Freescale Semiconductor
Application Note

Document Number:AN4757
Rev 0, 07/2013

从 S08 到 Kinetis E 系列移植指南

通过: **William Jiang**

内容

1 简介

Kinetis E (KE) 系列是采用稳健技术、新式 5 V I/O 焊盘和 ARM® Cortex® M0+ 内核的首款 Kinetis MCU。 此 KE 系列还针对不同的外设模块引入了许多新功能。 这些新功能使得 KE 系列有更低的功耗、更好的 EFT/ESD 保护和更高的性能。

本应用说明概述了 S08（主要指 S08P）与 KE 产品系列之间的主要差别，并提供了代码转换建议方法以帮助缩短学习时间。

2 设备概述

Kinetis E 系列设备面向家用电器细分市场，具有以下特性。

- 基于运行频率高达 20 MHz（对于 KE02）/48 MHz（对于其他 KE）的 ARM Cortex-M0+ 内核，提供位处理引擎 (BME)
- 高达 128 KB 闪存
- 高达 16 KB SRAM
- ROM（仅限某些 KE 成员）
- 结合内部参考时钟 (IRC) 的时钟发生器
- 晶体振荡器和 FLL
- 多达 3 个具有 LIN 从机功能的 SCI
- 多达 2 个 SPI
- 2 个 I2C
- 1 个 CAN（仅限某些 KE 成员）
- 3 个 16 位 FlexTimer

© 2013 Freescale Semiconductor, Inc.

- 1 个 PWT
- 1 个 32 位双通道 PIT
- RTC
- 符合 IEC60730 规范的看门狗
- CRC
- 多达 16 通道的 12 位 ADC，提供 8 级 FIFO
- 2 个 ACMP
- 16 通道 TSI 用于触控输入（仅限某些 KE 成员）
- 4 通道 DMA（仅限某些 DMA 成员）
- 多达 82 个 GPIO 引脚，8 个高驱动引脚支持 20 mA
- 支持 2.7–5.5 V 工作电压的嵌入式电压调节器，通电复位，可编程的低电压检测器
- 两种低功耗模式
- 长达 10 字节的独特芯片 ID
- 单线调试 (SWD) 接口
- 支持 –40 °C 到 +105 °C 的宽广温度范围

与 S08 设备（S08AC、S08P）相比，KE 系列采用 ARM Cortex-M0+ 内核、BME 和 32 位设备，大多数外围设备与 S08P 类似。 下表汇总了这些系列之间的主要特性比较：

表 1. S08AC、S08P 与 KE 产品系列的特性比较

特性	S08AC	S08P	KE 系列
内核平台	8 位 S08 内核，最大频率 40 MHz	低功耗 8 位 S08 内核，最大频率 20 MHz	高能效 32 位 Cortex-M0+ 内核，最高支持 48 MHz，可在一个周期内处理大多数指令
总线时钟	核心频率的一半，最高 20 MHz	与核心频率相同，最高 20 MHz	可与核心频率相同，最高 48 MHz
单周期 32 位 x 32 位乘法	仅支持 8 位 x 8 位	仅支持 8 位 x 8 位	有
调试	1 引脚调试模块 (BDM)	1 引脚调试模块 (BDM)	2 引脚串行线调试 (SWD)
嵌套向量中断控制器 (NVIC)	• 无 NVIC • 无 IPC • 不支持硬件嵌套中断 • 不支持中断向量重定位	• 无 NVIC • 包含中断优先级控制器 (IPC)，这需要软件代码支持嵌套中断。 • 不支持中断向量重定位	• 支持中断向量重定位，可在闪存或 RAM 中重定位。 • 真正的硬件中断嵌套，无需任何软件代码
数据访问字节顺序	高字节序	高字节序	低字节序
直接内存访问 (DMA)	无	无	已经为 KE06 成员添加
功率模式	• RUN • WAIT • STOP2（部分断电，最低功耗） • STOP3	• RUN • WAIT • STOP3（典型电流值 1.3 µA）	• 类似于 S08P
EEPROM	无	有	大多数成员有
闪存控制器 (FMC)	无	无	有
FlexTimer	无，只提供传统的 TPM	• 提供扩展的 TPM 功能用于支持电动机控制和电源应用 • TPM 功能向下兼容 • 在停止模式下不起作用	增强的 Flextimer，提供中间负载、全局时基、周期 TOF、故障极性控制、通道交换/反转控制和调试模式选项
系统节拍 (SysTick)	无	无	24 位定时器（内核时钟/16）
RTC	无	有	有

下一页继续介绍此表...

表 1. S08AC、S08P 与 KE 产品系列的特性比较 (继续)

特性	S08AC	S08P	KE 系列
ADC 模块	• 高达 10 位分辨率 • 寄存器访问仅限 8 位	• 支持高达 12 位分辨率 • 寄存器访问仅限 8 位	类似于 S08P，但寄存器访问为 32 位
TSI 模块	无	扫描结束中断（只限某些 S08P 成员）	仅限某些 KE 成员 • 扫描结束中断 • 超出范围中断 • 噪音检测
标识寄存器 (ID)	2 字节	8 字节	8 字节通用唯一标识符 (UUID) 加 2 字节 Kinetis ID

ARM Cortex M0+ 是市场上尺寸最小、功耗最低的 ARM 处理器，与其他所有 Cortex-M 内核兼容。以下是该处理器相比 8 位微处理器的优势。

- 这是迄今设计出来的能效最高的 32 位处理器，相比 8/16 位处理器，在能效方面具有明显的优势。在降低功耗的同时，仍能实现 2.15 CoreMark/MHz（ARM 编译器 5.0.3 版）的高性能，这是 8/16 位处理器的 2 到 40 倍，并且比 Cortex-M0 高 9%。
- 此外，它的编码效率也高于 8 位和 16 位 CPU。
- 由于采用了支持 C 语言的体系结构，为动态异常处理程序提供可重定位向量表（可将向量表移入 RAM），包含简单的指令集并整合了功能最丰富的开发生态系统，因此，与 8 位微处理器产品相比，能够使软件开发变得简单而快速。

有关此处理器的详细信息，请访问： arm.com。

3 编程模型

3.1 寄存器集

本节阐述 S08 系列可用的寄存器，以及 Kinetis E 系列使用的 ARM Cortex-M0+ 处理器。

以下是 S08 系列的寄存器集合。

- 一个累加器 A，即通用型 8 位寄存器，用作算术与逻辑运算以及内存加载/存储指令的目标和源。
- 一个 16 位索引寄存器 H:X，用作索引引用指针，X 可用作 8 位通用型寄存器。
- 一个 16 位堆栈指针（SP），指向自动后进先出（LIFO） 堆栈上的下一个可用位置。
- 除了 16 位程序计数器（PC）外，还有一个 8 位条件代码寄存器 （CCR），其中包含中断掩码 （I），以及五个标志用于指示最近执行的指令的结果。

以下是 KE 系列使用的 Cortex M0+ 内核寄存器的说明。

- 16 个 32 位寄存器 R0-R15
  - R0-R12 基本上可用于所有指令
  - R13 用作堆栈指针
  - R14 用作链接寄存器（用于子例程和异常返回）
  - R15 用作程序计数器

Cortex-M0+ 内核寄存器都不可直接寻址。由于所有的 Cortex-M0+ 内核寄存器都是 32 位，因此能够有效支持 32 位算术和逻辑运算。

除了 R0-R15 通用型寄存器外，Cortex M0+ 还有一些专用的寄存器，例如程序状态寄存器、异常屏蔽寄存器、中断屏蔽寄存器和控制寄存器。

- 程序状态寄存器（xPSR）：具有多个别名的单个 32 位寄存器，每个别名提供不同内容子集的视图。 PSR 结合了以下寄存器。
  - 应用程序状态寄存器（APSR）：APSR 基于用户立场包含了 ALU 状态标志。
  - 中断程序状态寄存器 （IPSR）：此寄存器包含当前正在执行的中断数（如果当前没有任何活动的中断，则为零），以供操作系统和异常处理使用。
  - 执行程序状态寄存器 （EPSR）：包含反映执行状态的位，不可直接访问。

