Freescale Semiconductor
Application Note

Document Number:AN4757
Rev 1, 11/2013

Migration Guide From S08 to Kinetis E Family

by: William Jiang

Contents

1 Introduction

The Kinetis E (KE) family is the first Kinetis MCU that employs robust technology, new 5 V I/O pad, and ARM® Cortex® M0+ core. This KE family also introduces many new features for different peripheral modules. With these new features, KE family has low power consumption, better EFT/ESD protection, and higher performance.

This application note outlines the key differences between S08 (specifically S08P) and KE product families, and provides code conversion recommendations to help shorten the learn curve.

2 Device overview

The Kinetis E family devices targeting the home appliance market segment has the following features.

- Based on ARM Cortex-M0+ core running up to 40 MHz (for KE02)/48 MHz (for other KEs) with Bit Manipulation Engine (BME)
- Up to 128 KB flash memory
- Up to 16 KB SRAM
- ROM (only for some KE parts)
- Clock generator with internal reference clock (IRC)
- Crystal oscillator and FLL
- Up to 3 x SCI with LIN slave capability
- Up to 2 x SPI

- 2 x I2C
- 1 x CAN (only for some KE parts)
- 3 x 16-bit FlexTimer
- 1 x PWT
- One 32-bit 2-channel PIT
- RTC
- IEC60730 compliant watchdog
- CRC
- Up to 16-ch 12-bit ADC with 8-level FIFO
- 2 x ACMP
- 16-channel TSI for touch input (only for some KE parts)
- 4-channel DMA (only for some KE parts)
- Up to 82 GPIO pins with 8 high-drive pins supporting 20 mA
- Embedded voltage regulator supporting 2.7–5.5 V operating voltage, power-on reset, and programmable low voltage detector
- Two low-power modes
- Unique chip ID with upto 10 bytes
- Single Wire Debug (SWD) interface
- Supports wide temperature range from –40 °C to +105 °C

Compared with S08 devices (S08AC, S08P), the KE family has ARM Cortex-M0+ core, BME, and 32-bit devices while most of peripherals are similar to S08P. The following table summarizes the comparison of main features among these families:

Table 1. Feature comparison among S08AC, S08P, and KE product families

Features	S08AC	S08P	KE family
Core platform	8-bit S08 core with maximum frequency of 40 MHz	Low-power 8-bit S08 core with maximum frequency of 20 MHz	High energy efficient 32-bit Cortex-M0+ core supporting up to 48 MHz with most of instructions in a single cycle
Bus clock	Half of core frequency, up to 20 MHz	Same as core frequency, up to 20 MHz	Can be same as core frequency, up to 48 MHz
Single cycle 32-bit x 32-bit multiply	Supports only 8-bit x 8-bit	Supports only 8-bit x 8-bit	Yes
Debug	1-pin debug module (BDM)	1-pin debug module (BDM)	2-pin serial wire debug (SWD)
Nested Vectored Interrupt Controller (NVIC)	• No NVIC • No IPC • Does not support hardware nested interrupt • Does not support interrupt vector relocation	• No NVIC • Has Interrupt Priority Controller (IPC) which requires software code to support nested interrupt. • Does not support interrupt vector relocation	• Supports interrupt vector relocation, which can relocate in flash or RAM. • True hardware interrupt nesting without any software code
Data Access Endianess	Big-endian	Big-endian	Little-endian
Direct Memory Access (DMA)	No	No	Added for some KE06 parts
Power mode	• RUN • WAIT • STOP2 (partial power down, lowest power consumption) • STOP3	• RUN • WAIT • STOP3 (typical 1.3 µA)	• Similar to S08P

Table continues on the next page...

Table 1. Feature comparison among S08AC, S08P, and KE product families (continued)

Features	S08AC	S08P	KE family
EEPROM	No	Yes	Yes for most parts
Flash Memory Controller (FMC)	No	No	Yes
FlexTimer	No, but only has legacy TPM	• Extended TPM function to support motor control and power applications • backward compatible with TPM function • Not functional in Stop mode	Enhanced Flextimer with intermediate load, global time base, periodic TOF, fault polarity control, channel swap/ invert control, debug mode function option
System Tick (Systick)	No	No	24-bit timer (Core clock/16)
RTC	No	Yes	Yes
ADC module	• Up to 10-bit resolution • Register access is 8-bit only	• Supports up to 12-bit resolution • register access is 8-bit only	Similar to S08P, but the register access is 32-bit
TSI module	No	End of scan interrupt (only available for some S08P parts)	Only available for some KE parts • End of scan interrupt • Out of range interrupt • Noise detection
Identification registers (ID)	2-byte	8-byte	8-byte Universal Unique Identifier (UUID) plus 2-byte Kinetis IDs

The ARM Cortex M0+ is the smallest, lowest-power ARM processor in the market, compatible with all other Cortex-M cores. Following are its advantages over 8-bit microcontrollers.

- It is the most energy-efficient 32-bit processor ever designed, has significant energy efficiency advantages over 8/16-bit processor with reduced power consumption while still achieving higher performance results of 2.15 CoreMark/MHz (ARM Compiler version 5.0.3), which is 2 to 40 times more than 8/16-bit, and 9% more than Cortex-M0.
- It also has higher code efficiency than 8-bit and 16-bit CPU.
- Because of its C-friendly architecture, relocatable vector table for dynamic exception handlers by moving the vector table into RAM, and simple instruction set together with richest development ecosystem, the software development is simple, fast, and much easier than 8-bit microcontroller products.

For more information on this processor, visit: **arm.com**.

3 Programmers model

3.1 Register set

This section defines the registers available for S08 family and ARM Cortex-M0+ processor used by the Kinetis E family.

Following is the register set for the S08 family.

- One accumulator A which is a general-purpose 8-bit register, used as destination and source of arithmetic and logic operations as well as memory load/store instructions.

- One 16-bit index register H:X which is used as index reference pointer, and X can be used as 8-bit general-purpose register.
- A 16-bit stack pointer SP which points at the next available location on the automatic last-in-first-out (LIFO) stack.
- In addition to a 16-bit program counter PC, there is an 8-bit Condition Code Register (CCR) that contains the interrupt mask (I) and five flags that indicate the results of the instruction recently executed.

Following is the description of registers of the Cortex M0+ core used by the KE family.

- 16 32-bit registers, R0-R15
  - R0-R12 are generally available for essentially all instructions.
  - R13 is used as the Stack Pointer
  - R14 is used as the Link Register (for subroutine and exception return)
  - R15 as the Program Counter.

None of the Cortex-M0+ core registers are directly addressable. Since all the registers of Cortex-M0+ core are 32-bit, it supports 32-bit arithmetic and logic operations efficiently.

Apart from R0-R15 general-purpose registers, Cortex M0+ has some special registers such as Program Status Register, Exception Mask Register, Interrupt Mask Register, and Control Register.

- Program Status Register (xPSR): It is a single 32-bit register with several aliases, each providing a view of a different subset of the contents. PSR combines the following registers.
  - Application Program Status Register (APSR): From the user point of view, the APSR contains the ALU status flags.
  - Interrupt Program Status Register (IPSR): This register contains the number of the currently executing interrupt (or zero if none is currently active) for operating system and exception handling use.
  - Execution Program Status Register (EPSR): It contains bits which reflect execution status and is not directly accessible.

For a detailed information on Cortex M0+ registers and their functions, see Cortex-M0 Devices Generic User Guide, available on **arm.com.**

3.2 Addressing mode

Addressing modes define the way the CPU accesses operands and data. The S08 supports a lot of addressing modes. These are listed as follows.

- Inherent Address Mode (INH)
- Relative Addressing Mode (REL),
- Direct Addressing Mode (DIR)
- Index Addressing Mode with
  - no offset (IX)
  - no offset with Post Increment (IX+)
  - 8-bit offset (IX1)
  - 8-bit offset with Post Increment (IX1+)
  - 16-bit offset (IX2)
  - SP-Relative 8-bit offset (SP1)
  - SP-Relative
  - 16-bit offset (SP2)

- Memory-to-memory address mode with
  - direct-to-direct
  - immediate to direct
  - indexed-to-direct with post increment
  - direct-to-indexed with post increment

The Cortex-M0+ addressing mode is very simple for memory access, and has an offset addressing mode as expressed in assembly language: [<Rn>,<offset>].

