Document Number: AN4463
Rev. 0, 03/2012

How To Develop a Robust Software in Noise Environment

by: Dennis Lui, T.C. Lun
System and Application, Microcontroller Solutions Group

Contents

1 Introduction

This application note describes how to implement a robust software when using MCU device of S08PT60, FL16 and AC60 in applications with heavy noise interferences. In general cases, the software design cannot change the physical media which couples the noise into the system, or reduce the absolute magnitude of noise generated from external sources. However, the software must be able to identify a particular event if it is a false alarm triggered by noise sources or it is a normal driven event and then make a smart decision on corresponding actions. For example, the software must not turn on a power control stage if there is any uncertainty on the requested action. Good defensive software design is one of the key factors to improve overall performance, system protection and operating stability in noise environments.

2 System Overview

A typical software design used for most of the microwave oven applications is illustrated here to show how to apply following software techniques in real case. Sample codes for device PT60, FL16 and AC60 are also included for each topic as a quick start guide to help user to adopt the techniques more quickly.

Following software technique is recommended for a good defensive software.

- Enable Watch-Dog to avoid code runaway
- Refresh data direction 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 re-confirm edge triggered event
- Enable slew rate control on output port
- Enable input glitch filter (PT60 build-in feature)

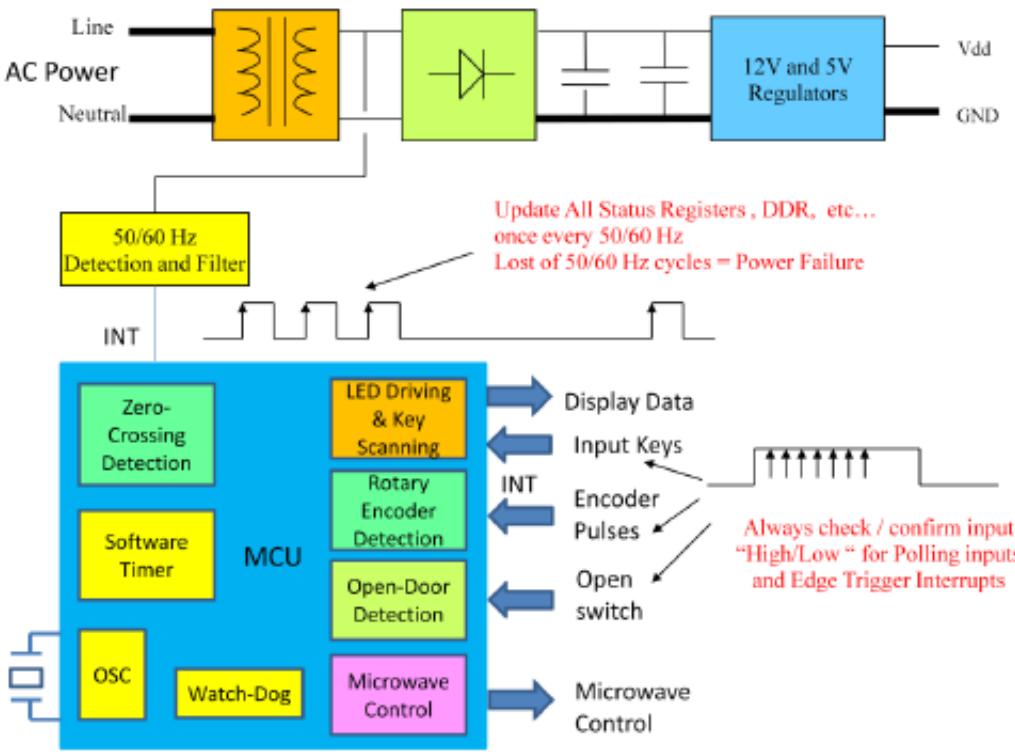

Figure 1. System Block Diagram

The system block diagram is shown in Figure 1 and the demo software is developed with basic requirements to enable the whole system working in normal condition and keep most of internal modules running in real application case. For example, multiplex I/O ports for LED display driving and input key scanning, configure the TPM module as a software timer to generate all timing information and enable input capture function for zero-crossing and rotary encoder detection. Detection of open-door error and loss of AC power are also implemented for safety protection. External crystal oscillator is selected as the reference clock source for the FLL module to generate the bus clock through the bus frequency divider. All unused pins are configured as output low to avoid high frequency noise coupling into the MCU.

3 Enable WATCHDOG

The Watch-Dog (WDOG) function forces a system reset, when the application software fails to execute as expected, for example, the running software jumps to an unexpected memory location or runs into an infinite loop when transient noise is injected into the MCU. To prevent a system reset from the WDOG timer when it is enabled, application software must reset the WDOG counter periodically. If the application program gets lost and fails to reset the WDOG counter before it times out, a system reset is generated to force the system back to a known starting point. It is recommended to put the WDOG refresh

routine in the main loop instead of sub-routines and interrupt routines. The WDOG function is initialized at the beginning of the main loop and the WDOG counter is reset periodically inside the forever loop. The WDOG reset code is defined by the macro __RESET_WATCHDOG().

