AN1800/D
(Motorola Order Number)

3/1999
REV. 0

PowerPC™

Freescale Semiconductor, Inc.

Application Note

Programming the Thermal Assist Unit in the MPC750 Microprocessor

Chuck Corley
PCSD Applications
risc10@email.sps.mot.com

This application note describes example software to program the thermal assist unit (TAU) feature of the MPC750 family of microprocessors. The TAU is an on-chip sensor that measures the instantaneous junction temperature of the microprocessor. TAU operation is described in the "Thermal Assist Unit" section of the *MPC750 RISC Microprocessor User's Manual* (order #: MPC750UM/AD). Some of the information from the user's manual is largely repeated in Section 1.1, "Thermal Assist Unit Overview," and Section 1.2, "Thermal Assist Unit Operation," as a background to understanding the software, but this application note should be used in conjunction with the user's manual. The following technical papers describe the physical implementation of the TAU:

[1] Hector Sanchez, Ross Philip, Jose Alvarez, Gianfranco Gerosa, "A CMOS Temperature Sensor for PowerPC™ RISC Microprocessor," Digest of Technical Papers 1997, IEEE Symposium on VLSI Circuits", pp. 13-14, June 1997 .

[2] Hector Sanchez, Belli Kuttanna, Tim Olson, Mike Alexander, Gian Gerosa, Ross Philip, and Jose Alvarez, "Thermal Management System for High Performance RISC Microprocessor,"Digest of Technical Papers COMPCON97, pp. 325-331 .

Note that the MPC750 is implemented in both a 2.6-volt version (part number MPC750A) and a 1.9-volt version (MPC750P). The functionality of the TAU will be identical in all members of the MPC750 family though process differences may affect the accuracy of the

This document contains information on a new product under development by Motorola. Motorola reserves the right to change or discontinue this product without notice.
© Motorola, Inc., 1999. All rights reserved.

© Freescale Semiconductor, Inc., 2004. All rights reserved.

thermal measurement reported. See the *MPC750 Hardware Specification* (order #: MPC750EC/AD) for electrical details including the resolution (4°C assumed in this application note) and accuracy (normally no worse than ± 16°C) of the TAU. A calibration capability, described in this application note, can be programmed to improve the accuracy of the temperature value returned.

To locate updates for this document or the reference materials, refer to the website at http://www.mot.com/PowerPC/.

1.1 Thermal Assist Unit Overview

The on-chip thermal assist unit (TAU) is composed of a thermal sensor, a digital-to-analog converter (DAC), a comparator, control logic, and three dedicated SPRs. See Figure 1 for a block diagram of the TAU.

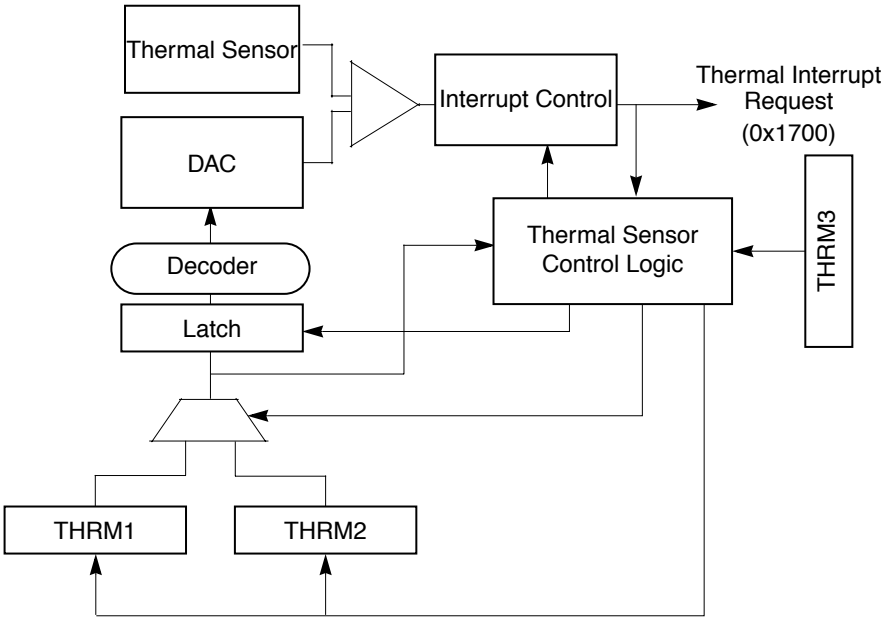

Figure 1. Thermal Assist Unit Block Diagram

The TAU provides thermal control by periodically comparing the MPC750's junction temperature against user-programmed thresholds, and generating a thermal management interrupt if the threshold values are crossed. The TAU also enables the user to determine the junction temperature through a software successive approximation routine.

