Freescale Semiconductor
Application Note

Document Number:AN4779
Rev 0, August 1, 2013

Kinetis E 系列的 EMC Design 提示

通过: **Dennis Lui 和 T.C. Lun**

内容

1 简介

充分考虑电磁兼容（EMC） 设计是确保系统在设计上可靠，能够在苛刻环境中毫无差错地运行，并且不会造成干扰的关键因素之一。 本应用说明提供了有关如何根据 EMC 要求在应用中使用 Kinetis E 系列 MCU 的设计提示。

其中介绍了有关硬件设计、印刷电路板（PCB） 布局和软件设置的各种技巧，以帮助客户在设计阶段初期对其产品应用 EMC 增强功能。 一般而言，末期发生的 EMC 问题更加复杂，需要花费更大的代价和更多的时间才能修复。 电路和 PCB 布局改造存在许多约束：

- 当所有组件或模块都安装在系统内部时。
- 由于这些整改需要的额外元件产生的更高成本的结构
- 解决方案可能要求在机械方面做出重大的设计变更，从而影响到项目的进度。

2 系统概述

我们以一个采用 Kinetis E 系列 MCU 的典型应用作为示例，来说明如何在实际开发中运用 EMC design 提示。

可从 freescale.com 下载的应用说明《AN4476：MC9S08PT60 的 EMC Design 注意事项》提供了有关 EMC 基本概念和理论的详细介绍，可帮助应用开发人员理解每条 EMC design 提示附带的理由。 请阅读该应用说明以及 freescale.com 上提供的其他 Kinetis E 系列文档，例如

《Kinetis E 参考手册》和《Kinetis E 子系列数据手册》，以了解设备特征、寄存器配置和固件编码的详细信息。 示例代码段是使用 IAR Embedded Workbench 6.40 编写的。

下面给出了一个典型的应用框图。

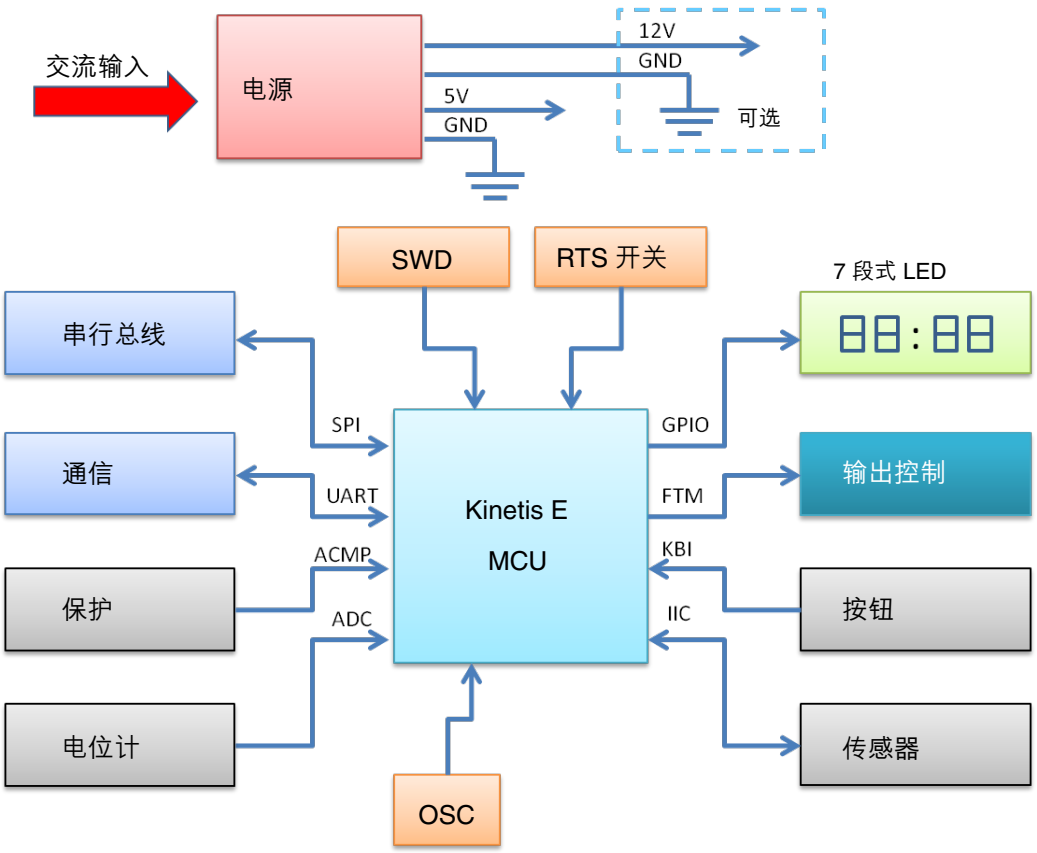

图 1. 典型应用框图

在电源模块中，交流线电压已转成低压并调节成 5 V。 整个系统，包括 MCU、GPIO、显示和模拟外设的主电压为 5 V。 在某些应用中，还需要为高功率控制电路提供 12 V 电压。 例如，大多数功率继电器开关由 12 V 驱动电路控制，但高电流阶段的电压则直接由交流线输入供应。

Kinetis E MCU 的应用包括：所有用户输入接口的信号检测，包括传统按钮，通过标准的 UART 串口与主机控制器进行的通信。通过 IIC 总线上的传感器设备或 ADC 引脚上的直接电压输入进行系统监控，使用特定顺序的 GPIO 引脚进行电源控制，以实现系统保护，通过模拟比较器输入进行硬件故障检测。

3 EMC design 提示

下列章节提供了 EMC design 提示，这些内容分硬件和软件不同的角度论及。 硬件或软件工程师可以根据其需求选择相关的章节，并在其设计中直接运用这些提示。

硬件设计提示涵盖板级考虑因素，这包括 PCB 布局设计技巧，以及不同类型的 I/O 端口的注意事项。 主要目标是利用 EMC 方面的知识来防止出现影响系统运行和稳定性的任何内部或外部噪声；最大程度地降低噪声源对敏感器件（例如 MCU）造成的耦合影响，减少干扰源的噪声量，并提高受体的抗扰度。

另一种方法是采用防御性软件设计思路，以解决在噪声环境中由软件对于错误触发事件不当的处理引起的 EMC 问题。 软件必须能够识别特定的事件是噪声源触发的错误警报，还是正常的从动事件。 然后它必须做出一个采取相应措施的正确决策。 例如，如果请求的操作具有任何不确定性，则 MCU 不能启动高功率控制阶段。

4 硬件设计

噪声环境中 MCU 应用的硬件注意事项包括 PCB 布局设计，以及外设接口的外部组件连接。

从板级别而言，PCB 布局是关乎从内部或外部噪声源是否会耦合噪声的关键因素。 布局中的迹线充当耦合路径，而迹线的几何因素（长度、宽度、形状和位置）会严重影响耦合效果。 在系统中妥当地排布电路板和电缆有助于将噪声源与系统分离，并提高系统的抗扰度。 下列章节介绍了用于实现可靠硬件设计的推荐技巧。

