Freescale Semiconductor
Application Note

Document Number:AN4779
Rev 0, August 1, 2013

EMC Design Tips for Kinetis E Family

by: Dennis Lui and T.C. Lun

Contents

1 Introduction....1
2 System overview....1
3 EMC design tips....2
4 Hardware design....3
5 Software design....7
6 Conclusion....17
7 References....17

1 Introduction

Electromagnetic Compatibility (EMC) design consideration is one of the critical factors to ensure a system is robust in design, able to operate flawlessly in harsh environments, and does not cause interference. This application note provides design tips on how to use Kinetis E series MCU in applications with EMC requirements.

Different techniques in hardware design, printed circuit board (PCB) layout, and software setting are illustrated here to help customers to apply EMC enhancements on their products at the beginning of the design phase. In general, EMC issues in final stages are more complicated, expensive, and time consuming to fix. There are many constraints on circuit and PCB layout modifications:

- When all component or module placements are fixed inside the system.
- Higher cost structure due to additional components used for those corrective actions.
- Solutions may invoke major design changes on mechanical aspects, which impacts project schedule.

2 System overview

A typical application using a Kinetis E series MCU is used as an example to demonstrate EMC design tips in practical use.

The application note AN4476: EMC Design Considerations for MC9S08PT60, available on freescale.com, provides detailed descriptions on basic EMC concepts and theory, which can help application developers understand the reason behind each of the EMC design tips. Read this application note along with other Kinetis E family documentation such as Kinetis E Reference Manual and Kinetis E Sub-Family Data Sheet, available on freescale.com, to understand the details of device characteristics, register configurations, and firmware coding. The example code snippets are written with IAR Embedded Workbench 6.40.

This is a typical application block diagram.

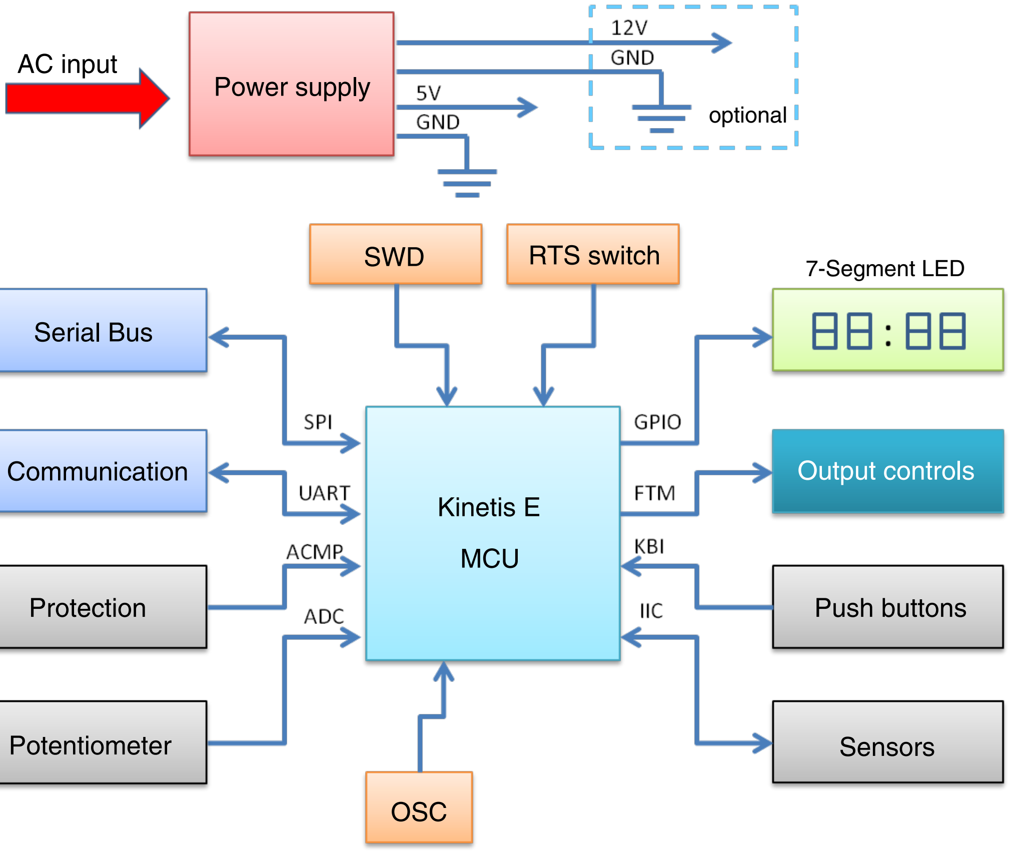


Figure 1. Typical application block diagram

The AC power line voltage is converted down and regulated to 5 V in the power supply block. The main supply for the whole system, including MCU, GPIO, display, and analog peripherals, is 5 V. In some applications, 12 V supply option is also required for high-power control circuits. For example, most power relay switches are controlled by 12 V driving circuits, but the high current stages are drawn from the AC power line input directly.

