How to use the on-chip temperature sensor

XMC1000

About this document

Scope and purpose

This application note provides information on how to use important temperature sensor functions in the SCU XMC Lib and external library in the DAVE™ Version 4 environment.

Intended audience

Engineers or developers who would like to use the temperature sensor of the XMC1000 product family.

Table of contents

About this document **1**
Table of contents **2**
1 XMC™ Lib functions supporting temperature sensor **3**
1.1 Calculation of current chip temperature 3
1.1.1 XMC_SCU_CalcTemperature 3
1.2 Installation of threshold values for temperature comparison 5
1.2.1 XMC_SCU_SetTempLowLimit 5
1.2.2 XMC_SCU_SetTempHighLimit 5
2 External library functions supporting the temperature sensor **9**
2.1 Calculation of the current chip temperature 9
2.1.1 XMC1000_CalcTemperature 9
2.2 Conversion of temperature to threshold values for temperature comparison 11
2.2.1 XMC1000_CalcTSEVAR 11
Revision history **15**

1 XMC™ Lib functions supporting temperature sensor

The XMC1000 family of devices provides a Temperature Sensor (DTS) peripheral which measures and indicates the current temperature. The DTS Temperature Sensor is also referred to as 'TSE' in the user documentation of the XMC1100, XMC1200 and XMC1300.

The start-up time of the DTS is less than than 15 µs with a measurement time of up to 10 ms. The permitted temperature sensor range is between -40°C (233 K) and 115°C (388 K). The accuracy of the DTS is typically +/-6°C for a junction temperature above 20°C. For more information, please refer to the respective device user documentation (Reference Manual, Data Sheet and Errata Sheet).

The XMC™ Lib consists of low level drivers which contain APIs for the XMC™ product family peripherals. The System Control Unit (SCU) driver library is a part of the XMC™ Lib which groups functions for controlling the General Control Unit including temperature monitoring, Clock Control Unit, Reset Control Unit and Interrupt System. In this document, we introduce two sets of code examples based on the XMC1400 in DAVE™ Version 4 environment to illustrate the usage of these DTS functions which are included in the SCU XMC™ Lib.

When enabled, the temperature measurement starts and the result is stored via bit field TSE_MON of the Temperature Sensor Counter2 Monitor Register (ANATSEMON). After storing the result, the temperature sensor continues with the next measurement. The SCU XMC™ Lib **XMC_SCU_CalcTemperature** function can be used to determine the current chip temperature using the TSE_MON value.

The temperature sensor is also capable of detecting low and high temperature events when the measurement result crosses the higher and/or lower threshold values. The threshold values are configurable via bit fields TSE_IL and TSE_IH in the temperature sensor low/high Temperature Interrupt Registers (ANATSEIL and ANATSEIH). The SCU XMC™ Lib **XMC_SCU_SetTempLowLimit** and **XMC_SCU_SetTempHighLimit** functions can be used to install the threshold values in the ANATSEIL and ANATSEIH registers.

The SCU XMC™ Lib **XMC_SCU_CalcTemperature**, **XMC_SCU_SetTempLowLimit** and **XMC_SCU_SetTempHighLimit** functions are available to be used in XMC1400. These functions can also be used for XMC1100, XMC1200, XMC1300 AB-step ES samples with a 2-byte user configuration sector version 0003_H or higher, and productive devices. The user configuration sector version is stored in the flash configuration sector 0, at address $10000FEA_H$.

The following sections provide more details for the previously mentioned DTS API functions.

1.1 Calculation of current chip temperature

1.1.1 XMC_SCU_CalcTemperature

The specification of the **XMC_SCU_CalcTemperature** XMC Lib function is :

- Input parameter : none
- Return status : chip temperature in degree Kelvin
- Prototype : unsigned long integer XMC_SCU_CalcTemperature (void)

