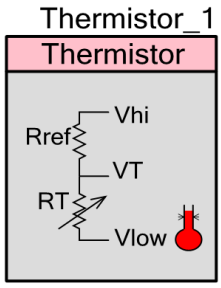
Thermistor Calculator
1.20

Features



- Adaptable for majority of negative temperature coefficient (NTC) thermistors
- Look-Up-Table (LUT) or equation implementation methods
- Selectable reference resistor, based on thermistor value
- Selectable temperature range
- Selectable calculation resolution for LUT method

General Description

The Thermistor Calculator component calculates the temperature based on a provided voltage measured from a thermistor. The component is adaptable to most NTC thermistors. It calculates the Steinhart-Hart equation coefficients based on the temperature range and corresponding user-provided reference resistances. The component provides API functions that use the generated coefficients to return the temperature value based on measured voltage values.

This component doesn't use an ADC or AMUX inside and thus requires those components to be placed separately in your projects.

When to use a Thermistor Calculator

The component has only one use case. The component-provided APIs are used to calculate temperature based on voltage values measured from a thermistor.

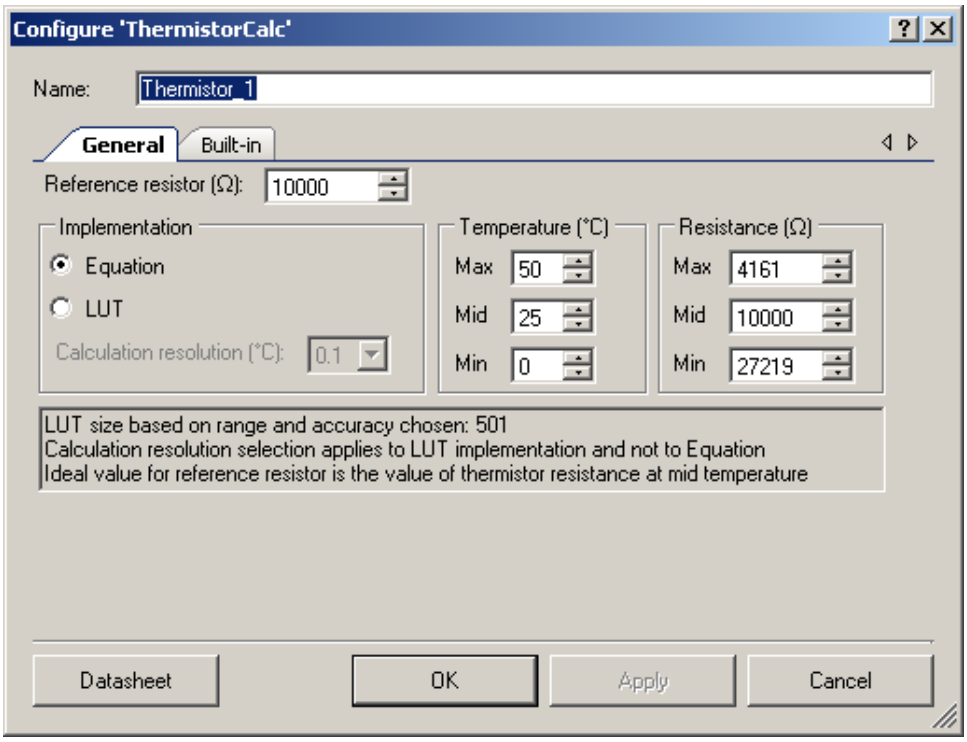
Input/Output Connections

This is a software component and doesn't have any input/output connections.

Parameters and Settings

Drag a Thermistor Calculator component onto your design and double-click it to open the Configure dialog. This dialog has one tab to guide you through the process of setting up the Thermistor Calculator component.

General Tab



The **General** tab provides the following parameters.

