Introduction

This driver for Atmel® | SMART ARM®-based microcontrollers provides an interface for the configuration and management of the device's Configurable Custom Logic functionality.

The following peripheral is used by this module:

- TSENS (Temperature Sensor)

The following devices can use this module:

- Atmel | SMART SAM C21

The outline of this documentation is as follows:

- Prerequisites
- Module Overview
- Special Considerations
- Extra Information
- Examples
- API Overview
# Table of Contents

Introduction......................................................................................................................1

1. Software License....................................................................................................... 4

2. Prerequisites..............................................................................................................5

3. Module Overview.......................................................................................................6
   3.1. Window Monitor............................................................................................................................6
   3.2. Events...........................................................................................................................................6

4. Special Considerations..............................................................................................7

5. Extra Information....................................................................................................... 8

6. Examples...................................................................................................................9

7. API Overview...........................................................................................................10
   7.1. Variable and Type Definitions.................................................................................................10
       7.1.1. Type tsens_callback_t................................................................................................. 10
   7.2. Structure Definitions................................................................................................................... 10
       7.2.1. Struct tsens_calibration............................................................................................... 10
       7.2.2. Struct tsens_config...................................................................................................... 10
       7.2.3. Struct tsens_events..................................................................................................... 10
       7.2.4. Struct tsens_module.................................................................................................... 11
       7.2.5. Struct tsens_window_config.........................................................................................11
   7.3. Macro Definitions........................................................................................................................11
       7.3.1. Module Status Flags.................................................................................................... 11
       7.3.2. Macro ERRATA_14476................................................................................................12
   7.4. Function Definitions....................................................................................................................12
       7.4.1. Driver Initialization and Configuration.......................................................................... 12
       7.4.2. Status Management.....................................................................................................13
       7.4.3. Enable, Disable, and Reset TSENS Module, Start Conversion and Read Result.......14
       7.4.4. Callback Management................................................................................................. 16
   7.5. Enumeration Definitions............................................................................................................. 17
       7.5.1. Enum tsens_callback...................................................................................................17
       7.5.2. Enum tsens_event_action........................................................................................... 18
       7.5.3. Enum tsens_window_mode.........................................................................................18

8. Extra Information for TSENS Driver.........................................................................19
   8.1. Acronym..................................................................................................................................... 19
   8.2. Dependencies.............................................................................................................................19
   8.3. Errata..........................................................................................................................................19
   8.4. Module History............................................................................................................................19

9. Examples for TSENS Driver.....................................................................................20
   9.1. Quick Start Guide for TSENS - Basic.........................................................................................20
9.1.1. Setup ........................................................................................................................... 20
9.1.2. Use Case .................................................................................................................. 21
9.2. Quick Start Guide for TSENS - Callback ..................................................................... 21
  9.2.1. Setup .................................................................................................................... 22
  9.2.2. Use Case .............................................................................................................. 23

10. Document Revision History ........................................................................................ 25
1. **Software License**

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. The name of Atmel may not be used to endorse or promote products derived from this software without specific prior written permission.

4. This software may only be redistributed and used in connection with an Atmel microcontroller product.

THIS SOFTWARE IS PROVIDED BY ATMEL "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT ARE EXPRESSLY AND SPECIFICALLY DISCLAIMED. IN NO EVENT SHALL ATMEL BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
2. **Prerequisites**

There are no prerequisites for this module.
3. **Module Overview**

The Temperature Sensor (TSENS) can be used to accurately measure the operating temperature of the device. TSENS accurately measures the operating temperature of the device by comparing the difference in two temperature dependent frequencies to a known frequency. The frequency of the temperature dependent oscillator (TOSC) is measured twice: first with the min configuration and next with the maximum configuration. The resulting signed value is proportional to the temperature and is corrected for offset by the contents of the OFFSET register.