4.1 单层 PCB

高成本多层 PCB 设计可让用户更灵活地铺排组件，进行信号路径布线、电源退耦和参考接地。

但是，PCB 的尺寸和形状受到机械规格的限制，在大多数场合下，这已成为 PCB 设计的主要障碍。 出于成本方面的考虑，单层双面 PCB 板是大多数家用电器应用的不错选择，但是，对于引脚数目众多的设备，更难设计出这样的 PCB。 下列章节介绍了如何在充分考虑 EMC 要求的条件下实现合理的 PCB 布局。

4.2 器件布局

器件布局必须符合列出的产品机械规格约束。

供参考的一般准则如下：

- 将螺孔和安装点的所有位置标记为禁用区。
- 根据固定位置的要求器件布局所有用户接口组件 （例如：显示面板、控制按钮和连接器）。
- 将高功率电路与低功率和噪声敏感电路分隔开来。
- 请将器件关联成组，并尽量将各组按照符合相关的信号流流向的逻辑顺序排列。
- 标识需要布局在 MCU 附近的所有关键组件，以及从 MCU 输入端口连接到电源或接地的外部组件 （例如：电源退耦电容器和输入信号滤波组件）。
- 最小化电源环路和接地环路构成的面积。
- 减少电源和 MCU 接地之间的共模阻抗。

可能需要付出相当大的精力才能完成一个有可能符合所有约束条件的可接受版本。

4.3 电源和接地布线

电源和接地平面的 PCB 布局对于板级 EMC 性能极其重要，尤其是在使用 5 V 和 12 V 的多电源系统中。

可以使用 PCB 布局技术将接地平面划分成两个部分，如下图所示。 其中一个部分定义为 12 V 电路的返回路径，另一个部分定义为 MCU 和其他关键组件的 5 V 返回路径。 来自 12 V 接地的噪声将不会通过接地迹线与 5 V 接地耦合。

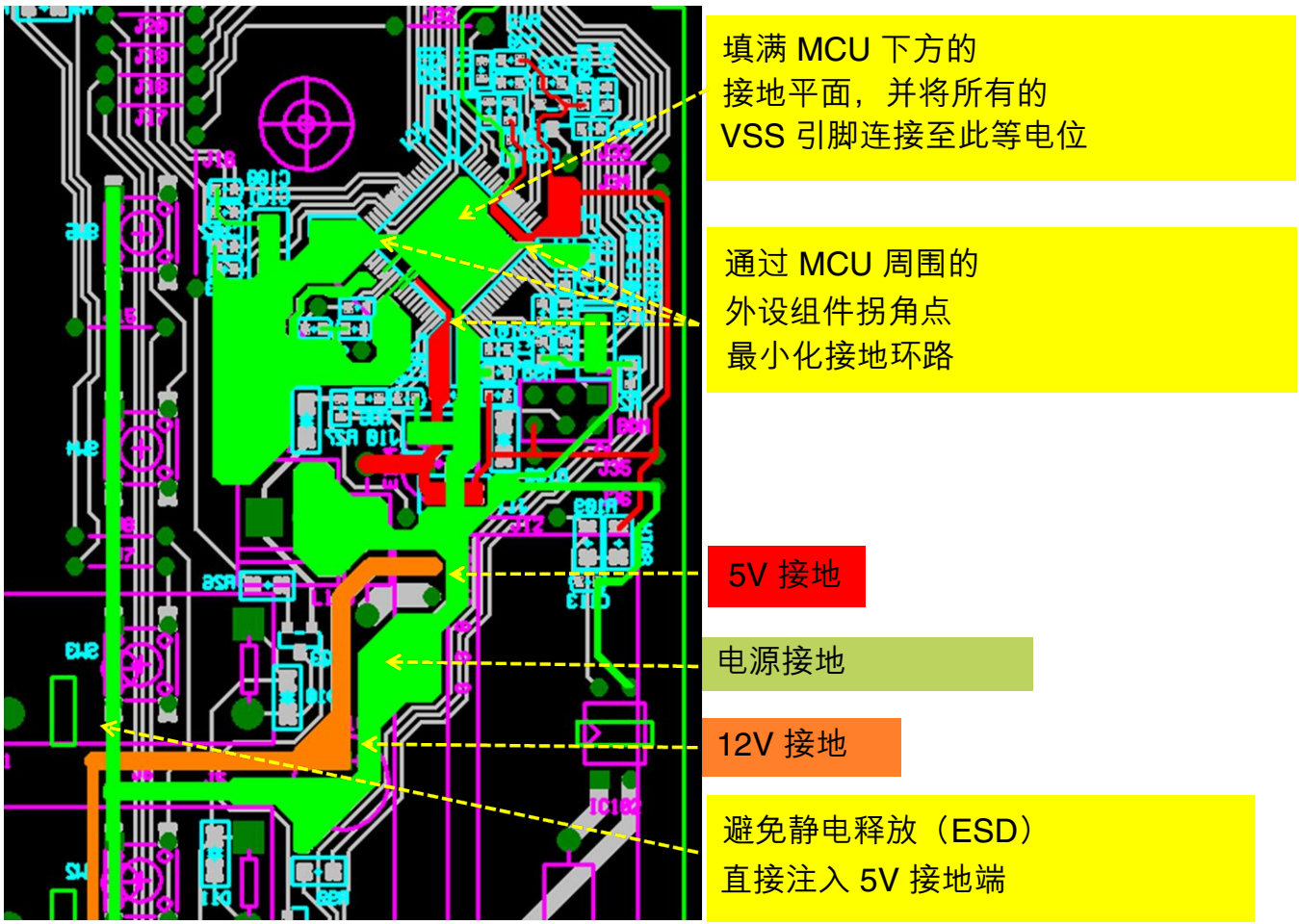

图 2. 电源和接地布线

在某些应用场合中，12 V 地就是用作 5 V 供电的那些容易在空气放电测试中的遭受 ESD 组件的返回路径，在空气放电测试中，这很容易造成 ESD（静电释放）损坏。 将 12 V 接地连接到这些 5 V 组件可以防止 ESD 放电能量直接耦合到 5 V 接地。 如果高能量流过 MCU 接地，有可能会强制 MCU 复位、停止甚至发生损坏。

PCB 布局中的 MCU 接地连接方法是保障 EMC 性能的关键因素。 该方法在 MCU 下方填充一个接地平面，并将所有 VSS 引脚连接在一起，这种做法充分考虑到了 EMC 方面的要求。 该方法确保所有 MCU VSS 引脚保持在相同的电位，同时可以最小化从 MCU 到高频噪声旁路电容器的电流返回路径上的电感。 对于 LQFP 封装，可以进一步将 MCU 接地平面扩展到封装拐角点，以实现较短的接地路径，并最大程度地减小围绕 MCU 的外设组件的环路面积。

4.4 退耦和旁路

用户有必要更好地理解退耦与旁路的概念，以免发生 EMC 实施不当的问题：

- 退耦用于隔离公用线上电路之间的噪声。 电源迹线就是从稳压器到 MCU 的公用线之一。
- 旁路是通过一个旁路电容来分流一个阻抗路径，以减少在该路径中的高频电流波动。