The TAU is controlled through three supervisor-level SPRs, accessed through the **mtspr** and **mfspr** instructions. Two of the SPRs (THRM1 and THRM2) provide temperature threshold values that can be compared to the junction temperature value, and control bits that enable comparison and thermal interrupt generation. The third SPR (THRM3) provides a TAU enable bit, a calibration value, and a sample interval timer. Note that all the bits in THRM1, THRM2, and THRM3 are cleared to 0 during a hard reset, and the TAU remains idle and in a low-power state until configured and enabled.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

The bit fields in the THRM1 and THRM2 SPRs are described in Table 1.

Table 1. THRM1 and THRM2 Bit Field Settings

Bits	Field	Description
0	TIN	Thermal management interrupt bit. Read only. This bit is set if the thermal sensor output crosses the threshold specified in the SPR. The state of this bit is valid only if TIV is set. The interpretation of the TIN bit is controlled by the TID bit.
1	TIV	Thermal management interrupt valid. Read only. This bit is set by the thermal assist logic to indicate that the thermal management interrupt (TIN) state is valid.
2–8	Threshold	Threshold value that the output of the thermal sensor is compared to. The threshold range is between 0° and 127°C, and each bit represents 1°C. Note that this is not the resolution of the thermal sensor.
9–28	—	Reserved. System software should clear these bits to 0.
29	TID	Thermal management interrupt direction bit. Selects the result of the temperature comparison to set TIN. If TID is cleared to 0, TIN is set and an interrupt occurs if the junction temperature exceeds the threshold. If TID is set to 1, TIN is set and an interrupt is indicated if the junction temperature is below the threshold.
30	TIE	Thermal management interrupt enable. Enables assertion of the thermal management interrupt signal. The thermal management interrupt is maskable by the MSR[EE] bit. If TIE is cleared to 0 and THRMn is valid, the TIN bit records the status of the junction temperature vs. threshold comparison without asserting an interrupt signal. This feature allows system software to make a successive approximation to estimate the junction temperature.
31	V	SPR valid bit. This bit is set to indicate that the SPR contains a valid threshold, TID, and TIE control bits. Setting THRM1[V], THRM2[V], and THRM3[E] to 1 enables operation of the thermal sensor.

The bit fields in the THRM3 SPR are described in Table 2.

Table 2. THRM3 Bit Field Settings

Bits	Name	Description
0–1	—	Reserved for future use. System software should clear these bits to 0.
2		Calibration sign bit. If set, add the value programmed by THRM3[3:6] to the thermal sensor (subtract from threshold value). If cleared, subtract the value programmed by THRM3[3:6] from the thermal sensor (add to the threshold value).
3		Calibration adjust bit. Adjust the calibration value by approximately 20°C.
4		Calibration adjust bit. Adjust the calibration value by approximately 8°C
5		Calibration adjust bit. Adjust the calibration value by approximately 4°C
6		Calibration adjust bit. Adjust the calibration value by approximately 2°C
7–17	—	Reserved for future use. System software should clear these bits to 0.
18–30	SITV	Sample interval timer value. Number of elapsed processor clock cycles before a junction temperature vs. threshold comparison result is sampled for TIN bit setting and interrupt generation. This is necessary due to the thermal sensor, DAC, and the analog comparator settling time being greater than the processor cycle time. The value should be configured to allow a sampling interval of 20 microseconds.
31	E	Enables the thermal sensor compare operation if either THRM1[V] or THRM2[V] is set to 1.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

1.2 Thermal Assist Unit Operation

The TAU can be programmed to operate in single- or dual-threshold modes, which results in the TAU generating a thermal management interrupt when one or both threshold values are crossed. In addition, with the appropriate software routine, the TAU can also directly determine the junction temperature. The following sections describe the configuration of the TAU to support these modes of operation.

1.2.1 TAU Single-Threshold Mode

When the TAU is configured for single-threshold mode, either THRM1 or THRM2 can be used to contain the threshold value, and a thermal management interrupt is generated when the threshold value is crossed. To configure the TAU for single-threshold operation, set the desired temperature threshold, TID, TIE, and V bits for either THRM1 or THRM2. The unused THRM*n* threshold SPR should be disabled by clearing the V bit to 0. In this discussion, THRM*n* refers to the THRM threshold SPR (THRM1 or THRM2) selected to contain the active threshold value.

After setting the desired operational parameters, the TAU is enabled by setting the THRM3[E] bit to 1, and placing a value allowing a sample interval of 20 microseconds or greater in the THRM3[SITV] field. The THRM3[SITV] setting determines the number of processor clock cycles between input to the DAC and sampling of the comparator output; accordingly, the use of a value smaller than recommended in the THRM3[SITV] field can cause inaccuracies in the sensed temperature.

If the junction temperature does not cross the programmed threshold, the THRM*n*[TIN] bit is cleared to 0 to indicate that no interrupt is required, and the THRM*n*[TIV] bit is set to 1 to indicate that the TIN bit state is valid. If the threshold value has been crossed, the THRM*n*[TIN] and THRM*n*[TIV] bits are set to 1, and a thermal management interrupt is generated if both the THRM*n*[TIE] and MSR[EE] bits are set to 1.

