Application Report
SLAA609A – September 2013 – Revised January 2015

Adjustment of ESIOSC Oscillator Frequency

Christian Kurz, Johann Zipperer MSP430

ABSTRACT

The MSP430FR698x Extended Scan Interface (ESI) uses two clock sources. These clocks are ACLK and a high-frequency clock generated by the ESI oscillator, ESIOSC. ESIOSC is realized as an RC-oscillator and shows a temperature and voltage dependency. However, a hardware-supported measurement of ESIOSC frequency and adjustment by software allows compensating for the frequency drift. This application report describes algorithms that enable initial ESIOSC frequency adjustments and compensate for frequency drifts caused by temperature and voltage changes during runtime.

Contents

List of Figures

1 Introduction

The Extended Scan Interface (ESI) is a peripheral module that performs user-defined measurement sequences and processes the measurement results with a programmable state machine. For example, a typical use case is rotation detection and rotation counting in applications like flow meters.

The ESI module consist of different blocks, and the most important is the timing state machine (TSM). There are 32 TSM control registers that are sequentially processed when a start trigger is seen. All of those control registers are identical, and their control bits define the settings for the analog front ends and the time period of each TSM register in use. The time period definition is especially important, because it allows adjustment of settle times or measurement interval times for the selected sensor solution. An accurate setting of the time periods is necessary, because the minimum settle times should be met and the time should not be too long because this increases the current consumption.

The time periods are defined with the TSM register by selecting a clock source, either ACLK or the high-frequency clock ESIOSC, and the number of clock cycles of that clock. The ESIOSC source is mainly chosen for precise settle time definition. For this reason, an accurate ESIOSC frequency is needed.

Hardware integrated in the ESI module allows measuring the ESIOSC frequency and adjusting it by software. This allows an initial setting of ESIOSC after starting the application or an ESIOSC recalibration during runtime.

The ESIOSC adjustment routine is executed only once after power-up, if the ESIOSC frequency drift caused by temperature and supply voltage changes is acceptable. However, such a software routine can also be periodically executed during the entire product lifetime. The period of running the calibration cycle must be defined by the application requirements; for example, frequency accuracy when affected by temperature changes and supply voltage changes.

2 ESIOSC Frequency Measurement

The internal clock generator, ESIOSC, allows ESI to operate independent from the microcontroller's SMCLK clock source. The frequency of the ESIOSC varies between individual units and drifts with temperature and supply voltage. The six ESICLKFQx control bits in the ESIOSC control register are used to adjust the ESIOSC frequency.

The ESIOSC frequency can be measured and adjusted by a software routine. The 8-bit wide counter ESICNT3 is used for measurement. Setting the control bits ESIOSC.ESIHFSEL and ESIOSC.ESICLKGON resets ESICNT3. Beginning with the second rising edge of ACLK, the ESIOSC clock cycles are counted for one complete ACLK period. Reading ESICNT3 while this measurement is ongoing always results in reading a 0x01. Therefore this is used as abort criteria in the code.

Figure 1 shows the measurement sequence. ESIOSC oscillator is off before the measurement starts. Setting the ESIHFSEL and ESICLKGON bits causes the ESIOSC oscillator to start, thus clearing the ESICNT3 counter, and starts the measurement one ACLK cycle later. The measurement itself, counts the number of ESIOSC cycles, starts within one complete ACLK cycle.