Accurately measures a temperature:

- ±1°C over 0°C ~ 60°C
- ±3°C over -40°C ~ 85°C
- ±5°C over -40°C ~ 105°C

The number of periods of GCLK_TSENS used for the measurement is defined by the GAIN register. The width of the resulting pulse is measured using a counter clocked by GCLK_TSENS in the up direction for the 1st phase and in the down 2nd phase. Register GAIN and OFFSET is loaded from NVM, or can also be fixed by user.

\[
VALUE = OFFSET + \left( \frac{f_{TOSCMIN} - f_{TOSCMAX}}{f_{GCLK}} \right) \times GAIN
\]

**Note:** If fix this bitfield, the relationship between GCLK frequency, GAIN and resolution as below:

<table>
<thead>
<tr>
<th>Resolution (°C)</th>
<th>GAIN@48MHz</th>
<th>GAIN@40MHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (1°C)</td>
<td>960</td>
<td>800</td>
</tr>
<tr>
<td>10 (0.1°C)</td>
<td>9600</td>
<td>8000</td>
</tr>
<tr>
<td>100 (0.01°C)</td>
<td>96000</td>
<td>80000</td>
</tr>
</tbody>
</table>

3.1. **Window Monitor**

The TSENS module window monitor function can be used to automatically compare the conversion result against a predefined pair of upper and lower threshold values.

3.2. **Events**

Event generation and event actions are configurable in the TSENS.

The TSENS has one actions that can be triggered upon event reception:

- Start conversion

The TSENS can generate the following output event:

- Window monitor

If the event actions are enabled in the configuration, any incoming event will trigger the action.

If the window monitor event is enabled, an event will be generated when the configured window condition is detected.
4. **Special Considerations**

There are no special considerations for this module.
5. Extra Information

For extra information, see Extra Information for TSENS Driver. This includes:

- Acronym
- Dependencies
- Errata
- Module History
6. **Examples**

For a list of examples related to this driver, see *Examples for TSENS Driver.*
7. **API Overview**

7.1. **Variable and Type Definitions**

7.1.1. **Type tsens_callback_t**

```c
typedef void(* tsens_callback_t )(enum tsens_callback)
```

Type of the callback functions.

7.2. **Structure Definitions**

7.2.1. **Struct tsens_calibration**

Calibration configuration structure.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint32_t</td>
<td>gain</td>
<td>Time amplifier gain</td>
</tr>
<tr>
<td>int32_t</td>
<td>offset</td>
<td>Offset correction</td>
</tr>
</tbody>
</table>

7.2.2. **Struct tsens_config**

Configuration structure for an TSENS instance. This structure should be initialized by the `tsens_get_config_defaults()` function before being modified by the user application.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>struct tsens_calibration</td>
<td>calibration</td>
<td>Calibration value</td>
</tr>
<tr>
<td>enum gclk_generator</td>
<td>clock_source</td>
<td>GCLK generator used to clock the peripheral</td>
</tr>
<tr>
<td>enum tsens_event_action</td>
<td>event_action</td>
<td>Event action to take on incoming event</td>
</tr>
<tr>
<td>bool</td>
<td>free_running</td>
<td>Enables free running mode if true</td>
</tr>
<tr>
<td>bool</td>
<td>run_in_standby</td>
<td>Enables TSENS in standby sleep mode if true</td>
</tr>
<tr>
<td>struct tsens_window_config</td>
<td>window</td>
<td>Window monitor configuration structure</td>
</tr>
</tbody>
</table>

7.2.3. **Struct tsens_events**

Event flags for the TSENS module. This is used to enable and disable events via `tsens_enable_events()` and `tsens_disable_events()`.
Table 7-3. Members

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool</td>
<td>generate_event_on_window_monitor</td>
<td>Enable event generation on window monitor</td>
</tr>
</tbody>
</table>

7.2.4. **Struct tsens_module**

TSENS software instance structure, used to retain software state information of an associated hardware module instance.

**Note:** The fields of this structure should not be altered by the user application; they are reserved for module-internal use only.

7.2.5. **Struct tsens_window_config**

Window monitor configuration structure.

Table 7-4. Members

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int32_t</td>
<td>window_lower_value</td>
<td>Lower window value</td>
</tr>
<tr>
<td>enum tsens_window_mode</td>
<td>window_mode</td>
<td>Selected window mode</td>
</tr>
<tr>
<td>int32_t</td>
<td>window_upper_value</td>
<td>Upper window value</td>
</tr>
</tbody>
</table>

7.3. **Macro Definitions**

7.3.1. **Module Status Flags**

TSENS status flags, returned by `tsens_get_status()` and cleared by `tsens_clear_status()`.

