device has several clock sources, namely the external system clock/crystal, the external RTC clock/crystal, the internal Fast IRC (FIRC) oscillator, the internal Slow IRC (SIRC) oscillator and the 128-kHz Low Power Oscillator (LPO). The SCG module incorporates the external system clock/crystal circuit, the FIRC, and the SIRC.

The external RTC clock/crystal is used as the clock source for the RTC module. The LPO clock source is used as an optional peripheral clock source only for some modules (for example, WDOG, LPTMR, and others) and is not used by the SCG module to create alternative clock frequencies. The three other clock sources are used by the SCG to provide the CPU/Platform clock, the Bus/Flash clock, and the alternative Peripheral clock. The PLL (in KE1xF) or LPFLL (in KE1xZ) are included to generate the CPU/Platform frequencies up to the maximum frequency.

The following figure shows the KE1xF clock diagram. There are some differences when compared to the KE1xZ diagram, mainly the PLL block being replaced by the LPFLL. In the KE1x family, the clock gate control, clock source selection, and clock divider for each module use the dedicated PCC module instead of the SIM module. See the *Clock management and distribution in KL28* (document AN5231) for details about the SCG and PCC modules.

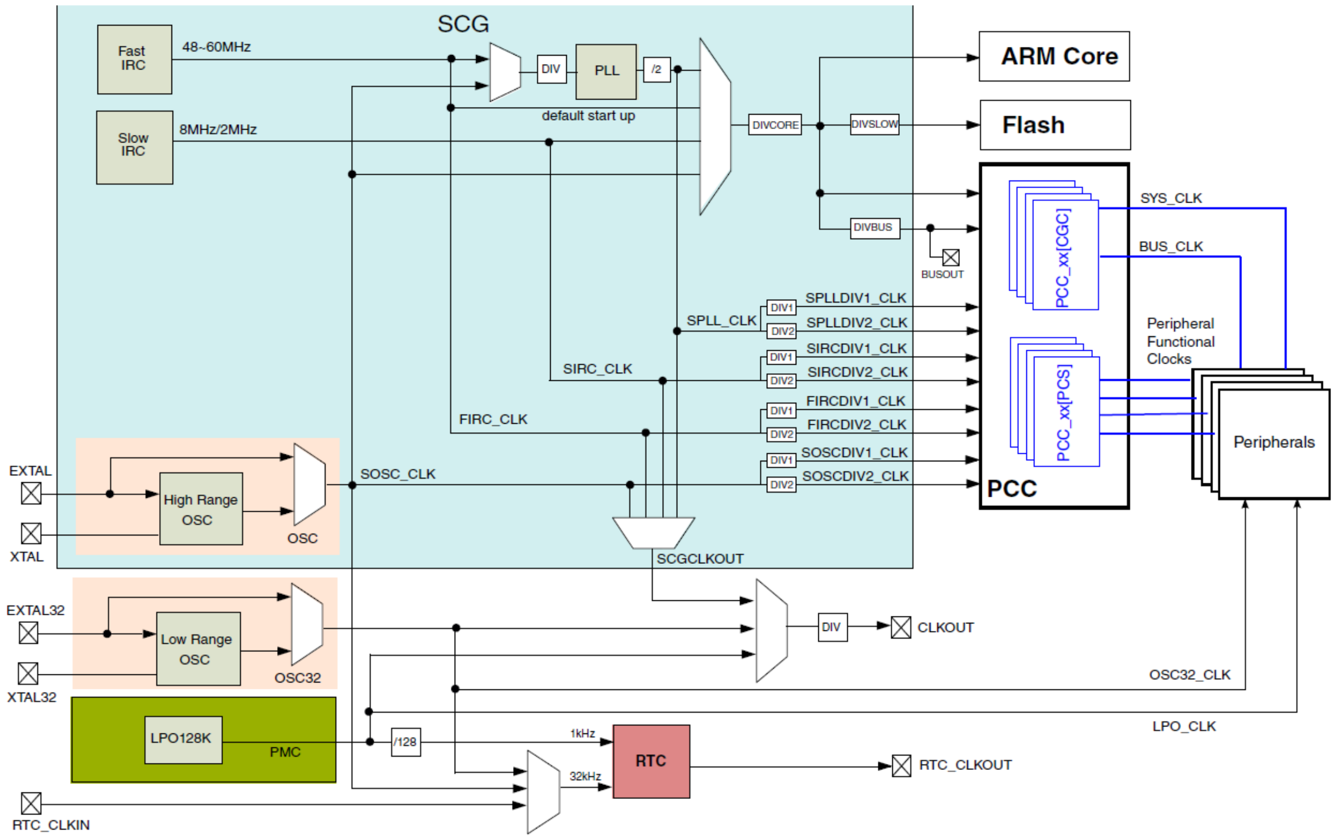

Figure 4. KE1xF clock diagram

The SDK v2.0 driver/example code for the KE1x clock includes:

- The board startup to put the MCU into the RUN/HSRUN mode. The FIRC, SIRC, and SYSOSC clocks are enabled. You may configure the core/bus clock frequency and the divider manually. By default, the core clock runs up to the maximum speed in the HSRUN mode.

```
void BOARD_BootClockRUN(void);
```

```
void BOARD_BootClockHSRUN(void);
```

- The module clock gate control. The `name` parameter is the peripherals’ clock name definition.

```
void CLOCK_EnableClock(clock_ip_name_t name);
```

```
void CLOCK_DisableClock(clock_ip_name_t name);
```

- The module clock source option and the divider setting. The `name` parameter is the peripherals’ clock name definition, the `src` parameter is the clock source definition. It may come from the SIRC, FIRC, SYSOSC, LPFLL/PLL, and other. The `divValue` parameter is the divider value, the `fracValue` parameter is the fraction multiply value.

```
void CLOCK_SetIpSrc(clock_ip_name_t name, clock_ip_src_t src);
```

```
void CLOCK_SetIpSrcDiv(clock_ip_name_t name, clock_ip_src_t src, uint8_t divValue,
uint8_t fracValue);
```

4.1.2. Power mode strategy

The KE0x Power Management Controller (PMC) provides multiple power options. The different modes of operation enable you to optimize the power consumption for the level of functionality needed. It supports the Run, Wait, and Stop modes which are easy to use both from the different power consumption level and the functional requirement. The I/O states are held in all modes.

- Run mode—the CPU clocks can run at a full speed and the internal supply is fully regulated.
- Wait mode—the CPU shuts down to save energy. The system clock and the bus clock run and a full regulation is maintained.
- Stop mode—the LVD is optionally enabled and the voltage regulator is in the standby mode.

Table 2. KE0x power modes

Power mode	Description	Core mode	Normal recovery method
Run	Allows the maximum performance of the chip. This is the default mode after the reset. The on-chip voltage regulator is on.	Run	—
Wait	Allows the peripherals to function while the core is in the Sleep mode reducing the power consumption. The NVIC is sensitive to the interrupts. The peripherals are clocked.	Sleep	Interrupt
Stop	Places the chip into the static state. It is the lowest-power mode that retains all registers while optionally maintaining the LVD protection. The NVIC is disabled. The AWIC is used to wake up from the interrupt. The peripheral clocks are stopped.	Deep sleep	Interrupt

When compared to the KE0x, the KE1x PMC has two regulation modes: the normal regulation and the low-power regulation. It provides more power modes for energy saving and different application usage. The normal regulation mode includes the user-configurable overdrive option for the High-Speed Run (HSRUN) mode.

The power modes in the KE1x are as follows:

Normal regulation:

- HSRUN (KE1xF only)—maximum 168-MHz core clock.

- RUN—maximum 120-MHz core clock (KE1xF)/maximum 72-MHz core clock (KE1xZ).
- WAIT—CPU sleep mode.
- STOP—CPU deep sleep mode; the low-power peripherals are optionally on.