In the code shown below, any one of these ports, P4.0, P4.1 or P4.2 is toggled when the temperature is at 25°C, above 25°C or below 25°C. Prior to calling the library function to determine the current chip temperature, the temperature sensor needs to be enabled via bit TSE_EN in the ANATSECTRL register. The **XMC_SCU_StartTempMeasurement** XMC™ Lib function performs this configuration.

```
#include "xmc_gpio.h"
```

```
#include "xmc_scu.h"
#define LED0 P4_0
#define LED1 P4_1
#define LED2 P4_2


/* Port pins output mode configuration */
XMC_GPIO_CONFIG_t LED_pin_config =
{
  .mode = XMC_GPIO_MODE_OUTPUT_PUSH_PULL,
  .output_level= 1U
};
void main(void)
{
       uint32_t temp_C = 0;
       uint32_t temp_k = 0;
       uint32_t limit = 0;
       uint32_t delay = 10000;

       /* Initialize port pins to output mode */
       XMC_GPIO_Init(LED0, &LED_pin_config);
       XMC_GPIO_Init(LED1, &LED_pin_config);
      XMC_GPIO_Init(LED2, &LED_pin_config);

       /* Enable DTS */
       XMC_SCU_StartTempMeasurement();

       while(1)
       {
              /* Calculate temperature of the chip in Kelvin */
              temp_k = XMC_SCU_CalcTemperature();

              /* Convert temperature to Celcius */
              temp_C = temp_k - 273;
              if(temp_C == 25)
              {
                     XMC_GPIO_ToggleOutput(LED0);
              }
              else if (temp_C > 25)
              {
```

```
                XMC_GPIO_ToggleOutput(LED1);
            }
            else if (temp_C < 25)
            {
                XMC_GPIO_ToggleOutput(LED2);
            }
            while(--delay);
            delay = 10000;
        }
        return 0;
}
```

1.2 Installation of threshold values for temperature comparison

1.2.1 XMC_SCU_SetTempLowLimit

The specification of the **XMC_SCU_SetTempLowLimit** XMC™ Lib function is :

Input parameter:	low threshold temperature in degree Kelvin (allowed range 233 to 388)
Return status:	status of equivalent low threshold value installation based on temperature provided
Prototype:	XMC_SCU_STATUS_t XMC_SCU_SetTempLowLimit (uint32_t temperature)

1.2.2 XMC_SCU_SetTempHighLimit

The specification of the **XMC_SCU_SetTempHighLimit** XMC™ Lib function is :

Input parameter:	high threshold temperature in degree Kelvin (allowed range 233 to 388)
Return status :	status of equivalent high threshold value installation based on temperature provided
Prototype:	XMC_SCU_STATUS_t XMC_SCU_SetTempHighLimit (uint32_t temperature)

In the code shown below, the DTS high temperature event happens when the temperature is above 0°C and the DTS low temperature event happens when the temperature is below 85°C. The temperature measurement is performed at room temperature. The appropriate threshold values shall be adapted according to the application. The **XMC_SCU_SetTempHighLimit** and **XMC_SCU_SetTempLowLimit** XMC™ Lib functions are used to convert these temperature points to the threshold values to be installed in the ANATSEIH and ANATSEIL registers, respectively. The actual measurement result is available at ANATSEMON. The result is compared against the configured limits in ANATSEIH and ANATSEIL registers. A high temperature event SRRAW.TSE_HIGH is triggered because ANATSEMON is less than ANATSEIH. A low temperature event SRRAW.TSE_LOW is triggered because ANATSEMON is more than ANATSEIL. For an input parameter which lies outside the permitted range, the function returns one value. In this example, add or subtract 1 Kelvin to/from the input parameter and re-run the function. If the function execution is successful, a zero value is returned.

In this example, interrupts are triggered for DTS high or DTS low temperature events. In the interrupt service routine, **IRQ1_Handler**, the interrupt is disabled to prevent continuous triggering of interrupts as the temperature does not change instantaneously. Upon entering the service routine for the first time, P4.0 is