7.3.1.1. **Macro TSENS_STATUS_RESULT_READY**

```c
#define TSENS_STATUS_RESULT_READY
TSENS result ready.
```

7.3.1.2. **Macro TSENS_STATUS_OVERRUN**

```c
#define TSENS_STATUS_OVERRUN
TSENS result overwritten before read.
```

7.3.1.3. **Macro TSENS_STATUS_WINDOW**

```c
#define TSENS_STATUS_WINDOW
Window monitor match.
```

7.3.1.4. **Macro TSENS_STATUS_OVERFLOW**

```c
#define TSENS_STATUS_OVERFLOW
TSENS result overflows.
7.3.2. Macro ERRATA_14476

#define ERRATA_14476

The magnitude of the temperature measurement value decreases with increasing temperature, i.e. it has a negative temperature coefficient. Errata reference: 14476.

7.4. Function Definitions

7.4.1. Driver Initialization and Configuration

7.4.1.1. Function tsens_init()

Initializes the TSENS.

```c
enum status_code tsens_init(
    struct tsens_config * config)
```

Initializes the TSENS device struct and the hardware module based on the given configuration struct values.

Table 7-5. Parameters

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>config</td>
<td>Pointer to the configuration struct</td>
</tr>
</tbody>
</table>

Returns

Status of the initialization procedure.

Table 7-6. Return Values

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATUS_OK</td>
<td>The initialization was successful</td>
</tr>
<tr>
<td>STATUS_ERR_INVALID_ARG</td>
<td>Invalid argument(s) were provided</td>
</tr>
<tr>
<td>STATUS_BUSY</td>
<td>The module is busy with a reset operation</td>
</tr>
<tr>
<td>STATUS_ERR_DENIED</td>
<td>The module is enabled</td>
</tr>
</tbody>
</table>

7.4.1.2. Function tsens_get_config_defaults()

Initializes an TSENS configuration structure to defaults.

```c
void tsens_get_config_defaults(
    struct tsens_config *const config)
```

Initializes a given TSENS configuration struct to a set of known default values. This function should be called on any new instance of the configuration struct before being modified by the user application.

The default configuration is as follows:

- GCLK generator 0 (GCLK main) clock source
- All events (input and generation) disabled
- Free running disabled
- Run in standby disabled
- Window monitor disabled
- Register GAIN value
- Register OFFSET value

**Note:** Register GAIN and OFFSET is loaded from NVM, or can also be fixed. If this bitfield is to be fixed, pay attention to the relationship between GCLK frequency, GAIN, and resolution. See Chapter Module Overview.

### Table 7-7. Parameters

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[out]</td>
<td>config</td>
<td>Pointer to configuration struct to initialize to default values</td>
</tr>
</tbody>
</table>

#### 7.4.2. Status Management

#### 7.4.2.1. Function tsens_get_status()

Retrieves the current module status.

```c
uint32_t tsens_get_status( void )
```

Retrieves the status of the module, giving overall state information.

**Returns**

Bit mask of TSENS status flags.

### Table 7-8. Return Values

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TSENS_STATUS_RESULT_READY</td>
<td>TSENS result is ready to be read</td>
</tr>
<tr>
<td>TSENS_STATUS_OVERRUN</td>
<td>TSENS result overwritten before read</td>
</tr>
<tr>
<td>TSENS_STATUS_WINDOW</td>
<td>TSENS has detected a value inside the set window range</td>
</tr>
<tr>
<td>TSENS_STATUS_OVERFLOW</td>
<td>TSENS result overflows</td>
</tr>
</tbody>
</table>

#### 7.4.2.2. Function tsens_clear_status()

Clears a module status flag.

```c
void tsens_clear_status(
    const uint32_t status_flags)
```

Clears the given status flag of the module.

### Table 7-9. Parameters

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>module_inst</td>
<td>Pointer to the TSENS software instance struct</td>
</tr>
<tr>
<td>[in]</td>
<td>status_flags</td>
<td>Bitmask of TSENS_STATUS_* flags to clear</td>
</tr>
</tbody>
</table>
7.4.3. Enable, Disable, and Reset TSENS Module, Start Conversion and Read Result

7.4.3.1. Function tsens_is_syncing()

Determines if the hardware module is currently synchronizing to the bus.