3.1 PT60 Sample code

The sample code for PT60 is given here. There is a new feature for WDOG function in PT60 which allows the user to re-configure the parameters used by the module, so the initial setup code is slightly different from traditional setting. The module should be un-locked before doing any modification.

```
#define __RESET_WATCHDOG() (void)(WDOG_CNT = 0xA602U, WDOG_CNT = 0xB480U)

void main(void) {

Sys_Init();
PE_low_level_init();
MicrowaveInit();

wdog_unlock();    // executing an unlock sequence
WDOG_CS1 = 0xA0; // Set WDOGA = 1 to allow reconfigure watchdog
WDOG_CS2 = 0x01;// Select internal 1 kHz as watchdog clock
WDOG_TMRH = 0x03;// Set watchdog counter to 1000
WDOG_TMRL = 0xE8;
WDOG_WINH = 0;    // disable window mode option
WDOG_WINL = 0;

EnableInterrupts;

for(;;) {

DisableInterrupts;  // disable interrupts

__RESET_WATCHDOG(); // Reset the watchdog counter

EnableInterrupts;   // enable interrupt

MicrowaveTask();    // Application main task

} /* loop forever */
  /* please make sure that you never leave main */
} /* end of Main */
```

3.2 FL16 Sample code

The sample code for FL16 is given here.

```
#define __RESET_WATCHDOG() (SRS = 0x55,SRS = 0xAA,(byte)ERR_OK)
void main(void) {
PE_low_level_init();
MicrowaveInit();
for(;;) {
__RESET_WATCHDOG(); // Reset the watchdog counter
MicrowaveTask(); // Application main task
} /* loop forever */
/* please make sure that you never leave main */
sure that you never leave main */
} /* end of Main */
```

3.3 AC60 Sample code

The sample code for AC60 is given here.

```
#define __RESET_WATCHDOG() (SRS = 0x00,(byte)ERR_OK)
void main(void) {
PE_low_level_init();
MicrowaveInit();
for(;;) {
__RESET_WATCHDOG(); // Reset the watchdog counter
MicrowaveTask(); // Application main task
} /* loop forever */
/* please make sure that you never leave main */
} /* end of Main */
```

4 Refresh Data Direction 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. The 50Hz or 60Hz periodic signal output from AC power supply through an optical coupling circuit can be used as a trigger signal for I/O direction registers update. The variable mStatusRegisterUpdate_d is set to TRUE at each falling edge of the 50Hz or 60Hz signal and then clear to FALSE at the end of the refresh register routine.

4.1 PT60 Sample Code

The sample code for PT60 is given here. The data port direction, input or output, is controlled through the input enable or output enable registers. After reset, all parallel I/O default to the Hi-Z state. The corresponding bit in output enable register (PTxOE) or input enable register (PTxIE) must be configured for output or input operation. Each port pin has an input enable bit and an output enable bit. The output enable state for each data port pin is defined by the macro IOstatusRegPTxDD_Value and the value for input enable state is equal to the one's complement of this macro value.

The Port C is configured with input and output multiplex function and the directional status is updated in another key scanning routine.

```
void StatusRegisterUpdate(void){
if(mStatusRegisterUpdate_d == TRUE){
IOstatusRegPTAIE_Port = ~IOstatusRegPTADD_Value;
IOstatusRegPTAOE_Port = IOstatusRegPTADD_Value;
IOstatusRegPTBIE_Port = ~IOstatusRegPTBDD_Value;
IOstatusRegPTBOE_Port = IOstatusRegPTBDD_Value;
/* Port C is used as Input and Output port and refresh
by key scanning routine
*/
// IOstatusRegPTCIE_Port = ~IOstatusRegPTCDD_Value;
// IOstatusRegPTCOE_Port = IOstatusRegPTCDD_Value;
IOstatusRegPTDIE_Port = ~IOstatusRegPTDDD_Value;
IOstatusRegPTDOE_Port = IOstatusRegPTDDD_Value;
IOstatusRegPTEIE_Port = ~IOstatusRegPTEDD_Value;
IOstatusRegPTEOE_Port = IOstatusRegPTEDD_Value;
IOstatusRegPTFIE_Port = ~IOstatusRegPTFDD_Value;
IOstatusRegPTFOE_Port = IOstatusRegPTFDD_Value;
IOstatusRegPTGIE_Port = ~IOstatusRegPTGDD_Value;
IOstatusRegPTGOE_Port = IOstatusRegPTGDD_Value;
IOstatusRegPTHIE_Port = ~IOstatusRegPTHDD_Value;
IOstatusRegPTHOE_Port = IOstatusRegPTHDD_Value;
mStatusRegisterUpdate_d = FALSE;
}
}
```

4.2 FL16 Sample Code

The sample code for FL16 is given here. The data port direction, input or output, is controlled through the data direction registers and each port pin has a register bit to control corresponding pin is an input or output. The data direction state for each data port pin is defined by the macro IOstatusRegPTxDD_Value.