有关 Cortex M0+ 寄存器及其功能的详细信息，请参见以下网站上提供的《Cortex-M0 设备常规用户指南》： **arm.com**

3.2 寻址模式

寻址模式定义了 CPU 访问操作数和数据的方式。 S08 支持多个寻址模式。 下面列出了这些模式。

- 固有寻址模式（INH）
- 相对寻址模式 （REL）
- 直接寻址模式 （DIR）
- 索引寻址模式，具体如下：
  - 无偏移 （IX）
  - 无偏移带后增量 （IX+）
  - 8 位偏移 （IX1）
  - 8 位偏移带后增量 （IX1+）
  - 16 位偏移 （IX2）
  - SP 相对 8 位偏移 （SP1）
  - SP 相对
  - 16 位偏移 （SP2）

- 内存间寻址模式，具体如下：
  - 直接-直接
  - 立即-直接
  - 索引-直接带后增量
  - 直接-索引带后增量

Cortex-M0+ 寻址模式可以很方便地应用到内存访问，并具有以汇编语言表达的偏移寻址模式：[<Rn>,<offset>]。

3.3 指令集

S08 支持 8 位指令集。 指令的长度可变，经扩展后可以包括指令流中长度为 1-4 字节的操作数。

Cortex-M0+ 支持部分 Thumb-2 指令集。 尽管大多数指令都是 16 位，但也有少量的 Thumb 指令是 32 位，例如 BL、DMB、DSB、ISB、MRS 和 MSR。 一个重要的差别在于，指令不能扩展为包括操作数数据。 有关指令的任何信息必须在指令本身内部编码。 这种限制的后果之一是无法在指令中指定任意 32 位常数或地址，因此，必须采用其他方法来实现此目的。 支持 Cortex-M0+ 的所有 C 编译器都毫无限制地允许程序员使用此功能。

3.4 工作模式

S08 可在以下模式下工作。

- 运行、等待和停止模式：等待和停止模式是低功耗模式，可分别通过执行 WAIT 和 STOP 指令进入。 有关这些模式的详细信息，请参见电源管理控制器 。
- 后台调试模式：后台模式功能通过 S08 内核中的后台调试控制器 (BDC) 管理。 在软件开发过程中，可以结合使用 BDC 和片上调试模块 (DBG) 进行 MCF 操作分析。 有关后台调试的详细信息，请参见调试 。
- 安全模式：处于安全模式时，将会限制从外部访问内部内存，以便只有从安全内存中提取的指令才能访问安全内存。 当代码从内部内存运行时，可以访问所有资源而不存在任何限制。

Cortex-M0+ 支持两种模式。

- 线程模式（用于用户进程）：此模式用于执行应用软件。
- 处理器模式：此模式用于处理异常情况，发生异常时，将自动进入此模式。

同时还定义了“权限”的概念。无权限的执行只能访问或者根本无法访问某些资源（例如，无权限代码无法屏蔽或取消屏蔽中断）。 处理器模式执行始终是有权限的。 默认情况下，线程模式也是通过权限执行的，但程序员可将线程模式配置为不使用权限执行。 可以使用这种配置来提供某种程度的系统保护，以防止不良或恶意程序的破坏。

3.5 堆栈

S08 堆栈朝低位 RAM 地址（即以降序）扩展。 堆栈可定位在使用 RAM 的 64 KB 地址空间内的任意位置，并可以采用任何大小，但不能超过可用的 RAM 量。 通常将其放置在内部 SRAM 中以实现最佳性能。 可按字节访问堆栈。

堆栈指针（SP）指向堆栈上的下一个可用位置，复位时将初始化为 0x00FF；此地址位于直接页面寄存器段或 RAM 中。但是，建议让应用代码重新初始化 SP，使其指向 RAM 的最后一个位置，如以下示例中所示：

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

在此情况下，至少需要耗时五个 CPU 周期才能重新初始化堆栈。

堆栈指针在存储或推送操作之后将会递减，在加载或弹出操作之前将会递增。

Cortex-M0+ 支持当前堆栈指针寻址到的满递减堆栈（类似于 S08 堆栈）。 但是，堆栈指针指示堆栈内存中的最后一个堆栈条目。 堆栈大小仅受可用 RAM 空间的限制。 Cortex-M0+ 堆栈指针通常初始化为已分配堆栈区域顶部上方的字（32 位）。 复位时初始化堆栈指针不需要任何 CPU 指令（即 CPU 周期）。 由于堆栈模块是满递减的，因此，堆栈指针在第一次存储之前会递减，并将第一个字放置在已分配区域顶部的堆栈上。 Cortex-M0+ 上的所有堆栈访问都以字大小进行。

3.6 异常和中断

S08P 具有一个不可屏蔽的中断源（SWI）和 39 个可屏蔽的中断源，其中包括一个外部可屏蔽中断 IRQ。 可以使用对应于最高优先级的最小向量编号固定中断优先级。 在启用中断优先级控制器后，则可以支持四个优先级。 实施嵌套中断方案时需要额外的软件开销。

Cortex-M0+ 具有集成的嵌套向量化中断控制器，最多支持 32 个独立的中断源。 共有四个中断优先级，支持真正的硬件嵌套中断方案。 Cortex-M0+ 还支持外部不可屏蔽中断 (NMI) 以及多个内部中断，例如硬故障、SVC 等。

3.7 向量表

S08P 向量表定位在内部闪存上端的固定地址上。 不支持向量表重新定位。 每个向量包含相应中断处理程序/服务例程的地址。 向量下端地址存储中断处理程序地址的高位字节，而向量上端地址存储中断处理程序地址的低位字节。POR 和其他复位向量位于闪存 0xFFFE–0xFFFF 的高端。

KE 系列向量默认位于地址 0x00000000 上。 在初始化期间，可将其重定位到闪存或 RAM 区域中的某个位置。 将向量表放置在内部 SRAM 中可以提高性能。 在向量表中，每个条目包含相应处理程序例程的起始地址。 位于向量表偏移 0 位置的第一个向量包含初始/起始 SP（监管器 SP），位于向量表偏移 4 位置的第二个向量包含起始 PC。 以下代码段显示了前两个向量条目：

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

要将向量表重新定位到偏移 vtor 位置，请使用以下代码段：

```
SCB_VTOR = vtor
```

3.8 复位和中断处理程序

对于 S08，在使用“从中断返回”(RTI) 指令时，中断服务例程就会结束。该指令可通过读出堆栈中以前保存的信息，将 CCR、A、X 和 PC 寄存器还原到中断前的值。

使用 C 代码时，为 S08 设备定义中断处理程序，则必须使用编译器指令来告诉编译器该处理程序是一个中断服务例程，而不是标准的 C 例程/函数。 以下示例显示了如何使用 CodeWarrior 定义中断服务例程：

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

Cortex-M0+ 支持硬件中的所有异常进入和退出序列，因此允许中断例程是符合 ARM 体系结构过程调用标准（AAPCS）的标准 C 函数。 只需通过引用相应的地址，就能在向量表中安置任何兼容的函数，并将其用作处理器。使用此方案时，代码开发员并不需要记住此类编译器指令。 以下代码段显示了如何使用 C 语言定义与标准函数相同的 Rtc_ISR ：

```
void Rtc_ISR(void)
{
 …
}
```