The Kinetis E MCU is used for all signal detections on user input interface from traditional pushbuttons, communications to host controller through standard UART serial port, system monitoring from sensor devices on IIC bus or direct voltage input at ADC pins, power controls using GPIO pins with specific sequential order for system protection purpose, and hardware fault detection at analog comparator inputs.

3 EMC design tips

The EMC design tips are discussed in the following sections, separate from hardware and software points of view. Hardware or software engineers can select the section according to their requirements and apply the tips directly into their design.

The hardware design tips cover board level considerations, which include design techniques on PCB layout and precautions for different type of I/O ports. The primary objective is to make use of EMC knowledge to prevent any internal or external noises that affect the system operation and stability; minimize the coupling effectiveness from the noise source to the victim (for example, the MCU), reduce the noise magnitude from the interfering source, and increase the noise immunity of the receptor.

A defensive software design concept is another way to address the EMC issues caused by improper software handling on false triggered events in a noisy environment. The software must be able to identify if a particular event is a false alarm triggered by noise sources, or a normal driven event. It must then make a smart decision on corresponding actions. For example, the MCU must not start a high-power control stage if there is any uncertainty on the requested action.

4 Hardware design

The hardware considerations for MCU application in noisy environment consist of PCB layout design and external component connections for peripheral interfaces.

At the board level, the PCB layout is the key factor for noise coupling from internal or external noise sources. The traces on the layout act as coupling paths, and the geometry factors of the traces (length, width, shape, and position) affect the coupling effectiveness significantly. A proper board and cable placement in the system can help to isolate noise sources from the system and increase the system immunity level. The following subsections describe techniques recommended for a robust hardware design.

4.1 Single-layer PCB

High cost multi-layer PCB design provides more flexibility on component placement, signal trace routing, power supply decoupling, and reference grounding.

However, the size and shape of the PCB are limited by the mechanical form factor, which is the key obstacle for PCB design in most cases. For cost consideration, a single-layer with double-side loading PCB is a good choice for most home appliance applications, but it is more challenging to design this kind of PCB with a high pin-count device. The following sections show you how to implement the PCB layout with good EMC practices.

4.2 Placement methods

Component placement must satisfy the list of mechanical constraints for the product.

The general guidelines for reference are as follows:

- Mark all positions for screw holes and mounting points as keep-out area.
- Place all user interface components with fixed position requirement. For example: display panel, control buttons, and connectors.
- Separate high-power circuitry from low-power and noise sensitive circuitry.
- Place associated components into small groups, and try to align the groups in a logic order which matches with the corresponding signal flow.
- Identify all critical components that need to be placed near the MCU; external components connected from MCU input ports to power or ground. For example: supply decoupling capacitors and filter components for input signals.
- Minimize the area formed by the power loops and ground loops.
- Reduce the common mode impedance from power and ground to the MCU.

It may require considerable effort to finalize an acceptable version which is able to fulfill all the constraints.

4.3 Power supply and ground routing

The PCB layout for power supply and ground plane are extremely important for EMC performance in board-level, especially in multi-supply system with 5 V and 12 V.

PCB layout technique is used to separate the ground plane into two portions as shown in this figure. One is defined as the return path for 12 V circuits, and the other is the 5 V return path for MCU and other critical components. The noise from the 12 V ground will not be coupled with the 5 V ground through the ground traces.