The Port C is configured with input and output multiplex function and the directional status is updated in another key scanning routine.

```
void StatusRegisterUpdate(void){
if(mStatusRegisterUpdate_d == TRUE){
IOstatusRegPTADD_Port = IOstatusRegPTADD_Value;
IOstatusRegPTBDD_Port = IOstatusRegPTBDD_Value;
/* Port C is used as Input and Output port and refresh
by key scanning routine
*/
// IOstatusRegPTCDD_Port = IOstatusRegPTCDD_Value
IOstatusRegPTDDD_Port = IOstatusRegPTDDD_Value;
mStatusRegisterUpdate_d = FALSE;
}
}
```

4.3 AC60 Sample Code

The sample code for AC60 is given here. The data port direction control scheme is similar to FL16.

The Port E is configured with input and output multiplex function and the directional status is updated in another key scanning routine.

```
void StatusRegisterUpdate(void){
if(mStatusRegisterUpdate_d == TRUE){
IOstatusRegPTADD_Port = IOstatusRegPTADD_Value;
IOstatusRegPTBDD_Port = IOstatusRegPTBDD_Value;
IOstatusRegPTCDD_Port = IOstatusRegPTCDD_Value;
IOstatusRegPTDDD_Port = IOstatusRegPTDDD_Value;
/* Port E is used as Input and Output port and refresh
by key scanning routine
// IOstatusRegPTEDD_Port = IOstatusRegPTEDD_Value;
IOstatusRegPTFDD_Port = IOstatusRegPTFDD_Value;
IOstatusRegPTGDD_Port = IOstatusRegPTGDD_Value;
mStatusRegisterUpdate_d = FALSE;
}
}
```

5 Fill Unused Memory

Unused memory, Flash or RAM should be filled with a pre-defined content such that the MCU does not execute any unexpected instruction when the normal execution flow is disturbed by external noise sources. It is recommended to fill all unused memory with No Operation (NOP) instruction. The MCU will not execute any command when the instruction pointer incorrectly points to those unused memory. The system will be reset by the WDOG function and forced back to the original starting point.

The unused memory can be filled by adding the FILL option in the linker parameter file (for example, Project.prm). The option FILL 0x9D instructs the linker to initialized the memory content with 0x9D which is equal to the object code value for a NOP instruction.

5.1 PT60 Sample Code

The sample code for PT60 is given here.

```
/* This is a linker parameter file for the mc9s08pt60 */
/* CodeWarrior will pass all the needed files to the linker by command line. But here you
may add your own files too. */
NAMES END
/* Here all RAM/ROM areas of the device are listed. Used in PLACEMENT below. */
SEGMENTS
Z_RAM = READ_WRITE 0x0040 TO 0x00FF FILL 0x9D;
RAM = READ_WRITE 0x0100 TO 0x0BCF FILL 0x9D;
RAM_CODE = READ_ONLY 0x0BD0 TO 0x103F FILL 0x9D;
ROM = READ_ONLY 0x3200 TO 0xF5FF FILL 0x9D;
ROM1 = READ_ONLY 0x1040 TO 0x2FFF FILL 0x9D;
ROM2 = READ_ONLY 0xFF80 TO 0xFFAF FILL 0x9D;
EEPROM = READ_ONLY 0x3100 TO 0x31FF FILL 0x9D;
FLASH_TO_RAM = READ_ONLY 0xF600 TO 0xFBFF RELOCATE_TO 0x0BD0;
USER_PARAM = READ_ONLY 0xFC00 TO 0xFDFF;
/* INTVECTS = READ_ONLY 0xFFB0 TO 0xFFFF; Reserved for Interrupt Vectors */
END
```

5.2 FL16 Sample Code

The sample code for FL16 is given here.

```
/* This is a linker parameter file for the mc9s08fl16 */
/* CodeWarrior will pass all the needed files to the linker by command line. But here you
may add your own files too. */
NAMES
END
SECTIONS
Z_RAM = READ_WRITE 0x0040 TO 0x005F FILL 0x9D;
RAM = READ_WRITE 0x0060 TO 0x043F FILL 0x9D;
ROM = READ_ONLY 0xC000 TO 0xFFAD FILL 0x9D;
END
PLACEMENT
DEFAULT_RAM, /* non-zero page variables */
INTO RAM;
DEFAULT_ROM, ROM_VAR, STRINGS INTO ROM;
_DATA_ZEROPAGE, /* zero page variables */
MY_ZEROPAGE INTO Z_RAM;
END
INIT _EntryPoint /* The entry point of the application. */
STACKSIZE 0x0080 /* Size of the system stack. */
```