3.3 Instruction set

The S08 supports an 8-bit instruction set. Instructions are of variable length and can extend to include operands in the instruction stream, varying from 1–4 bytes.

The Cortex-M0+ supports a subset of the Thumb-2 instruction set. Though most of the instructions are of 16 bits, there are few 32-bit thumb instructions such as BL, DMB, DSB, ISB, MRS, and MSR. One key difference is that instructions cannot extend to include operand data. Any information for the instruction must be encoded within the instruction itself. One consequence of this is that it is not possible to specify arbitrary 32-bit constants or addresses in instructions, so other methods must be used for this purpose. All the C compilers supporting Cortex-M0+ support this function transparently to the programmer.

3.4 Operating modes

The S08 can operate in the following modes.

- Run, Wait, and Stop modes: Both Wait and Stop modes are low-power modes and entered by executing WAIT and STOP instruction respectively. For more information on these modes, see Power Management Controller.
- Background Debug mode: The background mode functions are managed through the background debug controller (BDC) in the S08 core. The BDC, together with the on-chip debug module (DBG), provide the means for analyzing MCU operation during software development. For more information on background debug, see Debug.
- Secure mode: When in Secure mode, external access to internal memory is restricted, so that only instructions fetched from secure memory can access secure memory. When the code is running from internal memory, it can access all resources without any restriction.

The Cortex-M0+ supports two modes.

- Thread mode (used for user processes): This mode is used to execute application software.
- Handler mode: This mode is used to handle exceptions and is automatically entered when an exception occurs.

It also defines the concept of “privilege.” Unprivileged execution limits or excludes access to some resources (for instance, unprivileged code is unable to mask or unmask interrupts). Handler mode execution is always privileged. By default, Thread mode will also execute with privilege but the programmer may configure thread mode to execute without privilege. This configuration can be used to provide a degree of system protection from errant or malicious programs.

3.5 Stack

The S08 stack grows toward the low RAM addresses, that is, in descending order. The stack can be located anywhere in the 64-KB address space that has RAM and can be any size up to the amount of available RAM. Typically, for best performance, it will be located in internal SRAM. The stack is accessed in bytes.

The stack pointer (SP) points at the next available location on the stack, and is initialized to 0x00FF at reset; this address is in the Direct Page Register section or RAM. However, it is recommended that the application code reinitialize the SP to point to the last location of RAM as shown in the following example:

```
LDHX #RamLast+1 ; Point at next addr past RAM
TXS ; SP <-(H:X)-1
```

This requires at least five CPU cycles to reinitialize the stack.

The stack pointer is decremented after the store or push operation and incremented before load or pop operation.

The Cortex-M0+ supports a full descending stack addressed by the current stack pointer, similar to S08 stack. However, stack pointer indicates the last stacked item on the stack memory. The stack size is limited only by the available RAM space. The Cortex-M0+ stack pointer is typically initialized to the word (32-bit) above the top of the allocated stack area. The

initialization of the stack pointer at reset does not require any CPU instructions, that is, CPU cycles. Since the stack model is full descending, the stack pointer is decremented before the first store, thus placing the first word on the stack at the top of the allocated region. All stack accesses on Cortex-M0+ are word-sized.

3.6 Exceptions and interrupts

The S08P has one nonmaskable interrupt source (SWI) and 39 maskable interrupt sources including one external maskable interrupt IRQ. The interrupt priority is fixed with lowest vector number corresponding to the highest priority. With Interrupt Priority Controller enabled, it can support four priority levels. It requires additional software overhead to implement nested interrupt scheme.

The Cortex-M0+ has an integrated Nested Vectored Interrupt Controller which supports up to 32 separate interrupt sources. There are four interrupt priority levels, which support true hardware nested interrupt scheme. The Cortex-M0+ also supports an external Non-Maskable Interrupt (NMI) and several internal interrupts such as HardFault, SVC, and others.

3.7 Vector table

The S08P vector table is located at a fixed address on the high-end of the internal flash memory. It does not support relocation of vector table. Each vector contains the address of the corresponding interrupt handler/service routine. The vector low address stores the high bytes of the interrupt handler address, whereas the vector high address stores the low bytes of the interrupt handler address. The POR and other reset vector are located at the high-end of the flash memory 0xFFFE–0xFFFF.

The KE family vector table is located, by default, at address 0x00000000. It can be relocated during initialization to a location in either flash or RAM regions. Locating the vector table in internal SRAM can provide higher performance. Within the vector table, each entry contains the starting address of the corresponding handler routine. The first vector at vector table offset 0 contains the initial/start SP (supervisor SP), and the second vector at vector table offset 4 contains the start PC. The following code snippet shows the first two vector entries:

```
#define VECTOR_000      (pointer*)__BOOT_STACK_ADDRESS //   ARM core Initial Supervisor SP
#define VECTOR_001      __startup// 0x0000_0004  ARM core  Initial Program Counter
```

To relocate the vector table to the offset vtor, use the following code snippet:

```
SCB_VTOR = vtor
```

3.8 Reset and interrupt handlers

For S08, the interrupt service routine ends with a return-from-interrupt (RTI) instruction which restores the CCR, A, X, and PC registers to their pre-interrupt values by reading the previously saved information off the stack.

In C code, to define an interrupt handler for S08 devices, a compiler directive must be used to let the compiler know it is an interrupt service routine, instead of standard C routine/function. The following example shows how to define an interrupt service routine with CodeWarrior:

```
interrupt VectorNumber_Vrtc void Rtc_ISR(void)
{
    …
}
```

The Cortex-M0+ supports all exception entry and exit sequences in hardware and thus allows interrupt routines to be standard C functions, compliant with the ARM Architecture Procedure Call Standard (AAPCS). Any compliant function can be installed in the vector table as a handler, simply by referencing its address. With this scheme, the code developers do not need to remember such compiler directives. The following code snippet shows how to define the Rtc_ISR in C language which is same as standard function:

```
void Rtc_ISR(void)
{
    …
}
```

3.9 Memory

The S08 core can address 64 KB of memory space and divides the memory into five major segments as shown in this table.

Table 2. S08 memory map

Name	Address	Comment
Direct-page registers	0x0000–0x00xx	Upto 128 bytes
RAM	0x00xx–[1]	Includes some direct page locations
High-page registers	0x1800–0x18yy	System configuration
FLASH Memory	[2]–0xFFFF	Upto 60 KB
Vectors	0xFFCO–0xFFFF	Upto 32 x 2 bytes

1. The upper limit depends on the given device
2. The lower limit depends on the given device

The S08 core can access the direct-page registers/RAM in most efficient addressing mode and allows bit manipulation instructions to be used to set, clear, or test any bit in these registers with the BSET, BCLR, BRSET, and BRCLR instructions. This “bit-banding” is simple, but will require 2–5 CPU cycles.

The Cortex-M0+ processor can access 4 GB memory space, and divides the memory into several spaces. But it does not support such “bit-banding” operation. However, KE family added Bit Manipulation Engine (BME) to implement this “bit-banding” operation. For more detail on BME, see Bit Manipulation Engine (BME). The following table briefs KE02 system memory map.

Table 3. KE02 memory map

System 32-bit address range	Usage
0x0000_0000–0x07FF_FFFF	Program flash and read-only data (Includes exception vectors in first 196 bytes)
0x1000_0000–0x1000_00FF2	EEPROM
0x1FFF_FC00–0x1FFF_FFFF	SRAM_L: Lower SRAM
0x2000_0000–0x2000_0BFF	SRAM_U: Upper SRAM
0x4000_0000–0x4007_FFFF	AIPS peripherals
0x400F_F000–0x400F_FFFF	GPIO
0x400F_F000–0x400F_FFFF	Decorated AIPS peripherals space accessed via BME
0xE000_0000–0xE00F_FFFF	Private Peripherals
0xF000_2000–0xF000_2FFF	ROM table
0xF000_3000–0xF000_3FFF	Miscellaneous Control Module (MCM)
0xF800_0000–0xFFFF_FFFF	Single (core clock) cycle IOPORT
Others	Reserved

3.10 Access type

The S08 processor supports byte and bit accesses to memory. Bit accesses are supported via the bit-addressable region (direct page as aforementioned). Since the largest item which can be loaded from memory is of 8 bits and the destination is also of 8 bits, the sign of the loaded value is not important. To improve compiler efficiency, it implements LDHX, STHX, and CPHX instructions for 16-bit load/store and compare operations.