3.9 内存

S08 内核可以寻址 64 KB 的内存空间，并将内存分割成五个主要段，如下表中所示。

表 2. S08 内存映射

名称	地址	备注
直接页面寄存器	0x0000–0x00xx	多达 128 字节
RAM	0x00xx–[1]	包括某些直接页面位置
高页面寄存器	0x1800–0x18yy	系统配置
闪存	[2]–0xFFFF	多达 60 KB
向量	0xFFCO–0xFFFF	多达 32 x 2 字节

1. 上限取决于给定的设备
2. 下限取决于给定的设备

S08 内核能够以高效寻址模式访问直接页面寄存器/RAM，并允许使用 BSET、BCLR、BRSET 和 BRCLR 位处理指令来设置、清除或测试这些寄存器中的任何位。 这种“位带”操作十分简单，但需耗时 2–5 个 CPU 周期。

Cortex-M0+ 处理器可以访问 4 GB 内存空间，并将内存分割成多个空间。 但并不支持类似于“位带”的操作。 不过，KE 系列添加了位操作引擎 (BME) 以用于实现这种“位带”操作。 有关 BME 的详细信息，请参见位操作引擎（BME）。 下表简要描述了 KE02 系统内存映射。

表 3. KE02 内存映射

系统 32 位地址范围	用途
0x0000_0000–0x07FF_FFFF	程序闪存和只读数据（包括前 196 个字节中的异常向量）
0x1000_0000–0x1000_00FF2	EEPROM

下一页继续介绍此表...

表 3. KE02 内存映射 (继续)

系统 32 位地址范围	用途
0x1FFF_FC00–0x1FFF_FFFF	SRAM_L：下位 SRAM
0x2000_0000–0x2000_0BFF	SRAM_U：上位 SRAM
0x4000_0000–0x4007_FFFF	AIPS 外围设备
0x400F_F000–0x400F_FFFF	GPIO
0x400F_F000–0x400F_FFFF	通过 BME 访问的修饰 AIPS 外围设备空间
0xE000_0000–0xE00F_FFFF	专用外围设备
0xF000_2000–0xF000_2FFF	ROM 表
0xF000_3000–0xF000_3FFF	杂项控制模块 (MCM)
0xF800_0000–0xFFFF_FFFF	单周期（内核时钟）IOPORT
其他	保留

3.10 访问类型

S08 处理器支持对内存进行字节访问和位访问。 通过位可寻址区（如前所述的直接页面）支持位访问。 由于可从内存中加载的最大条目为 8 位，而目标也是 8 位，因此，所加载值的符号并不重要。 为了提高编译器的效率，该处理器针对 16 位加载/存储和比较运算执行 LDHX、STHX 和 CPHX 指令。

Cortex-M0+ 是 32 位处理器，所有内部寄存器均为 32 位。 支持 8 位字节、16 位半字和 32 位字内存传输。 在使用字节和半字时，程序员需要指定是将加载值处理为有符号还是无符号值。 在进行有符号加载时，加载值将进行符号扩展，以在目标寄存器中创建 32 位有符号值；进行无符号加载时，寄存器的上部将会清零。

Cortex-M0+ 还提供多重加载与存储指令，能够通过一条指令与相邻的内存块来回传输多个字。

3.11 位带

S08 位可寻址区支持独立位通过位处理指令对直接页面位置进行访问。 位设置/清除指令需耗时 5 个总线时钟周期。因此，对于位切换，则需耗时 10 个总线时钟周期。

KE04 和更新款的子系列支持位带。 上部 SRAM 区域 (SRAM_U) 是位带区。 位带可将内存别名区中的整个内存字映射到位带区中的单个位。 例如，写入一个别名字会设置或清除位带区中的相应位。 这样，便可以使用单个 LDR 指令从字对齐地址中直接访问位带区中的每个位，并通过 C/C++ 切换每个位，而无需执行读取-修改-写入指令序列。使用位带时，只需耗时两个内核时钟周期就能切换一个位，而使用读取-修改-写入序列时，则至少需要三个周期。 因此，使用 KE 系列中的位带可以明显地改善性能。

此外，所有 KE 系列都采用了 BME 技术来支持位带。 有关 BME 的详细信息，请参见位操作引擎（BME） 。

3.12 调试

下表提供了 S08 和 KE 系列中的调试模块在某些特性方面的比较。

表 4. S08P 与 KE 系列调试模块的比较

S08	KE 系列
使用单线 BDM 接口	使用串行线调试（SWD） 接口，该接口包括两条线：一条用于时钟，另一条用于数据 I/O。
支持三个硬件断点，或两个硬件断点加一个监测点	支持两个断点和两个监测点
支持 Loop1 捕获模式，可跟踪捕获 FIFO 中，用于代码跟踪的 8 字宽最新 COF 事件；另外还支持九种触发模式	无跟踪功能

有关调试的详细信息，请参见调试 。

3.13 电源管理

S08P 支持两种低功耗模式：等待和停止。

Cortex-M0+ 支持睡眠和深度睡眠模式。

- 在睡眠模式下，通常会将外部逻辑配置为停止处理器时钟，以最大程度地降低功耗。 NVIC 的电源和时钟保持为打开状态，以便在发生异常时能够退出睡眠模式。
- 在深度睡眠模式下，处理器可能会完全断电，通常只有外部唤醒中断控制器（WIC）处于活动状态。 在检测到任何未屏蔽外部中断时，WIC 将唤醒处理器。

可通过以下方法进入睡眠模式。

- 立即睡眠：等待中断（WFI）或等待事件（WFE）指令能使处理器立即进入睡眠模式。 检测到中断或调试事件时将退出该模式。
- 退出时睡眠：设置系统控制寄存器中的 SLEEPONEXIT 字段 (SCR[SLEEPONEXIT]) 可以在退出上一个未完成的 ISR 时，使处理器进入睡眠模式。 在此情况下，异常上下文将保留在堆栈中，以便能够立即处理唤醒处理器的异常。

此外，可通过设置 SCR[SLEEPDEEP] 进入深度睡眠模式。 如果设置了此字段，在进入睡眠模式时，处理器将向外部系统指示其可以进入更深度的睡眠。

实际上，KE 系列将睡眠模式实现为等待模式，将深度睡眠模式则实现为停止模式。

以下代码段显示了如何进入等待模式和停止模式：

```
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 嵌套向量中断控制器 (NVIC)

NVIC 是 ARM Cortex M 系列上的标配模块。 此模块与内核紧密集成，可以大大缩短进入与退出中断服务例程 (ISR) 的延迟时间。 需要耗时 15 个周期进入和退出 ISR，除非退出中断后再进入其他等待中的 ISR。 在此情况下，MCU 尾链以及退出和再进入操作则需要耗时 11 个周期。

NVIC 提供四个不同的中断优先级，可以使用这些优先级来控制为中断提供服务的顺序。 优先级为 0-3，其中 0 是最高优先级。 例如，在电机控制应用中，如果同时发生定时器中断和 UART，而运转电机的定时器中断比接收字符的 UART 中断更为紧急。 那么，必须将定时器优先级设置为高于 UART。这样，无需额外的软件代码，就能实现真正的硬件中断嵌套。

配备中断优先级控制器 (IPC) 的 S08 可以支持不同的中断优先级，但需要在中断服务例程中添加 prolog 和 epilog 软件代码。

以下代码段显示了如何在 KE 系列中使用给定的 IRQ 编号启用或禁用中断：

```
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);