Low-power regulation:

- VLPR—very low-power run mode, maximum 4-MHz core clock.
- VLPW—very low-power wait mode, maximum 1-MHz bus clock.
- VLPS—very low-power stop mode, the low-power peripherals are optionally on.

There are three low-power (VLPx) modes available on the KE1x. The regulation is in the low-power mode to save power and the clocks are limited to a maximum of 4 MHz from the SIRC or the external oscillator. The RTC, LPTimer, ADC, DAC, and CMP are optionally active in the VLPx modes. The ADC can be optionally on in the VLPx mode, but with a limited performance because the available clock source is the SIRC or the OSC (a 16-MHz crystal is the maximum).

Table 3. KE1x power modes

Power mode	Description	Core mode	Normal recovery method
Run	Default mode after the reset; the on-chip voltage regulator is on.	Run	—
High-Speed Run (KE1xF only)	Allows the maximum performance of the chip. The on-chip voltage regulator is on, but with a slightly elevated voltage output. In this state, the MCU is able to operate at a higher frequency when compared to the normal Run mode.	Run	—
Wait	Allows the peripherals to function while the core is in the sleep mode reducing the power consumption. The NVIC is sensitive to the interrupts. The peripherals are clocked.	Sleep	Interrupt
Stop	Places the chip into the static state. It is the lowest-power mode that retains all registers while maintaining the LVD protection. The NVIC is disabled. The AWIC is used to wake up from the interrupt. The peripheral clocks are stopped.	Deep Sleep	Interrupt
VLPR (Very Low-Power Run)	The on-chip voltage regulator is in a low-power mode that supplies only the power needed to run the chip at a reduced frequency. It is the reduced-frequency flash access mode (1 MHz). The LVD is off. The internal oscillator provides a low-power 4-MHz source for the core, the bus, and the peripheral clocks.	Run	—
VLPW (Very Low-Power Wait)	Same as the VLPR but with the core in the sleep mode to further reduce the power consumption. The NVIC remains sensitive to the interrupts (FCLK = ON). The on-chip voltage regulator is in a low-power mode that supplies only the power needed to run the chip at a reduced frequency.	Sleep	Interrupt
VLPS (Very Low-Power Stop)	Places the chip into the static state with the LVD operation off. It is the lowest-power mode with the ADC and the pin interrupts functional. The peripheral clocks are stopped, but the LPTMR, RTC, CMP, and DAC can be used. The NVIC is disabled (FCLK = OFF). The AWIC is used to wake up from the interrupt. The on-chip voltage regulator is in a low-power mode that supplies only the power needed to run the chip at a reduced frequency. The whole SRAM operates (the content is retained and the I/O states are held).	Deep Sleep	Interrupt

The SDK v2.0 driver/example code for the KE1x power mode configuration is as follows:

- Configure the MCU to different power modes. For the normal Stop mode setting, there is the `option` extended parameter to select the partial Stop mode.

```
status_t SMC_SetPowerModeRun(SMC_Type *base);
status_t SMC_SetPowerModeHsrun(SMC_Type *base);
status_t SMC_SetPowerModeWait(SMC_Type *base);
status_t SMC_SetPowerModeStop(SMC_Type *base, smc_partial_stop_option_t option);
status_t SMC_SetPowerModeVlpr(SMC_Type *base);
status_t SMC_SetPowerModeVlpw(SMC_Type *base);
status_t SMC_SetPowerModeVlps(SMC_Type *base);
```

4.1.3. Module interconnectivity

In the KE0x family, the module-to-module interconnections are fixed and cannot be re-routed. You may only do limited configuration using the SIM register.

The module interconnectivity in the KE1x scheme is based on the TRGMUX (shown in Figure 5) which is a newly designed module different from the existing Kinetis modules. The TRGMUX introduces an extremely flexible methodology of connecting various trigger sources to multiple pins/peripherals.

The TRGMUX is originally designed for a multi-core architecture, but it also supports a single-core architecture. It provides a very flexible solution when compared to the previous SIM-based solution. The TRGMUX design adds the TRGMUX1 level pre-trigger source for the TRGMUX0. The TRGMUX1 supports up to 32 trigger sources and has eight outputs. These eight outputs are the trigger inputs of the TRGMUX0. The TRGMUX0 supports up to 32 input sources, and its output are the target modules.

With the TRGMUX, each peripheral that accepts the external triggers usually has one specific 32-bit trigger-control register. Each control register supports up to four triggers, and each trigger can be selected from up to 32 inputs.

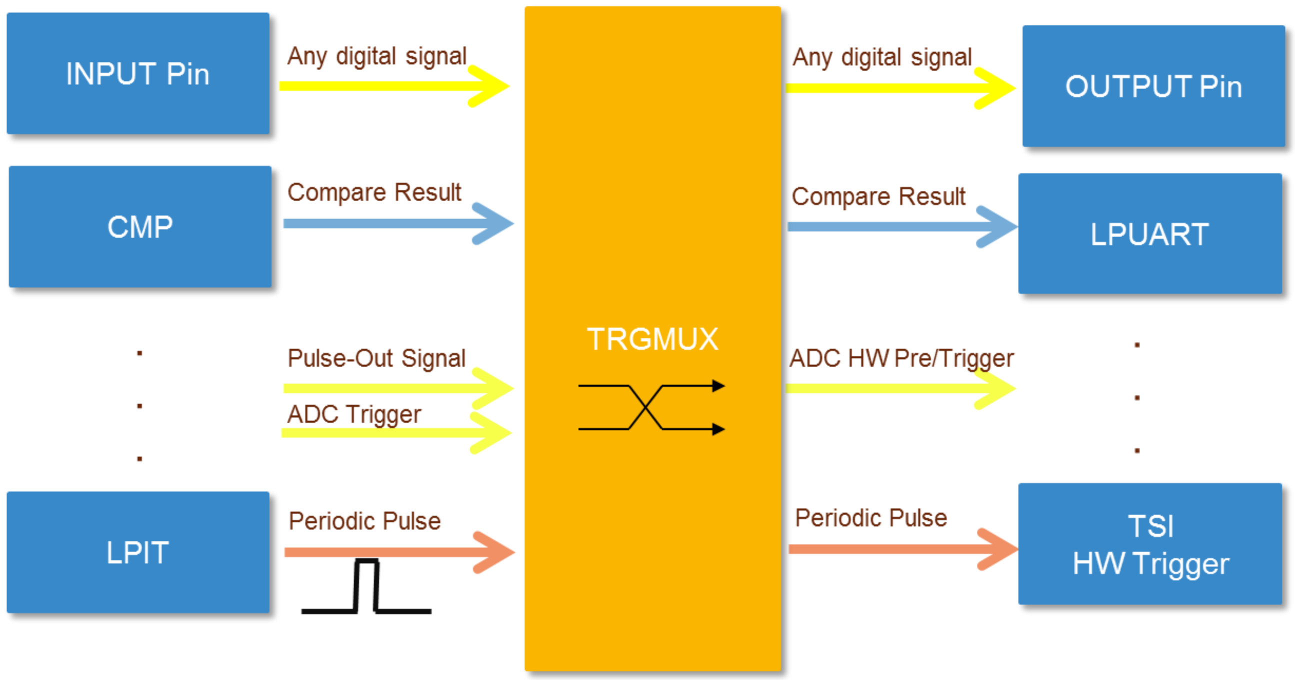

Figure 5. KE1x TRGMUX trigger scheme

The SDK v2.0 driver/example code for the KE1x TRGMUX configuration is as follows:

- Configure the MCU peripherals’ trigger source input. The `index` parameter selects the TRGMUX device collections. The `input` parameter selects the trigger input channel. The `trigger_src` parameter contains all peripheral trigger source options;

```
status_t TRGMUX_SetTriggerSource(TRGMUX_Type *base, trgmux_device_t index,
                                                    trgmux_trigger_input_t input,
trgmux_source_t trigger_src);
```

4.2. System level enhancement

To enhance safety, performance, and efficiency, the KE1x family provides more advanced modules and features than the KE0x, such as the flash/SRAM ECC, I/D cache, MPU and FAC, MMDVSQ, and eDMA, as shown in this table:

Table 4. System level enhancement

Feature	KE0xZ	KE1xZ	KE1xF
I/D cache	—	—	8 KB
Flash ECC	—	—	Yes
SRAM ECC	—	—	Yes
MPU	—	—	Yes
FAC	—	Yes	Yes
eDMA	—	8-channel	16-channel
MMDVSQ	—	Yes	—

4.2.1. Flash/SRAM ECC

- SRAM ECC—8-bit data with a 5-bit ECC, detection and correction of up to a 1-bit error, detect out up to a 2-bit error, and support of the ECC bits' error self-check.
- Flash ECC—64-bit data with 8-bit ECC, detection and correction of up to a 1-bit error, and support of the ECC bits' error self-check.