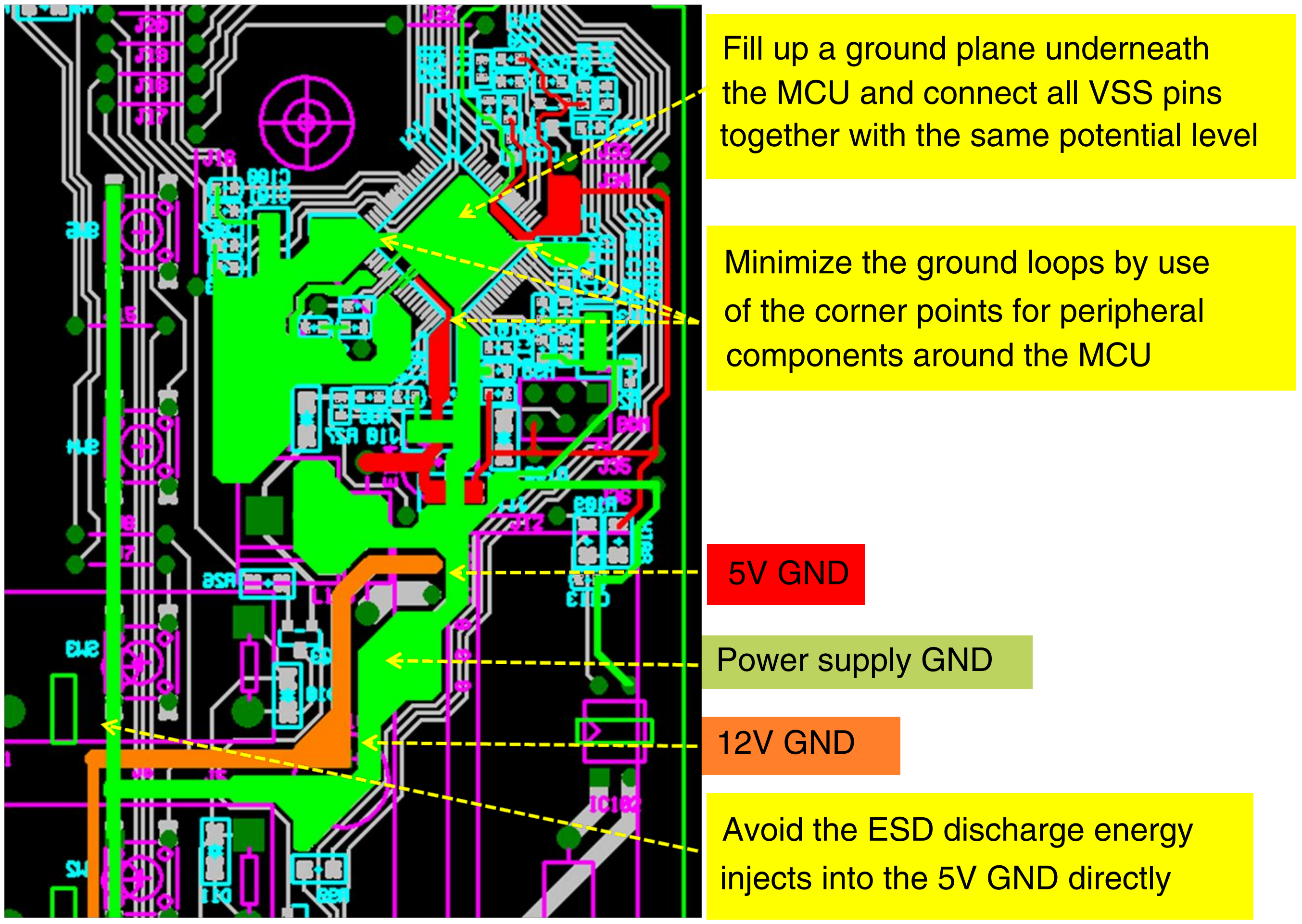

Fill up a ground plane underneath the MCU and connect all VSS pins together with the same potential level
Minimize the ground loops by use of the corner points for peripheral components around the MCU
5V GND
Power supply GND
12V GND
Avoid the ESD discharge energy injects into the 5V GND directly

Figure 2. Power supply and ground routing

In some application cases, the 12 V ground is intentionally used as the return path for components operated in 5 V supply rail and subject to ESD damage in air-discharge test. Connecting the 12 V ground to those 5 V components prevents the ESD discharge energy couples into the 5 V ground directly. The MCU may be forced to reset, halt, or even damage if high energy passes the MCU ground.

The MCU ground connection method in PCB layout is an essential factor of the EMC performance. It fills up a ground plane underneath the MCU and connects all VSS pins together, which is a good practice for EMC consideration. This method ensures all MCU VSS pins are kept at the same potential level, and also minimizes the inductance on current return path from MCU to bypass capacitors for high-frequency noise. For the LQFP package, the MCU ground plane can be further extended to the package corner points to achieve short ground paths with minimum loop area for other peripheral components around the MCU.

4.4 Decoupling and bypassing

It is necessary to have better understanding on the concepts of decoupling and bypassing to avoid any incorrect implementation for EMC issue:

- Decoupling is used to isolate noise between circuits on its common line. The power trace is one of the common lines from a voltage regulator to the MCU.
- Bypassing is used to reduce the high-frequency current flows in an impedance path by shunting that path with a bypass capacitor.

The effectiveness of adding decoupling and bypass capacitors for the MCU are very dependent on joining position and sequence as shown this figure. The guidelines of PCB layout on MCU supply pins (VDD and VSS) are as follows:

- Connect the power and ground traces from the power source to the decoupling capacitors and then connect them to the bypass capacitors before going to MCU's VDD and VSS pins.
- Place the power and ground traces in parallel to minimize the loop area.
- Place the bypass capacitor to each VDD and VSS pair as close as possible.