Reference resistor

The **Reference resistor** is connected to the thermistor, as shown in the symbol, for the constant voltage type of temperature measurement. The Rref and RT can be interchanged to get either increasing or decreasing voltage values with increasing temperatures. Ideally, the value of the reference resistor should be equal to the value of the thermistor at the middle of the temperature range required.

Range = 1Ω to 1GΩ (default 10000).

Implementation

You can obtain the temperature through Equation or LUT. The tradeoffs between the two methods are memory, speed, range, and resolution. The Equation method is more accurate and has a fixed range and accuracy. The Equation method uses more memory because it requires

the floating point math library. The LUT uses less memory and has a faster response time. The default is Equation.

Calculation resolution (°C)

If you choose the LUT implementation, this parameter becomes active to provide the LUT temperature accuracy.

The accuracy specified by this parameter is the accuracy of the temperature measurement. This implies that the accuracy corresponds to the conversion of the voltage to temperature. It does not account for the other inaccuracies in the system, such as the tolerance of the reference resistor, variation of the reference voltage, or the accuracy of the ADC.

Assuming an accurate voltage measurement, this parameter provides the accuracy of the temperature output of the component.

Options = 0.01, 0.05, 0.1 (default), 0.5, 1, 2 °C

Temperature (°C) / Resistance (Ω)

The first column is to specify maximum, middle, and minimum temperatures for the desired temperature range. The second column is to enter the resistances associated with the corresponding temperatures. The Steinhart-Hart coefficients are calculated based on the entries of this table.

These parameters also determine the temperature range for the LUT implementation. The highest and lowest temperature values entered in these parameters form the starting and ending values of the LUT.

Ranges:

- Temperature (max, mid, min) -80 to 325 °C. (default: max = 50, mid = 25 min = 0)
- Resistances (max, mid, min) 0 to 1MΩ (default max = 4161, mid = 10000 min = 27219)

Although the component supports a wide temperature range (-80 to 325 °C), it is recommended to used a standardized range (-40 to 125 °C) to get better accuracy results.

For standardized temperature ranges, you can usually find precision resistance values for defined temperatures in the datasheet of the particular thermistor you are using. For non-standardized temperature ranges (wider than -40 to 125 °C), you need to perform pre-measurements to obtain correct resistance values for particular temperatures. The Steinhart-Hart coefficients provide the best accuracy in standardized ranges. Anything beyond standardized ranges could provide less accuracy.

If you use the Thermistor Calculator component to measure temperatures in a wide range, even a standardized range, it is strongly recommended to split that wide temperature range to smaller sub-ranges to achieve maximum accuracy. For example range -40 to 125 °C should be split into three sub-ranges: -40° to 0°, 0° to 50°, and 50° to 125°. Splitting the range makes the resistance/temperature curve in each sub-range more linear, smoothing out inaccuracy in LUT entries, and providing the best temperature accuracy. If you have several sub-ranges, then you'll

need to determine which instance to use to calculate the voltage. The resistance calculation is independent of temperature range, so the resistance can be calculated using the function from any of the instances. Then, based on the resistance calculated, the instance for the applicable sub-range must be used to calculate the temperature.

Information

Additional details are provided based on the other parameters, such as the size of the LUT, and the appropriate reference resistor to choose. The LUT size is displayed in the first line and it is limited to 2001 steps. The LUT size is determined by following equation:

$$LUT_SIZE = (MAX_TEMP - MIN_TEMP) / Calculation\ resolution + 1$$

As shown in the equation, the LUT size depends on the range and accuracy. Thus, when the LUT size exceeds the limit, an error is provided next to the accuracy, that either the accuracy or the range has to be lowered.

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 "Thermistor_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 "Thermistor".