5.3 AC60 Sample Code

The sample code for AC60 is given here.

```
/* This is a linker parameter file for the mc9s08AC60 */
/* CodeWarrior will pass all the needed files to the linker by command line. But here you
may add your own files too. */
NAMES
END
SECTIONS
Z_RAM = READ_WRITE 0x0070 TO 0x008F FILL 0x9D;
RAM = READ_WRITE 0x0090 TO 0x086F FILL 0x9D;
ROM = READ_ONLY 0x1860 TO 0xFFAF FILL 0x9D;
```

```
END
PLACEMENT
DEFAULT_RAM, /* non-zero page variables */
INTO RAM;
DEFAULT_ROM, ROM_VAR, STRINGS INTO ROM;
_DATA_ZEROPAGE, /* zero page variables */
MY_ZEROPAGE INTO Z_RAM;
END
INIT _EntryPoint /* The entry point of the application. */
STACKSIZE 0x0080 /* Size of the system stack. */
```

6 Define All Interrupt Vectors

The reason to define the interrupt vectors for each unused interrupt function is to allow the MCU to jump into a pre-defined interrupt routine and back to previous execution step correctly when a particular unused interrupt associate flag is mis-triggered by a noise source. The interrupt function for each unused interrupt can be the same, so just one dummy interrupt routine can be used for all unused interrupt functions.

The interrupt routine for each interrupt function is assigned in the interrupt vector table as given in following sample code.

6.1 PT60 Sample Code

The sample code for PT60 is shown here.

```
void (* near const _vect[])(void) @0xFFB0 = { /* Interrupt vector table */
Cpu_Interrupt, /* Int.no. 39 Vnvm (at FFB0) Unassigned */
Cpu_Interrupt, /* Int.no. 38 Vkbi1 (at FFB2) Unassigned */
Cpu_Interrupt, /* Int.no. 37 Vkbi0 (at FFB4) Unassigned */
Cpu_Interrupt, /* Int.no. 36 Vtsi (at FFB6) Unassigned */
Cpu_Interrupt, /* Int.no. 35 Vrtc (at FFB8) Unassigned */
Cpu_Interrupt, /* Int.no. 34 Viic (at FFBA) Unassigned */
Cpu_Interrupt, /* Int.no. 33 Vspi1 (at FFBC) Unassigned */
Cpu_Interrupt, /* Int.no. 32 Vspi0 (at FFBE) Unassigned */
Cpu_Interrupt, /* Int.no. 31 Vsci2txd (at FFC0) Unassigned */
Cpu_Interrupt, /* Int.no. 30 Vsci2rxd (at FFC2) Unassigned */
Cpu_Interrupt, /* Int.no. 29 Vsci2err (at FFC4) Unassigned */
Cpu_Interrupt, /* Int.no. 28 Vsci1txd (at FFC6) Unassigned */
Cpu_Interrupt, /* Int.no. 27 Vsci1rxd (at FFC8) Unassigned */
Cpu_Interrupt, /* Int.no. 26 Vsci1err (at FFCA) Unassigned */
Cpu_Interrupt, /* Int.no. 25 Vsci0txd (at FFCC) Unassigned */
Cpu_Interrupt, /* Int.no. 24 Vsci0rxd (at FFCE) Unassigned */
Cpu_Interrupt, /* Int.no. 23 Vsci0err (at FFD0) Unassigned */
Cpu_Interrupt, /* Int.no. 22 Vadc (at FFD2) Unassigned */
Cpu_Interrupt, /* Int.no. 21 Vacmp (at FFD4) Unassigned */
Cpu_Interrupt, /* Int.no. 20 Vmtim1 (at FFD6) Unassigned */
Cpu_Interrupt, /* Int.no. 19 Vmtim0 (at FFD8) Unassigned */
Cpu_Interrupt, /* Int.no. 18 Vftm0ovf (at FFDA) Unassigned */
Vftm0ch1_Interrupt /* Int.no. 17 Vftm0ch1 (at FFDC Used */
Vftm0ch0_Interrupt /* Int.no. 16 Vftm0ch0 (at FFDE) Used */
Cpu_Interrupt /* Int.no. 15 Vftm1ovf (at FFE Unassigned */
Cpu_Interrupt /* Int.no. 14 Vftm1ch1 (at FF Unassigned */
Cpu_Interrupt, /* Int.no. 13 Vftm1ch0 (at FFE4) Unassigned */
Cpu_Interrupt, /* Int.no. 12 Vftm2ovf (at FFE6) Unassigned */
Cpu_Interrupt, /* Int.no. 11 Vftm2ch5 (at FFE8) Unassigned */
Cpu_Interrupt, /* Int.no. 10 Vftm2ch4 (at FFEA) Unassigned */
Vftm2ch3_Interrupt, /* Int.no. 9 Vftm2ch3 (at FFEC) Used */
Vftm2ch2_Interrupt, /* Int.no. 8 Vftm2ch2 (at FFEE) Used */
Cpu_Interrupt, /* Int.no. 7 Vftm2ch1 (at FFF0) Unassigned */
Cpu_Interrupt, /* Int.no. 6 Vftm2ch0 (at FFF2) Unassigned */
Cpu_Interrupt, /* Int.no. 5 Vftm2flt (at FFF4) Unassigned */
Cpu_Interrupt, /* Int.no. 4 Vclk (at FFF6) Unassigned */
Cpu_Interrupt, /* Int.no. 3 Vlvd (at FFF8) Unassigned */
```

```
Cpu_Interrupt /* Int.no. 2 VirgVwdog (at FFFA) Unassigned */
Cpu_Interrupt, /* Int.no. 1 Vswi (at FFFC) Unassigned */
_Startup /* Int.no. 0 Vreset (at FFFE) Reset vector */

};
```