A thermal management interrupt is held asserted internally until recognized by the MPC750's interrupt unit. Once a thermal management interrupt is recognized, further temperature sampling is suspended, and the THRM*n*[TIN] and THRM*n*[TIV] values are held until an **mtspr** instruction is executed to THRM*n*.

The execution of an **mtspr** instruction to THRM*n* anytime during TAU operation will clear the THRM*n*[TIV] bit to 0 and restart the temperature comparison. Executing an **mtspr** instruction to THRM3 will clear both THRM1[TIV] and THRM2[TIV] bits to 0, and restart temperature comparison in THRM*n* if the THRM3[E] bit is set to 1.

1.2.2 TAU Dual-Threshold Mode

The configuration and operation of the TAU's dual-threshold mode is similar to single-threshold mode, except both THRM1 and THRM2 are configured with desired threshold and TID values, and the TIE and V bits are set to 1. When the THRM3[E] bit is set to 1 to enable temperature measurement and comparison, the first comparison is made with THRM1. If no thermal management interrupt results from the comparison, the number of processor cycles specified in THRM3[SITV] elapses, and the next comparison is made with THRM2. If no thermal management interrupt results from the THRM2 comparison, the time specified by THRM3[SITV] again elapses, and the comparison returns to THRM1.

This sequence of comparisons continues until a thermal management interrupt occurs, or the TAU is disabled. When a comparison results in an interrupt, the comparison with the threshold SPR causing the interrupt is halted, but comparisons continue with the other threshold SPR. Following a thermal management interrupt, the interrupt service routine must read both THRM1 and THRM2 to determine which threshold was crossed. Note that it is possible for both threshold values to have been crossed, in which case the TAU ceases making temperature comparisons until an **mtspr** instruction is executed to one or both of the threshold SPRs.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

1.2.3 MPC750 Junction Temperature Determination

While the MPC750's TAU does not implement an analog-to-digital converter to enable the direct determination of the junction temperature, system software can execute a simple successive approximation routine to find the junction temperature.

The TAU configuration used to approximate the junction temperature is the same required for single-threshold mode, except that the threshold SPR selected has its TIE bit cleared to 0 to disable thermal management interrupt generation. Once the TAU is enabled, the successive approximation routine loads a threshold value into the active threshold SPR, and then continuously polls the threshold SPRs TIV bit until it is set to 1, indicating a valid TIN bit. The successive approximation routine can then evaluate the TIN bit value, and then increment or decrement the threshold value for another comparison. This process is continued until the junction temperature is determined.

1.2.4 TAU Calibration

Uncalibrated, the accuracy of the TAU will vary from part-to-part with process and temperature. Calibrated, the accuracy will more closely approach the resolution of the TAU (see the *MPC750 Hardware Specification*).

Bits 0–17 of THRM3 are defined as "reserved for future use" in the user's manual. However, bits 2–6 can be used to calibrate the TAU and are defined in Table 1. For an uncalibrated TAU, these bits should be cleared to 0 to indicate no adjustments made to the thermal sensor.

To calibrate the TAU, these bits are set to adjust the thermal sensor so that the TAU reading matches the junction temperature as determined by some other, more accurate, means. For example, an external thermocouple which has good thermal contact with the back of the die will provide an accurate junction temperature measurement if the die temperature is stable. Another approach is to adjust the thermal sensor so that the TAU reading is correct when the junction temperature is controlled by an external device, for example, a thermoelectric cooling plate thermally attached to the back of the die and controlled via a thermocouple fed back to an adjustable current source.

One other potential approach to calibration relies on the comparatively slow rate of change of junction temperature with time. During factory test and laboratory experiments, the junction temperature has been observed to rise from ambient temperature at a peak rate of approximately 1°C per second. Therefore, a calibration value taken as soon as possible after power-up by comparing the TAU sensor reading to ambient conditions (for example, room temperature) could improve accuracy over an uncalibrated TAU.

Note that each processor may have a different adjustment required and that once the calibration setting is determined, the value for bits 2–6 should be used thereafter for that part. The programs described herein use the convention that if TAU_Cal_Value[31] = 1 the calibration value is valid and should be used unless changed through recalibration.

1.3 Programming Examples

The following sections describe example programs to utilize the TAU in the modes described above. These programs have been proven on an MPC750 evaluation board running DINK, the diagnostic kernel available from Motorola PowerPC Applications. DINK, these routines, and test drivers to exercise the routines independent of the TAU are available at the website at http://www.mot.com/PowerPC.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