toggled when the temperature is higher than the threshold value for 0°C. P4.1 is toggled when the temperature is lower than the threshold value for 85°C. A dummy interrupt is triggered for the second interrupt event.

```
#include "xmc_gpio.h"
#include "xmc_scu.h"
#define LED0 P4_0
#define LED1 P4_1

/* Port pins output mode configuration */
XMC_GPIO_CONFIG_t LED_pin_config =
{
  .mode = XMC_GPIO_MODE_OUTPUT_PUSH_PULL,
  .output_level= 1U
};

void main(void)
{
      uint32_t temp_High_k;
      uint32_t temp_HighStatus;
      uint32_t temp_Low_k;
      uint32_t temp_LowStatus;

      /* Initialize port pins to output mode */
      XMC_GPIO_Init(LED0, &LED_pin_config);
      XMC_GPIO_Init(LED1, &LED_pin_config);

      /* Enable interrupt node 1 */
      NVIC_EnableIRQ(IRQ1_IRQn);

      /* Enable DTS */
      XMC_SCU_StartTempMeasurement();

      /* Convert DTS low temperature threshold value from °C to K */
      temp_Low_k = 85 + 273;
      temp_LowStatus = 0;

      /* Install DTS low temperature threshold value */
      temp_LowStatus = XMC_SCU_SetTempLowLimit(temp_Low_k);
      while (temp_LowStatus == 1)
      {
            temp_Low_k--;
```

```
            temp_LowStatus = XMC_SCU_SetTempLowLimit(temp_Low_k);
      }

      /* Enable service request on DTS temperature lower than expected
event*/
      XMC_SCU_INTERRUPT_EnableEvent(XMC_SCU_INTERRUPT_EVENT_TSE_LOW);

      /* Convert DTS high temperature threshold value from °C to K*/
      temp_High_k = 0 + 273;
      temp_HighStatus = 0;




      /* Install DTS high temperature threshold value*/
      temp_HighStatus = XMC_SCU_SetTempHighLimit(temp_High_k);
      while (temp_HighStatus == 1)
      {
            temp_High_k++;
            temp_HighStatus = XMC_SCU_SetTempHighLimit(temp_High_k);
      }

      /* Enable service request on DTS temperature higher than expected
event*/
      XMC_SCU_INTERRUPT_EnableEvent(XMC_SCU_INTERRUPT_EVENT_TSE_HIGH);

      while(1);

}

void IRQ1_Handler(void)
{
      /* Check if DTS temperature higher than expected event has occurred */
      if (1==XMC_SCU_HighTemperature())
      {
      /* Clear DTS high temperature event status */
      XMC_SCU_INTERRUPT_ClearEventStatus(XMC_SCU_INTERRUPT_EVENT_TSE_HIGH);

      /* Disable service request on DTS temperature higher than expected
event*/
      XMC_SCU_INTERRUPT_DisableEvent(XMC_SCU_INTERRUPT_EVENT_TSE_HIGH);
```

```
        /* User code goes here .. */
        XMC_GPIO_ToggleOutput(LED0);
        }

        /* Check if DTS temperature lower than expected event has occurred */
        if (1==XMC_SCU_LowTemperature())
        {
        /* Clear DTS low temperature event status */
        XMC_SCU_INTERRUPT_ClearEventStatus(XMC_SCU_INTERRUPT_EVENT_TSE_LOW);

        /* Disable service request on DTS temperature lower than expected
event*/

        XMC_SCU_INTERRUPT_DisableEvent(XMC_SCU_INTERRUPT_EVENT_TSE_LOW);

        /* User code goes here .. */
        XMC_GPIO_ToggleOutput(LED1);
        }
}
```

2 External library functions supporting the temperature sensor

For the XMC1100, XMC1200 and XMC1300 AA-step and AB-step EES, ES user configuration Version $0002_{H,}$ the external library **XMC1000_CalcTemperature**, **XMC1000_CalcTSEVAR** functions are available for the equivalent DTS features mentioned in Chapter 1. The XMC1000_tseRoutine.c must be added to the DAVE™ project. Two sets of code examples are introduced based on the XMC1300 in DAVE™ Version 4 environment to illustrate the usage of these external library DTS functions.

When enabled, the temperature measurement starts and the result is stored via bit field TSE_MON of the Temperature Sensor Counter2 Monitor Register (ANATSEMON). After storing the result, the temperature sensor continues with the next measurement. The external library **XMC1000_CalcTemperature** function can be used to determine the current chip temperature using the TSE_MON value.

The temperature sensor is also capable of detecting low and high temperature events when the measurement result crosses the higher and/or lower threshold values. The threshold values are configurable via bit fields TSE_IL and TSE_IH in the temperature sensor low/high Temperature Interrupt Registers (ANATSEIL and ANATSEIH). The external library **XMC1000_CalcTSEVAR** function can be used to convert the temperature to the threshold values to be installed in the ANATSEIL and ANATSEIH registers.

The following sections provide more details for the previously mentioned DTS API functions.

2.1 Calculation of the current chip temperature

2.1.1 XMC1000_CalcTemperature

The specification of the **XMC1000_CalcTemperature** external library function is :

- Input parameter: none
- Return status: chip temperature in degree Kelvin
- Prototype: unsigned long integer XMC1000_CalcTemperature (void)