6.2 FL16 Sample Code

The sample code for FL16 is given here.

```
extern near void _EntryPoint(void);
void (* near const _vect[])(void) @0xFFD2 = { /* Interrupt vector table */
Cpu_Interrupt, /* Int.no. 22 Vscitx (at FFD2) Unassigned */
Cpu_Interrupt, /* Int.no. 21 Vscirx (at FFD4) Unassigned */
Cpu_Interrupt, /* Int.no. 20 Vscierr (at FFD6) Unassigned */
Cpu_Interrupt, /* Int.no. 19 VReserved19 (at FFD8) Unassigned */
Cpu_Interrupt, /* Int.no. 18 VReserved18 (at FFDA) Unassigned */
Cpu_Interrupt, /* Int.no. 17 Vadc (at FFDC) Unassigned */
Cpu_Interrupt, /* Int.no. 16 Vtpm2ovf (at FFDE) Unassigned */
Encoder1_Interrupt, /* Int.no. 15 Vtpm2ch1 (at FFE0) Used */
Encoder2_Interrupt, /* Int.no. 14 Vtpm2ch0 (at FFE2) Used */
Cpu_Interrupt, /* Int.no. 13 Vtpm1ovf (at FFE4) Unassigned */
Cpu_Interrupt, /* Int.no. 12 VReserved12 (at FFE6) Unassigned */
Cpu_Interrupt, /* Int.no. 11 VReserved11 (at FFE8) Unassigned */
Zero_Interrupt, /* Int.no. 10 Vtpm1ch3 (at FFEA) Used */
SoftTimer_Interrupt, /* Int.no. 9 Vtpm1ch2 (at FFEC) Used */
Cpu_Interrupt, /* Int.no. 8 Vtpm1ch1 (at FFEE) Unassigned */
Cpu_Interrupt, /* Int.no. 7 Vtpm1ch0 (at FFF0) Unassigned */
Cpu_Interrupt, /* Int.no. 6 Vmtim (at FFF2) Unassigned */
Cpu_Interrupt, /* Int.no. 5 VReserved5 (at FFF4) Unassigned */
Cpu_Interrupt, /* Int.no. 4 VReserved4 (at FFF6) Unassigned */
Cpu_Interrupt, /* Int.no. 3 Vlvd (at FFF8) Unassigned */
Cpu_Interrupt, /* Int.no. 2 Virq (at FFFA) Unassigned */
Cpu_Interrupt, /* Int.no. 1 Vswi (at FFFC) Unassigned */
_EntryPoint /* Int.no. 0 Vreset (at FFFE) Reset vector */
};
```