1.3.1 Assembly Language Routines to Access the TAU

The following code provides assembly language subroutines that can be called from C to read and write the THERM*n* registers of the TAU. This syntax is compatible with the MetaWare High C/C++ compiler syntax used for DINK.

```
!file "readtau.s"
!  Assembly language routines to access the built-in Thermal Assist Unit
!  (TAU) of the MPC750.
!  Modification History:
!  990125 CJC Original
!     Register usage:
!        r3 = Arg1       new THERM spr value when writing
!              THERM spr value returned when reading
!        r11 = link register storage

      .data
      .global Report_Decrementer
Report_Decrementer:
      .long 0
      .global Report_TAU
Report_TAU:
      .long 0

      .text
      .global read_THERM1
      .global read_THERM2
      .global read_THERM3
      .global write_THERM1
      .global write_THERM2
      .global write_THERM3
      .global read_PVR
      .global Enable_Interrupts
      .global decrementer_handler
      .global TAU_handler

! Routines to read the TAU sprs.
read_THERM1:
      mflrr11  !Save the return address.
      mfsprr3,1020!Get THERM1.
      mtlrr11
      blr      !Return in r3

read_THERM2:
      mflrr11  !Save the return address.
      mfsprr3,1021!Get THERM2.
      mtlrr11
      blr      !Return in r3

read_THERM3:
      mflrr11  !Save the return address.
      mfsprr3,1022!Get THERM3.
```


For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

```
      mtlrr11
      blr      !Return in r3

! Routines to write the TAU sprs.
write_THERM1:
      mflrr11  !Save the return address.
      mtspr1020, r3!Write value in r3 into THERM1.
      mtlrr11
      blr

write_THERM2:
      mflrr11  !Save the return address.
      mtspr 1021, r3!Write value in r3 into THERM2.
      mtlrr11
      blr

write_THERM3:
      mflrr11  !Save the return address.
      mtspr1022, r3!Write value in r3 into THERM3.
      mtlrr11
      blr

! Routine to read the Processor Version Register.
read_PVR:
      mflrr11  !Save the return address.
      mfsprr3,287!Get PVR.
      mtlrr11
      blr      !Return in r3
```

1.3.2 Polling in the Single-Threshold Mode to Determine MPC750 Temperature

The following code provides a C language program which will print to the screen the uncalibrated junction temperature as determined by successive approximation. The TAU is accurate to 4°C over the range 0–127° so there are 32 possible values (THERM*n*[7:8] are don't cares). By successively halving the range, it takes five iterations to find the junction temperature to within 4 degrees, assuming that the temperature remains constant for the duration of the approximation. As stated above, the peak rate of change of temperature has been observed to be 1°C per second or less. Therefore, it is unlikely that the temperature will vary more than 1 degree over the duration of this program.

The THRM3[SITV] field should be programmed to a value allowing a sample interval of 20 microseconds or greater. Given the observed rate of change, there is little harm in using a large value of sample interval; also, to allow for a wide variety of core processor frequencies, SITV is set for 20 microseconds at 400 MHz.

```
/* file "tau.c"
 * Programs to measure junction temperature on the MPC750
 * using the Thermal Assist Unit (TAU).
 * Modification History:
 * 990125  CJC  Original
*/
#include "readTAU.s"  /* TAU read/write routines */
```


For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

```
#include "maximer.h" /* DINK I/O function definitions */

#define TAU_INTRPT_BIT 0x80000000 /* THERM1/2[0] = 1 */
#define TAU_INTRPT_VALID 0x40000000 /* THERM1/2[1] = 1 */
#define TAU_RESOLUTION 4 /* 4 degrees C */
#define TAU_THRESHOLD 0x3f800000  /* THERM1/2[2:8] mask  */
#define TAU_INTRPT_DIRECTION 0x4 /* THERM1/2[29] = 1 (interrupt if below) */
#define TAU_INTRPT_ENABLE 0x2 /* THERM1/2[30] = 1 */
#define TAU_SPR_VALID 0x1 /* THERM1/2/3[31] = 1 */
#define TAU_SITV 16000 /* THERM3[18:30] = 20 microseconds * 400MHz(max) *2 */
#define TRUE 1

/* Prototype declarations  */
int TAU_single_threshold(void);
void TAU_dual_threshold(void);
void TAU_calibrate(int);
void TAU_print_info(int temperature);

/* Global Variables  */
int TAU_Cal_Value = 0;
int Test_Temperature;   /* For testing only. */
extern int Report_Decrementer;   /* For flagging decrementer interrupt. */
extern int Report_TAU;         /* For flagging TAU interrupt. */
extern unsigned long THERM1, THERM2, THERM3;   /* For testing only. */

/*  Program to report calibrated junction temperature
 *  using the single threshold TAU method.
 */
int main(void)
{
      TAU_print_info(TAU_single_threshold());
      return(0);
}

/* Successive approximation routine to find the junction temperature.
 * Determines the threshold value just below the measured junction temperature
 * (assuming that the rate of temperature change is less than five
 * sampling intervals).
 * Therefore the TAU measured a value between this threshold and the next
 * threshold 4 degrees higher.  Returns an approximate temperature reading
 * which is this minimum threshold plus 2 degrees.
 */
int TAU_single_threshold(void)
{
      int i;
      int   min_temperature = 0;  /* Start at minimum value. */
      unsigned long PVR, threshold;
      unsigned long Therm1, Therm2, Therm3, Thermn;

      PVR = read_PVR() >> 16;   /* Check Processor Version Register.  */
      if ((PVR != 0x8) && (PVR != 0xc))  /* If not Arthur or G4,  */
```