}
}
```

有关 NVIC 和代码示例的详细信息，请参见以下网站上提供的《Kinetis L 外围模块快速参考指南》(KLQRUG)：
freescale.com

5 位操作引擎（BME）

位操作引擎 （BME） 为原子“读取-修改-写入”存储器对外围设备地址空间的操作提供硬件支持。 此架构功能也称为“修饰存储”，它定义了一种新的理念，不仅仅是将数据值读/写到编址存储器单元内，而且为存储器映射的外围设备的载入和存储操作提供了附加语义。 BME 修饰参考仅可用于处理器内核生成的系统总线事务，并针对基址为 0x4000_0000 到 0x4007_FFFF（AIPS 外围设备）的标准 512 KB 外围设备地址空间，部分针对从 0x400F_F000 开始的 GPIO 空间。 不能使用 BME 来访问其他内存区，包括 RAM 和闪存。

修饰语义嵌入在地址位 28–19 中，并在地址 0x4400_0000–0x5FFF_FFFF 上创建 448 MB 的空间。 这些位是从发送给外围总线控制器的实际地址中剥离而得，由 MBE 用来定义和控制其运算。 对于大多数 BME 命令而言，单个内核读或写总线周期将转换为原子读-修改-写序列，也就是说，一个不能分割的“先读后写”总线序列。

BME 支持修饰存储（逻辑与、逻辑或、位域插入）和修饰加载（加载-清除 1 位、加载-设置 1 位、无符号位域提取）。

外围地址位 31–29 始终是 010，也就是说，用于 BME 的外围内存空间从 0x40000000 开始。

数据大小由读取或写入运算指定，可为字节（8 位）、半字（16 位）或字（32 位）。

对于逻辑 AND、OR 和 XOR 存储运算，外围地址位 28–26 指定此表中提供的运算代码 (opcode)。

表 5. S08P 与 KE 的特性比较

运算	操作码	备注
与	001	在外围写入期间，将使用逻辑与运算将相应的外围数据与写入数据进行合并。
或	010	在外围写入期间，将使用逻辑或运算将相应的外围数据与写入数据进行合并。
异或	011	在外围写入期间，将使用逻辑异或运算将相应的外围数据与写入数据进行合并。

地址位 25–20 是“任意”位。 地址位 19–0 是外围空间的实际内存地址位。

对于逻辑位域插入 (BFI)、加载-清除/设置 1 位和位域提取 (UBFX) 存储运算，外围地址位 28–26 指定此表中提供的操作码。

表 6. S08P 与 KE 的功能比较

运算	操作码	备注
加载-清除	010	在外围读取期间，只将相应的位载入变量，并清除目标位。
加载-设置	011	在外围读取期间，只将相应的位载入变量，并设置目标位。
位域插入/提取	1xx	位 28 始终是 1，将会相应地使用位号位填充位 27–26。 在外围读取期间，会将供应位号中的相应位域提取到读取变量。 在外围写入期间，只将使用写入数据供应位号中的相应位域写入从位置供应位号开始的相应位域。 **注：** 位域插入/提取运算不能应用到从 0x400F_F000 开始的 GPIO 地址，因为地址位 19 由位宽度占用。 因此，无法使用此位域操作访问 GPIOx_PDOR/PSOR/PCOR/PTOR/PDIR/PDDR。 可以使用特殊的 C 语言方法，如以下代码段中所示。

对于无符号位域提取操作，地址位 18–0 是外围空间的实际内存地址位。

下图显示了不同运算的地址位表示形式。

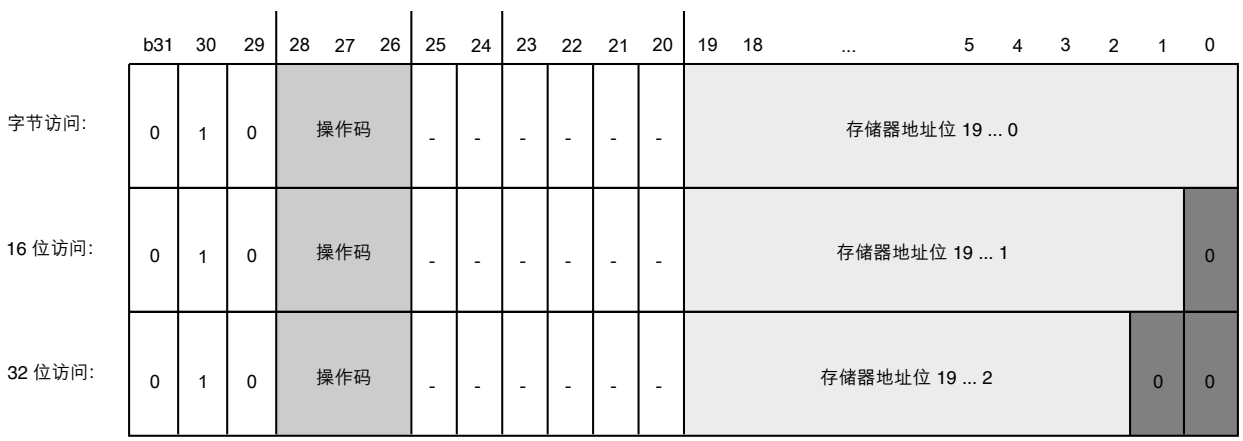

图 1. 与、或、异或存储的地址位

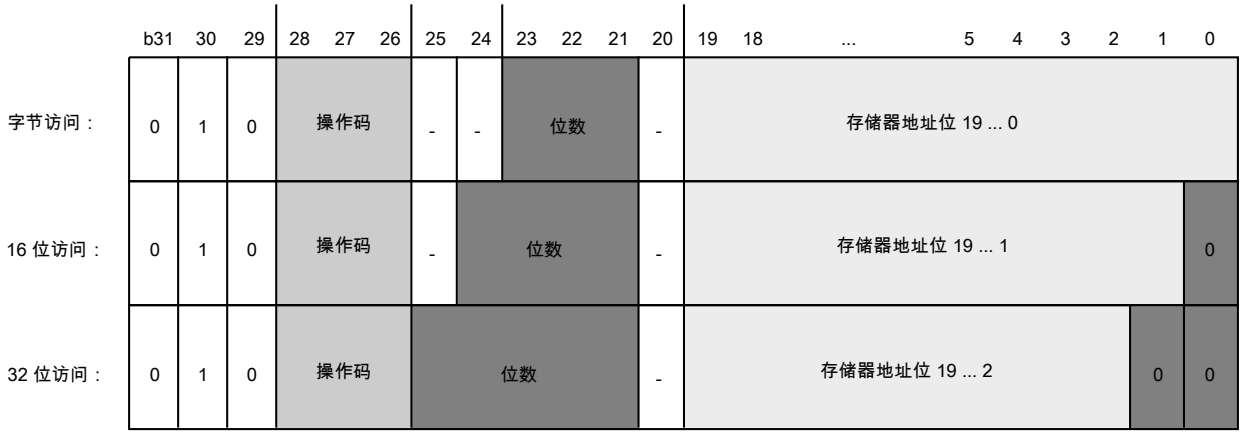

图 2. 载入-清除/设置 1 位运算的地址位

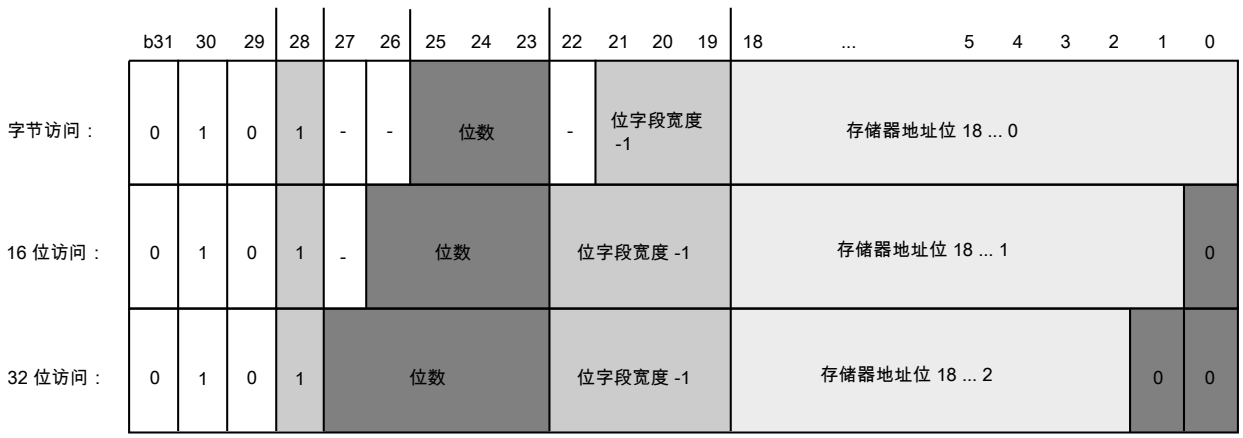

图 3. 位域插入/提取运算的地址位

以下代码段显示了如何使用 BME 运算：