```c
bool tsens_is_syncing( void )
```

Checks to see if the underlying hardware peripheral module(s) are currently synchronizing across multiple clock domains to the hardware bus. This function can be used to delay further operations on a module until such time that it is ready, to prevent blocking delays for synchronization in the user application.

**Returns**

Synchronization status of the underlying hardware module(s).

Table 7-10. Return Values

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>If the module synchronization is ongoing</td>
</tr>
<tr>
<td>false</td>
<td>If the module has completed synchronization</td>
</tr>
</tbody>
</table>

7.4.3.2. Function tsens_enable()

Enables the TSENS module.

```c
void tsens_enable( void )
```

Enables an TSENS module that has previously been configured.

7.4.3.3. Function tsens_disable()

Disables the TSENS module.

```c
void tsens_disable( void )
```

Disables an TSENS module that was previously enabled.

7.4.3.4. Function tsens_reset()

Resets the TSENS module.

```c
void tsens_reset( void )
```

Resets an TSENS module, clearing all module state and registers to their default values.

7.4.3.5. Function tsens_enable_events()

Enables an TSENS event output.

```c
void tsens_enable_events(
    struct tsens_events *const events)
```

Enables one or more input or output events to or from the TSENS module. See `tsens_events` for a list of events this module supports.

**Note:** Events cannot be altered while the module is enabled.
Table 7-11. Parameters

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>events</td>
<td>Struct containing flags of events to enable</td>
</tr>
</tbody>
</table>

### 7.4.3.6. Function tsens_disable_events()
Disables an TSENS event output.

```c
void tsens_disable_events(
    struct tsens_events *const events)
```

Disables one or more output events to or from the TSENS module. See `tsens_events` for a list of events this module supports.

**Note:** Events cannot be altered while the module is enabled.

Table 7-12. Parameters

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>events</td>
<td>Struct containing flags of events to disable</td>
</tr>
</tbody>
</table>

### 7.4.3.7. Function tsens_start_conversion()
Start a TSENS conversion.

```c
void tsens_start_conversion( void )
```

Start a new TSENS conversion.

### 7.4.3.8. Function tsens_read()
Reads the TSENS result.

```c
enum status_code tsens_read(
    int32_t * result)
```

Reads the result from a TSENS conversion that was previously started.

Table 7-13. Parameters

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[out]</td>
<td>result</td>
<td>Pointer to store the result value in</td>
</tr>
</tbody>
</table>

**Returns**
Status of the TSENS read request.
Table 7-14. Return Values

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATUS_OK</td>
<td>The result was retrieved successfully</td>
</tr>
<tr>
<td>STATUS_BUSY</td>
<td>A conversion result was not ready</td>
</tr>
<tr>
<td>STATUS_ERR_OVERFLOW</td>
<td>The result register has been overwritten by the TSENS module before the result was read by the software</td>
</tr>
</tbody>
</table>

7.4.4. Callback Management

7.4.4.1. Function tsens_register_callback()

Registers a callback.

```c
enum status_code tsens_register_callback(
    struct tsens_module *const module,
    tsens_callback_t callback_func,
    enum tsens_callback callback_type)
```

Registers a callback function which is implemented by the user.

**Note:** The callback must be enabled by for the interrupt handler to call it when the condition for the callback is met.

Table 7-15. Parameters

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>module</td>
<td>Pointer to TSENS software instance struct</td>
</tr>
<tr>
<td>[in]</td>
<td>callback_func</td>
<td>Pointer to callback function</td>
</tr>
<tr>
<td>[in]</td>
<td>callback_type</td>
<td>Callback type given by an enum</td>
</tr>
</tbody>
</table>

7.4.4.2. Function tsens_unregister_callback()

Unregisters a callback.

```c
enum status_code tsens_unregister_callback(
    struct tsens_module *const module,
    enum tsens_callback callback_type)
```

Unregisters a callback function which is implemented by the user.

Table 7-16. Parameters

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>module</td>
<td>Pointer to TSENS software instance struct</td>
</tr>
<tr>
<td>[in]</td>
<td>callback_type</td>
<td>Callback type given by an enum</td>
</tr>
</tbody>
</table>
7.4.3. **Function tsens_enable_callback()**

Enables callback.

```c
void tsens_enable_callback(
    enum tsens_callback callback_type)
```

Enables the callback function registered by `tsens_register_callback`. The callback function will be called from the interrupt handler when the conditions for the callback type are met.

**Table 7-17. Parameters**

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>callback_type</td>
<td>Callback type given by an enum</td>
</tr>
</tbody>
</table>

7.4.4. **Function tsens_disable_callback()**

Disables callback.