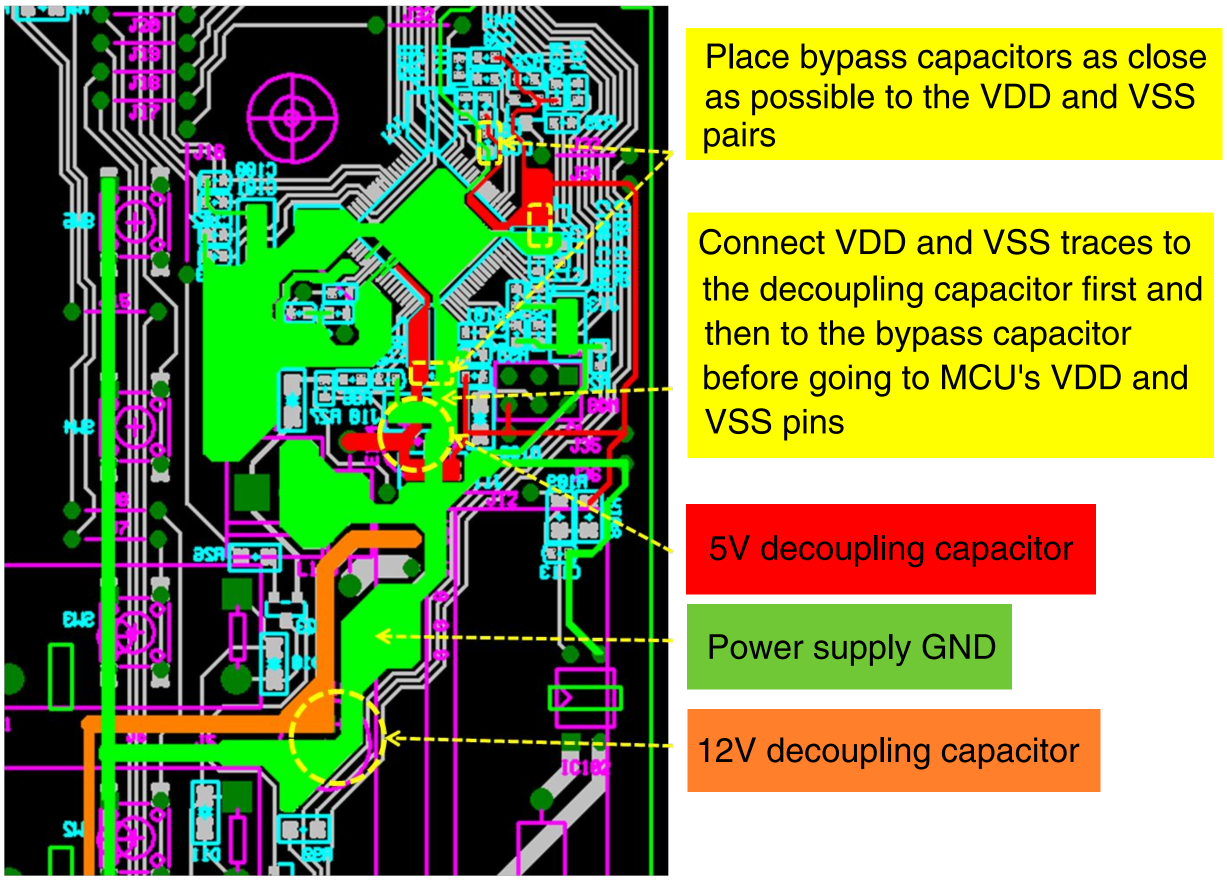

Place bypass capacitors as close as possible to the VDD and VSS pairs
Connect VDD and VSS traces to the decoupling capacitor first and then to the bypass capacitor before going to MCU's VDD and VSS pins
5V decoupling capacitor
Power supply GND
12V decoupling capacitor

Figure 3. Decoupling and bypassing

4.5 Crystal oscillator circuit

The crystal oscillator components connected at MCU EXTAL and XTAL pins are very sensitive to external noise.

The PCB routes ground traces in the form of a guard ring, along with the traces connecting to the EXTAL and XTAL pins can minimize the noise coupling into the crystal circuit. An example is shown in this figure, and the general guidelines are as follows:

- Do not place any signal trace (except the ground traces) near crystal circuit or across the bottom side of the circuit.
- Place the oscillator circuit components to the EXTAL and XTAL pins (crystal, feedback resistor, and loading capacitors) as close as possible.
- Select the internal oscillator as clock source for better EMC performance.
- Connect the ground of loading capacitor to the ground plane directly when in double-layer or multilayer PCB.
- Select minimum bus frequency to fulfill system requirements.
- Apply minimum trace length to oscillator circuit.
- Use suitable value of feedback resistor and loading capacitors.