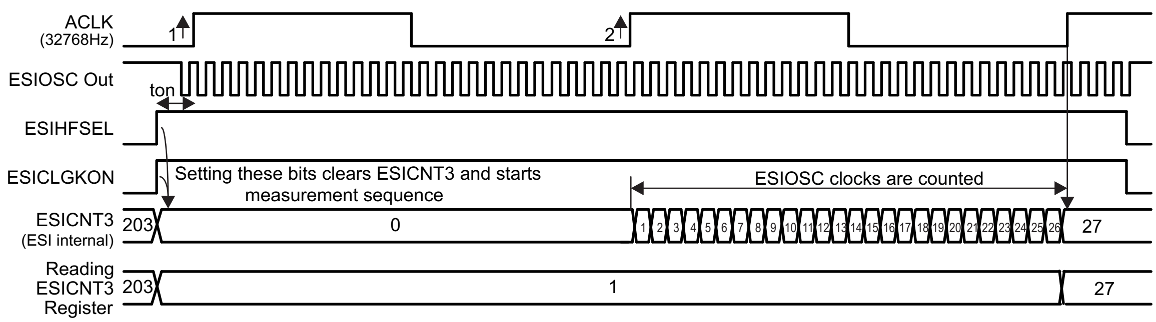


Figure 1. ESIOSC Measurement Sequence Timing Diagram Example

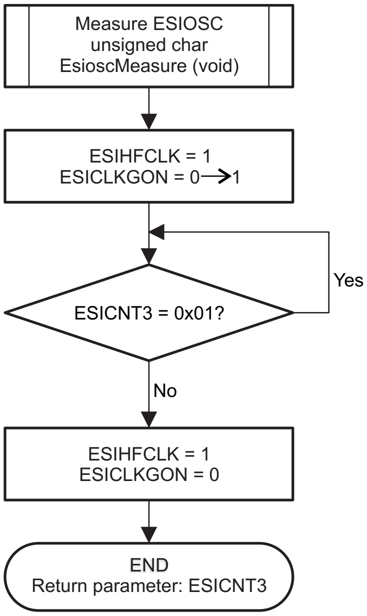
3 ESIOSC Measurement and Adjustment Software Functions

The following functions are used for measurement and adjustment of ESIOSC frequency. These API functions are derived from silicon test routines and may be used for reference.

- *EsioscMeasure()* function; measurement of ESIOSC frequency: This function measured the ESIOSC frequency. This function is also used to check the ESIOSC frequency after running *EsioscInit()* or *EsioscReCal()* functions.
- *EsioscInit()* function; adjustment of ESIOSC after power up: This function adjusts the ESIOSC frequency close to the selected target frequency. It is used as an initial setup of ESIOSC oscillator. This function has to be called only once. After calibration, the ESIOSC frequency may change due to supply voltage and temperature dependency. If this frequency drift is acceptable, no further ESIOSC adjustment is needed. In case, temperature and supply voltage drift should be compensated the *EsioscReCal()* function may be used.
- *EsioscReCal()* function; adjustment of ESIOSC during runtime: This function is used for adjusting the oscillator gradually during runtime. Instead of doing several measurements and step-by-step adjustment of ESIOSC frequency, like it is done in the *EsioscInit()* function, the *EsioscReCal()* is doing one measurement and one adjustment step then it returns to the caller. Adjusting the target frequency setting may requires several calls. The return value of this function provides information to the caller about the status of the calibration routine.

3.1 Measurement of ESIOSC Frequency

The measurement function uses the ESI hardware as described in Section 2. Figure 2 shows the flowchart and the source code for ESIOSC frequency measurement.




```
unsigned char EsioscMeasure()
{  unsigned int temp;

   ESIOSC &= ~(ESICLKGON);
   ESIOSC |= ESICLKGON + ESIHFSEL; // Start measurement

   do{
      temp = ESICNT3;          // Get counter value
   } while(temp == 0x01);      // When measurement has not
                               // finished yet, ESICNT3 value
                               // is 0x01

   ESIOSC &= ~(ESICLKGON); // Stop ESIOSC oscillator

   return temp; // Return parameter is ESICNT3 counter value
}
```

Figure 2. Flowchart and Source Code for ESIOSC Frequency Measurement

The *EsioscMeasure()* function allows you to measure the ESIOSC frequency with an accuracy of typically ±1 count (ratio of ESIOSC/ACLK). External distortions, variations of the 32-kHz crystal oscillator frequency, and voltage or temperature changes during measurement affect the measurement result. Multiple function calls and averaging of the measurement results help to reduce the impact of sporadic distortions and improve measurement accuracy. Example 1 invokes *EsioscMeasure()* multiple times.