6.3 AC60 Sample Code

The sample code for AC60 is given here.

```
extern near void _EntryPoint(void);
void (* near const _vect[])(void) @0xFFC6 = { /* Interrupt vector table */
Cpu_Interrupt, /* Int.no. 28 Vtpm3ovf (at FFC6) Unassigned */
Cpu_Interrupt, /* Int.no. 27 Vtpm3ch1 (at FFC8) Unassigned */
Cpu_Interrupt, /* Int.no. 26 Vtpm3ch0 (at FFCA) Unassigned */
Cpu_Interrupt, /* Int.no. 25 Vrti (at FFCC) Unassigned */
Cpu_Interrupt, /* Int.no. 24 Viic1 (at FFCE) Unassigned */
Cpu_Interrupt, /* Int.no. 23 Vadc1 (at FFD0) Unassigned */
Cpu_Interrupt, /* Int.no. 22 Vkeyboard1 (at FFD2) Unassigned */
Cpu_Interrupt, /* Int.no. 21 Vsci2tx (at FFD4) Unassigned */
Cpu_Interrupt, /* Int.no. 20 Vsci2rx (at FFD6) Unassigned */
Cpu_Interrupt, /* Int.no. 19 Vsci2err (at FFD8) Unassigned */
Cpu_Interrupt, /* Int.no. 18 Vsci1tx (at FFDA) Unassigned */
Cpu_Interrupt, /* Int.no. 17 Vsci1rx (at FFDC) Unassigned */
Cpu_Interrupt, /* Int.no. 16 Vsci1err (at FFDE) Unassigned */
Cpu_Interrupt, /* Int.no. 15 Vspi1 (at FFE0) Unassigned */
Cpu_Interrupt, /* Int.no. 14 Vtpm2ovf (at FFE2) Unassigned */
Encoder1_Interrupt, /* Int.no. 13 Vtpm2ch1 (at FFE4) Used */
Encoder2_Interrupt, /* Int.no. 12 Vtpm2ch0 (at FFE6) Used */
Cpu_Interrupt, /* Int.no. 11 Vtpm1ovf (at FFE8) Unassigned */
Cpu_Interrupt, /* Int.no. 10 VReserved10 (at FFEA) Unassigned */
```

```
Cpu_Interrupt, /* Int.no. 9 VReserved9 (at FFEC) Unassigned */
Zero_Interrupt, /* Int.no. 8 Vtpm1ch3 (at FFEE) Used */
SoftTimer_Interrupt, /* Int.no. 7 Vtpm1ch2 (at FFF0) Used */
Cpu_Interrupt, /* Int.no. 6 Vtpm1ch1 (at FFF2) Unassigned */
Cpu_Interrupt, /* Int.no. 5 Vtpm1ch0 (at FFF4) Unassigned */
Cpu_VicgInterrupt, /* Int.no. 4 Vicg (at FFF6) Used */
Cpu_Interrupt, /* Int.no. 3 Vlvd (at FFF8) Unassigned */
Cpu_Interrupt, /* Int.no. 2 Virq (at FFFA) Unassigned */
Cpu_Interrupt, /* Int.no. 1 Vswi (at FFFC) Unassigned */
_EntryPoint /* Int.no. 0 Vreset (at FFFE) Reset vector */
};
```

7 Select FLL Engaged Mode

It is recommended to enable the FLL engaged mode with external reference clock in the internal clock source (ICS) module which provides clock source option for the MCU. For example, a 4 MHz crystal oscillator is used as the reference clock and the target bus frequency is set to 4MHz. The 4MHz crystal clock frequency is divided by 128 to a 31.25 kHz low frequency clock by the reference divider first and then multiplied by 512 to 16MHz in the FLL module. The final bus clock is equal to FLL output divided by bus frequency divider, so the bus clock is 4MHz when the divider is 4.

The advantages of the frequency conversion using the FLL module instead of directly using the external crystal oscillator as the bus clock are:

- The impact of transient noise glitch on high frequency clock source (direct using crystal oscillator) is more significant compared to a low frequency clock source (divided by 128) 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 kind of short pulse noise due to the low-pass filter characteristic.

7.1 PT60 Sample Code

The sample code for PT60 is given here.

```
#define EXT_CLK_CRYST 4000 /* Use 4MHz crystal*/
#define BUS_CLK_HZ 4000000L /* set bus clock to 4MHz */
void FEI_to_FEE(void)
{
/* assume external crystal is 8Mhz or 4MHz
enable OSC high range and select oscillator output as OSCOUT
*/
ICS_OSCSC |= ICS_OSCSC_OSCEN_MASK
#if defined(CRYST_HIGH_GAIN)
| ICS_OSCSC_HGO_MASK /* Rs must be added and be large up to 200K */
12
#endif
| ICS_OSCSC_RANGE_MASK
| ICS_OSCSC_OSCOS_MASK;
asm{
nop
nop
}
/* wait for OSC to be initialized
*
*/
while(!(ICS_OSCSC & ICS_OSCSC_OSCINIT_MASK));
/* divide down external clock frequency to be within 31.25K to 39.0625K
*
*/
#if (EXT_CLK_CRYST == 8000)
/* 8MHz */
ICS_C1_RDIV = 3; /* now the divided frequency is 8000/256 = 31.25K */
```

```
#elif (EXT_CLK_CRYST == 4000)
/* 4MHz */
ICS_C1_RDIV = 2; /* now the divided frequency is 4000/128 = 31.25K */
#else
#error "Error: crystal value not supported!\n";
#endif
/* change FLL reference clock to external clock */
ICS_C1_IREFS = 0;
/* wait for the reference clock to be changed to external */
asm{
nop
nop
}
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 = 16MHz
*
*/
#if (BUS_CLK_HZ == 16000000)
// now system/bus clock is 16MHz
ICS_C2_BDIV = 0;
#elif (BUS_CLK_HZ == 4000000)
// ICS_C2_BDIV = 1; // now system/bus clock is 8MHz
ICS_C2_BDIV = 2; // now system/bus clock is 4MHz
#endif
/* clear Loss of lock sticky bit */
ICS_S |= ICS_S_LOLS_MASK;
}
```