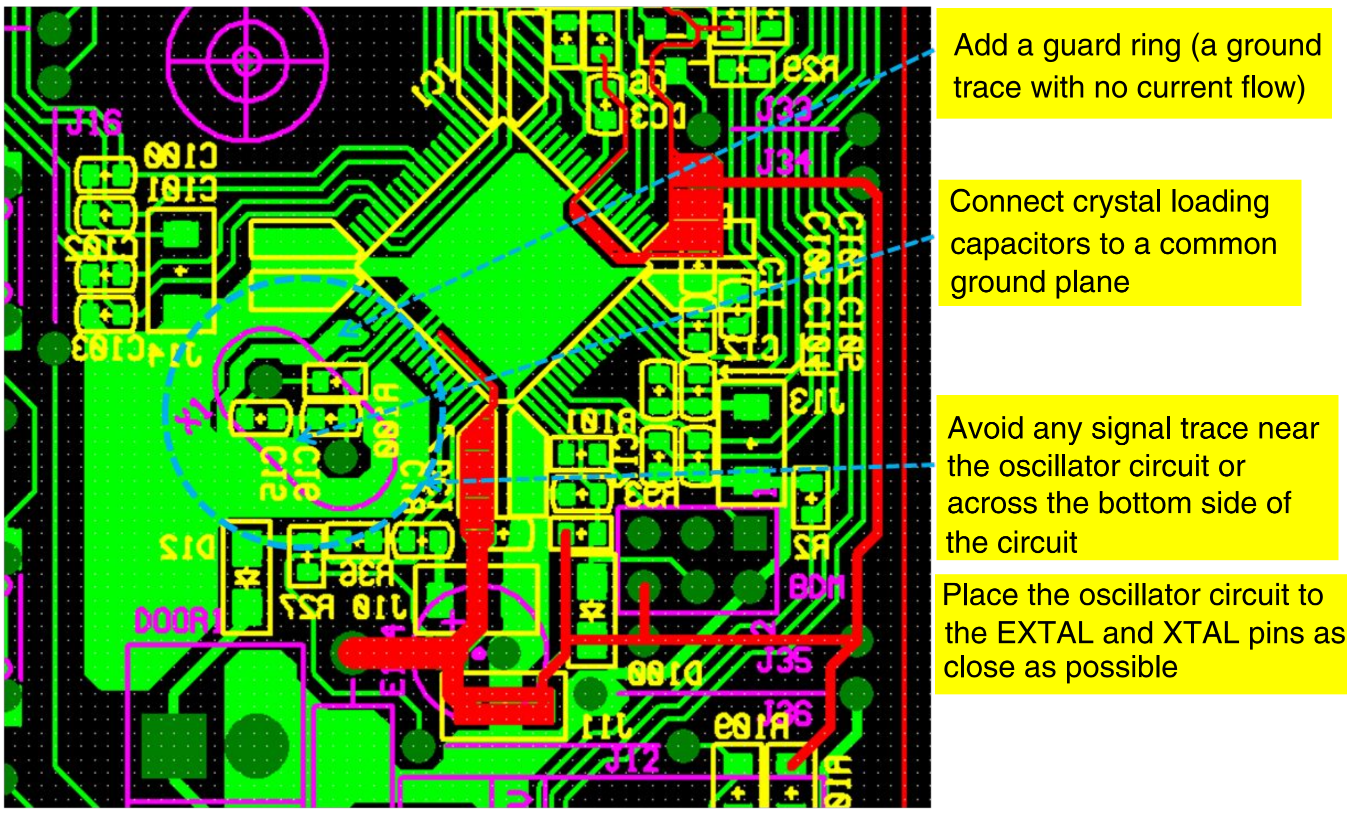

Add a guard ring (a ground trace with no current flow)
Connect crystal loading capacitors to a common ground plane
Avoid any signal trace near the oscillator circuit or across the bottom side of the circuit
Place the oscillator circuit to the EXTAL and XTAL pins as close as possible

Figure 4. Crystal oscillator circuit

4.6 Spacing and isolation

The isolation for different circuit blocks is important when an AC high-power circuit is involved side by side to a low-power circuit on the same board as shown in this figure.

In some cases, you may need to add a physical slot for better isolation if the board size is limited. Similarly, apply enough isolation space between the PCB trace and mounting screw holes or board edge for ESD consideration.