The execution time of the *EsioscMeasure()* function depends on when the function is invoked relative to the ACLK cycle. In the best case, *EsioscMeasure()* is invoked just before a rising ACLK edge, and the function takes two ACLK cycles. When *EsioscMeasure()* is started after a rising ACLK edge, the function takes three ACLK cycles.

Example 1. Example Code for Using EsioscMeasure() Function

```
#include "MSP430.h"
#include "ESI_ESIOSC.h"
void main(void)
{ unsigned char temp;
  WDTCTL = WDTPW | WDTHOLD;      // Stop watchdog timer

  // XT1 Setup
  PJSEL0 |= BIT4 + BIT5;

  CSCTL0_H = 0xA5;
  CSCTL1 = DCOFSEL_0;            // Set DCO= 1MHz
  CSCTL2 = SELA__LFXTCLK + SELS__DCOCLK + SELM__DCOCLK;
  CSCTL3 = DIVA__1 + DIVS__1 + DIVM__1;  // set all dividers
  CSCTL4 |= LFXTDRIVE_0;
  CSCTL4 &= ~LFXTOFF;

  do
  {    CSCTL5 &= ~LFXTOFFG;      // Clear XT1 fault flag
  SFRIFG1 &=  ~OFIFG;
  } while (SFRIFG1&OFIFG);       // Test oscillator fault flag

  temp = EsioscMeasure();        // ESIOSC frequency is measured
  temp = temp + EsioscMeasure(); // four times and average result
  temp = temp + EsioscMeasure(); // is calculated afterwards.
  temp = temp + EsioscMeasure(); //

  temp = temp /4;                // calculating average result

  while(1);                      // entire loop
}
```

3.2 Adjustment of ESIOSC After Power-Up

Figure 3 shows an adjustment algorithm that can be used for ESIOSC frequency adjustment after power-up.