The Cortex-M0+ is a 32-bit processor and all internal registers are 32-bit. Memory transfers of 8-bit bytes, 16-bit halfwords, and 32-bit words are supported. In the case of bytes and halfwords, the programmer needs to specify whether the loaded value is to be treated as signed or unsigned. In the case of signed loads, the loaded value is sign-extended to create a 32-bit signed value in the destination register; in the case of unsigned loads, the upper part of the register is cleared to zero.

The Cortex-M0+ also has Load and Store Multiple instructions which transfer multiple words in a single instruction to and from a contiguous block of memory.

3.11 Bit-banding

The S08 bit-addressable region supports atomic bit accesses to the direct page locations with bit manipulation instructions. The bit set/clear instructions take 5 bus clock cycles. So for a bit toggle, it takes 10 bus clock cycles.

The KE04 and the newer sub families support bit-banding. The upper SRAM area (SRAM_U) is the bit-band region. The bit banding maps a complete word of memory in an alias region of memory onto a single bit in the bit-band region. For example, writing to an alias word will set or clear the corresponding bit in the bitband region. This allows every individual bit in the bit-banding region to be directly accessible from a word-aligned address using a single LDR instruction, and individual bits to be toggled from C/C++ without performing a read-modify-write sequence of instructions. With bit-banding, only two core clock cycles are needed to toggle a bit, while it takes at least three cycles with a read-modify-write sequence. As a result, a significant performance improvement is seen with bit-banding in KE family.

In addition, all KE families employ BME techniques to support bit banding. For more information on BME, see Bit Manipulation Engine (BME).

3.12 Debug

This table presents a comparison of some of the features of the Debug module in S08 and KE families.

Table 4. Debug module comparison between S08P and KE family

S08	KE family
Uses single Wire BDM interface	Uses Serial Wire Debug (SWD) interface which includes two wires: one for clock and one for data I/O.
Supports three hardware breakpoints or two hardware breakpoint with one watchpoint	Supports two breakpoints and two watchpoints
Supports Loop1 capture mode to track most recent COF event captured into FIFO with 8-word width which is used for code trace, and nine trigger modes	No trace capability

For more information on debug, see Debug.

3.13 Power Management

The S08P supports two low-power modes: Wait and Stop.

The Cortex-M0+ supports Sleep and Deep Sleep modes.

- In Sleep mode, external logic is usually configured to stop the processor clock, minimizing power consumption. The power and clock to the NVIC is maintained, so that an exception can exit sleep mode.
- In Deep Sleep mode, the processor can be powered down completely, usually leaving only the external Wakeup Interrupt Controller (WIC) active. The WIC will wake the processor if any unmasked external interrupt is detected.

Sleep mode can be entered in the following ways.

- Sleep-now: The Wait-For-Interrupt (WFI) or Wait-For-Event (WFE) instructions cause the processor to enter Sleep mode immediately. Exit is on detection of an interrupt or debug event.
- Sleep-on-exit: Setting the SLEEPONEXIT field within the System Control Register (SCR[SLEEPONEXIT]) causes the processor to enter Sleep mode when the last pending ISR has exited. In this case, the exception context is left on the stack so that the exception which wakes the processor can be processed immediately.

In addition, Deep Sleep mode can be entered by setting SCR[SLEEPDEEP]. On entry to Sleep mode, if this field is set, the processor indicates to the external system that deeper sleep is possible.

Actually the KE family implements Sleep mode as Wait mode and Deep Sleep mode as Stop mode.

The following code snippets show how to enter Wait and Stop mode:

```
void wait (void)
{
        /* Clear the SLEEPDEEP bit to make sure we go into WAIT (sleep) mode instead
         * of deep sleep.
         */
         SCB_SCR &= ~SCB_SCR_SLEEPDEEP_MASK;

         /* WFI instruction will start entry into WAIT mode */
#ifndef KEIL
asm("WFI");
#else
    __wfi();
#endif
}
void stop (void)
{
         /* Set the SLEEPDEEP bit to enable deep sleep mode (STOP) */
         SCB_SCR |= SCB_SCR_SLEEPDEEP_MASK;

         /* WFI instruction will start entry into STOP mode */
#ifndef KEIL
         asm("WFI");
#else
         __wfi();
#endif
}
```

4 Nested Vectored Interrupt Controller (NVIC)

The NVIC is a standard module on the ARM Cortex M series. This module is closely integrated with the core and provides very low latency entering and exiting an interrupt service routine (ISR). It takes 15 cycles to enter and exit an ISR, unless the exit from the interrupt is into another pending ISR. In this case, the MCU tail-chains and the exit and re-entry takes 11 cycles.

The NVIC provides four different interrupt priorities which can be used to control the order in which interrupts must be serviced. Priorities are 0-3, with 0 receiving the highest priority. For example, in a motor-control application, if a timer interrupt and UART occur simultaneously, the timer interrupt that moves the motor is more critical than the UART interrupt receiving a character. The timer priority must be set higher than the UART. This supports true hardware interrupt nesting without any additional software code.

The S08 devices with Interrupt Priority Controller (IPC) can support different levels of interrupt priorities and requires prolog and epilog software code to be added in the interrupt service routine.

The following code snippet shows how to enable or disable an interrupt with the given IRQ number in the KE family:

```
void enable_irq (int irq)
{
    /* Make sure that the IRQ is an allowable number. Up to 32 is
     * used.
     *
     * NOTE: If you are using the interrupt definitions from the header
     * file, you MUST SUBTRACT 16!!!
     */
    if (irq > 32)
        printf("\nERR! Invalid IRQ value passed to enable irq function!\n");
    else
    {
      /* Set the ICPR and ISER registers accordingly */
      NVIC_ICPR = 1 << (irq%32);
      NVIC_ISER = 1 << (irq%32);
    }
}


void disable_irq (int irq)
{

    /* Make sure that the IRQ is an allowable number. Right now up to 32 is
     * used.
     *
     * NOTE: If you are using the interrupt definitions from the header
     * file, you MUST SUBTRACT 16!!!
     */
    if (irq > 32)
        printf("\nERR! Invalid IRQ value passed to disable irq function!\n");
    else
      /* Set the ICER register accordingly */
      NVIC_ICER = 1 << (irq%32);
}


void set_irq_priority (int irq, int prio)
{
    /*irq priority pointer*/
    uint32 *prio_reg;
    uint8 err = 0;
    uint8 div = 0;

    /* Make sure that the IRQ is an allowable number. Right now up to 32 is
     * used.
     *
     * NOTE: If you are using the interrupt definitions from the header
     * file, you MUST SUBTRACT 16!!!
     */
    if (irq > 32)
    {
        printf("\nERR! Invalid IRQ value passed to priority irq function!\n");
        err = 1;
    }

    if (prio > 3)
    {
```

```
        printf("\nERR! Invalid priority value passed to priority irq function!\n");
        err = 1;
    }

    if (err != 1)
    {
        /* Determine which of the NVICIPx corresponds to the irq */
        div = irq / 4;
        prio_reg = ((uint32*)&NVIC_IP(div));
        *prio_reg |= (((prio&0x3) << (8 - ARM_INTERRUPT_LEVEL_BITS))) << (((irq-
(div<<2)))<<3);

}
}
```

For more information on NVIC and code examples, see Kinetis L Peripheral Module Quick Reference Guide, KLQRUG, available on **freescale.com.**

5 Bit Manipulation Engine (BME)

The Bit Manipulation Engine (BME) provides hardware support for atomic read-modify-write memory operations to the peripheral address space. This architectural capability is also known as "decorated storage" as it defines a mechanism for providing additional semantics for load and store operations to memory-mapped peripherals beyond just the reading and writing of data values to the addressed memory locations. BME-decorated references are only available on system bus transactions generated by the processor core and targeted at the standard 512 KB peripheral address space based at 0x4000_0000 to 0x4007_FFFF (AIPS peripherals) and part of GPIO space starting from 0x400F_F000. BME can not be used to access other memory area including RAM, and flash.