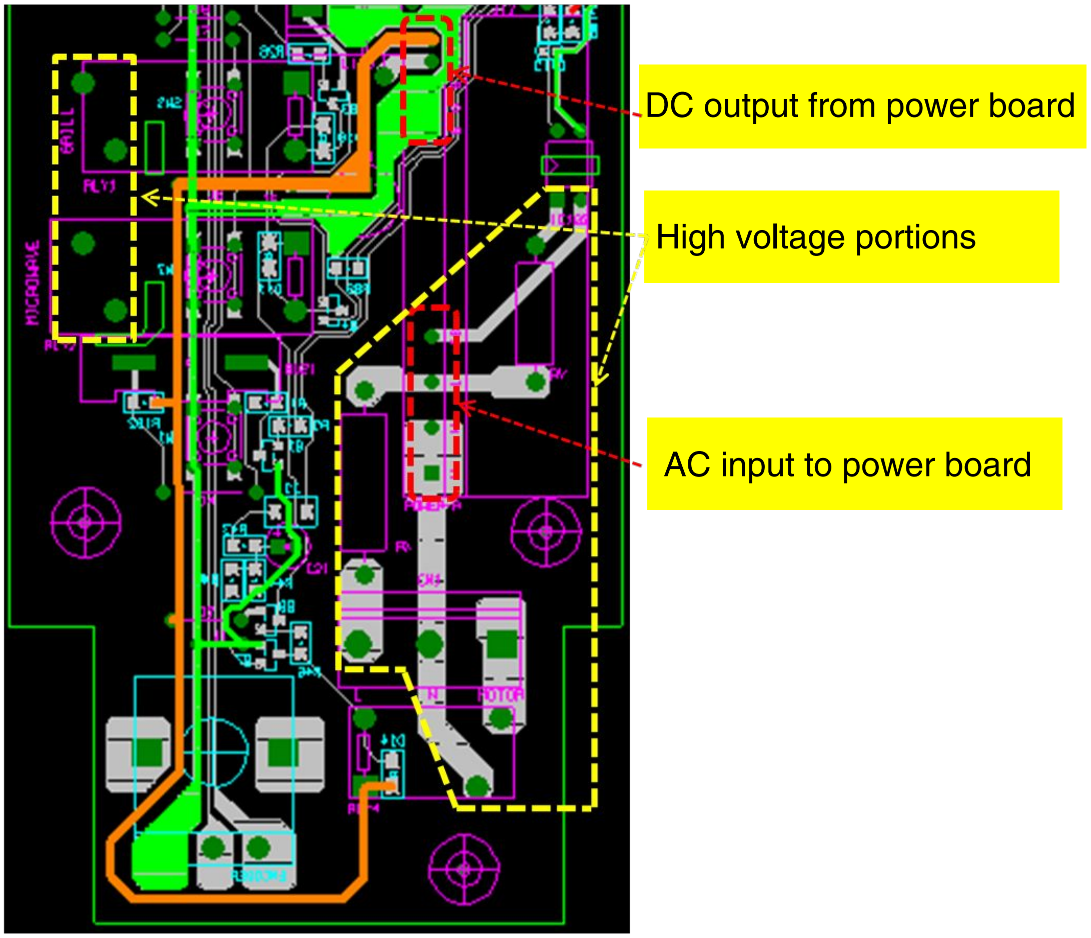


Figure 5. Spacing and isolation

4.7 Input and output port

The MCU I/O ports, configured as input function, are more sensitive to noise when compared to the output function.

In general, an RC filter is added for each input function pin to attenuate the noise injected into the pin from external noise sources. The placement of the filter should be close to the pin. The value of the RC filter depends on the input signal and its characteristics (digital or analog, and rate of change). The typical value of the series resistor is in the range of 100 Ω to 1 kΩ while the value of filtering capacitor is in the range of 1000 pF to 0.1 μF.

The RESET_b and NMI_b are special pins in the Kinetis E MCU. Placement of decoupling capacitor for RESET_b pin and the external pullup for both pins should be considered as power pin filtering. Minimizing the ground loop for the capacitor and the VDD loop for the pullup resistor of these pins is recommended.

Do not connect unused I/O pins to anything. Make it floating, and then set it as output low in software. Periodically refresh the state of the pin to avoid changes in state by noise. If floating pins are not allowed in a particular application, connect a 10 kΩ pulldown resistor for each unused pin. Do not connect any unused I/O pin to power or ground directly.

5 Software design

A good software design with EMC considerations improves overall system performance and operating stability in noisy environments.

In general, the software design cannot change the physical media which couples noise into the system, or reduce the absolute magnitude of noise generated from external sources. However, the software provides an intelligent method to select corrective actions in fault conditions and implement precautionary features for system protection. These software techniques are recommended for a good defensive software design:

- Enable WatchDog function to avoid code runaway.
- Refresh data direction setting registers periodically.
- Fill unused memory to avoid code runaway.
- Define all interrupt vectors, even those that are not used.
- Select Frequency-Locked Loop (FLL) engaged mode.
- Always reconfirm edge triggered event.
- Enable digital filter on input port.

5.1 Enable WatchDog function

The WatchDog (WDOG) function forces a system reset when the application software fails to execute as expected.

For example, an active software routine jumps into an unexpected memory location or runs into an infinite loop when transient noise is injected into the MCU. It is important to make sure that the system will not halt even the software loop is out of control in harsh conditions. Holding the MCU in an uncontrollable state is very dangerous and unacceptable, especially for high-power control applications with safety requirements. It is recommended to add the WDOG refresh routine in the main loop instead of sub-routines and interrupt routines. The sample code is given here.

```
#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 Refresh data direction setting registers

The input or output direction state for each port pin should be recovered to the expected condition, if it has been changed by any transient noise accidentally.

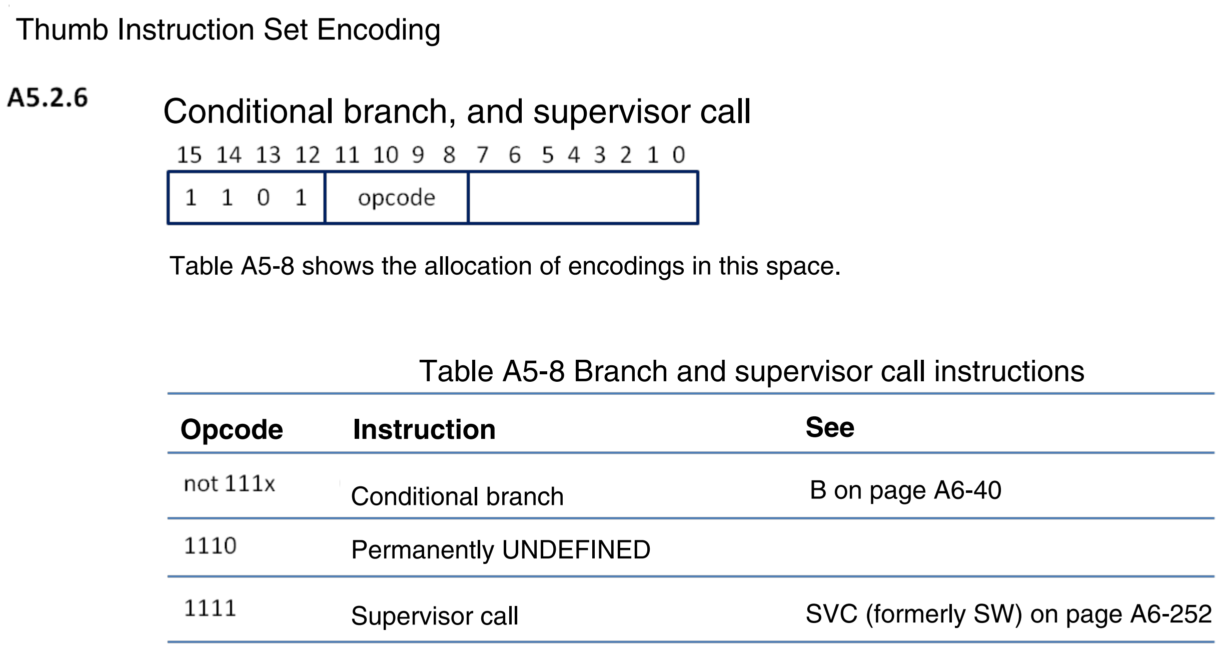
It is recommended to define a simple routine to refresh all data directions periodically. The refresh period depends on the application requirement and timing pattern of the injected noise. For AC power application, the 50 Hz or 60 Hz periodic signal captured from the AC power line through an optical coupling circuit can be used as a trigger signal. The sample code is given here.