```c
void tsens_disable_callback(
    enum tsens_callback callback_type)
```

Disables the callback function registered by the `tsens_register_callback`.

**Table 7-18. Parameters**

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>callback_type</td>
<td>Callback type given by an enum</td>
</tr>
</tbody>
</table>

7.4.5. **Function tsens_read_job()**

Read result from TSENS.

```c
void tsens_read_job(
    struct tsens_module *const module_inst,
    int32_t * result)
```

**Table 7-19. Parameters**

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>module_inst</td>
<td>Pointer to the TSENS software instance struct</td>
</tr>
<tr>
<td>[out]</td>
<td>result</td>
<td>Pointer to store the TSENS result</td>
</tr>
</tbody>
</table>

7.5. **Enumeration Definitions**

7.5.1. **Enum tsens_callback**

Callback types for TSENS callback driver.
Table 7-20. Members

<table>
<thead>
<tr>
<th>Enum value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TSENS_CALLBACK_RESULT_READY</td>
<td>Callback for result ready</td>
</tr>
<tr>
<td>TSENS_CALLBACK_OVERRUN</td>
<td>Callback when result overwritten before read</td>
</tr>
<tr>
<td>TSENS_CALLBACK_WINDOW</td>
<td>Callback when window is hit</td>
</tr>
<tr>
<td>TSENS_CALLBACK_OVF</td>
<td>Callback when the result overflows</td>
</tr>
</tbody>
</table>

7.5.2. Enum tsens_event_action

Enum for the possible actions to take on an incoming event.

Table 7-21. Members

<table>
<thead>
<tr>
<th>Enum value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TSENS_EVENT_ACTION_DISABLED</td>
<td>Event action disabled</td>
</tr>
<tr>
<td>TSENS_EVENT_ACTION_START_CONV</td>
<td>Start conversion</td>
</tr>
</tbody>
</table>

7.5.3. Enum tsens_window_mode

Enum for the possible window monitor modes for the TSENS.

Table 7-22. Members

<table>
<thead>
<tr>
<th>Enum value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TSENS_WINDOW_MODE_DISABLE</td>
<td>No window mode</td>
</tr>
<tr>
<td>TSENS_WINDOW_MODE_ABOVE</td>
<td>RESULT &gt; WINLT</td>
</tr>
<tr>
<td>TSENS_WINDOW_MODE_BELOW</td>
<td>RESULT &lt; WINUT</td>
</tr>
<tr>
<td>TSENS_WINDOW_MODE_INSIDE</td>
<td>WINLT &lt; RESULT &lt; WINUT</td>
</tr>
<tr>
<td>TSENS_WINDOW_MODE_OUTSIDE</td>
<td>!(WINLT &lt; RESULT &lt; WINUT)</td>
</tr>
<tr>
<td>TSENS_WINDOW_MODE_HYST_ABOVE</td>
<td>VALUE &gt; WINUT with hysteresis to WINLT</td>
</tr>
<tr>
<td>TSENS_WINDOW_MODE_HYST BELOW</td>
<td>VALUE &lt; WINLT with hysteresis to WINUT</td>
</tr>
</tbody>
</table>
8. Extra Information for TSENS Driver

8.1. Acronym

Below is a table listing the acronym used in this module, along with their intended meaning.

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TSENS</td>
<td>Temperature Sensor</td>
</tr>
</tbody>
</table>

8.2. Dependencies

This driver has no dependencies.

8.3. Errata

Errata reference: 14476.

The magnitude of the temperature measurement value decreases with increasing temperature, i.e. it has a negative temperature coefficient.

8.4. Module History

An overview of the module history is presented in the table below, with details on the enhancements and fixes made to the module since its first release. The current version of this corresponds to the newest version in the table.

<table>
<thead>
<tr>
<th>Changelog</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial Release</td>
</tr>
</tbody>
</table>
9. **Examples for TSENS Driver**

This is a list of the available Quick Start guides (QSGs) and example applications for SAM Temperature Sensor (TSENS) Driver. QSGs are simple examples with step-by-step instructions to configure and use this driver in a selection of use cases. Note that a QSG can be compiled as a standalone application or be added to the user application.

- Quick Start Guide for TSENS - Basic
- Quick Start Guide for TSENS - Callback

9.1. **Quick Start Guide for TSENS - Basic**

In this use case, the TSENS will be configured with the following settings:

- GCLK generator 0 (GCLK main) clock source
- Free running disabled
- Run in standby
- Window monitor disabled
- All events (input and generation) disabled
- Calibration value which read from NVM or user set

9.1.1. **Setup**

9.1.1.1. **Prerequisites**

There are no special setup requirements for this use-case.

Copy-paste the following setup code to your user application:

```c
void configure_tsens(void)
{
    struct tsens_config config_tsens;
    tsens_get_config_defaults(&config_tsens);
    tsens_init(&config_tsens);
    tsens_enable();
}
```

Add to user application initialization (typically the start of `main()`):

```c
configure_tsens();
```

9.1.1.2. **Workflow**

1. Configure the TSENS module.
   1. Create a TSENS module configuration struct, which can be filled out to adjust the configuration of a physical TSENS peripheral.
   ```c
   struct tsens_config config_tsens;
   ```
   2. Initialize the TSENS configuration struct with the module's default values.
   ```c
   tsens_get_config_defaults(&config_tsens);
   ```

**Note:** This should always be performed before using the configuration struct to ensure that all values are initialized to known default settings.
3. Set TSENS configurations.
   
   ```c
   tsens_init(&config_tsens);
   ```

4. Enable the TSENS module so that conversions can be made.
   
   ```c
   tsens_enable();
   ```

9.1.2. **Use Case**

9.1.2.1. **Code**

Copy-paste the following code to your user application:

```c
int32_t result;

tsens_start_conversion();

do {
    /* Wait for conversion to be done and read out result */
} while (tsens_read(&result) != STATUS_OK);

while (1) {
    /* Infinite loop */
}
```

9.1.2.2. **Workflow**

1. Start conversion.
   
   ```c
   tsens_start_conversion();
   ```

2. Wait until conversion is done and read result.
   
   ```c
   int32_t result;
   tsens_start_conversion();

   do {
       /* Wait for conversion to be done and read out result */
   } while (tsens_read(&result) != STATUS_OK);

   while (1) {
       /* Infinite loop */
   }
```

3. Enter an infinite loop once the conversion is complete.
   
   ```c
   while (1) {
       /* Infinite loop */
   }
```

9.2. **Quick Start Guide for TSENS - Callback**

In this use case, the TSENS will measure the temperature using interrupt driven conversion. When the temperature value has been measured, a callback will be called that signals the main application that the conversion is complete.

The TSENS will be set up as follows:

- GCLK generator 0 (GCLK main) clock source
- Free running disabled
- Run in standby
- Window monitor disabled
9.2.1. Setup

9.2.1.1. Prerequisites

There are no special setup requirements for this use-case.

9.2.1.2. Code

Add to the main application source file, outside of any functions:

```c
struct tsens_module tsens_instance;

int32_t tsens_result;
```

Callback function:

```c
volatile bool tsens_read_done = false;

static void tsens_complete_callback(enum tsens_callback i) {
    tsens_read_done = true;
}
```

Copy-paste the following setup code to your user application:

```c
static void configure_tsens(void) {
    struct tsens_config config_tsens;
    tsens_get_config_defaults(&config_tsens);
    tsens_init(&config_tsens);
    tsens_enable();
}

static void configure_tsens_callbacks(void) {
    tsens_register_callback(&tsens_instance, tsens_complete_callback, TSENS_CALLBACK_RESULT_READY);
    tsens_enable_callback(TSENS_CALLBACK_RESULT_READY);
}
```

Add to user application initialization (typically the start of `main()`):

```c
configure_tsens();
configure_tsens_callbacks();
```

9.2.1.3. Workflow

1. Create a module software instance structure for the TSENS module to store the TSENS driver state while it is in use.
   ```c
   struct tsens_module tsens_instance;
   ```

   **Note:** This should never go out of scope as long as the module is in use. In most cases, this should be global.

2. Create a variable for the TSENS sample to be stored in by the driver asynchronously.
   ```c
   int32_t tsens_result;
   ```
3. Create a callback function that will be called each time the TSENS completes an asynchronous read job.

