Please note that Cypress is an Infineon Technologies Company.
The document following this cover page is marked as “Cypress” document as this is the company that originally developed the product. Please note that Infineon will continue to offer the product to new and existing customers as part of the Infineon product portfolio.

**Continuity of document content**
The fact that Infineon offers the following product as part of the Infineon product portfolio does not lead to any changes to this document. Future revisions will occur when appropriate, and any changes will be set out on the document history page.

**Continuity of ordering part numbers**
Infineon continues to support existing part numbers. Please continue to use the ordering part numbers listed in the datasheet for ordering.
Die Temperature (DieTemp)

Features

- Accuracy of ±5 °C
- Range –40 °C to +140 °C (0xFFD8 to 0x008C)
- Blocking and non-blocking API

General Description

The Die Temperature (DieTemp) component provides an Application Programming Interface (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 non-blocking 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.”

Note that the device should not enter low power modes during temperature measurements. After calling DieTemp_Start() you should wait for the DieTemp_Query() API to report the status of the request that is different from CYRET_STARTED. For more information see the DieTemp_Query() API description.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DieTemp_Start()</td>
<td>Starts the SPC command to get the die temperature</td>
</tr>
<tr>
<td>DieTemp_Stop()</td>
<td>Stops the temperature reading</td>
</tr>
<tr>
<td>DieTemp_Query()</td>
<td>Queries the SPC to see if the temperature command is finished</td>
</tr>
<tr>
<td>DieTemp_GetTemp()</td>
<td>Sets up the command to get the temperature and blocks until finished</td>
</tr>
</tbody>
</table>

```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: There is no need to stop or disable this component. This component is naturally a slave that sends request to SPC through SPC API of cy_boot and waits for data to be ready or in case of non-blocking operation it sends request to SPC and user can manually poll the result.

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.

MISRA Compliance

This section describes the MISRA-C:2004 compliance and deviations for the component. There are two types of deviations defined:

▪ project deviations – deviations that are applicable for all PSOC Creator components
▪ specific deviations – deviations that are applicable only for this component

This section provides information on component-specific deviations. Project deviations are described in the MISRA Compliance section of the System Reference Guide along with information on the MISRA compliance verification environment.

The DieTemp component does not have any specific deviations.

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.

<table>
<thead>
<tr>
<th>Configuration</th>
<th>PSOC 3 (Keil_PK51)</th>
<th>PSOC 5LP (GCC)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Flash Bytes</td>
<td>SRAM Bytes</td>
</tr>
<tr>
<td>Default</td>
<td>339</td>
<td>0</td>
</tr>
</tbody>
</table>

Resources

The DieTemp uses the on-chip Temperature sensor to measure the internal die temperature.
**DC and AC Electrical Characteristics**

Specifications are valid for $-40 \, ^\circ C \leq T_A \leq 85 \, ^\circ C$ and $T_J \leq 100 \, ^\circ C$, except where noted.

Specifications are valid for $1.71 \, V$ to $5.5 \, V$, except where noted.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Conditions</th>
<th>Min</th>
<th>Typical</th>
<th>Max</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>Temp sensor accuracy</td>
<td>Range: $-40 , ^\circ C$ to $+85 , ^\circ C$</td>
<td>$-$</td>
<td>$\pm 5$</td>
<td>$-$</td>
<td>$^\circ C$</td>
<td></td>
</tr>
</tbody>
</table>

**Component Changes**

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

<table>
<thead>
<tr>
<th>Version</th>
<th>Description of Changes</th>
<th>Reason for Changes / Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.0.d</td>
<td>Minor datasheet edits.</td>
<td></td>
</tr>
<tr>
<td>2.0.c</td>
<td>Minor datasheet edits.</td>
<td></td>
</tr>
<tr>
<td>2.0.b</td>
<td>Datasheet update.</td>
<td>Added a note to not enter low power modes during temperature measurements.</td>
</tr>
<tr>
<td>2.0.a</td>
<td>Minor datasheets edits and updates. Removed PSoC 5 references.</td>
<td>PSoC 5 replaced by PSoC 5LP.</td>
</tr>
<tr>
<td>2.0</td>
<td>Removed CySpcStop() API call from DieTemp_Stop().</td>
<td>SPC cannot be stopped from runtime chip operation.</td>
</tr>
<tr>
<td></td>
<td>Added MISRA Compliance section.</td>
<td>The component does not have any specific deviations.</td>
</tr>
<tr>
<td>1.80</td>
<td>Added PSoC 5LP support.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DieTemp APIs were updated.</td>
<td>Due to changes to the cyboot SPC APIs.</td>
</tr>
<tr>
<td>1.70</td>
<td>The DieTemp_GetTemp() API was changed to read the DieTemp twice to fix an issue.</td>
<td>The DieTemp_GetTemp() API was returning the temperature even if the first read was unsuccessful.</td>
</tr>
<tr>
<td>1.60</td>
<td>The DieTemp.c file GetTemp API was edited to fix the power-on reset error output.</td>
<td>The DieTemp output on power-on reset was erroneous.</td>
</tr>
<tr>
<td>1.50.a</td>
<td>Added characterization data to datasheet</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Added information to the component that advertizes its compatibility with silicon revisions.</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>Minor datasheet edits and updates</td>
<td></td>
</tr>
<tr>
<td>1.50</td>
<td>Switch from cydevice.h to cydevice_trm.h.</td>
<td>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.</td>
</tr>
</tbody>
</table>