```
#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 Fill unused memory

Unused memory, flash memory or RAM should be filled with predefined content so that the MCU does not execute any unexpected instruction when the normal execution flow is disturbed by external noise sources.

One option is to fill all unused memory with instruction which is not defined in ARM® Cortex®-M0+ core. Figure 6 shows the opcode value of “1110” in conditional branch instruction is undefined, so it is recommended to fill all unused memory with “0xDEDE”. The execution of an undefined instruction will force the processor to go through the fault routine for appropriate action.



Thumb Instruction Set Encoding

A5.2.6 Conditional branch, and supervisor call

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	opcode											

Table A5-8 shows the allocation of encodings in this space.

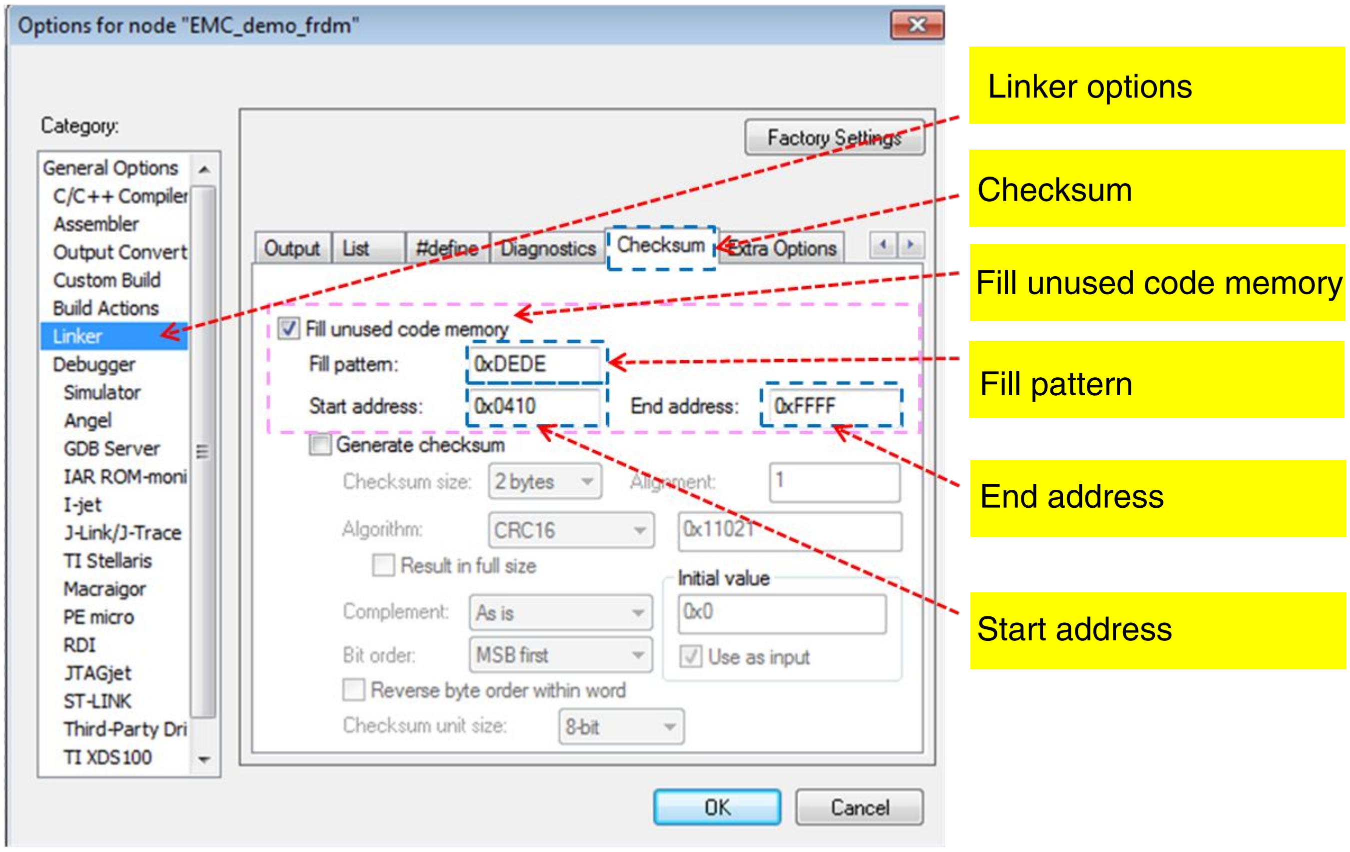
Table A5-8 Branch and supervisor call instructions