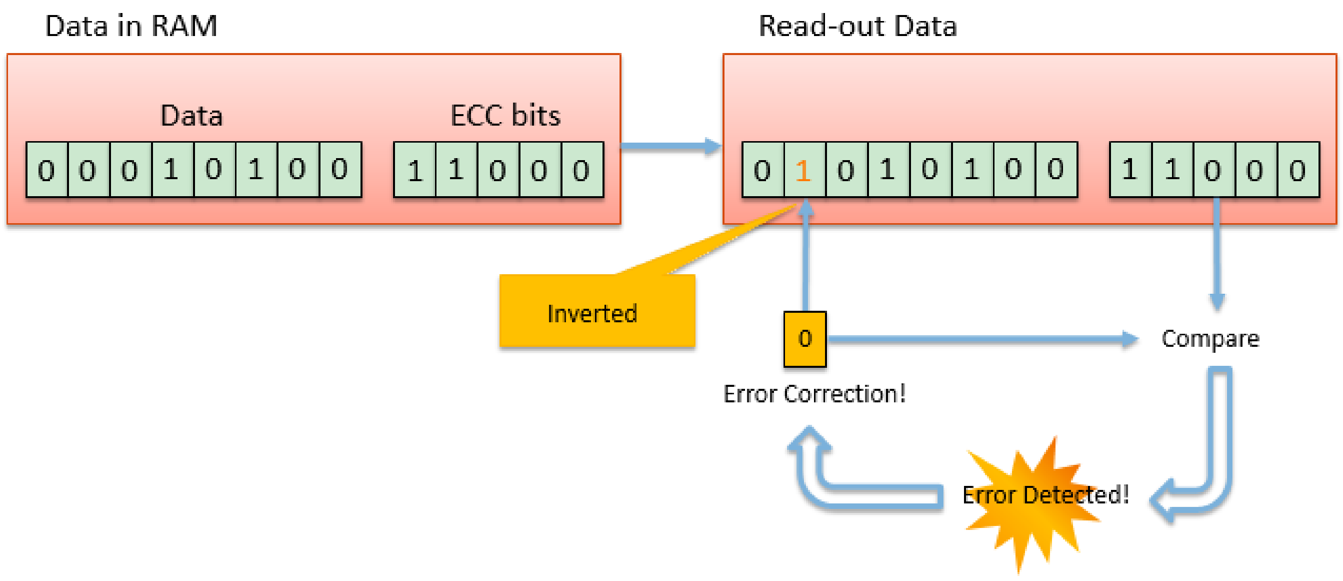

Figure 6. Example of SRAM ECC detection and correction

4.2.2. Cache

The KE1xF includes one 8-KB code cache to minimize the performance impact of the memory access latencies. The code cache is on the I/D bus, and there is no cache on the system bus.

The features of the cache are:

- 2-way set associative.
- Four word lines.
- The lines can be flushed individually.
- The entire cache can be flushed at once.
- Cache memory with a parity check.

Although the KE1xF core clock can run up to 168 MHz, the flash can't run at a frequency higher than 25 MHz. The system brings an 8-KB code cache that can pre-fetch the instructions and data for the CPU, which accelerates the P-flash data transfers and increases the CPU processing efficiency.

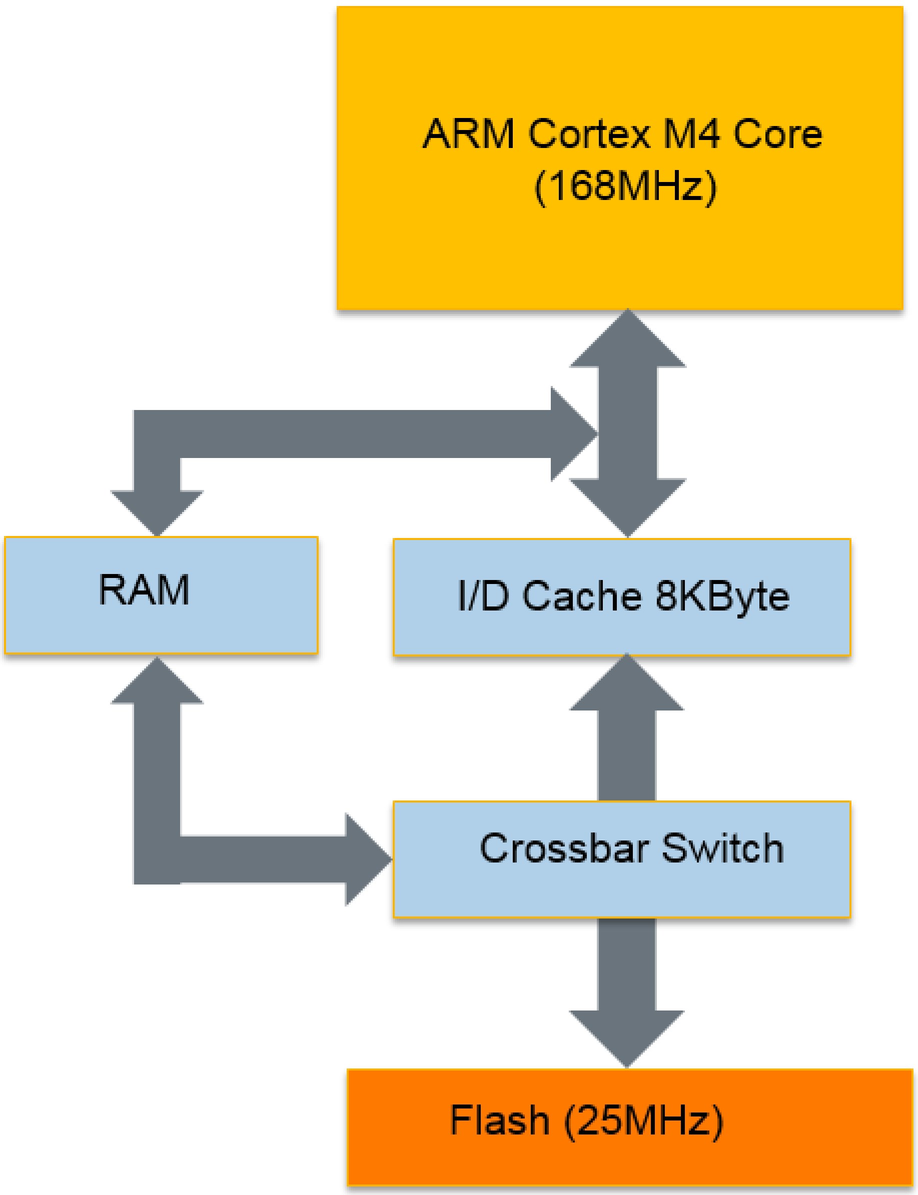

Figure 7. Example of cache access flash

4.2.3. MPU and FAC

The MPU on the KE1xF concurrently monitors all system bus transactions and evaluates their appropriateness using the pre-programmed region descriptors that define the memory spaces and their access rights. The memory references that have sufficient access-control rights are allowed to complete, while the references that are not mapped to any region descriptor or have insufficient rights are terminated with a protection-error response. The features of the MPU are:

- Support of up to eight memory regions.
- Read/Write/Execution permission arbitration.
- Region sizes may vary from 32 B to 4 GB.