The decoration semantic is embedded into address bits 28–19, creating a 448 MB space at addresses 0x4400_0000–0x5FFF_FFFF. These bits are stripped out of the actual address sent to the peripheral bus controller and used by the BME to define and control its operation. For most BME commands, a single core read or write bus cycle is converted into an atomic read-modify-write, that is, an indivisible "read followed by a write" bus sequence.

BME supports decorated stores (logical AND, logical OR, Bit Field Insert), and decorated loads (Load-and-clear 1 bit, Load-and-set 1 bit, Unsigned bit field extract).

The peripheral address bits 31–29 are always 010, which means the peripheral memory space used for BME starts from 0x40000000.

The data size is specified by the read or write operation and can be byte (8-bit), halfword (16-bit), or word (32-bit).

For logical AND, OR and XOR Store operations, the peripheral address bits 28–26 specify the operation code (opcode) as given in this table.

Table 5. Feature comparison between S08P and KE

Operation	Opcode	Comment
AND	001	During peripheral write, the corresponding peripheral data will be combined with the write data using logic AND operation.
OR	010	During peripheral write, the corresponding peripheral data will be combined with the write data using logic OR operation.

Table continues on the next page...

Table 5. Feature comparison between S08P and KE (continued)

Operation	Opcode	Comment
XOR	011	During peripheral write, the corresponding peripheral data will be combined with the write data using logic XOR operation.

The address bits 25–20 are “don’t care” bits. The address bits 19–0 are actual memory address bits for the peripheral space.

For storing Bit Field Insert (BFI), Load-and-Clear/Set 1 Bit, and Unsigned bit field extract (UBFX) operations, the peripheral address bits 28–26 specify the operation code (opcode) as given in this table.

Table 6. Feature comparison between S08P and KE

Operation	Opcode	Comment
Load-and-clear	010	During peripheral read, only the corresponding bit will be loaded to a variable and the target bit cleared.
Load-and-set	011	During peripheral read, only the corresponding bit will be loaded to a variable, and the target bit will be set.
Bit field insert/extract	1xx	Bit 28 is always 1, bits 27–26 will be filled with bit # bits accordingly. During peripheral read, the corresponding bit field from giving bit# will be extracted to the read variable. During peripheral write, only the corresponding bit field starting from the giving bit# of the location will be written with the corresponding bit field from the giving bit # of the write data. **NOTE:** Bit field insert/extract operation can not be applied to GPIO addresses starting from 0x400F_F000 as address bit 19 is occupied for bit width. So, GPIOx_PDOR/PSOR/PCOR/ PTOR/PDIR/ PDDR can not be accessed with this bit field operation. A special technique in C is used as demonstrated in the code snippet below.

For unsigned bit field extract operation, the address bits 18–0 are actual memory address bits for the peripheral space.

The following figures show the address bits representation for different operations.

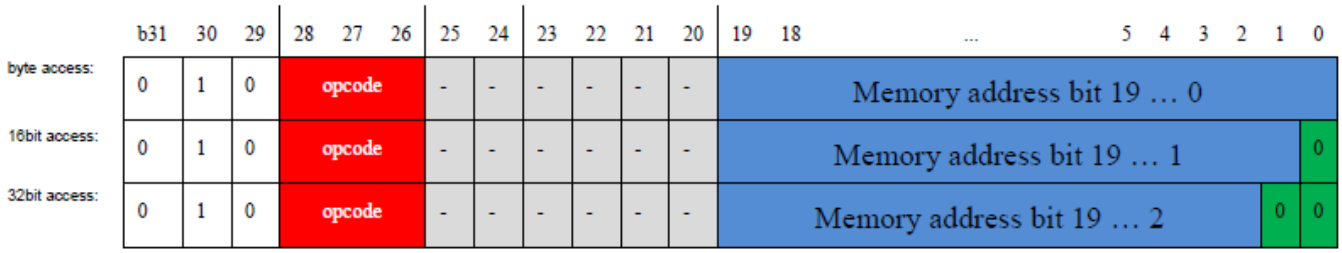

Figure 1. Address bits for AND, OR/XOR Store

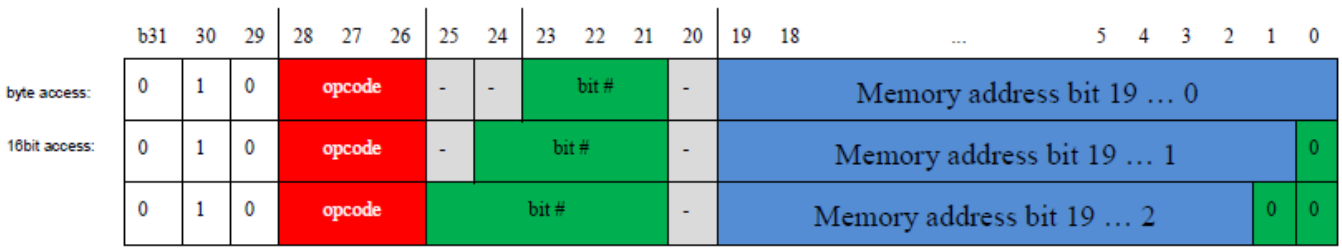

Figure 2. Address bits for Load-and-clear/set 1 bit operation

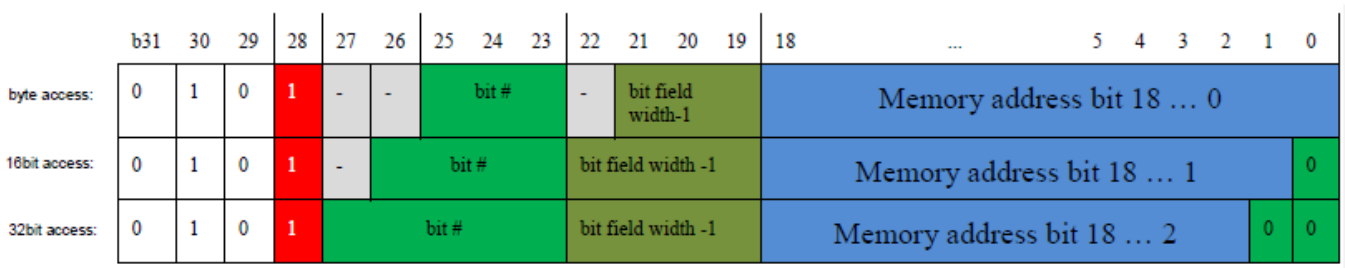

Figure 3. Address bits for bit field insert/extract operation

The following code snippet shows how to use BME operations:

```
// BME operation code
#define BME_OPCODE_AND1
#define BME_OPCODE_OR2
#define BME_OPCODE_XOR3
#define BME_OPCODE_BITFIELD4

//macro used to generate hardcoded AND address
#define BME_AND(ADDR)           (*(volatile uint32_t *)(((uint32_t)ADDR) |
(BME_OPCODE_AND<<26)))

BME_AND(&FTM2_OUTMASK) = 0x02;

//macro used to generate hardcoded OR address
#define BME_OR(ADDR)            (*(volatile uint32_t *)(((uint32_t)ADDR) |
(BME_OPCODE_OR<<26)))

BME_OR(&FTM2_OUTMASK) = 0x02;

//macro used to generate hardcoded XOR address
#define BME_XOR(ADDR)           (*(volatile uint32_t *)(((uint32_t)ADDR) |
(BME_OPCODE_XOR<<26)))

BME_XOR(&FTM2_OUTMASK) = 0x02;

//macro used to generate hardcoded bit field insert address
#define BME_BITFIELD_INSERT(ADDR,bit,width)        (*(volatile uint32_t *)
(((uint32_t)ADDR)    \
```

```
                                | (BME_OPCODE_BITFIELD <<26)  \
                                | ((bit & 0x1F)<<23) | ((width-1) & 0xF)<<19))

BME_BITFIELD_INSERT(&PORT_IOFLT,16,2) = (0x03 << 16);  // write 3 to PORT_IOFLT [10:8]

#define GPIO_ALIAS_OFFSET       0x000F0000L
#define GPIOB_PDOR_ALIAS        (((uint32_t)&GPIOB_PDOR)-GPIO_ALIAS_OFFSET)
BME_BITFIELD_INSERT(GPIOB_PDOR_ALIAS,19, 2) = (3<<19);  // write 3 to GPIOB_PDOR[20:19]

//macro used to generate hardcoded bit field extract address
#define BME_BITFIELD_EXTRACT(ADDR,bit,width)        (*(volatile uint32_t *)
(((uint32_t)ADDR)    \
                                | (BME_OPCODE_BITFIELD <<26)  \
                                | ((bit & 0x1F)<<23) | ((width-1) & 0xF)<<19))

 Data = BME_BITFIELD_EXTRACT(&ADC_R,0, 12);  // extract 12 bits: ADC_R[11:0]

//macro used to generate hardcoded load 1 bit clear address
#define BME_BIT_CLEAR(ADDR,bit)         (*(volatile uint32_t *)(((uint32_t)ADDR)    \
                                | (BME_OPCODE_BIT_CLEAR <<26)  \
                                | ((bit & 0x1F)<<21)))

//macro used to generate hardcoded load 1 bit set address
#define BME_BIT_SET(ADDR,bit)         (*(volatile uint32_t *)(((uint32_t)ADDR)    \
                                | (BME_OPCODE_BIT_SET <<26)  \
                                | ((bit & 0x1F)<<21)))
bit = BME_BIT_SET(&I2C0_S,1);    // read I2C0_S[IICIF] and then clear it by writing
                                 // 1 to it
bit = BME_BIT_CLEAR(&ACMP0_CS,5); // read ACMP0_CS[ACF] and then clear it
```