In the code shown below, any one of these ports, P0.0, P0.6 or P0.9 is toggled when the temperature is at 25°C, above 25°C or below 25°C. Prior to calling the library function to determine the current chip temperature, the temperature sensor needs to be enabled via bit TSE_EN in the ANATSECTRL register. The **XMC_SCU_StartTempMeasurement** XMC™ Lib function performs this configuration.

```
#include "xmc_gpio.h"
#include "xmc_scu.h"
#define LED0 P0_0
#define LED1 P0_6
#define LED2 P0_9


/* Port pins output mode configuration */
XMC_GPIO_CONFIG_t LED_pin_config =
{
  .mode = XMC_GPIO_MODE_OUTPUT_PUSH_PULL,
  .output_level= 1U
```

```
};

void main(void)
{
	uint32_t temp_C = 0;
	uint32_t temp_k = 0;
	uint32_t limit = 0;
	uint32_t delay = 10000;

	/* Initialize port pins to output mode */
	XMC_GPIO_Init(LED0, &LED_pin_config);
	XMC_GPIO_Init(LED1, &LED_pin_config);
	 XMC_GPIO_Init(LED2, &LED_pin_config);

	/* Enable DTS */
	XMC_SCU_StartTempMeasurement();

	while(1)
	{
		/* Calculate temperature of the chip in Kelvin */
		temp_k = XMC1000_CalcTemperature();

		/* Convert temperature to Celcius */
		temp_C = temp_k - 273;
		if(temp_C == 25)
		{
			XMC_GPIO_ToggleOutput(LED0);
		}
		else if (temp_C > 25)
		{
			XMC_GPIO_ToggleOutput(LED1);
		}
		else if (temp_C < 25)
		{
			XMC_GPIO_ToggleOutput(LED2);
		}
		while(--delay);
		delay = 10000;
	}
	return 0;
```

```
}
```

2.2 Conversion of temperature to threshold values for temperature comparison

2.2.1 XMC1000_CalcTSEVAR

The specification of the **XMC1000_CalTSEVAR** external library function is :

Input parameter: threshold temperature in degree Kelvin (permitted range 233 to 388)

Return status: equivalent threshold value for the temperature provided as an input parameter

Prototype: unsigned long XMC1000_CalcTSEVAR(uint32_t temperature)

In the code shown below, the DTS high temperature event happens when the temperature is above 0°C and the DTS low temperature event happens when the temperature is below 85°C. The temperature measurement is performed at room temperature. The appropriate threshold values shall be adapted according to the application. The **XMC1000_CalcTSEVAR** external library function is used to convert these temperature points to the threshold values. The threshold values are installed in the ANATSEIH and ANATSEIL registers. The actual measurement result is available at ANATSEMON. The result is compared against the configured limits in ANATSEIH and ANATSEIL registers. A high temperature event SRRAW.TSE_HIGH is triggered because ANATSEMON is less than ANATSEIH. A low temperature event SRRAW.TSE_LOW is triggered because ANATSEMON is more than ANATSEIL. If the function returns zero, add or minus 1 Kelvin to the input parameter and re-run the function. If the function execution is successful, the equivalent threshold value for the temperature (K) is returned.