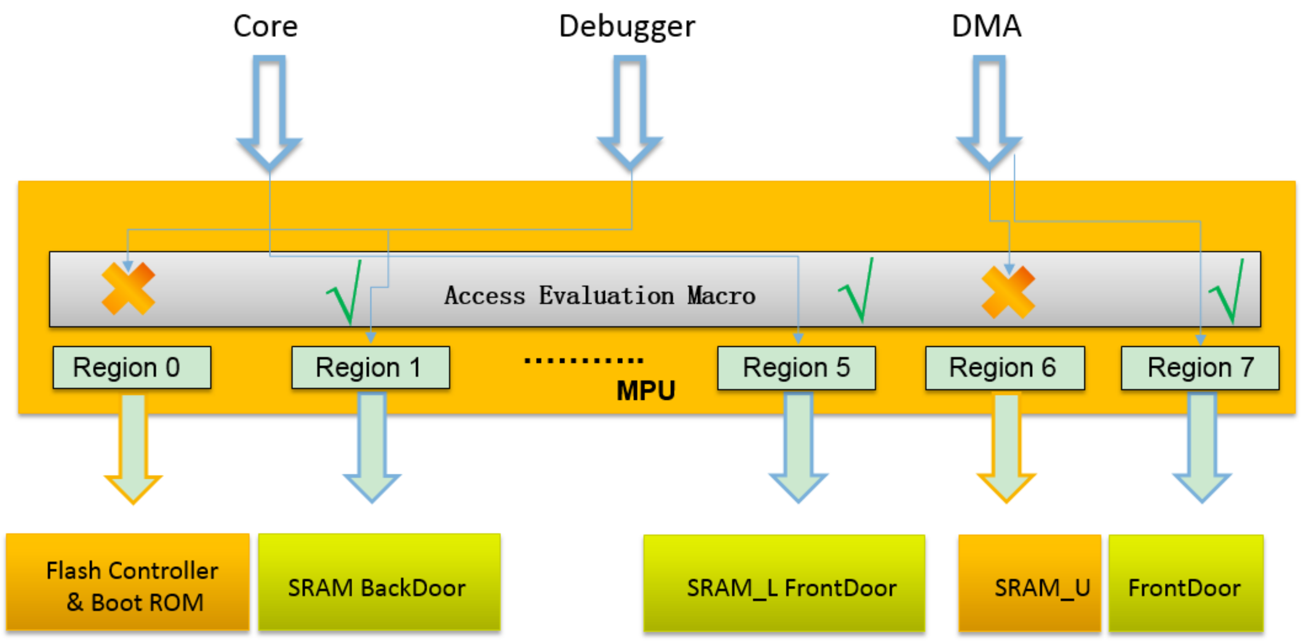

Figure 8. Example of MPU access memory

The Flash Access Control (FAC) is a native or third-party configurable memory protection scheme optimized to utilize the software libraries while offering the programmable restrictions to these libraries. The FAC is on both the KE1xF and KE1xZ and its key features are:

- Programmable flash memory divided into equal sizes; support for up to 64 segments.
- Cycle-by-cycle evaluation of access.
- Different secure states:
  - Supervisor/privileged secure state—Execute and Modify.
  - Mid-level state—Execute Only.
  - Unsecure state—No Access Rights.
- You may implement the access control logic in the Program Once area.
- Configured by the FXACC and FSACC registers.

4.2.4. MMDVSQ module

The ARM processor cores in the Cortex-M family implementing the ARMv6-M instruction set architecture do not include the hardware support for the integer-divide operations. The affected processors include the Cortex-M0+ core. However, in certain deeply embedded application spaces, the hardware support for this class of arithmetic operations (along with the unsigned square root function) is important to maximize the system performance and minimize the device power dissipation.
The MMDVSQ module is included in selected MCUs to serve as a memory-mapped co-processor located in a special address space (within the system memory map) that is accessible only to the processor core. The MMDVSQ module supports the execution of the integer divide operations defined in the ARMv7-M instruction set architecture and the unsigned integer square root operations. The supported integer divide operations include the 32/32 signed (SDIV) and unsigned (UDIV) calculations.

The MMDVSQ module is included only in the KE1xZ MCUs and its key features include:

- Support for 32/32 signed and unsigned divide (or remainder) calculations.
- Support for 32-bit unsigned square root calculations.
- More than 25 % performance improvement in running the math-intensive applications such as the sensorless PMSM FOC algorithms.
- The simple programming model includes the input data and result registers and a control/status register.

The generic block diagram of the processor core and the platform for this class of low-end MCUs is shown in the following figure. The MMDVSQ module's location (as a memory-mapped co-processor) is highlighted in yellow.