6 Clock modules

The Internal Clock Source (ICS) provides clock source options for the MCU. The module contains a frequency-locked loop (FLL) as a clock source that is controllable by either an internal or an external reference clock. The module can provide this FLL clock or either of the internal or external reference clocks as a source for the MCU system clock. There are also signals provided to control a low-power oscillator (OSC) module. These signals configure and enable the OSC module to generate its external crystal/resonator clock (OSCOUT) used by peripheral modules and as the ICS external reference clock source. The ICS external reference clock can be the external crystal/resonator (OSCOUT) supplied by an OSC, or it can be another external clock source.

The ICS and OSC structures on the KE family are very similar to S08P. The main difference between the KE02 subfamily and S08P is that the FLL multiplication factor in KE02 is fixed to 1024, which is double to that of S08P family. Therefore, the FLL output frequency is the reference clock frequency x 1024. The FLL output clock after the divider (BDIV) can be further divided down by half via SIM_BUSDIV[BUSDIV] to provide both the bus clock and the flash clock.

All the peripheral clocks can be gated via clock gating. By default after reset, all peripheral clocks except flash and SWD are gated off to conserve power. This is different from S08P which enables all peripheral clocks after reset.

The following figure shows the system clock diagram.

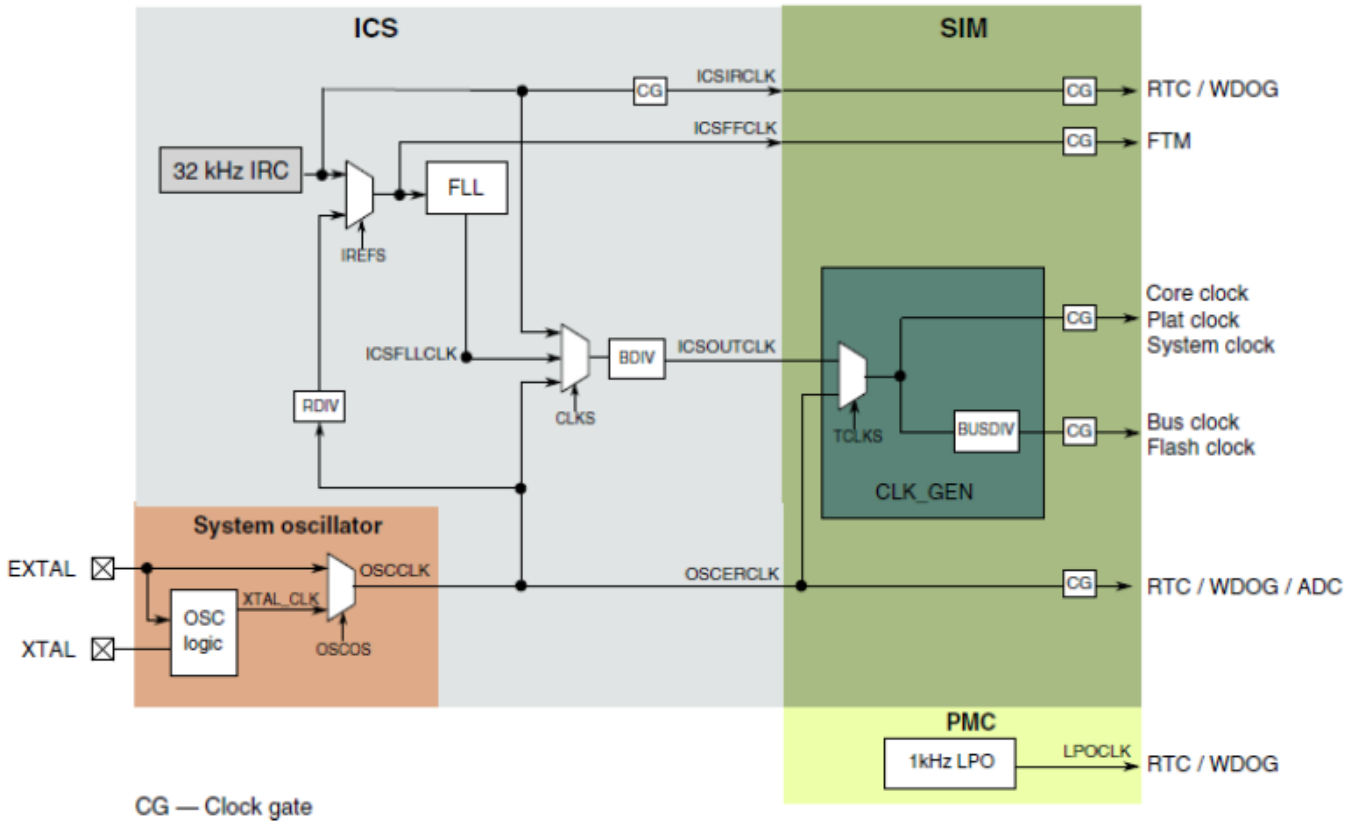

Figure 4. System clock diagram

The following code snippet shows how to initialize ICS and OSC modules to FEE mode from FEI mode with external crystal of 8 MHz:

```
/* assume external crystal is 8Mhz *
 */
/* enable OSC with high gain, high range and select oscillator output as OSCOUT
 *
 */
OSC_CR = OSC_CR_OSCEN_MASK
                            | OSC_CR_OSCSTEN_MASK          /* enable stop */
/* wait for OSC to be initialized
 *
 */
while(!(OSC_CR & OSC_CR_OSCINIT_MASK));

/* divide down external clock frequency to be within 31.25K to 39.0625K
 *
 */
          /* 8MHz */
           ICS_C1 = ICS_C1 & ~(ICS_C1_RDIV_MASK) | ICS_C1_RDIV(3); /* now the divided
                                                 frequency is 8000/256 = 31.25K */

/* change FLL reference clock to external clock */
ICS_C1 =  ICS_C1 & ~ICS_C1_IREFS_MASK;

while(ICS_S & ICS_S_IREFST_MASK);

/* wait for FLL to lock */
while(!(ICS_S & ICS_S_LOCK_MASK));

/* now FLL output clock is 31.25K*1024 = 32MHz
 *
 */
```

```
if(((ICS_C2 & ICS_C2_BDIV_MASK)>>5) != 1)
{
ICS_C2 = (ICS_C2 & ~(ICS_C2_BDIV_MASK)) | ICS_C2_BDIV(1);
}
/* now system/bus clock is the target frequency 16MHz
 *
 */
/* clear Loss of lock sticky bit */
ICS_S |= ICS_S_LOLS_MASK;
```

As mentioned in Figure 4 , it requires ICSOUTCLK be 40 MHz to achieve 40 MHz core/platform/system clock, so if using internal 32 KHz IRC as reference clock, it shall be trimmed to 39.0625 KHz , ICS_C2[BDIV] bit must be 0 and SIM_BUSDIV[BUSDIV] must be 1 (bus clock configured to 20 MHz which is max per data sheet). Typical trim value for 39.0625 KHz IRC is 0x4C, which can be written to ICS_C3 at startup.