```
// 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 时钟模块

内部时钟源（ICS）为 MCU 提供时钟源选项。 其中包含的锁频环（FLL）可用作时钟源，该时钟可由内部或外部参考时钟控制。 此模块可提供该 FLL 时钟或者内部或外部参考时钟作为 MCU 系统时钟的时钟源。 另外还提供了一些信号来控制低功耗振荡器（OSC）模块。 这些信号将配置和启用 OSC 模块，以生成供外围设备模块使用的外部晶振/谐振器时钟（OSCOUT）并用作 ICS 外部参考时钟源。 ICS 内部参考时钟可以是 OSC 提供的外部晶振/谐振器时钟（OSCOUT），也可以是其他外部时钟源。

KE 系列上的 ICS 和 OSC 结构与 S08P 十分相似。 KE02 子系列与 S08P 之间的主要差别在于，KE02 中的 FLL 倍增因子固定为 1024，即 S08P 系列的两倍。 因此，FLL 输出频率为参考时钟频率 x 1024。 可以通过 SIM_BUSDIV[BUSDIV] 将分频器（BDIV）后面的 FLL 输出时钟进一步对半分割，以便同时提供总线时钟和 Flash 时钟。

可通过时钟门控功能对所有外围时钟进行门控。 默认情况下，在复位后，除闪存和 SWD（串行线调试）以外的所有外围时钟都将阻断以节省电源。 这与 S08P 在复位后启用所有外围时钟不同。

下图给出了系统时钟示意图

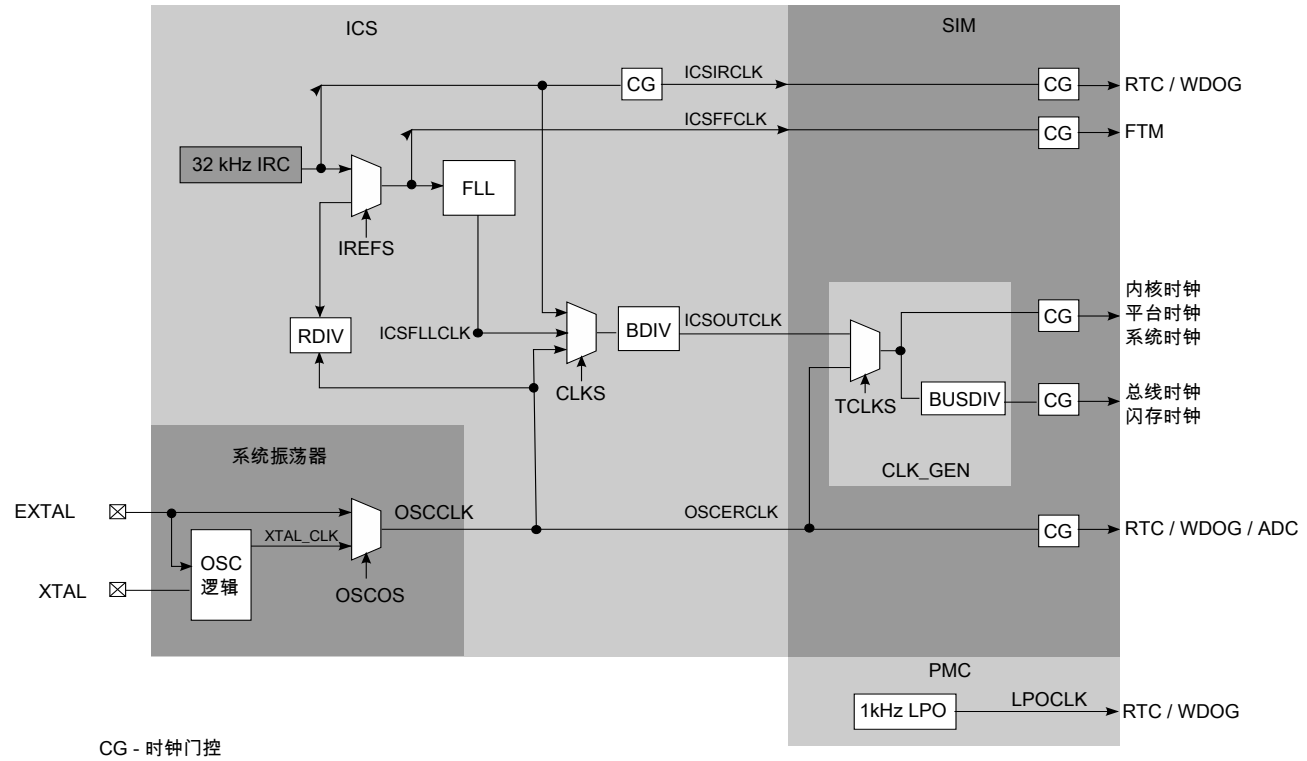

图 4. 系统时钟示意图

以下代码段显示了如何使用 8 MHz 外部晶振将 ICS 和 OSC 模块从 FEI 模式初始化为 FEE 模式：

```
/* 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*512*2 = 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;
```

以下代码段显示了如何门通/阻断 KBI0 时钟：

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

7 系统集成模块（SIM）

下表列出了 S08P、S08AC 和 KE 系列的 SIM 模块的差别。

表 7. S08 与 KE 产品系列的 SIM 模块比较

KE02	S08
包括 Kinetis 系列 ID、子系列 ID、修订版号和设备引脚 ID 的字段。	包括通用唯一标识符（UUID）
系统复位状态标志在 SRSID 寄存器的低半字中体现。 提供了三个新的状态标志：来自 SWD 的 MDM-AP 系统复位请求 (SIM_SRSID[MDMAP])、内核死锁 (SIM_SRSID[LOCKUP])，以及由于进入停止模式失败（原因通常是未通过 IIC 应答进入停止模式）而造成的停止模式应答错误复位 (SIM_SRSID[SACKERR])。 此外，Cortex-M0+ 内核可通过应用中断与复位控制寄存器生成软件复位。	SYS_SRS 寄存器包括只读状态标志，用于指示最近发生的复位的来源。 未提供直接软件复位机制。
时钟门由 SIM_SCGC 寄存器控制。	时钟门由 S08P 中的 SCG_C1 到 SCG_C4 寄存器控制。
引脚重新映射由 SIM_PINSEL 寄存器控制。	引脚重新映射由 S08P 中的 SYS_SOPT1 寄存器控制。
可以单独控制每个 Flextimer 引脚重新映射。	在 S08P 中，每个 Flextimer 模块的引脚只能以组为单位重组。
可以通过 SIM_BUSDIV 寄存器，针对总线时钟和闪存时钟将总线时钟进一步对半分频。	没有 SIM_BUSDIV 寄存器。 不能进一步对总线时钟和闪存时钟进行分频。
KE 系列中没有非法地址寄存器。	S08P 有非法地址寄存器。