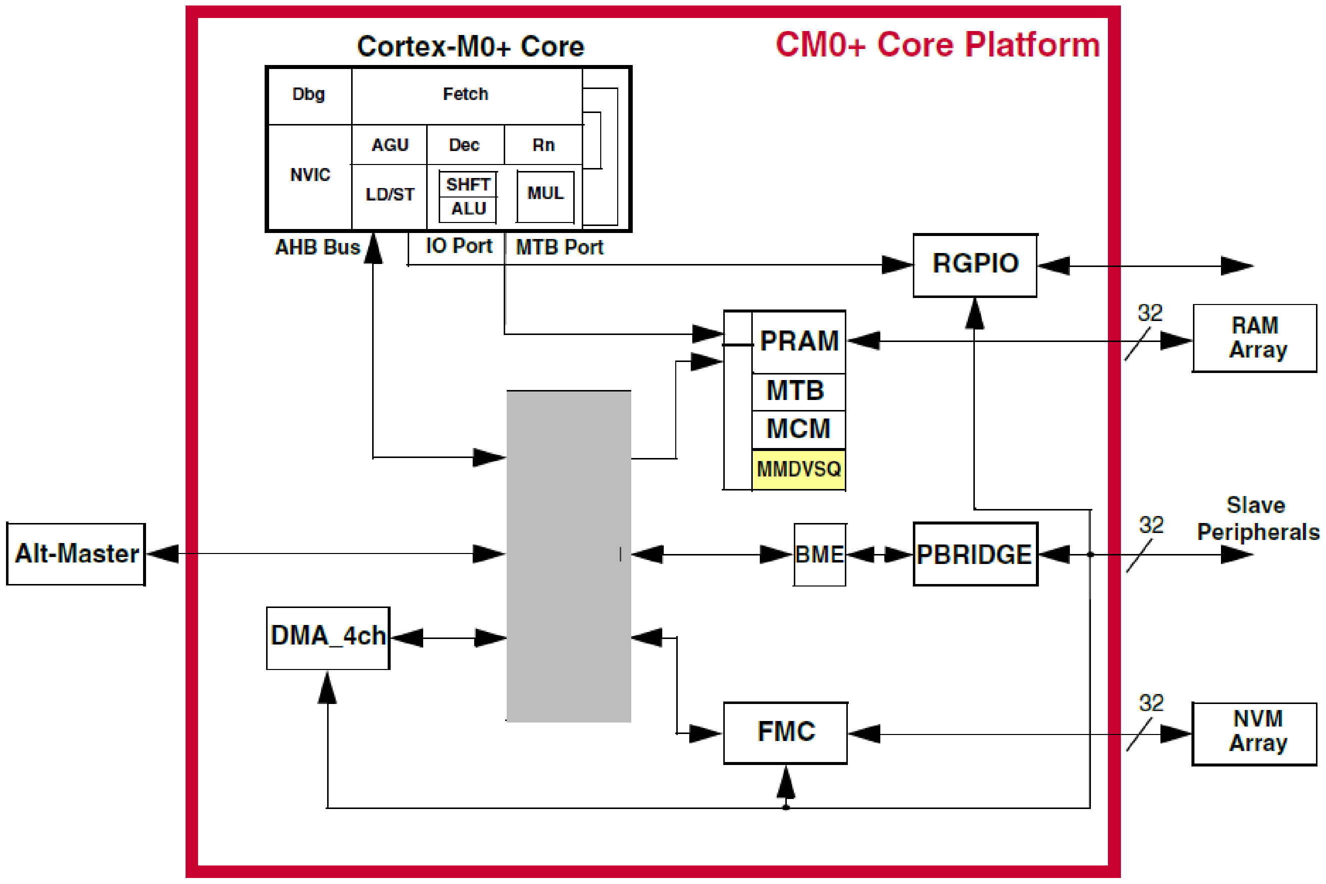

Figure 9. Generic Cortex-M0+ core platform block diagram

4.2.5. eDMA

The eDMA is a highly programmable data-transfer engine optimized to minimize any intervention required from the host processor. It is intended to be used in the applications where the data size to be transferred is statically known and not defined within the transferred data itself. The eDMA module has these key features:

- The whole data movement is done via the dual-address transfers—read from the source, write to

the destination.
    - Programmable source and destination addresses and transfer size.
    - Support for enhanced addressing modes.
- 16-channel (KE1xF) or 8-channel (KE1xZ) implementation which performs complex data transfers with a minimal intervention from the host processor.
- Transfer Control Descriptor (TCD) organized to support two-deep, nested transfer operations.
    - 32-byte TCD stored in the local memory for each channel.
    - An inner data-transfer loop defined by a minor byte transfer count.
    - An outer data-transfer loop defined by a major iteration count.
- Fixed-priority and round-robin channel arbitration.
- Channel completion reported via the optional interrupt requests.
- Support for complex data structures.

The following figure shows the KE1xF DMAMUX/eDMA scheme. It includes the DMA request mux which allows up to 63 DMA request signals to be mapped to any of the 16 DMA channels. The KE1xZ has a similar scheme, but the DMAMUX channel number is limited to eight.

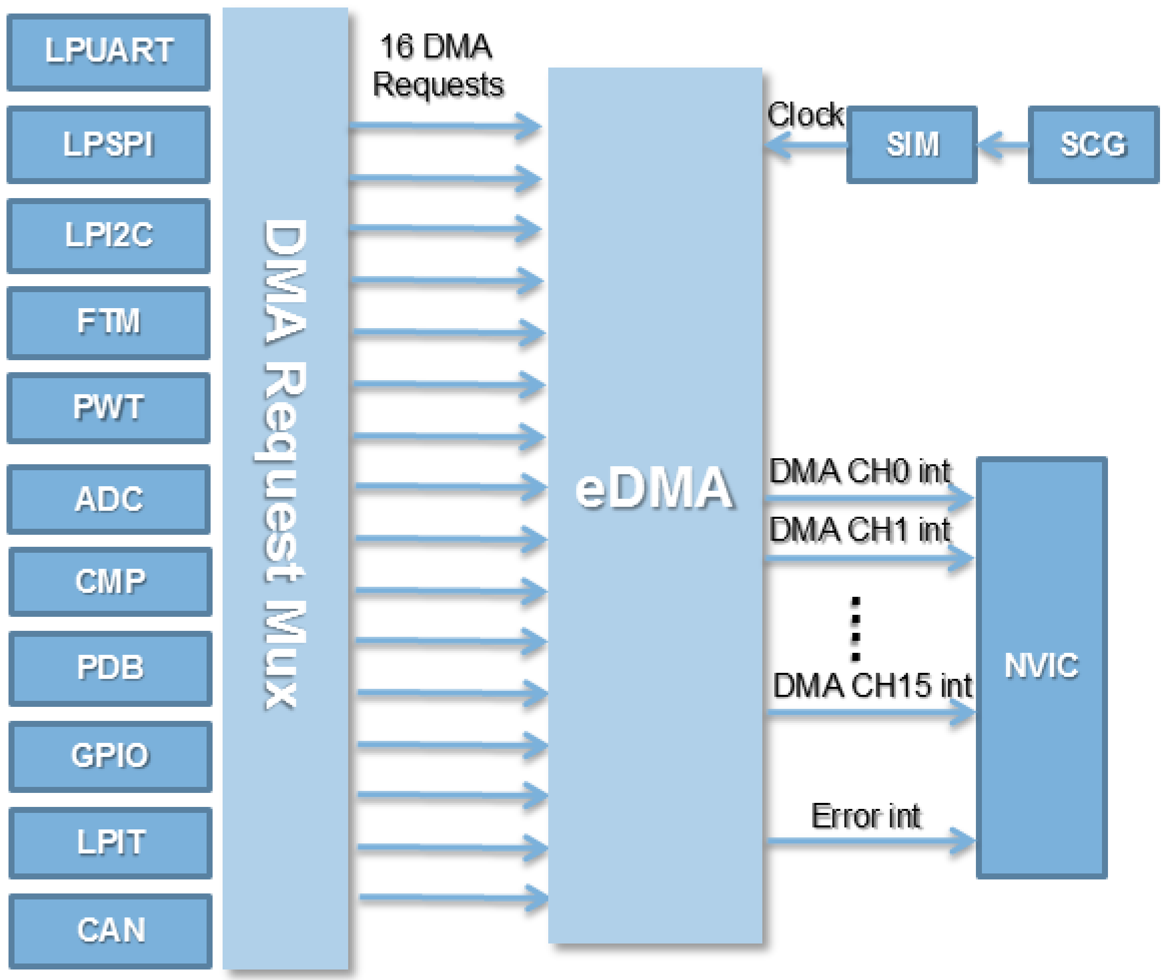

Figure 10. KE1xF DMAMUX/eDMA scheme

The SDK v2.0 driver/example code for the KE1x DMAMUX and eDMA features:

- DMAMUX initialization and enable/disable. The `channel` parameter is the DMAMUX channel number.

```
void DMAMUX_Init(DMAMUX_Type *base);

void DMAMUX_EnableChannel(DMAMUX_Type *base, uint32_t channel);

void DMAMUX_DisableChannel(DMAMUX_Type *base, uint32_t channel);
```

- DMAMUX channel source configuration. The `channel` parameter is the DMAMUX channel number and the `source` parameter specifies the peripheral source used to trigger the DMA transfer.

```
void DMAMUX_SetSource(DMAMUX_Type *base, uint32_t channel, uint8_t source);
```

- eDMA initialization and transfer configuration. The `config` parameter is the eDMA global configuration structure. Sometimes you may use just its default setting. Before calling the `EDMA_SetTransferConfig()` function, prepare the `edma_transfer_config_t` configuration structure and populate it with the desired source and destination transfer attributes.

```
void EDMA_Init(DMA_Type *base, const edma_config_t *config);

void EDMA_SetTransferConfig(DMA_Type *base, uint32_t channel, const
edma_transfer_config_t *config, edma_tcd_t *nextTcd);
```

- eDMA hardware channel request enable/disable. The `channel` parameter is the DMAMUX channel number.

```
void EDMA_EnableChannelRequest(DMA_Type *base, uint32_t channel);

void EDMA_DisableChannelRequest(DMA_Type *base, uint32_t channel);
```

4.3. Peripherals improvement

4.3.1. CAN modules

The KE1xF integrates up to two on-chip FlexCAN modules. It is a communication controller implementing the CAN protocol according to the ISO 11898-1 standard and the CAN 2.0 B protocol specifications. The general block diagram shown in the following figure describes the main sub-blocks implemented in the FlexCAN module, including one associated memory for storing of the message buffers, Receive Global Mask registers, Receive Individual Mask registers, Receive FIFO filters, and Receive FIFO ID filters.