The following code snippet shows how to gate on/off the KBI0 clock:

```
SIM_SCGC |= SIM_SCGC_KBI0_MASK;  // enable KBI0 clock
SIM_SCGC &= ~SIM_SCGC_KBI0_MASK;  // gate off KBI0 clock
```

7 System Integration Module (SIM)

This table lists the differences in the SIM modules of S08P, S08AC, and KE families.

Table 7. Comparison of SIM modules between S08 and KE product families

KE02	S08
Includes fields for Kinetis Family ID, Sub-Family ID, Revision Number, and Device Pin ID.	Includes Universal Unique Identifier (UUID)
The system reset status flags are reflected in low half word of SRSID register. There are three new status flags: MDM-AP System Reset Request (SIM_SRSID[MDMAP]) coming from SWD, Core Lockup (SIM_SRSID[LOCKUP]), and Stop Mode Acknowledge Error Reset (SIM_SRSID[SACKERR]) which is caused by the failure of entering Stop mode typically due to no acknowledge of entering Stop from IIC. In addition, the Cortex-M0+ core can generate software reset via the Application Interrupt and Reset Control Register.	The SYS_SRS register includes read-only status flags to indicate the source of the most recent reset. No direct software reset mechanism exists.
Clock gating is controlled by the SIM_SCGC register.	Clock gating is controlled by the SCG_C1 to SCG_C4 registers in S08P.
Pin remapping is controlled by the SIM_PINSEL register.	Pin remapping is controlled by the SYS_SOPT1 register in S08P.
Each Flextimer pin remapping can be individually controlled.	in S08P, each Flextimer module pins can only be rerouted in group.
The bus clock can be further divided by 2 for both bus and flash clock via the SIM_BUSDIV register.	There is no SIM_BUSDIV register. It can not further divide bus clock and flash clock down.
There is no illegal address register in KE family.	S08P has an illegal address register.

8 Power Management Controller

The Power Management Controller (PMC) module in KE family is similar to that in S08P. It supports Wait and Stop mode and has the power-on reset (POR) function. The low-voltage detect (LVD) and low-voltage warning interrupt can be enabled or disabled. If LVD is enabled in Stop (PMC_SPMSC1[LVDE] and PMC_SPMSC1[LVDSE] are both set) at the time the CPU executes a stop instruction, then the voltage regulator remains active during Stop mode. To get the lowest power consumption mode (Stop3), disable LVD in Stop mode by clearing these two fields.

The following code snippet shows how to enter Stop3 mode.

```
PMC_SPMSC1 = 0x00;//disable LVD;
stop();
```

9 Flash Memory, EEPROM, and Flash Memory Controller modules

The Flash Memory and EEPROM modules on the KE family are similar to S08P. On the KE family, the flash memory has cache, EEPROM has no cache. This will greatly improve the performance of flash access. However, all S08 devices have no flash cache. On the KE family, the flash memory can be accessed in word (32-bit), halfword (16-bit) or byte; however EEPROM can only be accessed in byte.

In addition, the KE family has another new flash operation feature: read-while-write. It allows read from flash while programming/erasing the flash. This is very useful for applications that do not need run code from RAM, especially for those devices with low memory footprint. This is enabled by setting Enable Stalling Flash Controller field in MCM_PLACR as given in this code.

```
MCM_PLACR |= MCM_PLACR_ESFC_MASK;
```

The flash memory controller is a memory acceleration unit that provides:

- an interface between bus masters and the 32-bit flash memory.
- a buffer and a cache that can accelerate flash memory data transfers

The flash memory controller provides two separate mechanisms for accelerating the interface between bus masters and flash memory. A 32-bit speculation buffer can prefetch the next 32-bit flash memory location and a 4-way, 4-set program flash memory cache can store previously accessed flash memory data for quick access times.

It not only allows instruction speculation and caching, but also data speculation and caching.

These different features can be enabled or disabled in the MCM_PLACR register.

There is a method to check whether the most recently programmed data is correct: invalidate the caches before reading the flash:

```
MCM_PLACR |= MCM_PLACR_CFCC_MASK;
```

This is to ensure that the cache contains the update value at the target location.

10 Pinout changes

The pinout of KE family is compatible with S08P. Both have 8 high-drive pins supporting 20 mA drive capability.

The KE family employs new I/O structure which is optimized for better transient protection and EMC performance. There are some changes to the pinout (from S08P) as given in this list.

- The debug port is replaced with two SWD pins: SWD_DIO pin, SWD_CLK pin. There is no BKGD function on the KE family.
- The KE family has a robust and reliable solution for use in harsh environments found in home appliances. The EFT can support 4.4 kV or higher and the Power ESD direct contact discharge is 20 kV or higher. Hardened IO-pad design to withstand system EMC and ESD interference includes
  - Filtering on GPIO/RST/IRQ for noise rejection
  - Drive strength control for EMI/EMC compliance
  - Default high impedance for all GPIOs (including unbonded pins)

11 Safety feature enhancement

11.1 Watchdog

The Watchdog (WDOG) Timer module is an independent timer that is available for system use. If it is not updated/refreshed with a certain data write sequence within a specified period of time, it will reset the MCU. It is used as a safety element to ascertain that the software is executing as planned and that the CPU is not stuck in an infinite loop or executing unintended code. It is designed as per IEC60730 safety standards.

This module is similar to the one in S08P family. Below are some guidelines when using this module:

- The WDOG register map is organized in big-endian mode similar to S08P, however the Cortex-M0+ core uses little-endian access mode. So when using 16-bit address mode to access WDOG registers such as WDOG_CNT, WDOG_TOVAL ,and WDOG_WIN, always take care that the high byte of value is in high byte address and low byte of value is in low byte address.
  For example, the following code snippet writes 0xA6 to WDOG_CNTL (high byte address), and 0x02 to WDOG_CNTH (low byte address).

```
WDOG_CNT = 0xA602;
```

- The counter refreshing sequence (that is, feeding WDOG) must be completed within 16 bus clocks, or it will generate reset.
- The unlock sequence write must be performed within 16 bus clocks for allowing updates to write-once configuration bits, otherwise it will generate reset.
- Software must make updates within 128 bus clocks after unlocking and before WDOG closing unlock window, or it will generate reset.

11.2 CRC

The Cyclic Redundancy Check (CRC) module generates 16/32-bit CRC code for error detection. It provides a programmable polynomial, WAS, and other parameters required to implement a 16-bit or 32-bit CRC standard. The 16/32-bit code is calculated for 32 bits of data at a time. This module is similar to that in S08P family, except for the following differences.

- Supports bitwise/bytewise transposition of input data or output data (the CRC result). S08P supports only bitwise transposition.
- Supports 32-bit access to the registers. S08P supports only 8-bit access to the registers.

For example, CRC_DATA register on the KE family is a 32-bit register, which is same as concatenation of four 8-bit registers from CRC_D0 to CRC_D3 on S08P.

12 Port Control and GPIO

The Port Control module controls glitch filter, pullup, and high drive. In KE family, there are four 32-bit registers, PORT_IOFLT, PORT_PUEL/H, and PORT_HDRVE. The functionality of these registers are similar to S08P with the only difference that these registers can be accessed in 32-bit on the KE family. This will greatly improve the control speed on multiple I/Os.

NOTE

- The PORT_PUEL/H are port pullup enable registers corresponding to combined PORT_PTxPEs registers in S08P. So, the code that access any PORT_PTxPE in S08P shall be changed accordingly. There are no port input and output enable registers (PORT_PTxIE, PORT_PTxOE) in the KE family. Instead, the KE family uses GPIOx_PDDR to control data direction and GPIOx_PIDR to enable or disable GPIO inputs.
- All GPIO pins are peripheral inputs by default after reset and in this case, the corresponding bits in Pin Data Input Registers will read 0. In order to correctly read a pin state, the corresponding bit in Port Input Disable Register (PIDR) must be cleared.

The GPIO module in KE family controls the direction, and the input and output data of I/O pins, also similar to the S08P family. However, the GPIO registers in KE family can be accessed in 32-bit. In addition, its GPIO can also be accessed from single (core clock) cycle IOPORT interface. So, this is called fast GPIO or FGPIO. The memory map of this FGPIO is starting from 0xF800_0000.

Accesses via the IOPORT interface (FGPIO memory space) occur in parallel with any instruction fetches and will therefore complete in a single cycle of core clock. See the following figure.