为 MCU 添加退耦和旁路电容器的效果在很大程度上取决于连接位置和顺序，如下图所示。 PCB 布局中 MCU 电源引脚（VDD 和 VSS）的准则如下：

- 将电源引出的电源和接地迹线依次连接到退耦电容器、旁路电容器以及 MCU 的 VDD 和 VSS 引脚。
- 并行排布电源和接地迹线以最小化环路面积。
- 将旁路电容器排布在尽量靠近每个 VDD-VSS 对的位置。

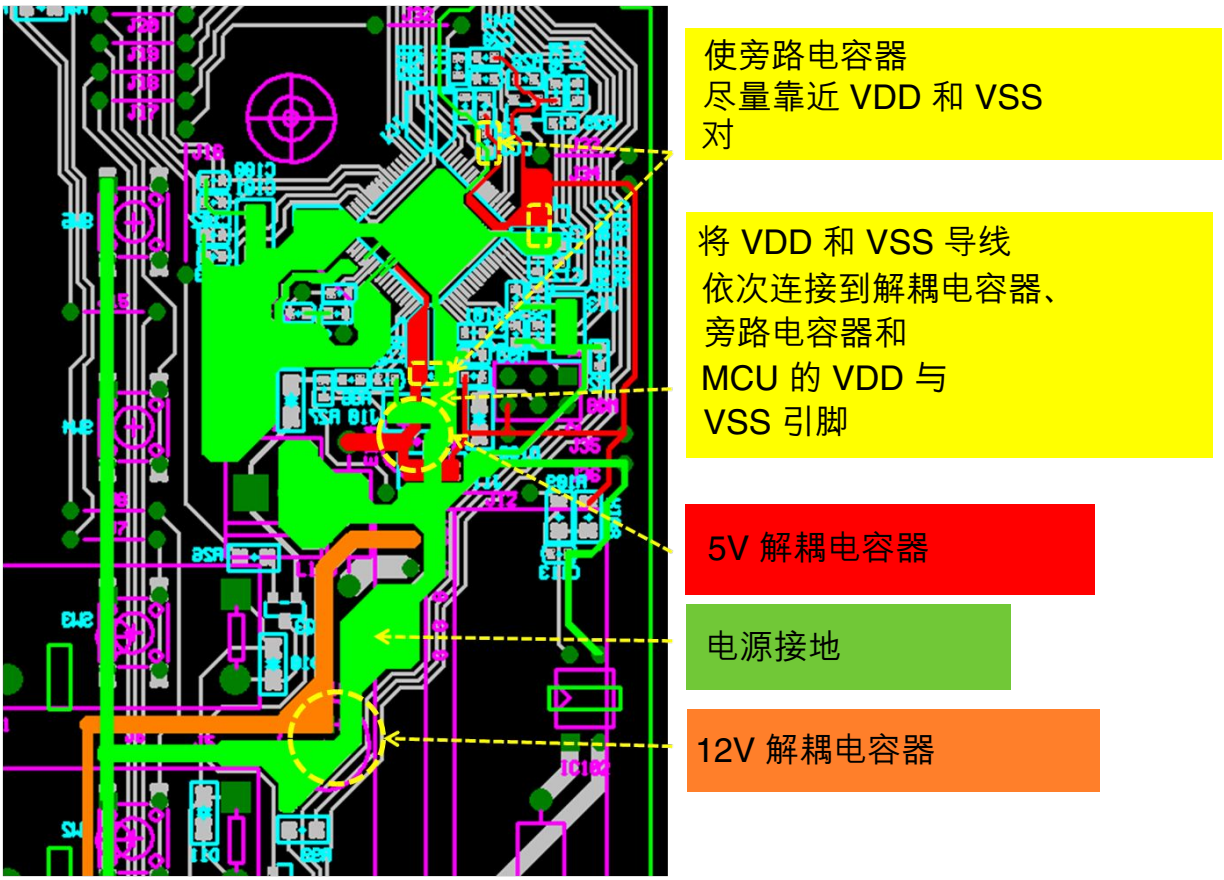

图 3. 退耦和旁路

4.5 晶体振荡器电路

连接到 MCU EXTAL 和 XTAL 引脚的晶体振荡器组件对外部噪声十分敏感。

PCB 以保护环的形式排布接地迹线，加上连接到 EXTAL 和 XTAL 引脚的迹线，可以最大程度地减少接入晶振电路的噪声。 下图给出了一个示例，一般准则如下：

- 不要在晶振电路附近或横跨电路底侧排布任何信号路径（接地线迹除外）。
- 将晶振电路组件（晶振、反馈电阻器和负载电容器）尽量排布在靠近 EXTAL 和 XTAL 引脚的位置。
- 选择内部振荡器作为时钟源以提高 EMC 性能。
- 如果使用双层或多层 PCB，请将负载电容器的接地直接连至接地平面。
- 选择最小总线频率以满足系统要求。
- 在振荡器电路上采用最小迹线长度。
- 使用参数值适当的反馈电阻器和负载电容器。

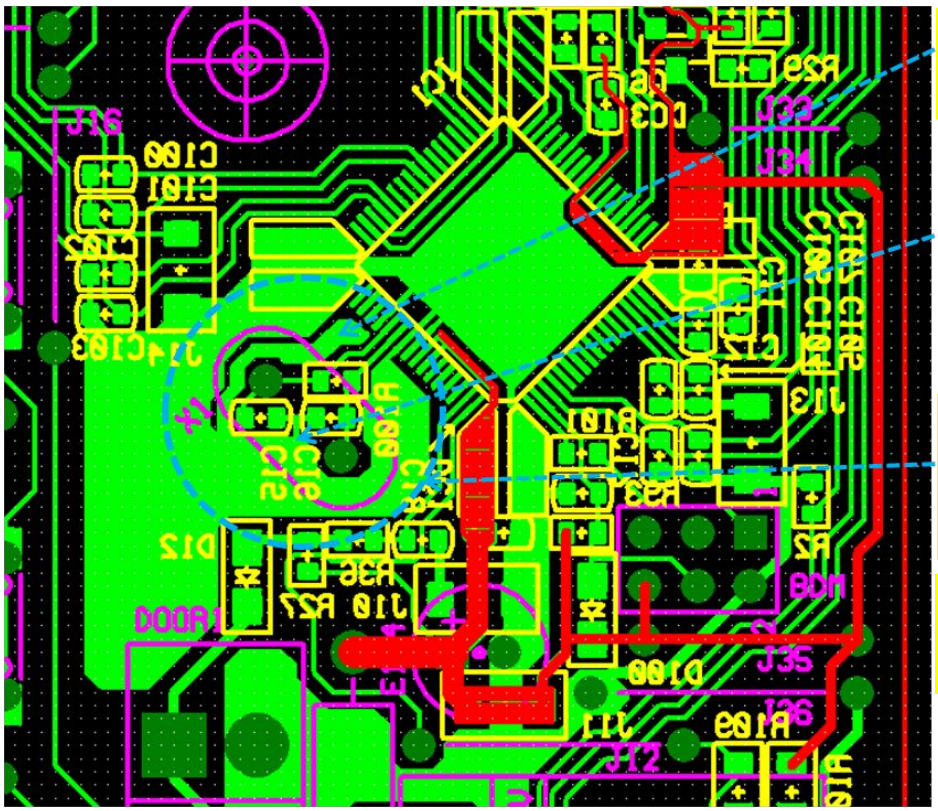

图 4. 晶体振荡器电路

4.6 间距和隔离

如果在同一块电路板上，交流高功率电路与低功率电路并列铺排（如下图所示），则隔离每个电路模块就很有必要。

在某些情况下，如果电路板的尺寸有限，则您可能需要添加物理插槽以提供更好的隔离。 同样，出于 ESD（静电释放）方面的考虑，需要在 PCB 迹线与安装螺孔或板边沿之间保留足够的间距。