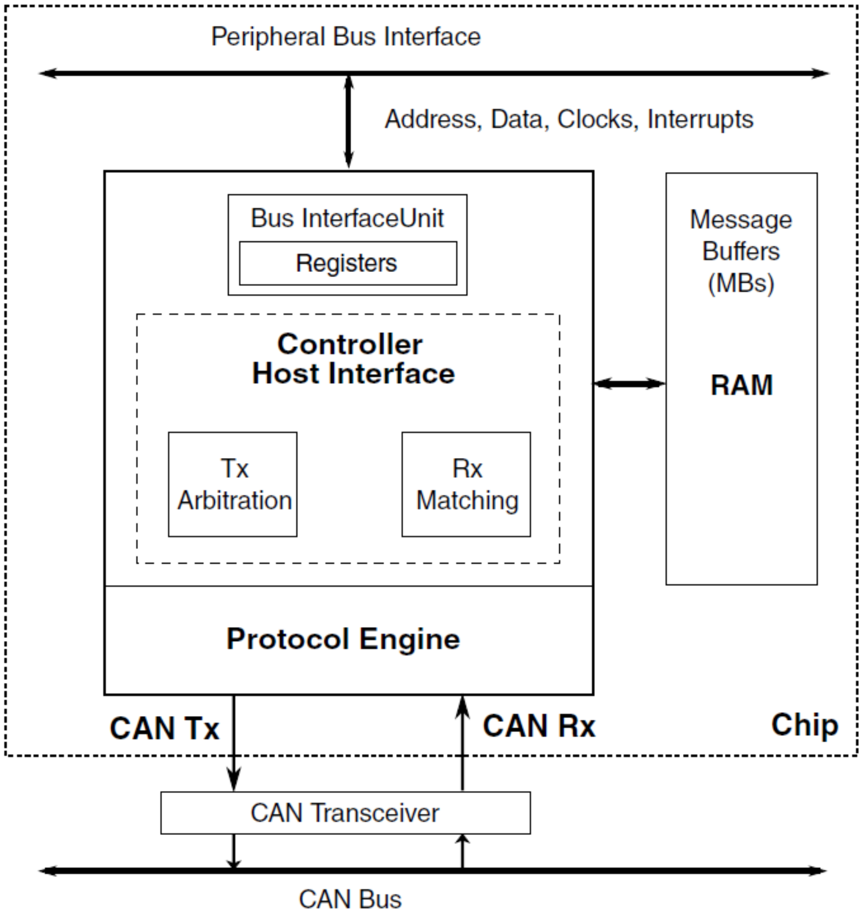

Figure 11. KE1xF FlexCAN block diagram

The FlexCAN module has these key features:

- Flexible message buffers (MBs) with a total of 16 message buffers (each being eight bytes long) which are configurable either as Rx or Tx.
- Each mailbox is configurable either as Rx or Tx and supports both the standard and extended messages.
- Flexible mailboxes (zero to eight bytes long).
- Support operational in the VLPR and VLPW modes with a programmable wakeup on the bus activity.
- Time stamp based on a 16-bit free-running timer with an optional external time tick.
- Bit time counting.
- Listen-only mode capability.
- Very good programming models.

In the KE06Z family, the MSCAN is used as the controller implementing the CAN 2.0A/B protocol, as defined in the Bosch specification (dated September 1991). The MSCAN uses an advanced buffer arrangement resulting in a predictable real-time behavior and a simplified application software. The following figure shows the MSCAN block diagram.

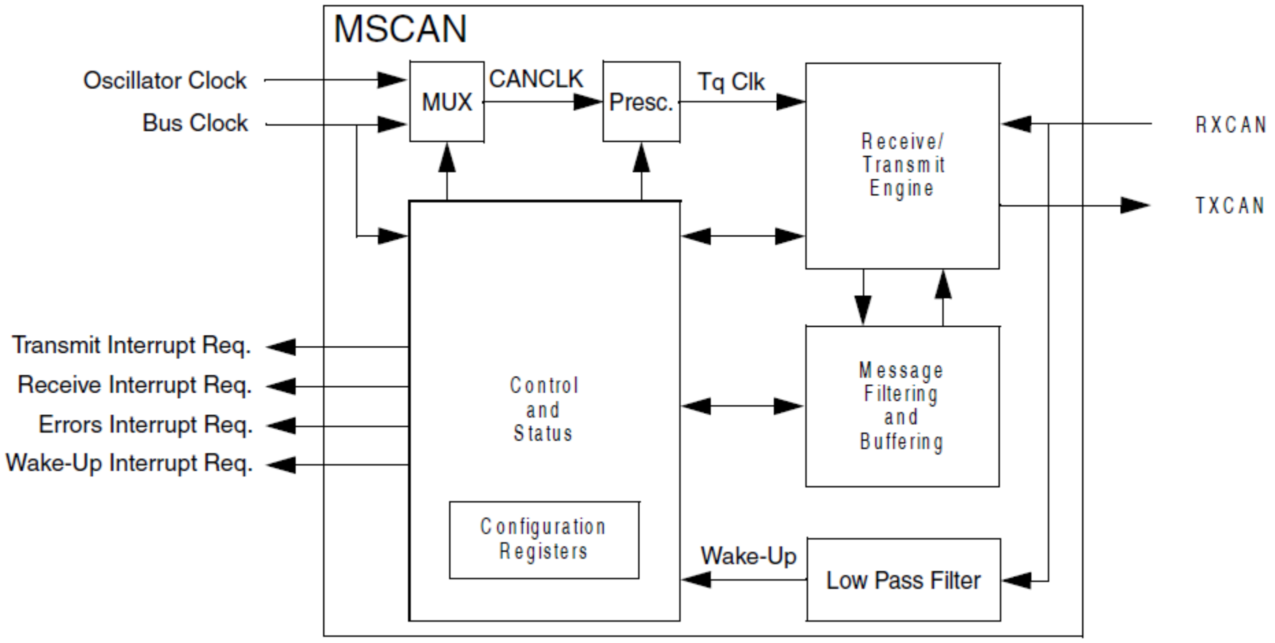

Figure 12. KE0xZ MSCAN block diagram

Both the FlexCAN and the MSCAN fully implement the CAN protocol specification and their structure is similar and compatible. The advantage of the FlexCAN is that it is more flexible than the MSCAN, contains much more sophisticated state, and can automatically respond to a remote frame (and more).

4.3.2. HMI

Besides the general I/O ports, the KE0x family uses the Keyboard Interrupt (KBI) module as the HMI interface. This on-chip peripheral module is called the Keyboard Interrupt module because it was originally designed to simplify the connection and use of the row-column matrices of the keyboard switches. These inputs are also useful as the extra external interrupt inputs and the external means to wake the MCU from the stop or wait low-power modes.

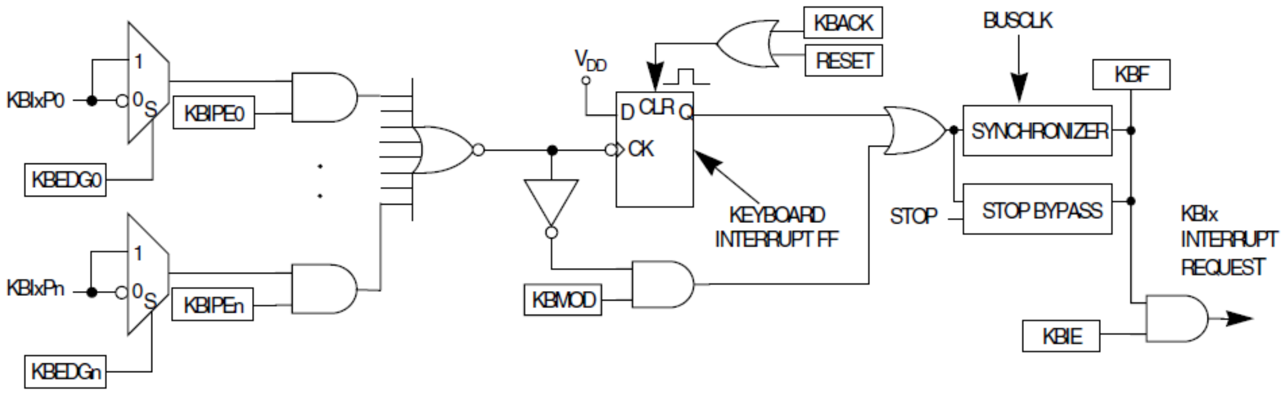

Figure 13. KE0x KBI block diagram

The KE1xZ uses the Touch Sensing Interface (TSI) to replace the KBI as the HMI interface. The TSI is the next generation HMI for many consumer and industry fields. The TSI replaces the traditional mechanical button/switch which helps to make the products more reliable and fashionable. Because the touch market is getting bigger and bigger, NXP does a lot to meet the marketing requirements—more keys, great robustness, high sensitivity, little software interference, easy to design, and no extra BOM cost.