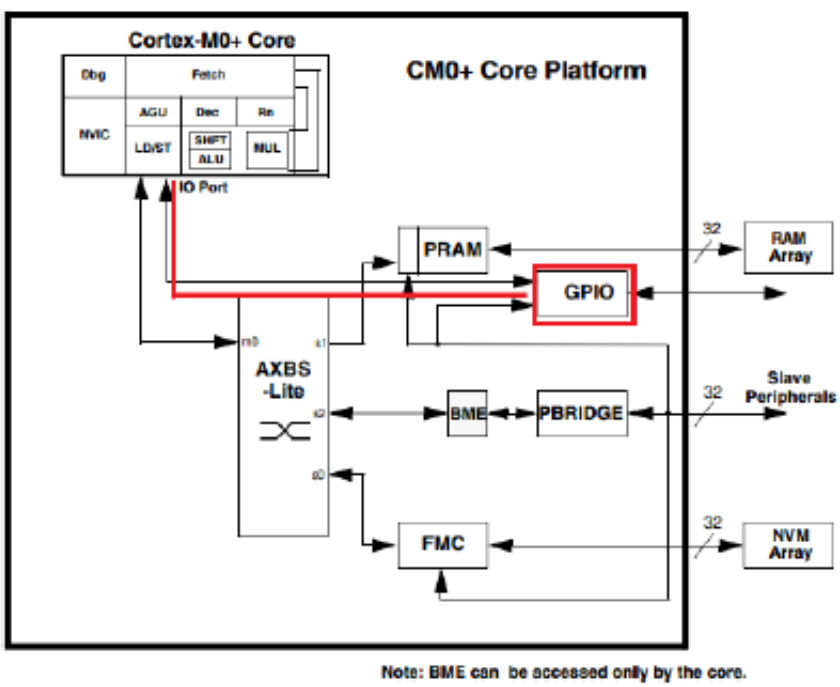

Figure 5. Cortex-M0+ core platform block diagram

Following is the list of main features of GPIO:

- Pin input data register visible in all digital pin-multiplexing modes
- Pin output data register with corresponding set/clear/toggle registers

- Pin data direction register
- Zero wait state access to GPIO registers through IOPORT

NOTE

All GPIO pins are peripheral inputs by default after reset and in this case, the corresponding bits in Pin Data Input Registers will read 0. In order to correctly read a pin state, the corresponding bit in Port Input Disable Register (PIDR) must be cleared.

The following code snippet demonstrates how to use Port control and GPIO:

```
PORT_IOFLT   = 0x0;   // Filter clock source is BUSCLK
PORT_PUEH    = 0x0;
PORT_PUEL     = 0x1;   // Enable pullup for PTA0
PORT_HDRVE = 0x0;   // No high drive pin

/* normal GPIO manipulation */
GPIOA_PDDR = 1<<0;  // PTA0 is set as output pin
GPIOA_PTOR = 1<<0;  // Toggle PTA0

GPIOA_PIDR  &= ~2;    // configure PTA2 pin as GPIO input, must clear PIDR bit
GPIOA_PDDR &= ~2;   // direction is INPUT


/* GPIO manipulation via IOPORT interface */
FGPIOA_PDDR = 1<<0;  // PTA0 is set as output pin
FGPIOA_PTOR = 1<<0;  // Toggle PTA0
```

13 Timer modules

13.1 RTC

The real-time counter (RTC) consists of one 16-bit counter, one 16-bit comparator, several binary-based and decimal-based prescaler dividers, three clock sources, one programmable periodic interrupt, and one programmable external toggle pulse output. The difference between S08P and KE family lies in the register access width.

Two 8-bit status and control registers (RTC_SC1 and RTC_SC2) in S08P are cascaded into single 32-bit register RTC_SC in KE family. Two 8-bit RTC modulo registers (RTC_MODH, RTC_MODL) in S08P are cascaded into single 32-bit RTC_MOD register.

Similar is the case for RTC counter registers. So the code to access RTC_SC1 and RTC_SC2, RTC_MODH, RTC_MODL, RTC_CNTH, RTC_CNTL in S08P shall be changed to access RTC_SC, RTC_MOD, RTC_CNT accordingly.

13.2 FlexTimer

The FlexTimer (FTM) is built upon a simple timer—TPM module used on S08 devices, and extends the functionality to meet the demands of motor control, digital lighting solutions, and power conversion. Several key enhancements are made to the Timer module as follows.

- Signed up counter
- Deadtime insertion hardware
- Fault control inputs
- Enhanced triggering functionality
- Initialization and polarity control

Key features of the FlexTimer module in S08 devices are given in the following list.

- FTM source clock is selectable.
  - Source clock can be the system clock, the fixed frequency clock, or an external clock.
  - Fixed frequency clock is an additional clock input to allow the selection of an on-chip clock source other than the system clock.
  - Selecting external clock connects FTM clock to a chip level input pin therefore allowing to synchronize the FTM counter with an off chip clock source
- Prescaler divide-by 1, 2, 4, 8, 16, 32, 64, or 128
- 16-bit counter
  - It can be a free-running counter or a counter with initial and final value.
  - The counting can be up or up-down.
- Each channel can be configured for input capture, output compare, or edge-aligned PWM mode.
- In Input Capture mode:
  - The capture can occur on rising edges, falling edges or both edges
  - An input filter can be selected for some channels

- In Output Compare mode, the output signal can be set, cleared, or toggled on match.
- All channels can be configured for center-aligned PWM mode.
- Each pair of channels can be combined to generate a PWM signal with independent control of both edges of the PWM signal.
- The FTM channels can operate as pairs with equal outputs, pairs with complementary outputs, or independent channels with independent outputs.
- The deadtime insertion is available for each complementary pair.
- Generation of match triggers
- Software control of PWM outputs
- Up to four fault inputs for global fault control with programmable polarity
- The polarity of each channel is configurable.
- Generation of an interrupt per channel
- Generation of an interrupt when the counter overflows
- Generation of an interrupt when the fault condition is detected
- Synchronized loading of write buffered FTM registers
- Write protection for critical registers
- Backward-compatible with TPM
- Testing of input captures for a stuck at 0 and 1 conditions
- Dual edge capture for pulse and period width measurement
- Global time base to sync different FlexTimer modules

Compared with S08P, the FlexTimer module in KE family has new enhanced features as listed below.

- Invert control/channel swap
- Fault input polarity control
- Intermediate load
- Software output control
- Debug mode function
- TOF frequency/ periodic TOF

Fault input polarity can be configured as active HIGH or LOW by clearing or setting FTMx_FLTPOL[FLTnPOL]. The FTM can be set to still function when the debug mode is entered. This is done by setting FTMx_CONF[BDMMODE].

The invert functionality swaps the signals between channel (n) and channel (n+1) outputs. The inverting operation is selected when (FTMEN = 1), (QUADEN = 0), (DECAPEN = 0), (COMBINE = 1), (COMP = 1), (CPWMS = 0), and (INVm = 1), where m represents a channel pair.

With intermediate load feature, the PWMLOAD register allows to update the MOD, CNTIN, and C(n)V registers with the content of the register buffer at a defined load point. In this case, it is not required to use the PWM synchronization. Following loading points/conditions are possible.

- When the FTM counter wraps from MOD value to CNTIN
- At the channel (j) match (FTM counter = C(j)V) when When CHjSEL = 1.

After the loading points are configured, they must be enabled by setting FTMx_PWMLOAD[LDOK]. This field must be set for the load to occur at the next loading points.

