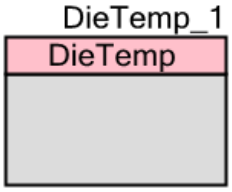
Die Temperature (DieTemp)
1.90



Features

- Accuracy of ±5 °C
- Range –40 °C to +140 °C (0xFFD8 to 0x008C)
- Blocking and non-blocking API
- Does not support PSoC 5 silicon

General Description

The Die Temperature (DieTemp) component provides an API to acquire the temperature of the die. The System Performance Controller (SPC) is used to get the die temperature. The API includes blocking and nonblocking calls.

When to Use a DieTemp

Use a DieTemp component when you want to measure the die temperature of a device.

Input/Output Connections

There are no Input/Output Connections on the DieTemp component. It is a software component only.

Component Parameters

The DieTemp has no configurable parameters other than standard Instance Name and Built-in parameters.

Application Programming Interface

Application Programming Interface (API) routines allow you to configure the component using software. The following table lists and describes the interface to each function. The subsequent sections cover each function in more detail.

By default, PSoC Creator assigns the instance name “DieTemp_1” to the first instance of a component in a given design. You can rename it to any unique value that follows the syntactic

rules for identifiers. The instance name becomes the prefix of every global function name, variable, and constant symbol. For readability, the instance name used in the following table is "DieTemp."

Function	Description
DieTemp_Start()	Starts the SPC command to get the die temperature
DieTemp_Stop()	Stops the temperature reading
DieTemp_Query()	Queries the SPC to see if the temperature command is finished
DieTemp_GetTemp()	Sets up the command to get the temperature and blocks until finished

cystatus DieTemp_Start(void)

Description: Sends the command and parameters to the SPC to start a Die Temperature reading. This function returns before the SPC finishes. This function call must always be paired with a call to the DieTemp_Query() API to complete the Die Temperature reading..

Parameters: void

Return Value: CYRET_STARTED if the SPC command was started successfully.
CYRET_UNKNOWN if the SPC command failed.
CYRET_LOCKED if the SPC was busy.

Side Effects: None

void DieTemp_Stop(void)

Description: Stops the temperature reading.

Parameters: None

Return Value: None

Side Effects: None

cystatus DieTemp_Query(int16 * temperature)

Description: Checks to see if the SPC command started by DieTemp_Start() has finished. If the command has not finished, the temperature value is not read and returned. The caller will need to poll this function while the return status remains CYRET_STARTED.

This can be used only in conjunction with the DieTemp_Start() API to successfully get the correct Die Temperature.

The Die Temperature reading returned on the first sequence of DieTemp_Start() followed by DieTemp_Query() can be unreliable, so you must do this sequence twice and use the value returned from the second sequence.

Parameters: int16 * temperature: Address to store the temperature in degrees Celsius

Return Value: CYRET_SUCCESS if the temperature command completed successfully.

CYRET_UNKNOWN if the there was an SPC failure.

CYRET_STARTED if the temperature command has not completed.

CYRET_TIMEOUT if waited too long before reading data.

Side Effects: None

cystatus DieTemp_GetTemp(int16 * temperature)

Description: Sends the command and parameters to the SPC to start a Die Temperature reading and waits until it fails or completes. This is a blocking API.

This function reads the Die Temperature twice and returns the second value to work around an issue in the silicon that causes the first value read to be unreliable.

Parameters: int16 * temperature: Address to store the temperature in degree of Celsius

Return Value: CYRET_SUCCESS if the command was completed successfully.

Along with additional status codes from DieTemp_Start() or DieTemp_Query().

Side Effects: None

Sample Firmware Source Code

PSoC Creator provides many example projects that include schematics and example code in the Find Example Project dialog. For component-specific examples, open the dialog from the Component Catalog or an instance of the component in a schematic. For general examples, open the dialog from the Start Page or **File** menu. As needed, use the **Filter Options** in the dialog to narrow the list of projects available to select.

Refer to the “Find Example Project” topic in the PSoC Creator Help for more information.

Resources

The DieTemp uses the on-chip Temperature sensor to measure the internal die temperature.

API Memory Usage

The component memory usage varies significantly, depending on the compiler, device, number of APIs used and component configuration. The following table provides the memory usage for all APIs available in the given component configuration.

The measurements have been done with the associated compiler configured in Release mode with optimization set for Size. For a specific design the map file generated by the compiler can be analyzed to determine the memory usage.

Configuration	PSoC 3 (Keil_PK51)		PSoC 5 (GCC)		PSoC 5LP (GCC)	
	Flash Bytes	SRAM Bytes	Flash Bytes	SRAM Bytes	Flash Bytes	SRAM Bytes
Default	339	0	N/A	N/A	210	0

DC and AC Electrical Characteristics

Specifications are valid for –40 °C ≤ T_A ≤ 85 °C and T_J ≤ 100 °C, except where noted.
Specifications are valid for 1.71 V to 5.5 V, except where noted.

Parameter	Description	Conditions	Min	Typical	Max	Units
	Temp sensor accuracy	Range: –40 °C to +85 °C	–	±5	–	°C

Component Changes

This section lists the major changes in the component from the previous version.

Version	Description of Changes	Reason for Changes / Impact
1.90.a	Minor datasheet edit.	
1.90	Fixed the issue that caused temperature reading errors on PSoC 5LP when the code was compiled with speed optimization.	On PSoC 5LP, the second temperature reading request was performed when SPC was not ready (in some cases).
1.80	Added PSoC 5LP support.	
	DieTemp APIs were updated.	Due to changes to the cyboot SPC APIs.
1.70	The DieTemp_GetTemp() API was changed to read the DieTemp twice to fix an issue.	The DieTemp_GetTemp() API was returning the temperature even if the first read was unsuccessful.
1.60	The *DieTemp.c* file GetTemp API was edited to fix the power-on reset error output.	The DieTemp output on power-on reset was erroneous.
1.50.a	Added characterization data to datasheet	
	Added information to the component that advertizes its compatibility with silicon revisions.	The tool returns an error/warns if the component is used on incompatible silicon. This component is not compatible with PSoC 3 ES2 or PSoC 5.
	Minor datasheet edits and updates	
1.50	Switch from *cydevice.h* to *cydevice_trm.h*.	The *cydevice.h* file has been made obsolete, so the APIs and generated code provided with PSoC Creator are not included with *cydevice_trm.h*.

© Cypress Semiconductor Corporation, 2012-2015. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

PSoC® is a registered trademark, and PSoC Creator™ and Programmable System-on-Chip™ are trademarks of Cypress Semiconductor Corp. All other trademarks or registered trademarks referenced herein are property of the respective corporations.

Any Source Code (software and/or firmware) is owned by Cypress Semiconductor Corporation (Cypress) and is protected by and subject to worldwide patent protection (United States and foreign), United States copyright laws and international treaty provisions. Cypress hereby grants to licensee a personal, non-exclusive, non-transferable license to copy, use, modify, create derivative works of, and compile the Cypress Source Code and derivative works for the sole purpose of creating custom software and or firmware in support of licensee product to be used only in conjunction with a Cypress integrated circuit as specified in the applicable agreement. Any reproduction, modification, translation, compilation, or representation of this Source Code except as specified above is prohibited without the express written permission of Cypress.

Disclaimer: CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress' product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

Use may be limited by and subject to the applicable Cypress software license agreement.