The TSI provides the touch sensing detection on the capacitive touch sensors. The external capacitive touch sensor is typically formed on a PCB and the sensor electrodes are connected to the TSI input channels via the I/O pins in the device. The TSI operates in the switching integration mode to achieve low power consumption, high sensitivity, and advanced EMC robustness. It supports both the self-cap (Figure 14) and mutual-cap (Figure 15) sensors. In the self-cap mode, the TSI requires only one pin for each touch sensor. In the mutual-cap mode, the sensing is done using a capacitive touch matrix in various TX-RX configurations. The TSI requires one pin per a TX line and one pin per a RX line. The TSI fully supports the NXP Touch Sensing Software (TSS) library which provides a solid capacitive measurement module to implement the touch keyboard, rotaries, and sliders.

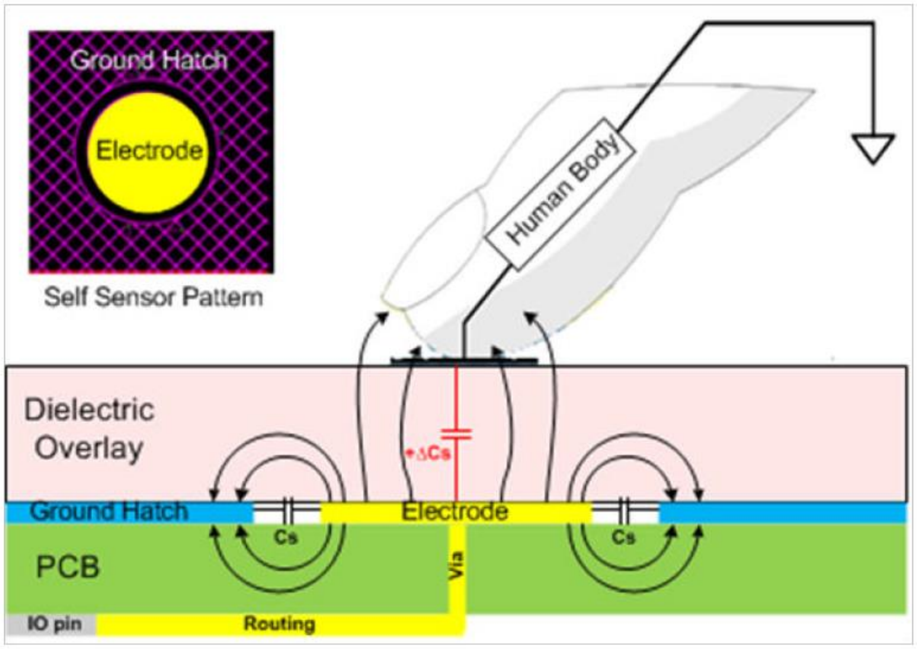

Figure 14. Self-cap touch sensor structure and electric field

The self-cap sensor structure is as follows:

- Cs: intrinsic self-capacitance. It is usually 10 pF ~ 50 pF.
- ΔCs: touch-generated self-capacitance. It is usually 0.3 pF ~ 2 pF.
- Sensor sensitivity: ΔCs/Cs. It is usually 1 % ~ 10 %.

The intrinsic performance depends on the electrode pattern design, the thickness/dielectric of the overlay, and the PCB routing.

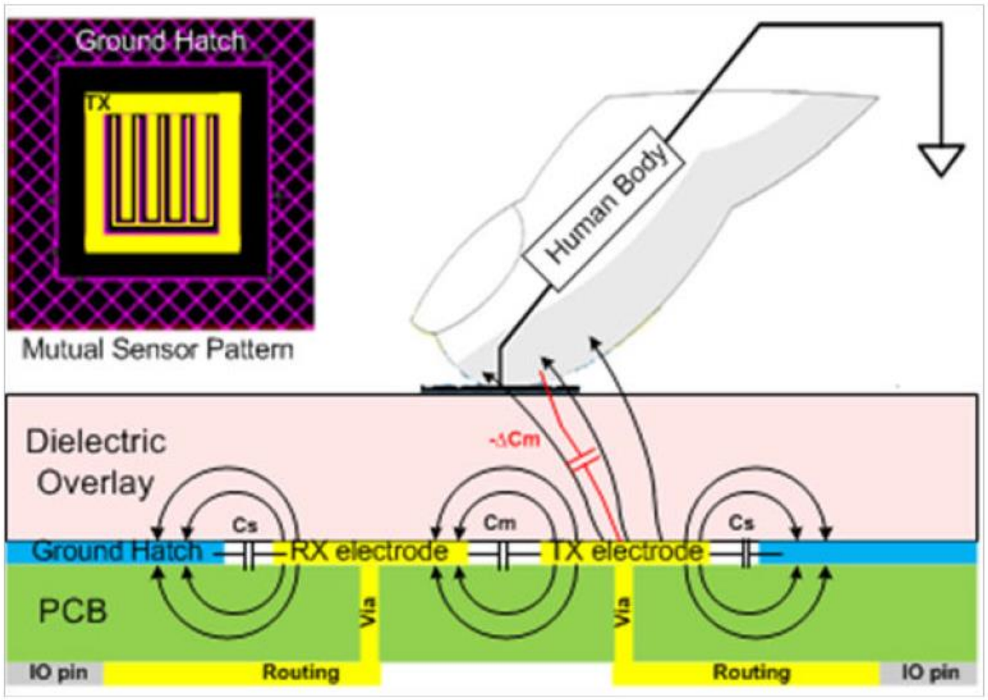

Figure 15. Mutual-cap touch sensor structure and electric field

The mutual-cap sensor structure is as follows:

- Cm: intrinsic mutual-cap. It is usually 2 pF ~ 10 pF.
- ΔCm: touch-reduced mutual-cap. It is usually 0.3 pF ~ 2 pF.
- Cs: parasitic self-cap. It is usually 10 pF ~ 50 pF.
- Sensor sensitivity: ΔCm/Cm. It is usually 1 % ~ 20 %.

The intrinsic performance depends on the electrode pattern design, thickness/dielectric of the overlay, and PCB routing.

For the software part, the SDK v2.0 provides comprehensive TSI module drivers. The NXP Touch software library is designed as a supplement to speed up the development of touch applications and it is available as the binary and the source code.