The following code snippet shows how to enable intermediate load feature with channel 2 match:

```
FTM2_MOD = 1200;
  FTM2_CNTIN = 200;
  FTM2_MODE = 0x05; /* FTM features are enabled ans write protection is disabled */
  FTM2_COMBINE = 0x23;  /* Combine is enabled, Output CH0 and CH1 are complementary */
  FTM2_SYNCONF = 0x4;   // CNTIN register is updated with its buffer value by the
                                                    // PWM synchronization.
 FTM2_C0SC = 0x28;   /* No Interrupts; High True pulses on Edge Aligned PWM */
  FTM2_C1SC = 0x28;
  FTM2_C0V = 300;  /* 25% pulse width */
  FTM2_C1V = 1100; /* 91% pulse width */
  FTM2_SC = 0x08;     /*  CLK source is System clock / 1 */
  FTM2_SYNC = 0x80; /*   OUTMASK register is updated with the value of its buffer only by
the PWM synchronization.  */
  FTM2_C0V = 150;  /* 25% pulse width */
  FTM2_C1V = 550;
  FTM2_MOD = 600;
  FTM2_PWMLOAD = 0x203 ; /* enable load on both load points */
```

The software output control forces the channel output according to software-defined values at a specific time in the PWM generation. The software output control is selected when (FTMEN = 1), (QUADEN = 0), (DECAPEN = 0), (COMBINE = 1), (CPWMS = 0), and (CHnOC = 1). The CHnOC field enables the software output control for a specific channel output and the CHnOCV selects the value that is forced to this channel output.

The TOF frequency can be set by FTMx_CONF[NUMTOF]. The TOF flag will be set for the first counter overflow, and not set for the next N overflows where N = NUMTOF and not 0. This is useful when the PWM event needs to be changed only after several PWM cycles. This greatly reduces CPU overload.

In addition, the user must be aware of the following two other differences.

- Nearly all of the FTM channels on the KE family can be remapped individually to different pins, but on S08P, only FTM2 channels can have alternate pin reassignment in group. This is done by using Pin Selection Register (SIM_PINSEL).
- The registers can be accessed in 32-bit for better performance.

13.3 PIT

The KE family Periodic Interrupt Timer (PIT) is an array of timers/channels that can be used to raise interrupts and triggers. It is a 32-bit module and each timer is a count down timer with the initial value in PIT_LDVALn register. Each time a timer reaches 0, it will generate a trigger pulse and set the interrupt flag. A new interrupt may be generated if enabled by setting PIT_TCTRLn[TIE] and only after the previous one is cleared. The counter period can be restarted by first disabling and then enabling the timer with PIT_TCTRLn[TEN]. The current counter value of the timer can be read via the PIT_CVALn register.

It is possible to change the counter period without restarting the timer by writing PIT_LDVALn register with the new load value and this value will take effect after the next trigger event.

The timers in PIT can be chained as if those timers were only one timer.

NOTE

The module must be enabled by clearing PIT_MCR[MDIS] before any other setup is done.

The following code snippet shows how to set timer chain (timer 1 and 0) to count 10000 cycles before generating PIT channel1 interrupt:

```
enable_irq(23);// enable PIT channel 1/timer1 interrupt
PIT_MCR = 0; // enable PIT
PIT_LDVAL0 = 1000-1;
PIT_LDVAL1 = 10-1;
```

```
PIT_TCTRL0 = PIT_TCTRL_TEN_MASK;
PIT_TCTRL1 = PIT_TCTRL_TIE_MASK|PIT_TCTRL_TEN_MASK
                      | PIT_TCTRL_CHN_MASK;   // Enable PIT Chain Mode for timer1:0

#define VECTOR_039 pit_ch1_isr
void pit_ch1_isr(void)
   {
            // clear PIT ch1/timer1 interrupt flag
            PIT_TFLG1  = PIT_TFLG_TIF_MASK;
            printf("\tEntered PIT CH1 ISR *\n");
   }
```

14 Debug

The KE family implements a 2-pin Serial Wire Debug (SWD) port: SWD_CLK, and SWD_DIO. The SWD is based on the ARM CoreSight architecture. The basic debug functionality includes processor halt, single-step, processor core register access, Reset and HardFault Vector Catch, software breakpoints, and full system memory access. For more information on this architecture, visit **arm.com**.

Through the ARM Debug Access Port (DAP), the debugger has access to the status and control elements, implemented as registers on the DAP bus. These registers provide additional control and status for low power mode recovery and typical run-control scenarios. The status register bits also provide a means for the debugger to get updated status of the core without having to initiate a bus transaction across the crossbar switch, thus remaining less intrusive during a debug session. It is important to note that these DAP control and status registers are not memory mapped within the system memory map and are only accessible via the Debug Access Port using SWD.

The KE family Debug module supports two breakpoints and two watchpoints. But there is no Micro Trace Buffer (MTB), so the user can't get the trace function using SWD module.

NOTE

During reset, do not try to read bit 2 (security state) of the MDM-AP Status Register. This bit is valid only when the device is not in reset state.

15 ADC module

The 12-bit analog-to-digital converter (ADC) is a successive approximation ADC which is faster than sigma delta while still having similar good accuracy.

Following is a list of features of the ADC module.

- Linear Successive Approximation algorithm with 8-, 10-, or 12-bit resolution
- Up to 16 external analog inputs, external pin inputs, and 5 internal analog inputs including internal bandgap, temperature sensor, and references
- Output formatted in 8-, 10-, or 12-bit right-justified unsigned format
- Single or Continuous Conversion (automatic return to idle after single conversion)
- Support up to eight result FIFO with selectable FIFO depth
- Configurable sample time and conversion speed/power
- Conversion complete flag and interrupt
- Input clock selectable from up to four sources
- Operation in wait or stop3 modes for lower noise operation
- Asynchronous clock source for lower noise operation
- Selectable asynchronous hardware conversion trigger
- Automatic compare with interrupt for less-than, or greater-than or equal-to, programmable value

The ADC module in KE family is similar to that in S08P, with the only difference that the registers can be accessed in 32-bit. The ADC 12-bit result can be read in one cycle. So, it is much faster than S08P.

The following code snippet shows how to initialize the ADC and read ADC result register in the ADC handler :

```
void ADC_init(void)
{
/* The following code segment demonstrates how to initialize ADC by low-power mode, long
sample time, bus frequency, hardware triggered from AD1, AD3, AD5, and AD7 external pins
with 4-level FIFO enabled */
ADC_APCTL1 = ADC_APCTL1_ADPC6_MASK | ADC_APCTL1_ADPC5_MASK | ADC_APCTL1_ADPC3_MASK |
ADC_APCTL1_ADPC1_MASK;
ADC_SC3 = ADC_SC3_ADLPC_MASK | ADC_SC3_ADLSMP_MASK | ADC_SC3_MODE1_MASK;
// setting hardware trigger
ADC_SC2 = ADC_SC2_ADTRG_MASK ;
//4-Level FIFO
ADC_SC4 = ADC_SC4_AFDEP1_MASK | ADC_SC4_AFDEP0_MASK;
// dummy the 1st channel
ADC_SC1 = ADC_SC1_ADCH0_MASK;
// dummy the 2nd channel
ADC_SC1 = ADC_SC1_ADCH1_MASK | ADC_SC1_ADCH0_MASK;
// dummy the 3rd channel
ADCSC1 = ADCSC1_ADCH2_MASK | ADC_SC1_ADCH0_MASK;
// dummy the 4th channel and ADC starts conversion
ADC_SC1 = ADC_SC1_AIEN_MASK | ADC_SC1_ADCH2_MASK | ADC_SC1_ADCH1_MASK | ADC_SC1_ADCH0_MASK;
}

/* FIFO ADC interrupt service routine */
unsigned short buffer[4];
void ADC_isr(void)
{
      /* The following code segment demonstrates read AD result FIFO */
      // read conversion result of channel 1 and COCO bit is cleared
      buffer[0] = ADCR;
      // read conversion result of channel 3
      buffer[1] = ADCR;
      // read conversion result of channel 5
      buffer[2] = ADCR;
      // read conversion result of channel 7
      buffer[3] = ADCR;
}
```

16 References

- KLQRUG, Kinetis L Peripheral Module Quick Reference, available on **freescale.com**
- AN4347, Transitioning Applications from S08AC and S08FL Family to S08PT Family, available on **freescale.com**
- AN4560: PWM synchronization using Kinetis FlexTimers, available on **freescale.com**
- Cortex -M0 Devices Generic User Guide, available on **arm.com**

17 Glossary

ACMP	Analog Comparator
BME	Bit Manipulate Engine
CRC	Cyclic Redundant Check
DMA	Direct Memory Access
FMC	Flash Memory Controller
IRC	Internal Reference Clock
PWT	Pulse Width Timer
RTC	Real Time Counter

Table continues on the next page...

18 Revision history

Revision number	Date	Substantive changes
0	07/2013	Initial release
1	11/2013	Added 40 MHz support

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number AN4757
Revision 1, 11/2013