7.2 FL16 Sample Code

The sample code for FL16 is given here.

```
/* System clock initialization */
/* Test if the device trim value is stored on the specified address */
if (*(unsigned char*)0xFFAF != 0xFF) {
/* Initialize ICSTRM register from a non volatile memory */
ICSTRM = *(unsigned char*)0xFFAF;
/* Initialize ICSSC register from a non volatile memory */
ICSSC = (unsigned char)((*(unsigned char*)0xFFAE) & (unsigned char)0x01);
}
/* ICSC1: CLKS=0,RDIV=2,IREFS=0,IRCLKEN=1,IREFSTEN=0 */
setReg8(ICSC1, 0x12); /* Initialization of the ICS control register 1 */
/* ICSC2: BDIV=1,RANGE=1,HGO=1,LP=0,EREFS=1,ERCLKEN=1,EREFSTEN=0 */
setReg8(ICSC2, 0x76); /* Initialization of the ICS control register 2 */
/* Wait until the initialization of the ext. crystal oscillator is completed */
while(!ICSSC_OSCINIT) {
SRS = 0x55; /* Reset watchdog counter write 55, AA */
SRS = 0xAA;
}
/* ICSSC: DRST_DRS=0,DMX32=0 */
clrReg8Bits(ICSSC, 0xE0); /* Initialization of the ICS status and control */
/* Wait until the FLL switches to Low range DCO mode */
while((ICSSC & 0xC0) != 0x00) {
SRS = 0x55; /* Reset watchdog counter write 55, AA */
SRS = 0xAA;
}
```

7.3 AC60 Sample Code

The sample code for AC60 is given here. The 4MHz crystal clock frequency is multiplied by factor of 4 to a 16Mhz frequency clock in the FLL module. The final bus clock is equal to FLL output divided by the reduced frequency divider plus a fixed divider (by 2), so the bus clock is 4MHz when the whole division factor is 4.

There is no reference clock divider when external crystal source is selected as reference clock for FLL module, so the crystal frequency is not divided down before the multiplication process in the FLL.

```
/* System clock initialization */
/* ICGC1: HGO=1,RANGE=1,REFS=1,CLKS1=1,CLKS0=1,OSCSTEN=1,LOCD=0,??=0 */
setReg8(ICGC1, 0xFC); // Select FEE mode, Ext Xtal =4MHz
/* ICGC2: LOLRE=0,MFD2=0,MFD1=0,MFD0=0,LOCRE=0,RFD2=0,RFD1=0,RFD0=1 */
setReg8(ICGC2, 0x01); // Multiplication factor = 4, Divison factor = 2
if (*(unsigned char*)0xFFBE != 0xFF) { /* Test if the device trim value is stored on the
specified address */
ICGTRM = *(unsigned char*)0xFFBE; /* Initialize ICGTRM register from a non volatile memory */
}
while(!ICGS1_ERCS) { /* Wait until external reference is not stable */
}
```

8 Re-Confirm Edge Triggered

Multiple reading on input data for each edge triggered interrupt service is almost an essential technique to confirm if the input event is valid and driven by determined sources. The zero-crossing detection routine given here is one of the basic function for most of home appliance applications to detect the zero-crossing point in AC power system. The voltage level at zero-crossing point is almost equal to zero, the controller is allowed to switch on or off any power control stage with minimum loading on the power system.

The 50Hz or 60Hz signal coupling from the AC power line is applied to a timer input capture pin in MCU and the interrupt service routine will be called for each falling edge detected at this pin. Multiple reading of the pin status is defined in a “for loop” to ensure all readings from the pin status are the same before the TRUE flag is set up for this event.

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

8.1 PT60 / FL16 / AC60 Sample Code

Same sample code for PT60 / FL16 / AC60 and is given here.

```
/* Random Delay Loop */
uint8_t RandomDelay(void){
uint16_t random_16bit = RANDOM_COUNTER;
mRandomDelayCount = TPMxCnVLvalue(random_16bit);
mRandomDelayCount &= gRandomDelayCountMask_c;
return mRandomDelayCount;
}
void Zero_OnCapture(void)
{
/* Write your code here ... */
uint8_t ZeroTriggerDebounce[gNumberOfZeroTriggerDebounce_c];
uint8_t iRead = 0;
for (iRead = 0; iRead < gNumberOfZeroTriggerDebounce_c; iRead++){
uint8_t idelay;
idelay = RandomDelay();
while(idelay > 0){
--idelay;
```

```
asm(nop);
}
ZeroTriggerDebounce[iRead] = ZeroTrigger_GetPinValue();
if (iRead !=0){
if((ZeroTriggerDebounce[iRead] != ZeroTriggerStateIdle) &&
(ZeroTriggerDebounce[iRead] == ZeroTriggerDebounce[iRead - 1])){
mZeroTriggerRead = ZeroTriggerDebounce[iRead];
}else {
mZeroTriggerRead = ZeroTriggerStateIdle;
}
}
}
if (mZeroTriggerRead == ZeroTriggerStateActive){
mZeroTriggerVaild_d = TRUE;
}else{
mZeroTriggerVaild_d = FALSE;
}
/* Action for Vaild Zero Trigger */
}
```

9 Input Glitch Filter (PT60 Build-in Feature)

The Input Glitch Filter is a new feature in PT60 MCU that provides, a simple low-pass filter for each port pin that is configured as a digital input. The glitch width threshold can be easily adjusted by setting registers PORT_IOFLTn and PORT_FCLKDIV between 1~4096 BUSCLKs (or 1~128 LPOCLKs). Glitches that are shorter than the selected clock width will be filtered out; glitches that are more than twice the selected clock width will be allowed to pass to the internal circuitry.