- TSI self-cap mode initialization function. The `config` parameter is the configuration structure for the self-cap mode. Populate it with the desired charge current/voltage, oscillator frequency, sensitivity configuration, and other attributes before calling this function.

```
void TSI_InitSelfCapMode(TSI_Type *base, const tsi_selfCap_config_t *config)
```

- TSI mutual-cap mode initialization function. The `config` parameter is the configuration structure for the mutual-cap mode. Populate it with the desired charge current/voltage, oscillator frequency, sensitivity configuration, and other attributes before calling this function.

```
void TSI_InitMutualCapMode(TSI_Type *base, const tsi_mutualCap_config_t *config)
```

- Key calibration function. It is used to calibrate the TSI idle value. Save it as the TSI baseline for the future key detection.

```
void TSI_keyCalibrate(void);
```

- Key detection function. It represents different types of the signal-processing algorithms. The primary purpose of a key detector algorithm is to determine whether an electrode was touched or not and calculate the normalized signal to compare with the baseline. The `current_key_id` parameter represents the pointer to the key channel. The function return value is the key event which indicates the momentary actions such as “touched”, “released”, and so on.

```
uint8_t TSI_keyDetect(uint8_t *current_key_id);
```

4.3.3. Communication interfaces

Both the KE0x and KE1x families include a rich variety of communication interfaces such as the I^{2}C, SPI, and UART. On the KE1x, they are improved to work in the low-power modes which avoid waking up the CPU frequently to further reduce the power consumption. The communication interfaces are re-named as LPI^{2}C, LPSPI, and LPUART. They are functional in the Stop/VLPS modes if the clock they use remains enabled. The following figure shows the module clock for the LPUART. This example figure also applies to the clocking of the LPSPI, LPI^{2}C, FlexIO, and LPIT.

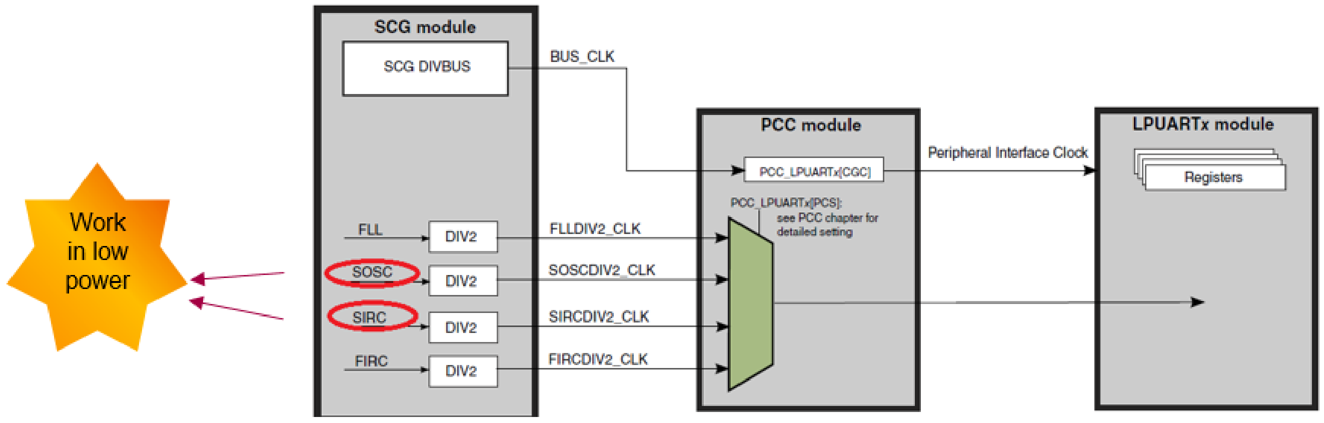

Figure 16. Module clock for LPUART

4.4. Pinout compatibility

Both the KE0x and KE1x families have multiple packages and pinout choices for different application usages (see Figure 17). They are mostly pin-to-pin compatible for different series, so there is no extra layout cost when migrating.

For example, the KE06Z and KE18F both have the CAN interface, so the application field may be similar and it is possible to migrate both the hardware and software. With the 64LQFP package difference between them, there are only two pins to pay attention to. The pin 9 on the KE06Z is used as the VREFL, but it is used as the VREFH on the KE18F. The default NMI pin is located at pin 19 on the KE06Z, but it is located at pin 45 on the KE18F. The other 62 pins are all compatible and only some peripheral instances or channels change so the porting is very easy.

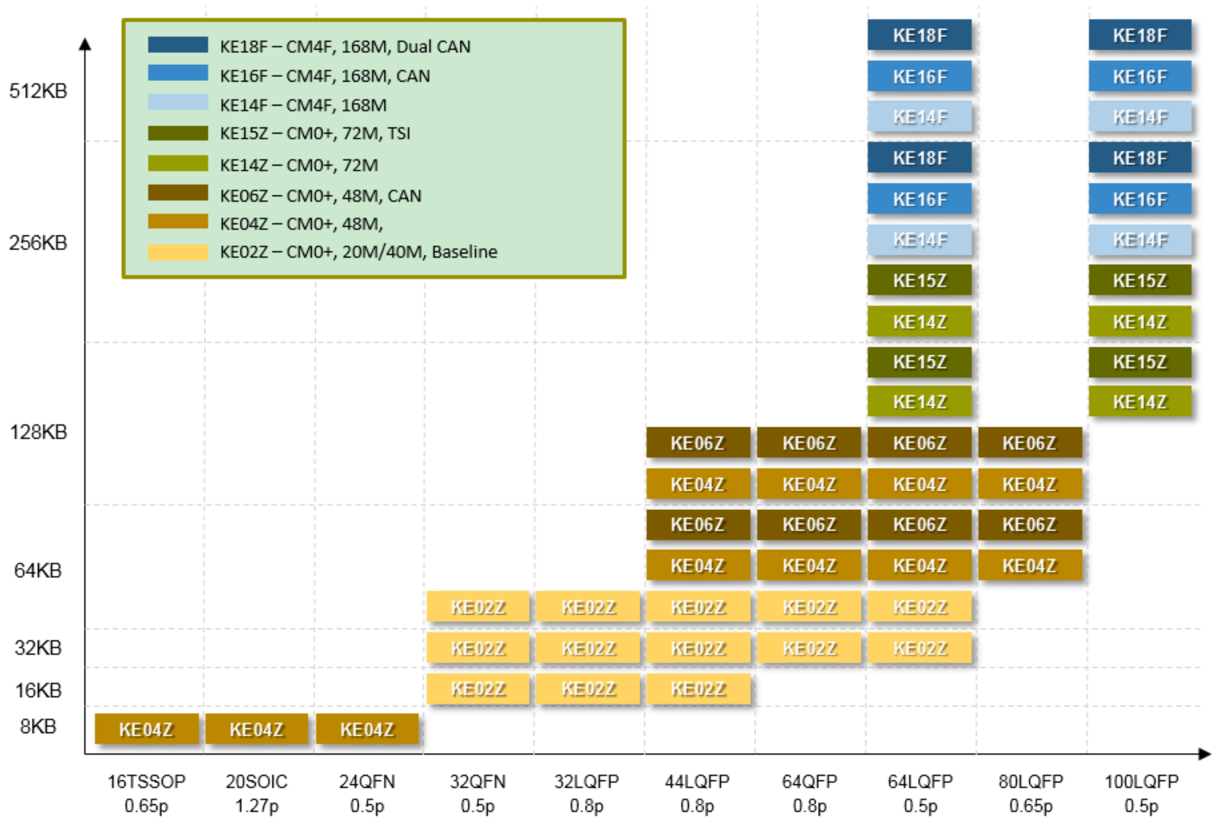

Figure 17. KE0x and KE1x package option chart

5. Conclusion

The KE0x family is the first entry-level Kinetis MCU that employs a robust technology, the 5-V I/O pad, and the ARM Cortex-M0+ core. The KE1x is an extension of the existing KE0x MCU family with an enhanced CPU performance and additional memories and peripherals. This application note outlines the key differences and improvements between the KE0x and KE1x MCU families and introduces the basic Kinetis SDK v2.0 software packages. This document helps you to easily migrate from the KE0x family to the KE1x family.

6. Revision History

This table summarizes the changes done to this document since the initial release:

Table 5. Revision history

Revision number	Date	Substantive changes
0	09/2016	Initial release.

Document Number: AN5332
Rev. 0
09/2016