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

```
        return(-1);   /* we're done. (No TAU)  Return error. */

      write_THERM2(Therm2 = 0);   /* Disable unused THERMn threshold SPR.*/
      Therm3 = TAU_Cal_Value | TAU_SITV | TAU_SPR_VALID;
      write_THERM3(Therm3);  /* Initialize THERM3 SITV and enable.*/

/* Start with a threshold of one-half total range (0-127 degrees).*/
      threshold = TAU_RESOLUTION << 4; /* Initial threshold of 64 degrees */

      for (i=4;i>= 0; i--)/* By fifth iteration, we will know the temperature.*/
      {
         /*  Shift the threshold value to THERM1[2:8] and enable. */
         /*  Clear TID bit so that the thermal interrupt  bit (TIN) will be set */
         /*  when the temperature is above the threshold.  */
         Therm1 = (threshold << 23) | TAU_SPR_VALID;
         write_THERM1(Therm1);  /* Write to THERM1. */

         while (!((Thermn = read_THERM1()) & TAU_INTRPT_VALID))
         {
         }  /* Wait for SITV (Sample Interval Timer) to expire.  */
         if (Thermn & TAU_INTRPT_BIT)  /* Check thermal interrupt (TIN) bit.  */
         {  /*  Temperature is greater than threshold  */
            min_temperature = threshold;   /* Threshold is the new minimum. */
               /*Raise the threshold half the difference */
            threshold += TAU_RESOLUTION / 2 << i;
         }
         else
         {        /*  Temperature is lower than threshold  */
               /*Lower the threshold half the difference */
            threshold -= TAU_RESOLUTION / 2 << i;
      }
      return(min_temperature + 2);
}

/* Routine to print out the junction temperature measurement obtained
 * from the TAU (Thermal Assist Unit).
 * Could be called from anywhere including an interrupt routine in DINK.
 * Enter with temperature to be displayed.
 * Will print temperature.
 */
 void TAU_print_info(int temperature)
{
      int bias = 0;
      printf("Junction temperature: %d C ", temperature);

      if (TAU_Cal_Value) /* Is calibration value valid? (At least bit 31 set?) */
      {
         if(TAU_Cal_Value & 0x02000000)
            bias = 2;
         if(TAU_Cal_Value & 0x04000000)
            bias += 4;
```

Freescale Semiconductor, Inc.

```
         if(TAU_Cal_Value & 0x08000000)
            bias += 8;
         if(TAU_Cal_Value & 0x10000000)
            bias += 20;
         if(TAU_Cal_Value & 0x20000000)
            printf("Calibration = -%d C\n", bias);
         else
            printf("Calibration = +%d C\n", bias);
      }
      else
      {
         printf("Uncalibrated.\n");
      }
}
```

1.3.3 Calibrating the TAU to a Known Temperature

Tne code that follows provides a C language program which will prompt the user for a known junction temperature. (The included files, defines, and global variables of the previous program are not repeated here.) The user could respond with a stable thermocouple reading or the stable temperature controller setting of a thermoelectric cooler attached to the die. The program then calls the single-mode threshold program of Section 1.3.2, “Polling in the Single-Threshold Mode to Determine MPC750 Temperature,” to determine the junction temperature measurement and compute a calibration value.

As described above, for an inexpensive, non-intrusive calibration the subroutine shown below could be called immediately at the first power-on reset with predicted ambient temperature conditions. The resulting calibration value could then be saved and used for all subsequent TAU accesses for the life of the part. Or the part could be recalibrated later using a more sophisticated method. By the time the main program below is called with sufficient resources loaded to prompt the user for a response, the die has probably already risen above ambient conditions and this technique is no longer applicable.

```
/*  Program to prompt the user for a calibration value
 *   and then report calibrated junction temperature.
 */
int main(void)
{
      int cal_temp;
      printf("Enter calibrate temperature: ");
      scanf("%d",&cal_temp);
      printf("\n");
      TAU_calibrate(cal_temp);
      TAU_print_info(TAU_single_threshold());
      return(0);
}

/*  Routine to set the calibration value for the Thermal Assist Unit.
 *  Undocumented in the MPC750 User's Manual, the TAU can actually be
 *  biased by setting reserved bits THERM3[2:6].
 *    THERM3[2] = sign bit (set means subtract the calibration value from
threshold)
 *   THERM3[3] = adjust by 20 degrees C
 *   THERM3[4] = adjust by 8 degrees C
 *   THERM3[5] = adjust by 4 degrees C
```