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. This feature is not available in FL16 and AC60.

9.1 PT60 Sample Code

The sample code for PT60 is given here.

```
#ifdef PortFilterEnable
setReg8Bits(PORT_FCLKDIV, 0x02); // set FLTDIV1 to Bus clock divided by 8
setReg8Bits(PORT_IOFLT0, 0x55); // select FLTDIV1 for Port A/B/C/D
setReg8Bits(PORT_IOFLT1, 0x55); // select FLTDIV1 for Port E/F/G/H
#endif
```

10 Slew Rate Control

Slew rate control can be enabled for each port pin by setting the corresponding bit in one of the slew rate control registers (PTxSEn). When enabled, slew control limits the rate at which an output can transition. This reduces EMC emissions. Slew rate control has no effect on pins which are configured as inputs. This feature is not available in PT60.

10.1 FL16 Sample Code

The sample code for FL16 is given here.

```
/* PTASE: PTASE7=1,PTASE6=1,PTASE4=1,PTASE3=1,PTASE2=1,PTASE1=1,PTASE0=1 */
setReg8Bits(PTASE, 0xDF);
/* PTBSE: PTBSE7=1,PTBSE6=1,PTBSE5=1,PTBSE4=1
```

```
PTBSE3=1,PTBSE2=1,PTBSE1=1,PTBSE0=1 */
setReg8(PTBSE, 0xFF);
/* PTCSE: PTCSE7=1,PTCSE6=1,PTCSE5=1,PTCSE4=1
PTCSE3=1,PTCSE2=1,PTCSE1=1,PTCSE0=1 */
setReg8(PTCSE, 0xFF);
/* PTDSE: PTDSE5=1,PTDSE4=1,PTDSE3=1,PTDSE2=1,PTDSE1=1,PTDSE0=1 */
setReg8Bits(PTDSE, 0x3F);
```

10.2 AC60 Sample Code

The sample code for AC60 is given here.

```
/* PTASE: PTASE1=1,PTASE0=1 */
setReg8Bits(PTASE, 0x03);
/* PTBSE: PTBSE3=1,PTBSE2=1,PTBSE1=1,PTBSE0=1 */
setReg8Bits(PTBSE, 0x0F);
/* PTCSE: PTCSE5=1,PTCSE4=1,PTCSE3=1,PTCSE2=1,PTCSE1=1,PTCSE0=1 */
setReg8Bits(PTCSE, 0x3F);
/* PTDSE: PTDSE3=1,PTDSE2=1,PTDSE1=1,PTDSE0=1 */
setReg8Bits(PTDSE, 0x0F);
/* PTESE:PTESE7=1,PTESE6=1,PTESE5=1,PTESE4=1,PTESE3=1,PTESE2=1,PTESE1=1,PTESE0=1 */
setReg8(PTESE, 0xFF);
/* PTFSE: PTFSE5=1,PTFSE4=1,PTFSE1=1,PTFSE0=1 */
setReg8Bits(PTFSE, 0x33);
/* PTGSE: PTGSE6=1,PTGSE5=1,PTGSE3=1,PTGSE2=1,PTGSE1=1,PTGSE0=1 */
setReg8Bits(PTGSE, 0x6F);
```

11 Conclusion

This application note provides guidelines for developing software to be used in noisy environments. The software techniques for overall system stability enhancement are demonstrated with PT60, FL16 and AC60 devices as examples, to help customer to design in Freescale MCU quickly.

12 References

- Ott, H., “Noise Reduction Techniques in Electronic Systems. New York: Wiley, 1976.
- Ott, H., “Digital Circuit Grounding and Interconnection,” Proceedings of the IEEE Symposium on Electromagnetic Compatibility, pp. 292-297, Aug. 1981.
- C.R. Paul., “Introduction to Electromagnetic Compatibility,” John Wiley Interscience, 1992.
- Martin O,Hara., “EMC at Component and PCB Level,” Newnes, 1998.
- Keith Armstrong., “PCB Design Techniques for Lowest-Cost EMC Compliance, Part1,” Electronics and Communication Engineering Journal, August 1999.
- Mark I. Montrose., “Printed Circuit Board Techniques for EMC Compliance,” IEEE press series 2000.
- T,C, Lun., “Designing for Board Level Electromagnetic Compatibility,” Freescale Application Note, AN2321. 2002.
- Ross Carlton, Greg Racino, John Suchyta., “Improving the Transient Immunity Performance of Microcontroller-Based Applications,” Freescale Application Note, AN2764. 2005.

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

Document Number: AN4463
Rev. 0, 03/2012