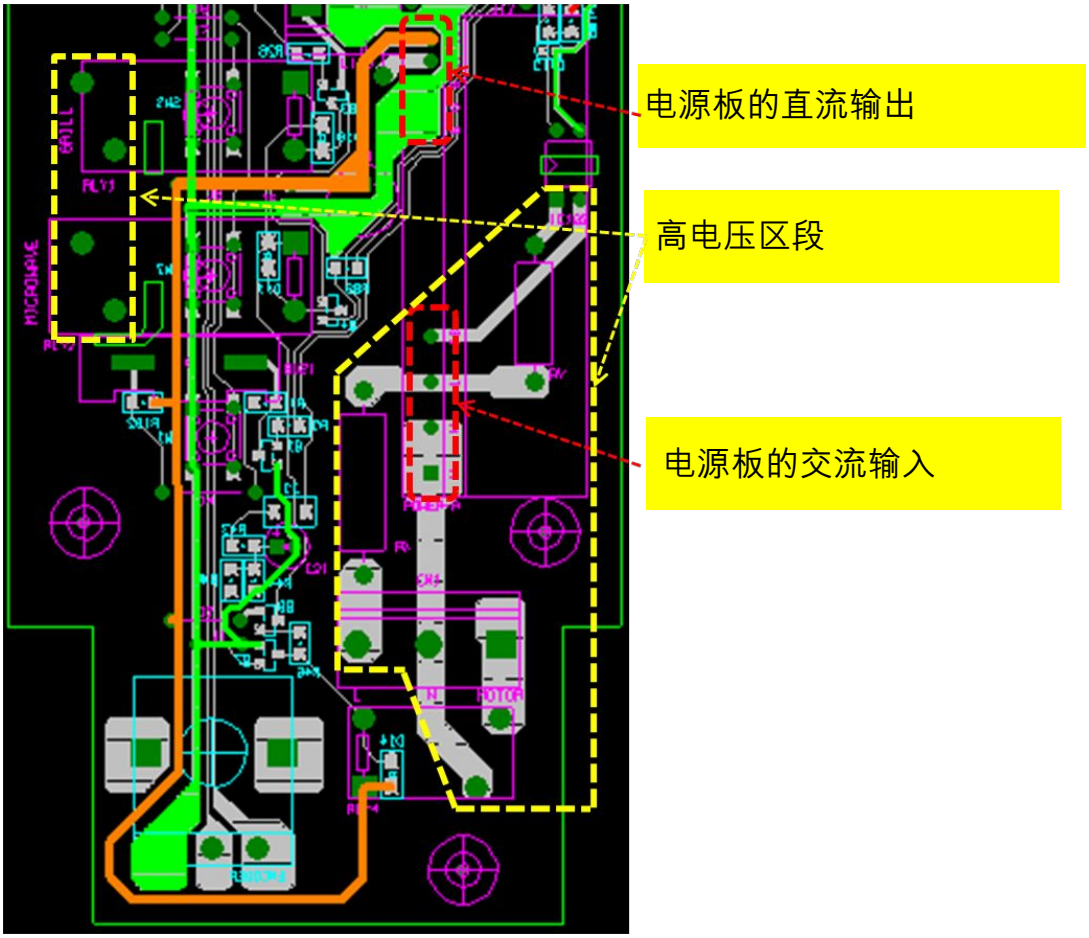

图 5. 间隔和隔离

4.7 输入和输出端口

与输出功能相比，配置为输入功能的 MCU I/O 端口对噪声更加敏感。

一般会给每个输入功能引脚添加一个 RC 滤波器，用于衰减外部噪声源注入到引脚中的噪声。 该滤波器的位置应靠近引脚。 RC 滤波器的值取决于输入信号及其特征（数字或模拟，以及变化率）。 串联电阻器的典型值在 100 Ω 到 1 kΩ 的范围内，而滤波电容器的典型值在 1000 pF 到 0.1 µF 的范围内。

RESET_b 和 NMI_b 是 Kinetis E MCU 中的特殊引脚。 由于电源引脚滤波方面的原因，RESET_b 退耦电容和这两个管脚的外部上拉的布局都应被当做电源管脚滤波。 建议最小化电容器的接地环路以及这些引脚的上拉电阻的 VDD 环路。

不要将未使用的 I/O 引脚连接到任何元件。 请将此类引脚悬空，并在软件中将其设置为低输出。 定期刷新引脚状态，以避免噪声引起其状态变化。 如果特定应用中不允许悬空引脚，请为每个未使用的引脚连接一个 10 kΩ 下拉电阻。 不要将任何未使用的 I/O 引脚直连到电源或接地。

5 软件设计

在充分考虑 EMC 要求的条件下进行合理的软件设计可以改善噪声环境中的系统整体性能和工作稳定性。

一般而言，软件设计无法改变会在系统中耦合噪声的物理媒介，或者减少外部源产生的噪声绝对值。 但是，软件可以提供智能方法让用户在故障状态下选择纠正措施，以及实施预防性功能来实现系统保护。 我们建议采用以下软件技巧来实现良好的防御性软件设计：

- 启用看门狗功能以避免代码失控。
- 定期刷新数据方向设置寄存器。
- 填充未使用的内存以避免代码失控。
- 定义所有中断向量，即使有些向量未使用。
- 选择锁频环（FLL）啮合模式。
- 始终再次确认边沿触发事件。
- 在输入端口上启用数字滤波器。

5.1 启用看门狗功能

当应用软件无法按预期执行时，看门狗 (WDOG) 功能会强制系统复位。

例如，在 MCU 中注入瞬态噪声时，激活的软件例程将跳转到意外的内存位置或进入无限循环。 必须确保即使在苛刻的条件下软件循环失控时，系统也不会停止。 将 MCU 保持在不可控状态是非常危险且不可接受的，尤其是对于安全要求较高的高功率控制应用。 建议在主循环，而不是子例程和中断例程中添加 WDOG 刷新例程。 下面提供了示例代码。

```
#define wdog_unlock() WDOG_CNT = 0x20C5; WDOG_CNT = 0x28D9
#define WDOG_CLK (WDOG_CLK_INTERNAL_1KHZ)

void wdog_enable(void)
{
/* First unlock the watchdog so that we can write to registers */
 wdog_unlock();

/* NOTE: the following write sequence must be completed within 128 buc clocks
 *
 */
/* enable watchdog */

#if (WDOG_CLK == WDOG_CLK_INTERNAL_32KHZ)
 WDOG_CS2 = 2;    /* use internal reference clock (32K) as clock source */
#elif (WDOG_CLK == WDOG_CLK_INTERNAL_1KHZ)
 WDOG_CS2 = 1;    /* use internal 1K clock as clock source */
#elif (WDOG_CLK == WDOG_CLK_EXTERNAL)
 WDOG_CS2 = 3;    /* use external clock as clock source */
#elif (WDOG_CLK == WDOG_CLK_BUS)
 WDOG_CS2 = 0;    /* use bus clock as clock source */
#else
#error "not supported WDOG clock source\n";
#endif
 WDOG_TOVALH = 0x03;
 WDOG_TOVALL = 0xE8; // ~1s

 WDOG_CS1 = 0x20
 | WDOG_CS1_EN_MASK
 //| WDOG_CS1_INT_MASK
    //| WDOG_CS1_STOP_MASK
 //| WDOG_CS1_WAIT_MASK
    //| WDOG_CS1_DBG_MASK // debug enable
    ;
}

void wdog_refresh(void){

 DisableInterrupts;    // disable interrupts

 WDOG_CNT = 0x02A6;    //Refresh sequence of writing 0x02A6
 WDOG_CNT = 0x80B4;    // and then 0x80B4 within 16 bus clocks
```

```
 EnableInterrupts;    // enable interrupts

}

void main (void){

 wdog_enable();    // enable Watch-Dog function

 for(;;){

 wdog_refresh();    // Reset the Watch-Dog counter

 MicrowaveTask();    // Application main task

 }
}
```