8 电源管理控制器

KE 系列中的电源管理控制器（PMC）与 S08P 中的类似。 支持等待和停止模式，并具有通电复位（POR）功能。 可以启用或禁用低电压检测（LVD）和低电压警告中断。 当 CPU 执行停止指令时，如果 LVD 在停止模式下启用（同时设置了 PMC_SPMSC1[LVDE] 和 PMC_SPMSC1[LVDSE]），那么电压调节器在停止模式下将继续保持激活状态。 要获得最低功耗模式 (Stop3)，请通过清除这两个字段，在停止模式下禁用 LVD。

以下代码段显示了如何进入 Stop3 模式。

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

9 闪存、EEPROM 和闪存控制器模块

KE 系列上的闪存与 EEPROM 模块与 S08P 上的模块十分相似。 KE 系列的闪存带有缓存，而 EEPROM 则没有缓存。 这极大地改善了闪存存取的性能。 但是，所有 S08 设备都没有闪存缓存。 在 KE 系列中，可按字（32 位）、半字（16 位）或字节访问闪存；但只能按字节访问 EEPROM。

此外，KE 系列具有其他新的闪存操作功能：可同时读写。 允许在编程/擦写闪存的同时读取闪存。 对于不需要从 RAM 运行代码的应用，尤其是内存占用空间较小的设备，此功能十分有用。 可通过在 MCM_PLACR 中设置“启用停顿闪存控制器”字段来启用此功能，如以下代码中所示。

```
MCM_PLACR |= MCM_PLACR_ESFC_MASK;
```

闪存控制器是一个内存加速单元，有以下特点：

- 在总线主控与 32 位闪存之间提供接口。
- 提供可加快闪存数据传输速度的缓冲器和缓存。

闪存控制器拥有两个单独机制，用于加快总线主控与闪存之间的接口的速度。 32 位前瞻缓冲器可预取下一个 32 位闪存位置，而 4 路、4 组的程序闪存缓存可存储之前访问的闪存数据，以便快速访问。

不仅支持指令推测和缓存，而且还支持数据推测和缓存。

可以在 MCM_PLACR 寄存器中启用或禁用这些不同的功能。

可以使用以下方法来检查最近编程的数据是否正确：在读取闪存之前使缓存失效：

```
MCM_PLACR |= MCM_PLACR_CFCC_MASK;
```

这是为了确保缓存包含目标位置的更新值。

10 引脚分配变化

KE 系列的引脚分配与 S08P 兼容。 两者都有 8 个支持 20 mA 驱动能力的高驱动引脚。

KE 系列采用经过优化的新 I/O 结构，能够改善瞬态保护和 EMC 性能。 以下列表提供了引脚分配存在的变化（以 S08P 为例）。

- 将调试端口替换为两个 SWD 引脚：SWD_DIO 和 SWD_CLK 引脚。 KE 系列中没有 BKGD 功能。
- KE 系列使用了稳健可靠的解决方案，可以应对家用电器面对的苛刻使用环境。 EFT 可以支持 4.4 kV 或更高电压，电源 ESD 直接接触放电电压为 20 kV 或更高。 强化的 IO 焊盘设计可以承受系统 EMC 和 ESD 干扰，具体特性包括：
  - 在 GPIO/RST/IRQ 上进行滤波以实现噪声抑制
  - 驱动强度控制符合 EMI/EMC 规范
  - 所有 GPIO 默认使用高阻抗（包括非粘合引脚）

11 安全功能增强

11.1 看门狗

看门狗 (WDOG) 定时器模块是可供系统使用的独立定时器。 如果在指定的时间段内未使用特定数据写入序列将其更新/刷新，则复位 MCU。 可以将其用作一种安全元件，以确保软件按计划运行，同时确保 CPU 不会陷于无限循环或执行意外代码。 看门狗是根据 IEC60730 安全标准设计的。

此模块与 S08P 系列中的对应模块类似。 以下是有关使用此模块的一些准则：

- WDOG 寄存器映射以类似于 S08P 的 big-endian 模式进行组织，但是，Cortex-M0+ 内核使用 little-endian 访问模式。 因此，在使用 16 位地址模式访问 WDOG_CNT、WDOG_TOVAL 和 WDOG_WIN 等 WDOG 寄存器时，始终要记住值的高位字节在高位字节地址中，值的低位字节在低位字节地址中。
  例如，以下代码段会将 0xA6 写入 WDOG_CNTL（高位字节地址），将 0x02 写入 WDOG_CNTH（低位字节地址）。

```
WDOG_CNT = 0xA602;
```

- 刷新序列（即向 WDOG 馈送数据）的计数器必须在 16 个总线时钟周期内完成，否则将出现复位。
- 解锁序列写入必须在 16 个总线时钟周期内完成，以便对一次写入配置位进行更新，否则将出现复位。
- 软件必须在解锁后、WDOG 关闭解锁窗口前的 128 个总线时钟周期内完成更新，否则将出现复位。

11.2 CRC

循环冗余检查（CRC）模块为错误检测生成 16/32 位 CRC 代码。 它提供了可编程的多项式、WAS 以及实施 16 位或 32 位 CRC 标准所需的其他参数。 可以一次性计算 32 位数据的 16/32 位代码。 此模块与 S08P 系列中的模块类似，不过仍存在以下差别：

- 支持输入数据或输出数据的按位/按字节移位（CRC 作用的结果）。 S08P 仅支持按位移位。
- 支持对寄存器进行 32 位访问。 S08P 仅支持对寄存器进行 8 位访问。

假如 KE 系列上的 CRC_DATA 寄存器是一个 32 位寄存器，那么，它相当于在 S08P 上串联了 CRC_D0 到 CRC_D3 这四个 8 位寄存器。

12 端口控制和 GPIO

端口控制模块控制数字干扰滤波器、上拉和高驱动。 在 KE 系列中，有四个 32 位寄存器：PORT_IOFLT、PORT_PUEL/H 和 PORT_HDRVE。 这些寄存器与 S08P 中的对应寄存器在功能上类似，唯一的差别在于，KE 系列中的这些寄存器可以在 32 位模式下访问。 这极大地改善了针对多个 I/O 的控制速度。

注

- PORT_PUEL/H 是对应于 S08P 中 PORT_PTxPE 组合寄存器的端口上拉启用寄存器。 因此，需要相应地更改用于访问 S08P 中任何 PORT_PTxPE 的代码。 KE 系列中没有端口输入和输出启用寄存器（PORT_PTxIE、PORT_PTxOE）。 KE 系列改为使用 GPIOx_PDDR 来控制数据方向，使用 GPIOx_PIDR 来启用或禁用 GPIO 输入。
- 在复位后，所有 GPIO 引脚将默认为外围输入，在此情况下，引脚数据输入寄存器中的相应位显示为 0。 要正确读取引脚状态，必须清除端口输入禁用寄存器 (PIDR) 中的相应位。

KE 系列中的 GPIO 模块控制方向以及 I/O 引脚的输入和输出数据，这一点也与 S08P 系列类似。 但是，KE 系列中的 GPIO 寄存器可以在 32 位模式下访问。 此外，还可以通过单周期（内核时钟）IOPORT 接口访问它的 GPIO。 这称为快速 GPIO 或 FGPIO。 此 FGPIO 的内存映射从 0xF800_0000 开始。