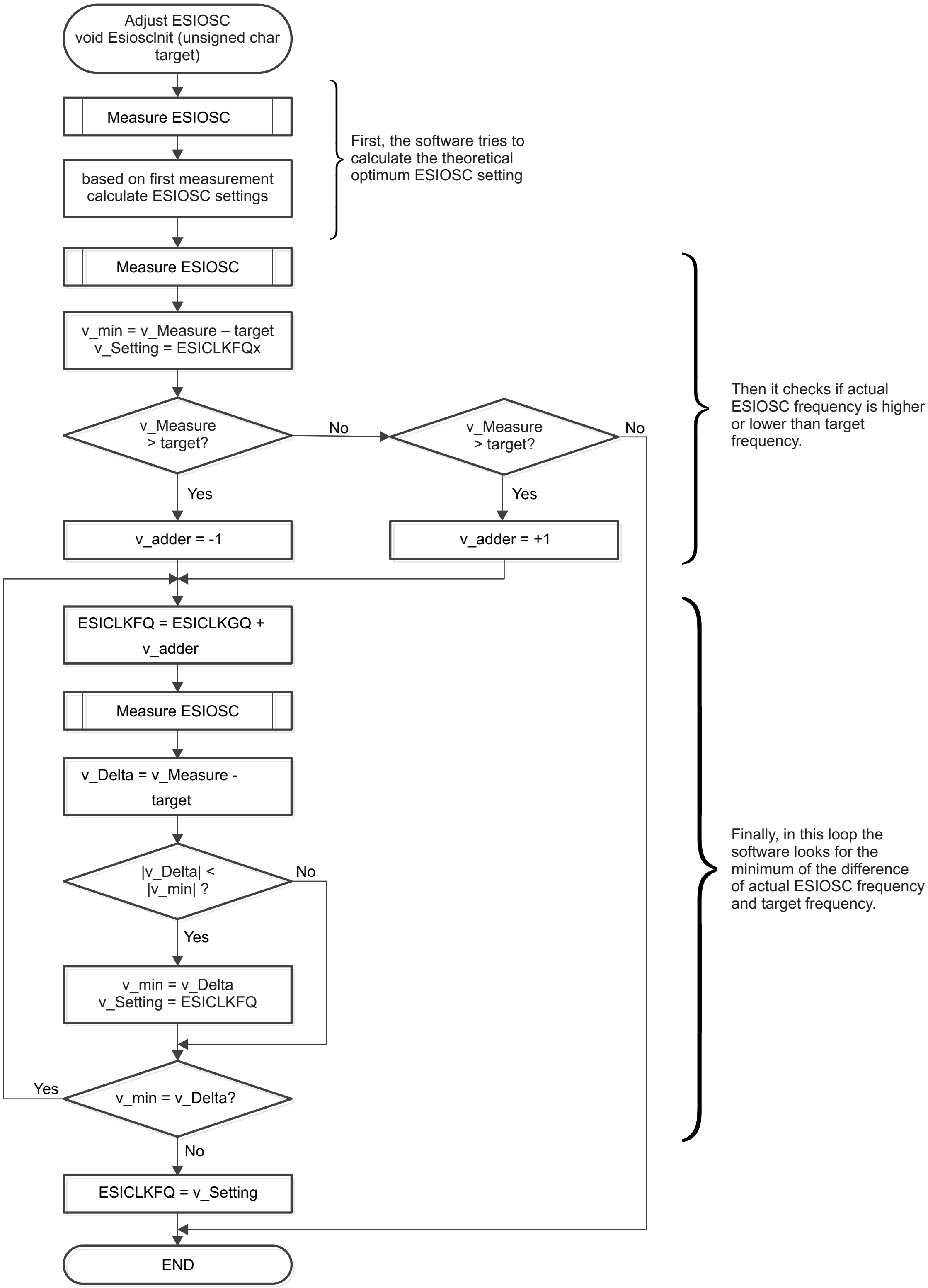


Figure 3. Flowchart for ESIOSC Adjustment After Power-On

The adjustment function consists of two steps. The first measurement is used to find out if the ESIOSC frequency is above or below the target frequency. Based on that determination, "v_adder" is set to -1 or +1. This variable is later used as and adjustment value for the ESIOSC frequency.

The second step is a loop that is repeatedly executed until a minimum difference between target frequency and ESIOSC frequency is found.

Note that *EsioscInit()* uses ACLK as reference clock. To ensure an accurate measurement it is important that ACLK is stable before calling the function (a 32-kHz crystal may require hundreds of ms).

The *EsioscInit()* function uses the argument *target*. This argument allows definition of the frequency ratio for adjustment. The adjusted ESIOSC frequency can be calculated with Equation 1:

$$\text{ESIOSC frequency} = \text{target} \times f_{ACLK} \tag{1}$$

Example for defining a target value:

$$f_{ACLK} = 32768\text{ Hz, Required ESIOSC frequency} = 4.8\text{ MHz} \tag{2}$$

Transpose Equation 1 to calculate the target value:

$$\text{target} = \frac{\text{ESIOSC frequency}}{32768\text{ Hz}} = \frac{4.8\text{ MHz}}{32768\text{Hz}} = 146.48 \tag{3}$$

target must be an integer value, and using 146 results in the following ESIOSC frequency:

$$\text{ESIOSC frequency} = 146 * 32768\text{ Hz} \approx 4.78\text{ MHz} \tag{4}$$

Checking for underflow and overflow can be done in software by checking the ESICLKFQx bits:

- (ESIOSC.ESICLKFQx AND 0x3F00) = 0x0000 → Underflow occurred
- (ESIOSC.ESICLKFQx AND 0x3F00) = 0x3F00 → Overflow occurred