5.2 刷新数据方向设置寄存器

如果任何瞬态噪声意外地改变了某个端口引脚的输入或输出方向状态，应将其恢复到预期状态。

建议定义一个简单例程来定期刷新所有数据方向。 刷新周期取决于应用要求以及所注入噪声的定时模式。 对于交流电应用，可以使用通过光学耦合电路从交流电源线捕获到的 50 Hz 或 60 Hz 周期信号作为触发信号。 下面提供了示例代码。

```
#define UnABase PortBaseABCD // PortBaseABCD
#define UnAPort PortA // Port
#define UnAPins 0x7C // Bit 6,5,4,3,2
#define UnAPullupBase PullupBaseABCD // Pullup Base Address

#define Unused_A_Dir_Out()    GPIO_PDDR_REG(UnABase) |= ((uint32_t)UnAPins<<UnAPort)
#define Unused_A_Dir_In()    GPIO_PDDR_REG(UnABase) &= ~((uint32_t)UnAPins<<UnAPort)
#define Unused_A_InDis()    GPIO_PIDR_REG(UnABase) |= ((uint32_t)UnAPins<<UnAPort)

#define Unused_A_Toggle()    GPIO_PTOR_REG(UnABase) |= ((uint32_t)UnAPins<<UnAPort)
#define Unused_A_High() GPIO_PSOR_REG(UnABase) |= ((uint32_t)UnAPins<<UnAPort)
#define Unused_A_Low()    GPIO_PCOR_REG(UnABase) |= ((uint32_t)UnAPins<<UnAPort)
.
.
.

void StatusRegisterUpdate(void){
 if(mStatusRegisterUpdate_d == TRUE){

 Unused_A_InDis();
 Unused_A_Low();
 Unused_A_Dir_Out();

 Unused_B_InDis();
 Unused_B_Low();
 Unused_B_Dir_Out();

/* Port C is used as Input and Output port and refresh
by key scanning routine
*/
 //Unused_C_InDis();
 //Unused_C_Low();
 //Unused_C_Dir_Out();

 Unused_D_InDis();
 Unused_D_Low();
 Unused_D_Dir_Out();

 Unused_E_InDis();
 Unused_E_Low();
 Unused_E_Dir_Out();
```

```
 Unused_F_InDis();
 Unused_F_Low();
 Unused_F_Dir_Out();

 Unused_G_InDis();
 Unused_G_Low();
 Unused_G_Dir_Out();

 Unused_H_InDis();
 Unused_H_Low();
 Unused_H_Dir_Out();

 mStatusRegisterUpdate_d = FALSE;
 }
}
```

5.3 填充未使用的内存

应使用预定义的内容填充未使用的内存、闪存或 RAM，以便在正常执行流程而受到外部噪声源干扰时，MCU 不会执行任何意外的指令。

一种做法是使用 ARM® Cortex®-M0+ 内核中未定义的指令填充所有未使用的内存。 图 6 显示条件转移指令中的 opcode 值“1110”未定义，因此建议使用“0xDEDE”填充所有未使用的内存。 执行未定义的指令会强制处理器遍历所有故障例程以选择适当的操作。

Thumb 指令集编码

A5.2.6 条件分支和管理程序调用

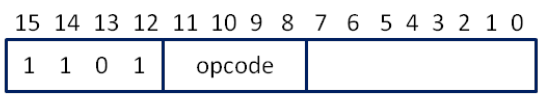

表 A5-8 显示了此空间中的编码分配。

表 A5-8：跳变和管理程序调用指令

操作码	指令	参见
not 111x	条件分支	第 A6-40 页上的 B 部分
1110	永久未定义	
1111	管理程序调用	第 A6-252 页上的 SVC（以前称 SW

图 6. 未定义的 opcode

可以遵照以下步骤以及图 7 中所示的配置，在 IAR Embedded Workbench IDE 中填充未使用的内存位置：

- 按热键组合 [Alt + F7] 访问项目选项菜单。
- 在类别中选择“Linker”（链接器）选项，并选择“Checksum”（校验和）选项卡。
- 单击“Fill unused code memory”（填充未使用的代码内存），然后填写“Fill pattern”（填充模式）、“Start address”（起始地址）和“End address”（结束地址）的值。