通过 IOPORT 接口（FGPIO 内存空间）执行的访问与任何指令的获取保持同步，因此可在一个内核时钟周期内完成。参见下图。

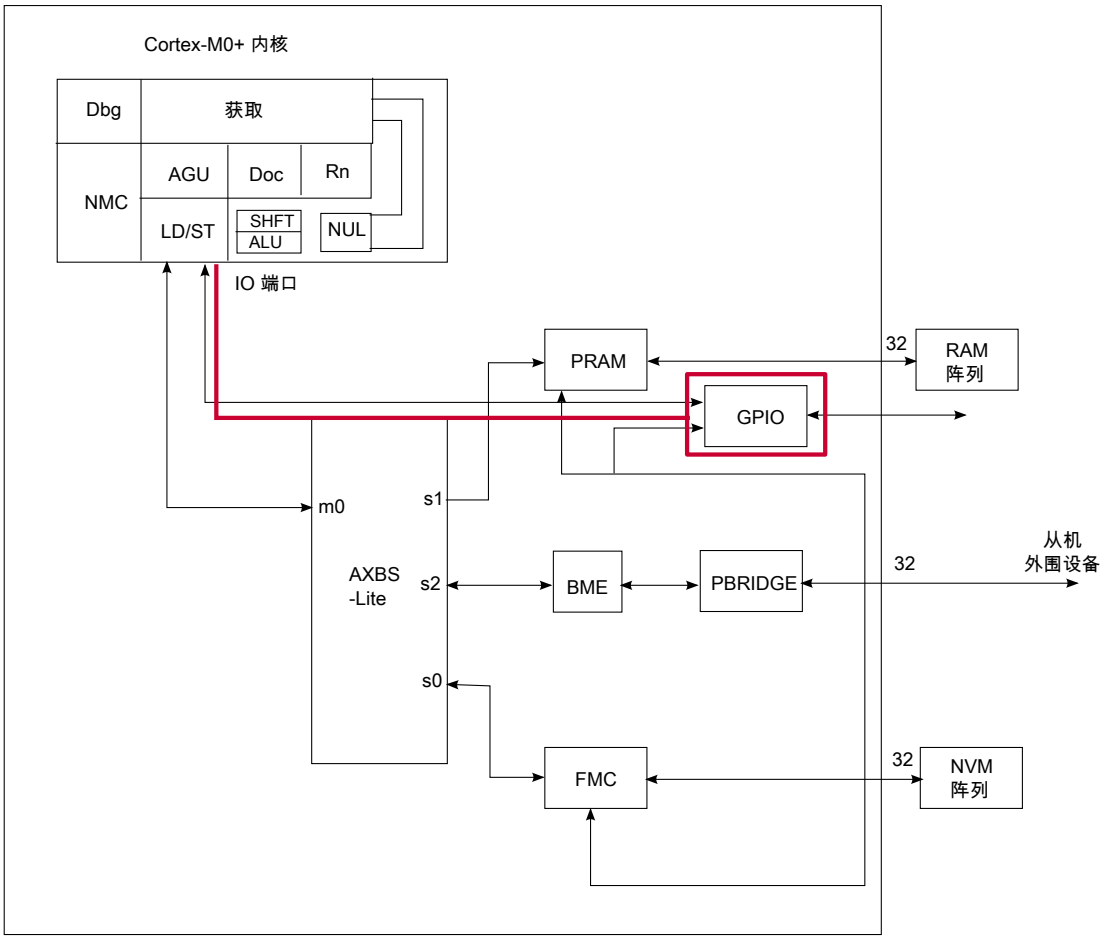

图 5. Cortex-M0+ 内核平台框图

以下是 GPIO 主要特性的列表：

- 在所有数字引脚多路复用模式下均显示引脚输入数据寄存器
- 引脚输出数据寄存器，具有对应的设置/清除/切换寄存器
- 引脚数据方向寄存器
- 可通过 IOPORT 对 GPIO 寄存器执行零等待状态访问

注

复位后，所有 GPIO 引脚将默认为外围输入，在此情况下，引脚数据输入寄存器中的相应位显示为 0。 要正确读取引脚状态, 必须清除端口输入禁用寄存器 (PIDR) 中的相应位。

以下代码段演示了如何使用端口控制和 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 定时器模块

13.1 RTC

实时计数器 (RTC) 由一个 16 位计数器、一个 16 位比较器、多个二进制和十进制预分频器因子、三个时钟源、一个可编程周期中断和一个可编程外部切换脉冲输出组成。 S08P 与 KE 系列之间的差别在于寄存器访问宽度。

S08P 中的两个 8 位状态与控制寄存器（RTC_SC1 和 RTC_SC2），级联后则相当于 KE 系列中的一个 32 位寄存器 RTC_SC。 S08P 中的两个 8 位 RTC 模数寄存器（RTC_MODH、RTC_MODL），级联后则相当于一个 32 位 RTC_MOD 寄存器。

RTC 计数器寄存器的情况与此类似。因此，S08P 中用于访问 RTC_SC1、RTC_SC2、RTC_MODH、RTC_MODL、RTC_CNTH 和 RTC_CNTL 的代码需相应地更改为访问 RTC_SC、RTC_MOD 和 RTC_CNT。

13.2 FlexTimer

FlexTimer（FTM）以简单定时器，即 S08 设备上使用的 TPM 模块为基础而设计，并将其功能扩展，从而可满足电动机控制、数字灯光解决方案和电源转换的需求。 下面将介绍对定时器模块所做的多个关键性增强部分。

- 带符号的递增计数器
- 死区时间插入硬件
- 故障控制输入
- 增强的触发功能
- 初始化和极性控制

以下列表介绍了 S08 设备中 FlexTimer 模块的主要特性。

- FTM 源时钟可选。
  - 源时钟包括系统时钟、固定频率时钟或外部时钟。
  - 固定频率时钟是附加的时钟输入，允许选择除系统时钟以外的片上时钟源。
  - 选择外部时钟可将 FTM 时钟连接到芯片级输入引脚，因此，可将 FTM 计数器与片下时钟源同步
- 1、2、4、8、16、32、64 或 128 分频的预分频器
- 16 位计数器
  - 可以是自由运行计数器，也可以是带有初始值和最终值的计数器。
  - 计数可以是递增或递减计数。
- 可为输入捕获、输出比较或边沿对齐的 PWM 模式配置每个通道。
- 在输入捕获模式下：
  - 捕获可以在上升沿和/或下降沿发生
  - 可为某些通道选择输入滤波器

- 在输出比较模式下，可以在匹配时设置、清除或切换输出信号。
- 可为中心对齐 PWM 模式配置所有通道。
- 每个通道对都可以组合，以生成一个 PWM 信号，该信号独立于 PWM 信号两个边沿的控制。
- FTM 通道可以作为提供相同输出的对、提供互补输出的对，或者提供独立输出的独立通道进行操作。
- 针对每个互补对提供死区时间插入。
- 生成匹配触发器
- PWM 输出的软件控制
- 提供多达四个故障输入，从而结合可编程极性进行全局故障控制

- 每个通道的极性可配置。
- 为每个通道生成一个中断
- 计数器溢出时生成中断
- 检测到故障状态时生成中断
- 同步加载写入缓冲 FTM 寄存器
- 为关键寄存器提供写保护
- 与 TPM 向下兼容
- 陷入 0 和 1 状态时测试输入捕获
- 可进行双边沿捕获以便进行脉宽和期限测量
- 为同步不同的 FlexTimer 模块提供全局时基

与 S08P 相比，KE 系列中的 FlexTimer 模块具有下列新的增强特性。

- 反转控制/通道交换
- 故障输入极性控制
- 中间负载
- 软件输出控制
- 调试模式功能
- TOF 频率/周期 TOF

可通过清除或设置 FTMx_FLTPOL[FLTnPOL]，将故障输入极性配置为高电平有效或低电平有效。 FTM 可设置为在进入调试模式后仍起作用。 为此，可以设置 FTMx_CONF[BDMMODE]。

反转功能可以交换通道（n) 和通道（n+1) 输出之间的信号。 当条件为（FTMEN = 1)、(QUADEN = 0)、(DECAPEN = 0)、(COMBINE = 1)、(COMP = 1)、(CPWMS = 0) 和（INVm = 1)（其中的 m 代表通道对）时，将选择反转操作。