In this example, interrupts are triggered for DTS high or DTS low temperature events. In the interrupt service routine, **IRQ1_Handler**, the interrupt is disabled to prevent continuous triggering of interrupts since the temperature does not change instantaneously. Upon entering the service routine for the first time, P0.0 is toggled when the temperature is higher than the threshold value for 0°C. P0.9 is toggled when the temperature is lower than the threshold value for 85°C. A dummy interrupt is triggered for the second interrupt event.

```
#include "xmc_gpio.h"
#include "xmc_scu.h"
#define LED0 P0_0
#define LED1 P0_9

/* Port pins output mode configuration */
XMC_GPIO_CONFIG_t LED_pin_config =
{
  .mode = XMC_GPIO_MODE_OUTPUT_PUSH_PULL,
  .output_level= 1U
};

void main(void)
{
       uint32_t temp_High_k;
```

```
	uint32_t temp_HighStatus;
	uint32_t temp_Low_k;
	uint32_t temp_LowStatus;

	/* Initialize port pins to output mode */
	XMC_GPIO_Init(LED0, &LED_pin_config);
	XMC_GPIO_Init(LED1, &LED_pin_config);

	/* Enable SCU interrupt node 1 */
	NVIC_EnableIRQ(SCU_1_IRQn);

	/* Enable DTS */
	XMC_SCU_StartTempMeasurement();

	/* Convert DTS low temperature threshold value from °C to K */
	temp_Low_k = 85 + 273;
	temp_LowStatus = 0;

	/* Convert temperature in Kelvin to threshold value and install DTS
      low temperature threshold value */
	while (temp_LowStatus == 0)
	{
		temp_LowStatus = XMC1000_CalcTSEVAR(temp_Low_k);
		if (temp_LowStatus == 0)
		{
			temp_Low_k--;
		}
	}
	SCU_ANALOG->ANATSEIL = temp_LowStatus;

	/* Enable service request on DTS temperature lower than expected
event*/
	XMC_SCU_INTERRUPT_EnableEvent(XMC_SCU_INTERRUPT_EVENT_TSE_LOW);

	/* Convert DTS high temperature threshold value from °C to K*/
	temp_High_k = 0 + 273;
	temp_HighStatus = 0;
```

```
      /* Convert temperature in Kelvin to threshold value and install DTS
     high temperature threshold value */
      while (temp_HighStatus == 0)
      {
            temp_HighStatus = XMC1000_CalcTSEVAR(temp_High_k);
            if (temp_HighStatus == 0)
            {
                  temp_High_k++;
            }
      }
      SCU_ANALOG->ANATSEIH = temp_HighStatus;

      /* Enable service request on DTS temperature higher than expected
     event*/
      XMC_SCU_INTERRUPT_EnableEvent(XMC_SCU_INTERRUPT_EVENT_TSE_HIGH);

      while(1);

}

void SCU_1_IRQHandler (void)
{
      /* Check if DTS temperature higher than expected event has occurred */
      if (1==XMC_SCU_HighTemperature())
      {
      /* Clear DTS high temperature event status */
      XMC_SCU_INTERRUPT_ClearEventStatus(XMC_SCU_INTERRUPT_EVENT_TSE_HIGH);

      /* Disable service request on DTS temperature higher than expected
event*/
      XMC_SCU_INTERRUPT_DisableEvent(XMC_SCU_INTERRUPT_EVENT_TSE_HIGH);

      /* User code goes here .. */
      XMC_GPIO_ToggleOutput(LED0);
      }

      /* Check if DTS temperature lower than expected event has occurred */
      if (1==XMC_SCU_LowTemperature())
      {
```

```
        /* Clear DTS low temperature event status */
        XMC_SCU_INTERRUPT_ClearEventStatus(XMC_SCU_INTERRUPT_EVENT_TSE_LOW);

        /* Disable service request on DTS temperature lower than expected
event*/

        XMC_SCU_INTERRUPT_DisableEvent(XMC_SCU_INTERRUPT_EVENT_TSE_LOW);

        /* User code goes here .. */
        XMC_GPIO_ToggleOutput(LED1);
        }
}
```

[1] A Reference. See the XMC1000 reference manual, data sheet at http://www.infineon.com/XMC1000

[2] A Reference. See DAVE™ at http://www.infineon.com/DAVE

Revision history

Major changes since the last revision

Page or Reference	Description of change
All	Changed in Application Note template.
All	Added code examples based on SCU XMC Lib functions in DAVE™ Version 4 environment.
All	Adapted code examples based on DTS external library functions in DAVE™ Version 4 environment.

Trademarks of Infineon Technologies AG
AURIX™, C166™, CanPAK™, CIPOS™, CoolGaN™, CoolMOS™, CoolSET™, CoolSiC™, CORECONTROL™, CROSSAVE™, DAVE™, DI-POL™, DrBlade™, EasyPIM™, EconoBRIDGE™, EconoDUAL™, EconoPACK™, EconoPIM™, EiceDRIVER™, eupec™, FCOS™, HITFET™, HybridPACK™, Infineon™, ISOFACE™, IsoPACK™, i-Wafer™, MIPAQ™, ModSTACK™, my-d™, NovalithIC™, OmniTune™, OPTIGA™, OptiMOS™, ORIGA™, POWERCODE™, PRIMARION™, PrimePACK™, PrimeSTACK™, PROFET™, PRO-SIL™, RASIC™, REAL3™, ReverSave™, SatRIC™, SIEGET™, SIPMOS™, SmartLEWIS™, SOLID FLASH™, SPOC™, TEMPFET™, thinQ!™, TRENCHSTOP™, TriCore™.

Trademarks updated August 2015

Other Trademarks
All referenced product or service names and trademarks are the property of their respective owners.

Edition 2016-02

Published by

Infineon Technologies AG

81726 Munich, Germany

© 2016 Infineon Technologies AG.
All Rights Reserved.

Do you have a question about this document?

Email: erratum@infineon.com

Document reference
AP32323

IMPORTANT NOTICE

The information contained in this application note is given as a hint for the implementation of the product only and shall in no event be regarded as a description or warranty of a certain functionality, condition or quality of the product. Before implementation of the product, the recipient of this application note must verify any function and other technical information given herein in the real application. Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind (including without limitation warranties of non-infringement of intellectual property rights of any third party) with respect to any and all information given in this application note.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer's technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

For further information on the product, technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies office (www.infineon.com).

WARNINGS

Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies' products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.