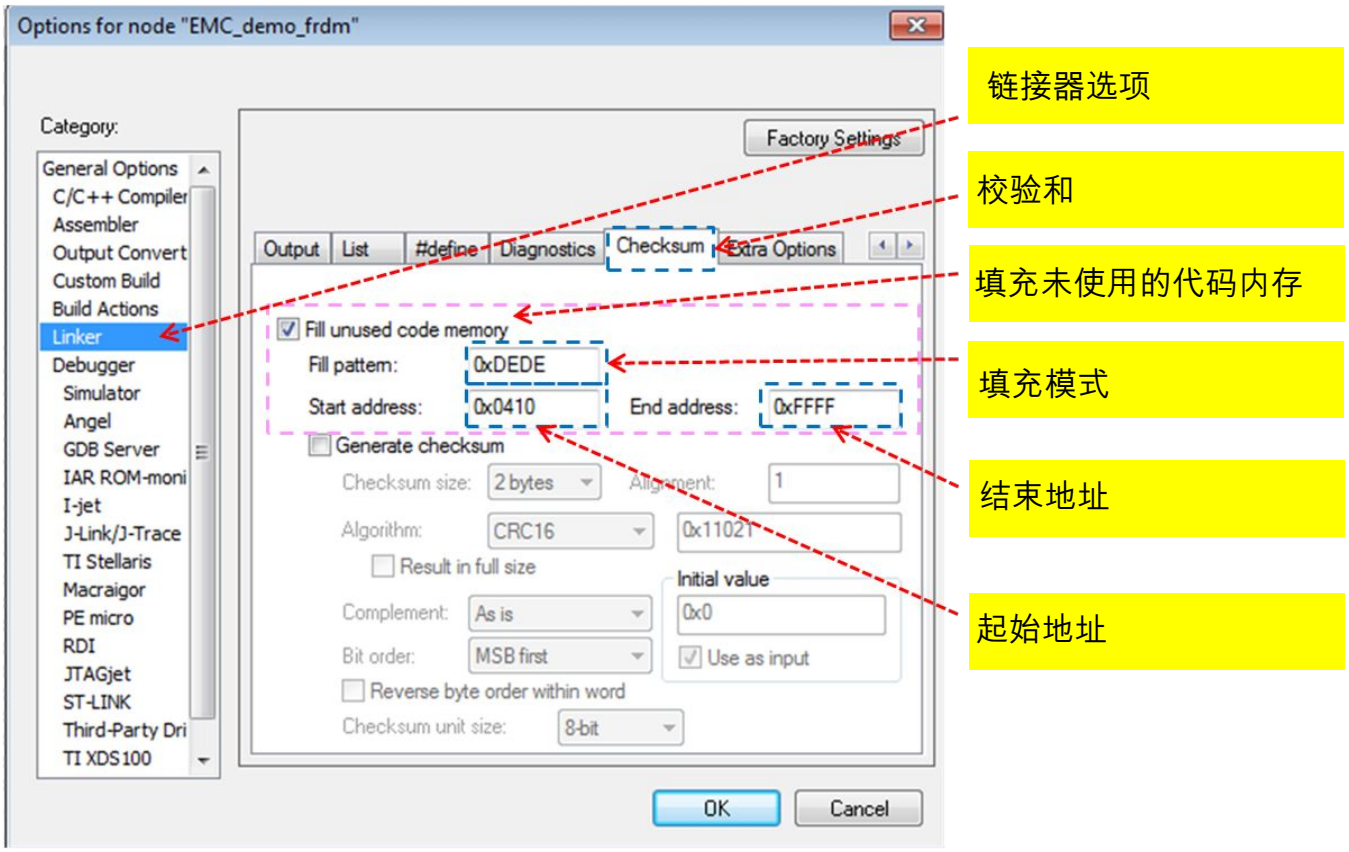

图 7. IAR 链接器选项

下面提供了接收系统复位请求的硬故障中断服务例程的示例代码。

```
void hardfault_isr(void)
{
 uint32_t temp;

 temp = SCB_AIRCR;
 temp &= 0x0000FFFF;
 SCB_AIRCR = temp | 0x05FA0000 | SCB_AIRCR_SYSRESETREQ_MASK;

 return;
}
```