For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

```
 *   THERM3[6] = adjust by 2 degrees C
 *
 *  Enter with a known junction temperature.
 *  This routine will call TAU_single_threshold() to determine TAU reading
 *  and set the global variable TAU_Cal_Value.
 *
 *  As a convention, TAU_Cal_Value bit 31 will be set to a one (valid) to
 *  distinguish a calibration value of zero from an uninitialized TAU_Cal_Value.
 */
void TAU_calibrate(int actual_temperature)
{
      int difference_error, TAU_measured_value;
      TAU_Cal_Value = 0;
      if ((TAU_measured_value = TAU_single_threshold()) < 0)
         return;   /* If TAU_single_threshold returned -1, there is no TAU.  */
      /* Find error = calibration value,
         then actual will equal measured + Cal_Value  */
      difference_error = actual_temperature - TAU_measured_value;
      /* Set TAU_Cal_Value to show that it is valid even if zero. */
      TAU_Cal_Value = 1;
      if (difference_error < 0)  /* What should sign bit be? */
      {
         TAU_Cal_Value = TAU_Cal_Value | 0x20000000; /* Set sign bit (THERM3[2])
*/
         difference_error = - difference_error;
      }
      if (difference_error >= 20)   /* Is the difference >= 20? */
      {
         TAU_Cal_Value = TAU_Cal_Value | 0x10000000 ; /*Set bit (THERM3[3]) */
         difference_error -= 20;
      }
      if (difference_error >= 8)   /* Is the difference >= 8? */
      {
         TAU_Cal_Value = TAU_Cal_Value | 0x08000000 ; /* Set bit (THERM3[4]) */
         difference_error -= 8;
      }
      if (difference_error >= 4)   /* Is the difference >= 4? */
      {
         TAU_Cal_Value = TAU_Cal_Value | 0x04000000 ; /* Set bit (THERM3[5]) */
         difference_error -= 4;
      }
      if (difference_error >= 2)   /* Is the difference >= 2? */
      {
         TAU_Cal_Value = TAU_Cal_Value | 0x02000000 ; /* set bit (THERM3[6]) */
      }
}
```

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

1.3.4 Using the TAU Interrupts in Dual-Threshold Mode

The following code provides an assembly language interrupt service routine which sets a flag Report_TAU on the occurrence of a TAU exception (0x1700 exception).

There are many potential uses for the dual-threshold TAU interrupts including user warnings of cooling system failure, automatic power-down, instruction throttling, etc. This program uses the dual-thresholds to track changes in the junction temperature at 4 degree intervals. THERM1 is set to interrupt if the junction temperature drops below a value t – 4 and THERM2 is set to interrupt if the junction temperature rises above t + 4. After reporting one of these events and the new temperature, t, the thresholds are adjusted by 4 degrees to wait for the next change in junction temperatute.

Settling time of the TAU is not critical in this program and has been increased to the maximum. (An early attempt to use this program to measure from a lower threshold t to an upper threshold t + 4 resulted in momentary jittering back and forth between two temperatures as the junction temperature alternately was above and then below the new threshold. This jittering was independent of settling time. The program was revised to measure a temperature ± 4°C where the threshold crossed is in the middle of the range and the result is very stable.)

Note that enabling external interrupts (via setting MSR[EE] = 1) inevitably enables the decrementer interrupt as well, so a decrementer interrupt service routine must be provided that at least returns code execution back to the interrupted program.

```
! Routine to Enable Interrupts.
Enable_Interrupts:
      mflrr11   !Save the return address.
      mfmsrr3   !Get MSR.
      orir3,r3,0x8000! Set the MSR[EE] bit (#16)
      mtmsrr3   !Interrupts can begin immediately after this instr.
      mtlrr11
      blr       !Return

! Interrupt routine to initialize the decrementer
! to keep it from interfering with the TAU interrupt
! after MSR[EE] is set.
! Actually, an rfi is all that is required but for
! debug we are going to set a flag visible to the main program.

      .org 0x900
      .text
decrementer_handler:
      mtsprsprg3,r3! Save r3 and r4 so we can use them
      mtsprsprg2,r4

      lisr3,0x8000! Set a flag to show it occurred for debug

      lisr4, Report_Decrementer@h! get high order address
      stwr3,Report_Decrementer@l(r4)! store flag to address

      mfsprr3,sprg3! Restore r3 and r4
      mfsprr4,sprg2

      rfi       ! Return from interrupt
```


For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

```
! Interrupt routine for the Thermal Assist Unit (TAU)
! on MPC750 and G4.
! This code generates an exception when the temperature goes
! below or above an 8degree C range.
! Rather than report the event, this routine only stores the values
! for use in a notification routine that is polling the values.
! Otherwise, we would have to store a lot more state.

      .org 0x1700
      .text
TAU_handler:
      mtsprsprg3,r3! Save r3 and r4 so we can use them
      mtsprsprg2,r4

      lisr3,0x8000! Set a flag to show a TAU interrupt occurred

      lisr4, Report_TAU@h! get high order address
      stwr3,Report_TAU@l(r4)! store flag to address

      mfsprr3,sprg3! Restore r3 and r4
      mfsprr4,sprg2

      rfi         ! Return from interrupt
```