```c
volatile bool tsens_read_done = false;

static void tsens_complete_callback(enum tsens_callback i)
{
    tsens_read_done = true;
}
```

4. Configure the TSENS module.
   1. Create a TSENS module configuration struct, which can be filled out to adjust the configuration of a physical TSENS peripheral.
```c
struct tsens_config config_tsens;
```
   2. Initialize the TSENS configuration struct with the module's default values.
```c
tsens_get_config_defaults(&config_tsens);
```

**Note:** This should always be performed before using the configuration struct to ensure that all values are initialized to known default settings.

3. Set TSENS configurations.
```c
tsens_init(&config_tsens);
```

4. Enable the TSENS module so that conversions can be made.
```c
tsens_enable();
```

5. Register and enable the TSENS read complete callback handler.
   1. Register the user-provided read complete callback function with the driver, so that it will be run when an asynchronous read job completes.
```c
tsens_register_callback(&tsens_instance, tsens_complete_callback, TSENS_CALLBACK_RESULT_READY);
```
   2. Enable the read complete callback so that it will generate callbacks.
```c
tsens_enable_callback(TSENS_CALLBACK_RESULT_READY);
```

9.2.2. **Use Case**

9.2.2.1. **Code**

Copy-paste the following code to your user application:

```c
system_interrupt_enable(SYSTEM_INTERRUPT_MODULE_TSENS);
system_interrupt_enable_global();

tsens_read_job(&tsens_instance, &tsens_result);

while (tsens_read_done == false) {
    /* Wait for asynchronous TSENS read to complete */
}

while (1) {
    /* Infinite loop */
}
9.2.2.2. Workflow

1. Enable interrupts, so that callbacks can be generated by the driver.

```c
system_interrupt_enable(SYSM_INTERRUPT_MODULE_TSENS);
system_interrupt_enable_global();
```

2. Start an asynchronous TSENS conversion, to store TSENS sample into the variable and generate a callback when complete.

```c
tsens_read_job(&tsens_instance, &tsens_result);
```

3. Wait until the asynchronous conversion is complete.

```c
while (tsens_read_done == false) {
    /* Wait for asynchronous TSENS read to complete */
}
```

4. Enter an infinite loop once the conversion is complete.

```c
while (1) {
    /* Infinite loop */
}
```
10. Document Revision History

<table>
<thead>
<tr>
<th>Doc. Rev.</th>
<th>Date</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>42542A</td>
<td>12/2015</td>
<td>Initial document release</td>
</tr>
</tbody>
</table>
Atmel® and others are registered trademarks or trademarks of Atmel Corporation in U.S. and other countries. ARM® and others are registered trademarks of ARM Ltd. Other terms and product names may be trademarks of others.

DISCLAIMER: The information in this document is provided in connection with Atmel products. No license, express or implied, by estoppel or otherwise, to any intellectual property right is granted by this document or in connection with the sale of Atmel products. EXCEPT AS SET FORTH IN THE ATMEL TERMS AND CONDITIONS OF SALES LOCATED ON THE ATMEL WEBSITE, ATMEL ASSUMES NO LIABILITY WHATSOEVER AND DISCLAIMS ANY EXPRESS, IMPLIED OR STATUTORY WARRANTY RELATING TO ITS PRODUCTS INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL ATMEL BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, PUNITIVE, SPECIAL OR INCIDENTAL DAMAGES (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS AND PROFITS, BUSINESS INTERRUPTION, OR LOSS OF INFORMATION) ARISING OUT OF THE USE OR INABILITY TO USE THIS DOCUMENT, EVEN IF ATMEL HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Atmel makes no representations or warranties with respect to the accuracy or completeness of the contents of this document and reserves the right to make changes to specifications and products descriptions at any time without notice. Atmel does not make any commitment to update the information contained herein. Unless specifically provided otherwise, Atmel products are not suitable for, and shall not be used in, automotive applications. Atmel products are not intended, authorized, or warranted for use as components in applications intended to support or sustain life.

SAFETY-CRITICAL, MILITARY, AND AUTOMOTIVE APPLICATIONS DISCLAIMER: Atmel products are not designed for and will not be used in connection with any applications where the failure of such products would reasonably be expected to result in significant personal injury or death (“Safety-Critical Applications”) without an Atmel officer’s specific written consent. Safety-Critical Applications include, without limitation, life support devices and systems, equipment or systems for the operation of nuclear facilities and weapons systems. Atmel products are not designed nor intended for use in military or aerospace applications or environments unless specifically designated by Atmel as military-grade. Atmel products are not designed nor intended for use in automotive applications unless specifically designated by Atmel as automotive-grade.