注

有关详细信息，请参见 arm.com 上的《ARM Cortex-M0+ 设备通用用户指南》。

5.4 定义所有中断向量

如果为每个未使用的中断功能定义中断向量，可以在噪声源错误触发了特定的未使用中断标志时，运行软件跳转到预定义的中断例程。

在执行中断功能后，MCU 能够正确恢复执行步骤。 下面提供了示例代码。

```
/* Interrupt Vector Table Function Pointers */
typedef void pointer(void);

extern void __startup(void);
extern unsigned long __BOOT_STACK_ADDRESS[];
extern void __iar_program_start(void);
```

```
extern void SRTC_ISR(void);
extern unsigned long __initial_sp[];
extern void Reset_Handler( void );
#if (defined(__GNUC__))
extern unsigned long _estack;
extern void __thumb_startup(void);
#define VECTOR_000 (pointer*)&_estack    // ARM core Initial Supervisor SP
#define VECTOR_001 __thumb_startup    // 0x0000_0004 1 - ARM core Initial Program Counter
//#define VECTOR_001 __startup //__thumb_startup
 // 0x0000_0004 1 - ARM core Initial Program Counter

#elif (defined(KEIL))
#define VECTOR_000 (pointer*)__initial_sp    // ARM core Initial Supervisor SP
#define VECTOR_001 Reset_Handler    // 0x0000_0004 1 - ARM core Initial Program Counter
#else
 // Address Vector IRQ Source module Source description

#define VECTOR_000 (pointer*)__BOOT_STACK_ADDRESS    // ARM core Init Supervisor SP
#define VECTOR_001 __startup    // 0x0000_0004 1 - ARM core Init Program Counter
#endif
#define VECTOR_002 default_isr // 0x0000_0008 2 - ARM core NMI
#define VECTOR_003 hardfault_isr // 0x0000_000C 3 - ARM core Hard Fault
#define VECTOR_004 default_isr // 0x0000_0010 4 -
#define VECTOR_005 default_isr // 0x0000_0014 5 - ARM core Bus Fault
#define VECTOR_006 default_isr // 0x0000_0018 6 - ARM core Usage Fault
#define VECTOR_007 default_isr // 0x0000_001C 7 -
#define VECTOR_008 default_isr // 0x0000_0020 8 -
#define VECTOR_009 default_isr // 0x0000_0024 9 -
#define VECTOR_010 default_isr // 0x0000_0028 10 -
#define VECTOR_011 SVC_isr // 0x0000_002C 11 - ARM core SVCall
#define VECTOR_012 default_isr // 0x0000_0030 12 - ARM core Debug Monitor
#define VECTOR_013 default_isr // 0x0000_0034 13 -
#define VECTOR_014 default_isr // 0x0000_0038 14 - ARM core PendableSrvReq
#define VECTOR_015 default_isr // 0x0000_003C 15 - ARM core SysTick
#define VECTOR_016 default_isr // 0x0000_0040 16 0 Reserved DMA DMA 0 complete
#define VECTOR_017 default_isr // 0x0000_0044 17 1 Reserved DMA DMA 1 complete
#define VECTOR_018 default_isr // 0x0000_0048 18 2 Reserved DMA DMA 2 complete
#define VECTOR_019 default_isr // 0x0000_004C 19 3 Reserved DMA DMA 3 complete
#define VECTOR_020 default_isr // 0x0000_0050 20 4 Reserved MCM MCM
#define VECTOR_021 default_isr // 0x0000_0054 21 5 NVM FTMRH flash memory
#define VECTOR_022 default_isr // 0x0000_0058 22 6 PMC LVD,LVW interrupt
#define VECTOR_023 default_isr // 0x0000_005C 23 7 LLWU LLWU/IRQ
#define VECTOR_024 default_isr // 0x0000_0060 24 8 I2C0 I2C
#define VECTOR_025 default_isr // 0x0000_0064 25 9 - --
#define VECTOR_026 default_isr // 0x0000_0068 26 10 SPI0 SPI0
#define VECTOR_027 default_isr // 0x0000_006C 27 11 SPI1 SPI1
#define VECTOR_028 default_isr // 0x0000_0070 28 12 SCI0 UART0
#define VECTOR_029 default_isr // 0x0000_0074 29 13 SCI1 UART1
#define VECTOR_030 default_isr // 0x0000_0078 30 14 SCI2 UART2
#define VECTOR_031 default_isr // 0x0000_007C 31 15 ADC0 ADC complete
#define VECTOR_032 default_isr // 0x0000_0080 32 16 ACMP0 ACMP0
#define VECTOR_033 default_isr // 0x0000_0084 33 17 FTM0 FlexTimer0
#define VECTOR_034 default_isr // 0x0000_0088 34 18 FTM1 FlexTimer1
#define VECTOR_035 default_isr // 0x0000_008C 35 19 FTM2 FlexTimer2
#define VECTOR_036 default_isr // 0x0000_0090 36 20 RTC RTC overflow
#define VECTOR_037 default_isr // 0x0000_0094 37 21 ACMP1 ACMP1
#define VECTOR_038 default_isr // 0x0000_0098 38 22 PIT_CH0 PIT_CH0 overflow
#define VECTOR_039 default_isr // 0x0000_009C 39 23    PIT_CH1 PIT_CH1 overflow
#define VECTOR_040 default_isr // 0x0000_00A0 40 24 KBI0 Keyboard0 interrupt
#define VECTOR_041 default_isr // 0x0000_00A4 41 25 KBI1 Keyboard1 interrupt
#define VECTOR_042 default_isr // 0x0000_00A8 42 26 Reserved ---
#define VECTOR_043 default_isr // 0x0000_00AC 43 27 ICS ICS loss of lock
#define VECTOR_044 default_isr // 0x0000_00B0 44 28 WDOG Watchdog timeout
#define VECTOR_045 default_isr // 0x0000_00B4 45 29 Reserved
#define VECTOR_046 default_isr // 0x0000_00B8 46 30 Reserved
#define VECTOR_047 default_isr // 0x0000_00BC 47 31 Reserved
// END of real vector table
/
*******************************************************************************************
**********************/
```

```
#define VECTOR_048 default_isr // 0x0000_00C0 48 32 Reserved
#define VECTOR_049 default_isr // 0x0000_00C4 49 33 Reserved
#define VECTOR_050 default_isr // 0x0000_00C8 50 34 Reserved
#define VECTOR_051 default_isr // 0x0000_00CC 51 35 Reserved
#define VECTOR_052 default_isr // 0x0000_00D0 52 36 Reserved
#define VECTOR_053 default_isr // 0x0000_00D4 53 37 Reserved
#define VECTOR_054 default_isr // 0x0000_00D8 54 38 Reserved
#define VECTOR_055 default_isr // 0x0000_00DC 55 39 Reserved
#define VECTOR_056 default_isr // 0x0000_00E0 56 40 Reserved
#define VECTOR_057 default_isr // 0x0000_00E4 57 41 Reserved
#define VECTOR_058 default_isr // 0x0000_00E8 58 42 Reserved
#define VECTOR_059 default_isr // 0x0000_00EC 59 43 Reserved
#define VECTOR_060 default_isr // 0x0000_00F0 60 44 Reserved
#define VECTOR_061 default_isr // 0x0000_00F4 61 45 Reserved
#define VECTOR_062 default_isr // 0x0000_00F8 62 46 Reserved
#define VECTOR_063 default_isr // 0x0000_00FC 63 47 Reserved
#define VECTOR_064 default_isr // 0x0000_0100 64 48 Reserved
#define VECTOR_065 default_isr // 0x0000_0104 65 49 Reserved
#define VECTOR_066 default_isr // 0x0000_0108 66 50 Reserved
#define VECTOR_067 default_isr // 0x0000_010C 67 51 Reserved
#define VECTOR_068 default_isr // 0x0000_0110 68 52 Reserved
#define VECTOR_069 default_isr // 0x0000_0114 69 53 Reserved
#define VECTOR_070 default_isr // 0x0000_0118 70 54 Reserved
#define VECTOR_071 default_isr // 0x0000_011C 71 55 Reserved
#define VECTOR_072 default_isr // 0x0000_0120 72 56 Reserved
#define VECTOR_073 default_isr // 0x0000_0124 73 57 Reserved
#define VECTOR_074 default_isr // 0x0000_0128 74 58 Reserved
#define VECTOR_075 default_isr // 0x0000_012C 75 59 Reserved
#define VECTOR_076 default_isr // 0x0000_0130 76 60 Reserved
#define VECTOR_077 default_isr // 0x0000_0134 77 61 Reserved
#define VECTOR_078 default_isr // 0x0000_0138 78 62 Reserved
#define VECTOR_079 default_isr // 0x0000_013C 79 63 Reserved
#define VECTOR_080 default_isr // 0x0000_0140 80 64 Reserved
#define VECTOR_081 default_isr // 0x0000_0144 81 65 Reserved
#define VECTOR_082 default_isr // 0x0000_0148 82 66 Reserved
#define VECTOR_083 default_isr // 0x0000_014C 83 67
#define VECTOR_084 default_isr // 0x0000_0150 84 68
#define VECTOR_085 default_isr // 0x0000_0154 85 69
#define VECTOR_086 default_isr // 0x0000_0158 86 70
#define VECTOR_087 default_isr // 0x0000_015C 87 71
#define VECTOR_088 default_isr // 0x0000_0160 88 72
#define VECTOR_089 default_isr // 0x0000_0164 89 73
#define VECTOR_090 default_isr // 0x0000_0168 90 74
#define VECTOR_091 default_isr // 0x0000_016C 91 75
#define VECTOR_092 default_isr // 0x0000_0170 92 76
#define VECTOR_093 default_isr // 0x0000_0174 93 77
#define VECTOR_094 default_isr // 0x0000_0178 94 78
#define VECTOR_095 default_isr // 0x0000_017C 95 79
#define VECTOR_096 default_isr // 0x0000_0180 96 80
#define VECTOR_097 default_isr // 0x0000_0184 97 81
#define VECTOR_098 default_isr // 0x0000_0188 98 82
#define VECTOR_099 default_isr // 0x0000_018C 99 83

#ifdef USE_BOOTLOADER
#else
#define CONFIG_1    (pointer*)0xffffffff
#define CONFIG_2    (pointer*)0xffffffff
#define CONFIG_3    (pointer*)0xffffffff
#define CONFIG_4    (pointer*)0xfffefffe
#endif
#endif /*__VECTORS_H*/
```

5.5 选择 FLL 啮合模式

建议对内部时钟源 (ICS) 模块中的内部或外部参考时钟启用 FLL 啮合模式，此模块为 MCU 提供了时钟源选项。

参考时钟源先将低频率除以参考分频频率，然后在 FLL 模块中将频率倍上去。 最终的内核或总线时钟等于 FLL 输出频率除以内核或总线分频器。

在 ICS 模块中转换频率的优势如下：

- 与低频时钟源（参考分频器后面）相比，在相对于时钟周期的干扰宽度方面，高频时钟源（参考分频器前面）上的瞬态噪声干扰影响更加明显。
- 一般而言，由于低通滤波器的特征，FLL 模块的响应速度不足以对此类短脉噪声做出反应。