Opcode	Instruction	See
not 111x	Conditional branch	B on page A6-40
1110	Permanently UNDEFINED	
1111	Supervisor call	SVC (formerly SW) on page A6-252

Figure 6. Undefined opcode

The unused memory locations can be filled in IAR Embedded Workbench IDE by following steps and the configuration is shown in Figure 7 :

- Access the project option menu by pressing hot key combination of [Alt + F7].
- Select Linker option in category and the tab of Checksum.
- Click the "Fill unused code memory" and fill in the value for Fill pattern, Start address and End address.



☑ Fill unused code memory
Fill pattern: 0xDEDE
Start address: 0x0410 End address: 0xFFFF
☐ Generate checksum
Checksum size: 2 bytes Alignment: 1
Algorithm: CRC16 0x11021
☐ Result in full size
Initial value: 0x0
Complement: As is
Bit order: MSB first ☑ Use as input
☐ Reverse byte order within word
Checksum unit size: 8-bit

Figure 7. IAR linker options

The sample code for hard fault interrupt service routine with system reset request is given here.

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

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

  return;
}
```

NOTE

For details, see ARM Cortex-M0+ Devices Generic User Guide, available on arm.com.

5.4 Define all interrupt vectors

Defining the interrupt vectors for each unused interrupt function allows the running software to jump into a predefined interrupt routine when a particular unused interrupt flag is false-triggered by a noise source.

The MCU is able to resume the execution steps correctly after the interrupt function. The sample code is given here.

```
/* 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*)0xfffeffff
#endif
#endif /*__VECTORS_H*/
```

5.5 Select FLL engaged mode

It is recommended to enable the FLL engaged mode with internal or external reference clock in the internal clock source (ICS) module, which provides clock source option for the MCU.

The reference clock source first divides the lower frequency by reference divider and then multiplies the frequency up in FLL module. The final core or bus clock is equal to FLL output frequency divided by the core or bus frequency divider.

The advantages of the frequency conversion in the ICS module are:

- The impact of transient noise glitch on high-frequency clock source (before the reference divider) is more significant compared to a low-frequency clock source (after the reference divider) in terms of the glitch width against the clock cycle.
- In general, the response of the FLL module is not fast enough to react to such kinds of short pulse noise due to the low-pass filter characteristic.

The sample code is given here.

```
#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 Reconfirm edge triggered

Multiple reading on input data for each edge-triggered interrupt service is an important technique to confirm if the input event is valid and driven by determined sources.

The timing slot between each successive reading inside the loop should be adjusted with some kind of irregular pattern, such that an evenly distributed noise pattern will not be recognized as a valid event. A simple random delay function is inserted between each reading so the overall repeat period is not consistent. The random delay variable can be equal to a free-running counter value captured when there is an interrupt trigger even.

The sample code is given here.

```
/* 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 Enable digital filter

The digital filter is a feature in Kinetis E MCU that provides a simple low-pass filter characteristic for each port pin that is configured as a digital input.

The filter width in clock size is the same for all enabled digital filters within one port, and should be changed only when all digital filters for that port are disabled. This configurable filter provides an adaptive way to handle different types of transient noises with deterministic pulse width in nature, which are difficult to handle by traditional analog filters.

The sample code is given here.

```
#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 Conclusion

EMC design tips are illustrated in this application note to help customers apply EMC considerations in the early design phase using Kinetis E MCU.

Detailed descriptions on hardware and software techniques are listed as a quick reference for customer to adapt a Freescale solution more effectively.

7 References

These documents are available on freescale.com.

1. AN4438: *EMC Design Considerations for MC9S08PT60* by T.C. Lun., 2012.
2. AN4476: *System Design Guideline for 5V 8-bit families in Home Appliance Applications*, by T.C. Lun, Dennis Lui, 2012.
3. AN4463: *How to Develop a Robust Software in Noise Environment*, by Dennis Lui, T.C. Lun, 2012.

4. AN2321: *Designing for Board Level Electromagnetic Compatibility*, by T.C. Lun, 2002.
5. AN2764: *Improving the Transient Immunity Performance of Microcontroller-Based Applications*, by Ross Carlton, Greg Racino, and John Suchyta, 2005.

Document Number AN4779
Revision 0, August 1, 2013