The following code provides a C language program which polls the value of the Report_TAU flag and when it is set, calls the TAU_dual_threshold routine to determine which threshold was exceeded and adjusts the thresholds before enabling the next interrupt.

```
/*  Routine to enable the TAU interrupt and then poll
 *  a flag until a TAU interrupt occurs.  On an interrupt,
 *  report (possibly calibrated) junction temperature
 *  using the dual threshold TAU method.
 */
#define TAU_SITV 0x3ffe /* THERM3[18:30] = maximum (20 secs at 409 MHz */

int main(void)
{
      write_THERM1((20 << 23)| TAU_INTRPT_DIRECTION | \
         TAU_INTRPT_ENABLE | TAU_SPR_VALID);
         /* Initialize TAU THERM1 threshold = 20C */
      write_THERM2((((20 + 8) >>23) | \
         TAU_INTRPT_ENABLE | TAU_SPR_VALID) & ~TAU_INTRPT_DIRECTION);
               /* Initialize TAU THERM2 threshold = 20 + 8C */
      write_THERM3(TAU_Cal_Value | TAU_SITV | TAU_SPR_VALID);
               /* Initialize THERM3 and enable TAU */

      Enable_Interrupts();    /*  Allow  external  interrupts  (and  TAU  and
decrementer) */
```


For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

```
      while (TRUE)
      {
         if (Report_Decrementer)
         {
            printf("Decrementer interrupt\n");
            Report_Decrementer = 0;
         }
         if (Report_TAU)
         {
            Report_TAU = 0;
            printf("TAU interrupt\n");
            TAU_dual_threshold();
         }
      }
      return(0);
}

/* An interrupt service routine which utilizes the two threshold values of
 * THERM1 and THERM2.  Exceeding THERM1 could prompt the user to take action
 * to reduce temperature and exceeding THERM2 could begin taking unilateral
 * action, e.g. instruction throttling, to reduce temperature.
 *
 * However in this routine, THERM1 is set to detect junction temperature dropping
 * below some value t-4.  THERM2 is set to detect junction temperature rising
 * above t+4. This is used to track junction temperature changes for test
purposes.
 *
 * Instead of actually being invoked by an exception, this version polls a
 * flag set in the TAU interrupt service routine indicating a thermal interrupt
has
 * occurred.  This simple approach increases the resources available to take
 * advantage of the interrupt without having to save a lot of state in the
 * exception handler.
 */
void TAU_dual_threshold(void)
{
      int min_temperature = 0;  /* Storage for temperature t-4 */
      unsigned long threshold = 0; /* Storage for threshold crossed. */
      unsigned long Therm1, Therm2, Therm3;

      Therm1 = read_THERM1(); /*Get TAU values. */
      Therm2 = read_THERM2();
      Therm3 = read_THERM3();
      write_THERM3(0); /* Turn TAU off while adjusting thresholds.*/
      if (Therm1 & TAU_INTRPT_BIT) /* Was THERM1 the cause of interrupt? */
      {        /* Yes, junction temperature has dropped below t-4. */
         if (Therm2 & TAU_INTRPT_BIT) /* Did THERM2 also signal interrupt? */
         {
            printf("An  error  has  occurred.  Junction  temperature  <t-4  and
>t+4\n");
         }
         threshold = (Therm1 & TAU_THRESHOLD) >> 23;  /* Get low threshold. */
```


For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

```
         min_temperature = threshold - 4;  /* Reduce low threshold by 4 degrees.
*/
         if (min_temperature < 0) min_temperature = 0; /* Can't go below zero. */
         write_THERM1((min_temperature << 23)| TAU_INTRPT_DIRECTION | \
            TAU_INTRPT_ENABLE  |  TAU_SPR_VALID);   /*  Set  new  thresholds  and
enable */
         write_THERM2((((min_temperature + 8) << 23) | \
            TAU_INTRPT_ENABLE | TAU_SPR_VALID) & ~TAU_INTRPT_DIRECTION);
         write_THERM3(TAU_Cal_Value  |  TAU_SITV  |  TAU_SPR_VALID); /*Re-enable
intrpts*/
      }
      else if (Therm2 & TAU_INTRPT_BIT) /* Was THERM2 the cause of the interrupt?
*/
         {     /* Yes, junction temperature has risen above t+4. */
            threshold = (Therm2 & TAU_THRESHOLD) >> 23;  /* Get high threshold */
            min_temperature = threshold - 4;  /* New low threshold is high - 4 */
            if (min_temperature > 116) /* Lower threshold can't go above 116.*/
               min_temperature = 116;
            write_THERM1((min_temperature << 23)| TAU_INTRPT_DIRECTION | \
               TAU_INTRPT_ENABLE  |  TAU_SPR_VALID);   /*  Set  new  threshold  and
enable */
            write_THERM2((((min_temperature + 8) << 23) | \
               TAU_INTRPT_ENABLE | TAU_SPR_VALID) & ~TAU_INTRPT_DIRECTION);
            write_THERM3(TAU_Cal_Value | TAU_SITV | TAU_SPR_VALID); /*Re-enable
*/
         else /* If it wasn't THERM1 or THERM2 that caused the interrupt, ERROR!
*/
         {
               printf("A unexplained TAU interrupt has occurred.\n");
         }
      TAU_print_info(min_temperature  +  4);     /*  Print  out  the  current
temperature. */
      return;
}
```