下面提供了示例代码。

```
#define EXT_CLK_CRYST    4000    /* in KHz */
#define BUS_CLK_4MHz    /* define bus frequency */

void FEI_to_FEE(void)
{
    /* assume external crystal is 8Mhz or 4MHz
    *
    */
    /* 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 */
#if defined(CRYST_HIGH_GAIN)
    | OSC_CR_HGO_MASK /* Rs must be added and be large up to 200K */
#endif
#if (EXT_CLK_CRYST >=4000)
    | OSC_CR_RANGE_MASK
#endif
    | OSC_CR_OSCOS_MASK;    /* for crystal only */
#if defined(IAR)
    asm(
    "nop \n"
    "nop \n"
    );
#elif defined(__MWERKS__)
    asm{
    nop
    nop
};
#endif
    /* 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
    *
    */

 #if (EXT_CLK_CRYST == 8000) ||    (EXT_CLK_CRYST == 10000)
 /* 8MHz */
 ICS_C1 = ICS_C1 & ~(ICS_C1_RDIV_MASK) | ICS_C1_RDIV(3);    /* 8000/256 = 31.25K */
 #elif (EXT_CLK_CRYST == 4000)
 /* 4MHz */
 ICS_C1 = ICS_C1 & ~(ICS_C1_RDIV_MASK) | ICS_C1_RDIV(2);    /* 4000/128 = 31.25K */
 #elif (EXT_CLK_CRYST == 16000)
 /* 16MHz */
 ICS_C1 = ICS_C1 & ~(ICS_C1_RDIV_MASK) | ICS_C1_RDIV(4);    /* 16000/512 = 31.25K */

 #elif (EXT_CLK_CRYST == 20000)
 /* 20MHz */
 ICS_C1 = ICS_C1 & ~(ICS_C1_RDIV_MASK) | ICS_C1_RDIV(4);    /* 20000/512 = 39.0625K */

 #elif (EXT_CLK_CRYST == 32)
 ICS_C1 = ICS_C1 & ~(ICS_C1_RDIV_MASK);
 #else
 #error "Error: crystal value not supported!\n";
```

```
 #endif

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

    /* wait for the reference clock to be changed to external */
#if defined(IAR)
    asm(
    "nop \n"
    "nop \n"
    );
#elif defined(__MWERKS__)
    asm{
    nop
    nop
};
#endif
    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);

    }

#if defined(BUS_CLK_4MHZ)

 ICS_C2 = (ICS_C2 & ~(ICS_C2_BDIV_MASK)) | ICS_C2_BDIV(3);    // divided by 8

#elif defined(BUS_CLK_8MHZ)

 ICS_C2 = (ICS_C2 & ~(ICS_C2_BDIV_MASK)) | ICS_C2_BDIV(2); // divided by 4
#else
 ICS_C2 = (ICS_C2 & ~(ICS_C2_BDIV_MASK)) | ICS_C2_BDIV(1); // divided by 2
#endif

    /* now system/bus clock is the target frequency
    *
    */
    /* clear Loss of lock sticky bit */
    ICS_S |= ICS_S_LOLS_MASK;
}
```

5.6 再次确认边沿触发事件

多次读取每个边沿触发中断服务的输入数据，是确认输入事件是否有效，并且是否由指定源驱动的重要技巧。

应该使用某种不规则模式调整每两次读取循环数据的时隙，以防止将均匀分布的噪声模式识别为有效事件。 可以在每两次读数之间插入一个简单的随机延迟函数，使整体重复周期不一致。 随机延迟变量可以等于发生中断触发事件时捕捉到的自由运行计数器值。

下面提供了示例代码。

```
/* Random Delay Loop */
uint8_t RandomDelay(void){

uint32_t random_32bit = RANDOM_COUNTER;

 mRandomDelayCount = TPMxCnVLvalue(random_32bit);
```

```
 mRandomDelayCount &= gRandomDelayCountMask_c;
 return mRandomDelayCount;
}

for (iKey = 0; iKey < KeyDebounce; iKey++){

// random delay
 uint8_t idelay;

 idelay = RandomDelay();

 while(idelay > 0){
 --idelay;
 delay_1ms();
 }

 KeyScanValue[iKey] = Sw2Pin_Read();

 if (iKey != 0){

 if ((KeyScanValue[iKey] == KeyScanValue[iKey - 1])&&(KeyScanValue[iKey]==0)){

 KeyDetected_d = TRUE;

 }else{

 KeyDetected_d = FALSE;

 }

 }else {
 KeyDetected_d = FALSE;
 }

}
```

5.7 启用数字滤波器

数字滤波器是 Kinetis E MCU 中的一个功能组件，针对配置为数字输入的每个端口引脚提供简单低通滤波特性。

在一个端口中启用的所有数字滤波器的滤波宽度（以时钟大小为单位）相同，只有在禁用该端口的所有数字滤波器时才能更改此宽度。 这个可配置的滤波器能够以自适应的方式处理不同类型的瞬态噪声，这些噪声本质上具有一定的脉冲宽度，如使用传统的模拟滤波器则很难处理。

下面提供了示例代码。

```
#define PortFilterEnable

#ifdef PortFilterEnable
PORT_IOFLT = PORT_IOFLT_FLTDIV3(LPOCLK_2) // Set FLTDIV3 to LPOCLK divided by 2
  | PORT_IOFLT_FLTDIV2(BUSCLK_64) // Set FLTDIV2 to BUSCLK divided by 64
  | PORT_IOFLT_FLTDIV1(BUSCLK_8) // Set FLTDIV1 to BUSCLK divided by 8
  | PORT_IOFLT_FLTNMI(SEL_FLFDIV3) // Select FLTDIV3 for NMI
  | PORT_IOFLT_FLTKBI1(SEL_FLFDIV2) // Select FLTDIV2 for KBI1
  | PORT_IOFLT_FLTKBI0(SEL_FLFDIV2) // Select FLTDIV2 for KBI0
  | PORT_IOFLT_FLTRST(SEL_FLFDIV3) // Select FLTDIV3 for RST
  | PORT_IOFLT_FLTH(SEL_FLFDIV1) // Select FLTDIV1 for Port H
  | PORT_IOFLT_FLTG(SEL_FLFDIV1) // Select FLTDIV1 for Port G
  | PORT_IOFLT_FLTF(SEL_FLFDIV1) // Select FLTDIV1 for Port F
  | PORT_IOFLT_FLTE(SEL_FLFDIV1) // Select FLTDIV1 for Port E
  | PORT_IOFLT_FLTD(SEL_FLFDIV1) // Select FLTDIV1 for Port D
  | PORT_IOFLT_FLTC(SEL_FLFDIV1) // Select FLTDIV1 for Port C
  | PORT_IOFLT_FLTB(SEL_FLFDIV1) // Select FLTDIV1 for Port B
  | PORT_IOFLT_FLTA(SEL_FLFDIV1); // Select FLTDIV1 for Port A
#endif
```

6 结论

本应用说明中所述的 EMC design 提示可帮助客户在使用 Kinetis E 系列微控制器的前期设计阶段，充分考虑到 EMC 方面的要求。

我们以快速参考的形式提供了有关硬件和软件技巧的详细说明，让客户更有效地采用 Freescale 解决方案。

7 参考

freescale.com 上提供了以下文档。

1. 《AN4438：*MC9S08PT60* 的 *EMC Design* 注意事项》，由 T.C. Lun 在 2012 年编写。
2. 《AN4476：家用电器应用中使用的 *5V 8* 位系列系统设计指导》，由 T.C. Lun、Dennis Lui 在 2012 年编写。
3. 《AN4463：如何开发在噪声环境中使用的可靠软件》，由 Dennis Lui、T.C. Lun 在 2012 年编写。
4. 《AN2321：板级电磁兼容设计》，由 T.C. Lun 在 2002 年编写。
5. 《AN2764：改善基于微控制器的应用的瞬态抗扰性能》，由 Ross Carlton、Greg Racino 和 John Suchyta 在 2005 年编写。

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number AN4779
Revision 0, August 1, 2013