Example 2. Example Code for Using EsioscInit() Function

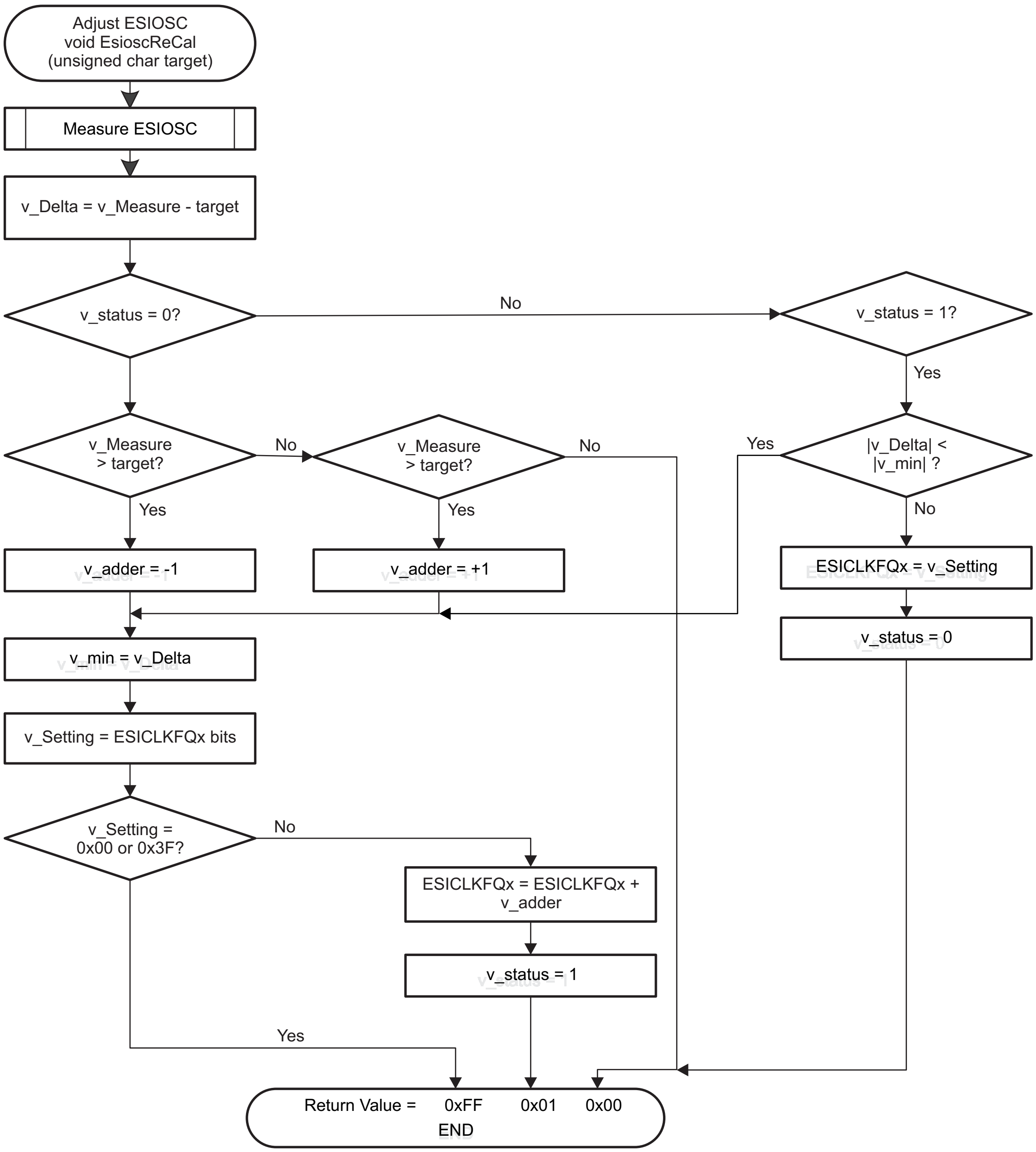
```
#include "MSP430.h"
#include "ESI_ESIOSC.h"
void main(void)
{ unsigned char temp;
  WDTCTL = WDTPW | WDTHOLD;      // Stop watchdog timer

  // XT1 Setup
  PJSEL0 |= BIT4 + BIT5;

  CSCTL0_H = 0xA5;
  CSCTL1 = DCOFSEL_0;            // Set DCO= 1MHz
  CSCTL2 = SELA__LFXTCLK + SELS__DCOCLK + SELM__DCOCLK;
  CSCTL3 = DIVA__1 + DIVS__1 + DIVM__1;  // set all dividers
  CSCTL4 |= LFXTDRIVE_0;
  CSCTL4 &= ~LFXTOFF;

  do
  {    CSCTL5 &= ~LFXTOFFG;      // Clear XT1 fault flag
 SFRIFG1 & ~OFIFG;
  } while (SFRIFG1&OFIFG);       // Test oscillator fault flag

  EsioscInit(ESIOSC_Default);    // default setting = 4.8MHz
  if ((ESIOSC&0x3F00)==0x0000)
 printf("Warning: Underflow happened!");
  if ((ESIOSC&0x3F00)==0x3F00)
 printf("Warning: Overflow happened!");

  ESIOSC_Initialization;         // initialize ESI and start operation
  while(1);                      // entire loop
}
```