因其中间负载特性，PWMLOAD 寄存器允许使用寄存器缓冲器在定义加载点的内容更新 MOD、CNTIN 和 C(n)V 寄存器。 在此情况下，则不需要使用 PWM 同步。 支持以下加载点/条件。

- 当 FTM 计数器从 MOD 值变化为 CNTIN 时
- 语句 When CHjSEL = 1 则是进行通道（j) 匹配（FTM 计数器 = C(j)V)。

配置加载点后，必须通过设置 FTMx_PWMLOAD[LDOK] 启用这些加载点。 必须设置此字段，以便在后续加载点发生加载。

以下代码段显示了如何使用通道 2 匹配启用中间加载功能：

```
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 */
```

在生成 PWM 时，软件输出控制将根据特定时间的软件定义值强制设定通道输出值。 当条件为（FTMEN = 1)、(QUADEN = 0)、(DECAPEN = 0)、(COMBINE = 1)、(CPWMS = 0) 和（CHnOC = 1)，将选择软件输出控制。 CHnOC 字段可针对特定的通道输出启用软件输出控制，CHnOCV 可选择强制到此通道输出的值。

可通过 FTMx_CONF[NUMTOF] 设置 TOF 频率。 将会针对第一次发生的计数器溢出设置 TOF 标志，而不会针对后续的 N 次溢出设置此标志（其中的 N = NUMTOF 且不为 0)。 如果只需要在若干个 PWM 周期后更改 PWM 事件，则此功能十分有用。 这会大大减少 CPU 过载现象。

此外，用户必须了解下列其他两个差异。

- 在 KE 系列中，几乎所有的 FTM 通道都可单独重新映射到不同的引脚；而在 S08P 中，只有 FTM2 通道才能在组中进行替换引脚的重新分配。 可以使用引脚选择寄存器（SIM_PINSEL) 实现这种重新映射。
- 可以使用 32 位访问寄存器以改善性能。

13.3 PIT

KE 系列周期中断定时器 (PIT) 是一系列可用于引发中断和触发器的定时器/通道。 这是一个 32 位模块，每个定时器是倒计数计时器，该定时器的初始值则在 PIT_LDVALn 寄存器中指定。 每当定时器达到 0 时，将生成触发脉冲并设置中断标志。 如果已通过设置 PIT_TCTRLn[TIE] 启用了相应的功能，则可以生成新的中断，不过只会在清除上一个中断后才生成。 通过使用 PIT_TCTRLn[TEN] 先禁用，而后再启用定时器，即可以重新开始计数器周期。 可通过 PIT_CVALn 寄存器读取定时器的当前计数器值。

可以在不重新启动定时器的情况下更改计数器周期，方法是使用新的加载值写入 PIT_LDVALn 寄存器，此值将在发生下一个触发事件后生效。

可以链接 PIT 中的定时器，就如同这些定时器是一个整体一样。

注

在执行其他任何设置之前，必须通过清除 PIT_MCR[MDIS] 启用该模块。

以下代码段显示了如何设置定时器链（定时器 1 和 0），以便在生成 PIT channel1 中断之前先计数 10000 个周期：

```
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 调试

KE 系列实施了 2 引脚串行线调试 (SWD) 端口：SWD_CLK 和 SWD_DIO。 SWD 基于 ARM CoreSight 体系结构。 基本调试功能包括处理器停止、单步、处理器内核寄存器访问、复位和硬故障向量捕获、软件断点和全面系统内存访问。有关此体系结构的详细信息，请访问： arm.com。

通过 ARM 调试访问端口（DAP），调试器可访问 DAP 总线上被用作寄存器的状态和控制单元。 这些寄存器提供了低功耗模式恢复和典型运行控制场景的其他控制和状态。 状态寄存器位还为调试器提供了用于获取内核的最新状态，而同时无需启动交叉开关中的总线事务的方式，从而避免在调试会话期间过多的打扰。 重要的是要注意，这些 DAP 控制和状态寄存器在系统内存映射内未进行内存映射，而且只可使用 SWD 通过调试访问端口进行访问。

KE 系列调试模块支持两个断点和两个监测点。 但其并不提供微跟踪缓冲器（MTB），因此，用户无法使用 SWD 模块获得迹函数。

注

在复位期间，请不要尝试读取 MDM-AP 状态寄存器的位 2 (安全状态)。 仅当设备处于非复位状态时，此位才有效。

15 ADC 模块

12 位模数转换器 (ADC) 是一种逐次逼近 ADC，其速度比 Σ-Δ 转换器要快，同时还能保持同等的高精度。

以下是 ADC 模块特性的列表。

- 采用 8 位、10 位或 12 位分辨率的线性逐次逼近算法
- 多达 16 个外部模拟输入、外部引脚输入以及 5 个内部模拟输入，包括内部带隙、温度传感器和参考电压
- 8 位、10 位或 12 位靠右对齐、无符号输出格式
- 单一或连续转换（单一转换后自动返回到空闲状态）
- 支持高达 8 个结果 FIFO，可选择 FIFO 深度
- 可配置采样时间和转换速度/功率
- 转换完成标志和中断
- 可从多达 4 个源中选择输入时钟
- 在等待或 stop3 模式下运行，即可实现低噪声运行
- 异步时钟源，即可实现低噪声运行
- 可选异步硬件转换触发器
- 自动与可编程值进行比较（小于、大于或等于），根据结果生成中断

KE 系列中的 ADC 模块与 S08P 中的类似，唯一的差别在于，前者的寄存器可以在 32 位模式下访问。 可以在一个周期内读取 ADC 12 位结果。 因此，它的速度比 S08P 要快得多。

以下代码段显示了如何初始化该 ADC，以及读取 ADC 处理器中的 ADC 结果寄存器：

```
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 参考

- 《KLQRUG：Kinetis L 外围模块快速参考》，可从以下网站获得： **freescale.com**
- 《AN4347：将 S08AC 和 S08FL 系列中的应用移植到 S08PT 系列》，可从以下网站获得： **freescale.com**
- 《AN4560：使用 Kinetis FlexTimer 进行 PWM 同步》，可从以下网站获得： **freescale.com**
- 《Cortex -M0 设备常规用户指南》，可从以下网站获得： **arm.com**

17 术语表

ACMP	模拟比较器
BME	位操作引擎
CRC	循环冗余检查
DMA	直接内存访问
FMC	闪存控制器
IRC	内部参考时钟
PWT	脉冲宽度定时器
RTC	实时计数器
SWD	串行线调试

18 修订历史记录

修订版号	日期	重要改动
0	07/2013	首版

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

本文档中的信息仅供系统和软件实施方 使用飞思卡尔产品。 未包含基于本文档信息设计或加工 任何集成电路的任何明确或隐含的版权许可授权。飞思卡尔保留对此处任何产品进行更改的权利， 如有更改，恕不另行通知。

飞思卡尔对其产品在任何特定用途方面的适用性不做任何担保、声明或保证，也不承担因为应用 或使用产品或电路所产生的任何责任， 明确拒绝承担包括但不局限于因果性或附带损害在内的所有责任。 飞思卡尔 数据表和/或规格中所提供的“典型”参数在不同应用中可能 并且确实不同，实际性能会随时间而有所变化。 所有工作参数， 包括“典型值”在内，在每个 客户应用中必须经由客户的技术专家进行验证。 飞思卡尔未转让与其专利权及其他 权利相关的许可。 飞思卡尔销售产品时 遵循以下网址中包含的标准销售条款和条件： freescale.com/SalesTermsandConditions 。

Freescale, Freescale logo, and Kinetis are trademarks of Freescale Semiconductor, Inc., Reg. U.S. Pat. & Tm. Off. All other product or service names are the property of their respective owners.

© 2013 飞思卡尔半导体有限公司

Document Number AN4757
Revision 0, 07/2013