1.4 Experimental Results

The single-threshold routine of Section 1.3.2, "Polling in the Single-Threshold Mode to Determine MPC750 Temperature" was used to measure junction temperature on a small sample (four parts) of MPC750P with similar processing (same date code). A thermoelectric cooler was attached to the parts to control the junction temperature. The thermal connection between the thermoelectric cooler and the parts was surprisingly sensitive to the effectiveness of the thermal grease connecting the thermoelectric cooler and the die. Properly thermally connected, the TAU measurement matched the controller temperature to within 10 degrees worst case. Calibration could not significantly improve accuracy on this sample. An MPC750A part (Part #5) with significantly different processing was measured by attaching a thermocouple and running the microprocessor at different frequencies to get a range of stable operating temperatures. The non-linearity and inaccuracies in measurements of part #5 may result from poor thermal connection between the thermocouple and back of the die. We suspect that the TAU may have given a more accurate measurement of the high temperature readings than the thermocouple did.

The results are plotted in Figure 2 where the ideal of measured temperature on the y axis equal to actual temperature on the x-axis is plotted as the heavy solid line with slope of 1.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

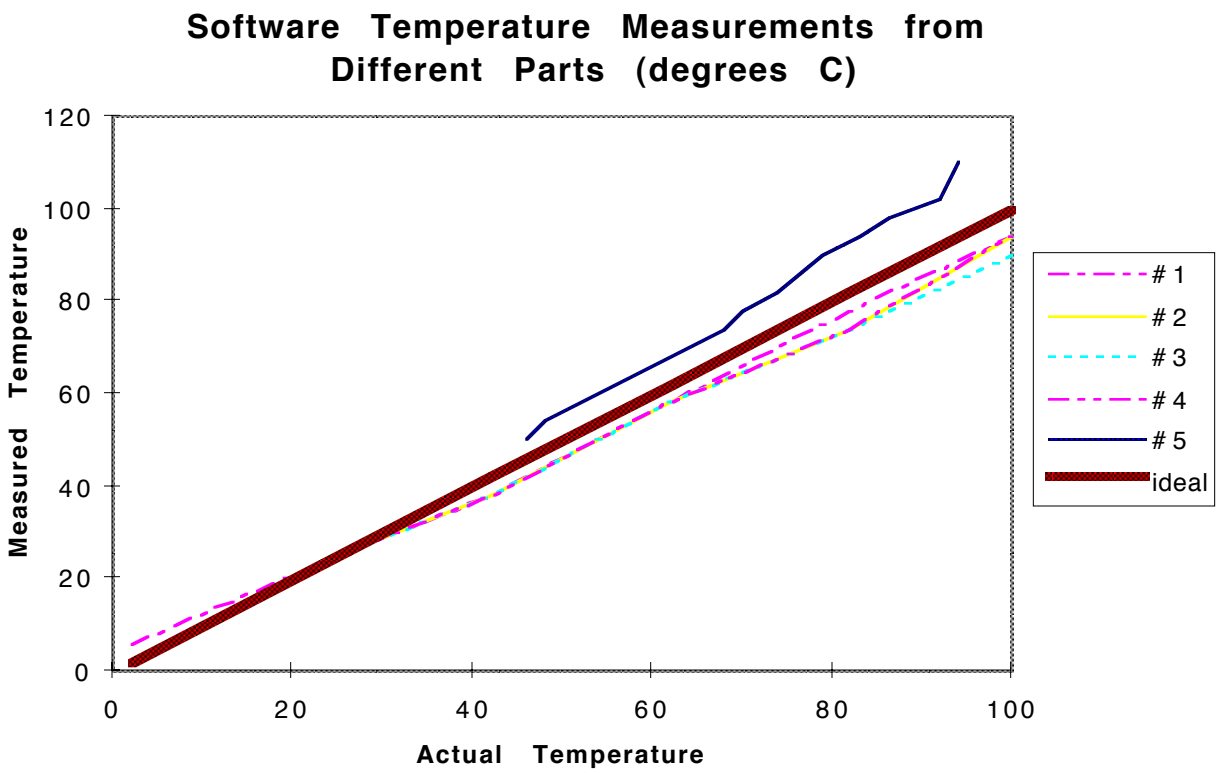

Figure 2. Graph of Experimental Results

The same parts were calibrated to the known temperature. The slope and error of the TAU results for individual parts, calibrated and uncalibrated, are shown in Table 3

Table 3. Experimental Results

Part No.	Slope of measured temperature to actual (without calibration)	Average Error (uncalibrated)	Maximum Error (uncalibrated)	Slope of measured temperature to actual (with calibration)	Average Error (calibrated)	Maximum Error (calibrated)
ideal	1	±16°C	±16°C	1	±4°C	±4°C
1	.91	-2	-6	.91	-2	-6
2	.94	-3	-6	.92	-4	-8
3	.89	-4	-10	.89	-4	-10
4	.89	0	-6	.92	-4	-8
5	1.12	+8	+16	1.16	+2	+6

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

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)
support@freescale.com

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 Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.
Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters which may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including “Typicals” must be validated for each customer application by customer’s technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.

freescale™ semiconductor