3.3 Adjustment of ESIOSC During Runtime

Recalibration during operation time helps to compensate for various aging effects of the sensors. However recalibration of the ESIOSC oscillator must be avoided while a TSM measurement sequence is in progress. To start recalibration, first synchronize with the end of a TSM sequence (by using ESIIFG1 interrupt).

The *EsioscReCal()* function described in this section does not use loops, the main software must call this function several times before the final setting is stable. This approach results in short execution times of the function to ensure completion before the next TSM measurement sequence starts.

Figure 4 shows a flowchart of the *EsioscReCal()* function.



NOTE: v_status is a global variable that is initialized as 0.
Return values =

- 0xFF is used for underflow or overflow
- 0x01 is used for measurement not yet completed
- 0x00 is used for measurement is completed

Figure 4. Flowchart for ESIOSC Adjustment During Runtime

The *target* parameter in the *EsioscReCal()* function has the same functionality that it does in the *EsioscInit()* function. Equation 1, Equation 2, Equation 3, and Equation 4 in Section 3.2 describe how this parameter is defined.

When the *EsioscReCal()* function is called the first time, v_status is 0. This causes the start of a new adjustment sequence. When the function is called a second time, the v_status is 1, and the *EsioscRecal()* function performs a further adjustment step and tunes ESIOSC accordingly. The global variable "v_status" remains 1 as long the adjustment is in progress.

The execution time of the EsioscReCal() function varies with the exact moment in time the function is invoked. This is because a synchronization to ACLK is done. The *EsioscReCal()* code shown in Example 3 can take up to 150 MCLK cycles.

Example 3. Example Pseudo-Code for Using EsioscReCal() Function

```
#include "MSP430.h"
#include "ESI_ESIOSC.h"
void main(void)
{ BasicSetupOfMCU();      // for example, Stop watchdog timer
  WaitTillAclkIsStable(); // XT1 setting and wait for 32kHz oscillator

  EsioscInit(ESIOSC_Default);     // default setting = 4.8MHz
  if ((ESIOSC&0x3F00)==0x0000)
 printf("Warning: Underflow happened!");
  if ((ESIOSC&0x3F00)==0x3F00)
 printf("Warning: Overflow happened!");

  ESIOSC_Initialization();        // initialize ESI and start operation
  while(1)
  { __bis_SR_register(LPM3_bits | GIE); }  // entire loop: go to LPM3
}
void __InterruptServiceRoutine_ESI(void)
{ unsigned char RetVal;

  if (ESIIFG1 & ESIINT2)
  {  ESIINT2 &= ~ESIIFG1;        // clear interrupt flag
     RetVal = EsioscReCal(ESIOSC_Default);  // default setting = 4.8MHz
     if (RetVal == 0xFF)
      printf("Warning: Underflow or Overflow happened!");
     if (RetVal == 0x01)
      printf("ESIOSC adjustment in progress. Further function calls needed.");
     if (RetVal == 0x00)
      printf("ESIOSC adjustment completed.");
}
```

4 References

- *MSP430FR58xx, MSP430FR59xx, MSP430FR68xx, and MSP430FR69xx Family User's Guide* (SLAU367)
- *MSP430FR698x(1), MSP430FR598x(1) Mixed-Signal Microcontrollers* (SLAS789)
- *MSP430FR688x(1), MSP430FR588x(1) Mixed-Signal Microcontrollers* (SLASE32)

Revision History

Changes from Original (December 2013) to A Revision **Page**

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Products		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265