Function	Description
uint32 Thermistor_GetResistance (int16 vReference, int16 vThermistor)	The digital values of the voltages across the reference resistor and the thermistor are passed to this function as parameters. These can be considered as the inputs to the component. The function returns (outputs) the resistance, based on the voltage values.
int16 Thermistor_GetTemperature (uint32 resT)	The value of the thermistor resistance is passed to this function as a parameter. The function returns (outputs) the temperature, based on the resistance value. The method used to calculate the temperature is dependent on whether Equation or LUT was selected.

uint32 Thermistor_GetResistance(int16 vReference, int16 vThermistor)

Description: The digital values of the voltages across the reference resistor and the thermistor are passed to this function as parameters. These can be considered as the inputs to the component. The function returns (outputs) the resistance, based on the voltage values.

Parameters: vReference is the voltage across the reference resistor.

vThermistor is the voltage across the thermistor. The ratio of these two voltages is used by this function. Therefore, the units for both parameters must be the same.

Return Value: The return value is the resistance across the thermistor. The return type is 32-bit unsigned integer as shown in the function prototype provided above. The value returned is the resistance in Ohms.

Side Effects: None

int16 Thermistor_GetTemperature (uint32 resT)

Description: The value of the thermistor resistance is passed to this function as a parameter. The function returns (outputs) the temperature, based on the resistance value. The method used to calculate the temperature is dependent on whether Equation or LUT was selected.

Parameters: resT is the resistance across the thermistor in Ohms.

Return Value: The return value is the temperature in 1/100ths of degrees C. For example, the return value is 2345, when the actual temperature is 23.45 degrees C.

Side Effects: None

Sample Firmware Source Code

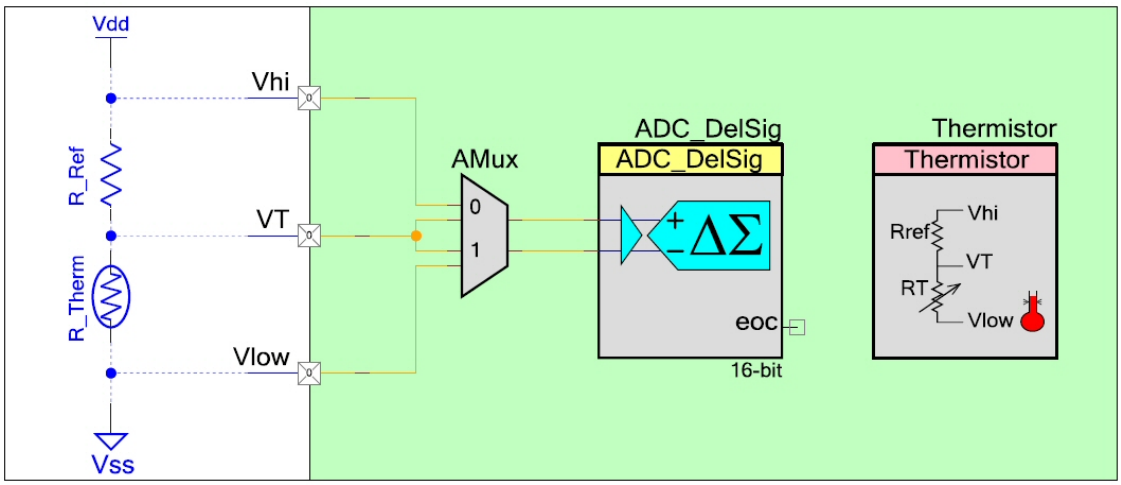
PSoC Creator provides numerous example projects that include schematics and example code in the Find Example Project dialog. For component-specific examples, open the dialog from the Component Catalog or an instance of the component in a schematic. For general examples, open the dialog from the Start 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.

Functional Description

The overall project requires the connection of an external resistor and thermistor to PSoC. The voltages of the externally connected resistor and thermistor are measured by using an ADC and the values are passed to the thermistor component through an API call. The return value of this API call is the temperature. This method of excluding the ADC from the component makes the ADC available for other functions in a project. The block diagram for the overall system is provided in the following diagram.

Figure 1. Block Diagram of the thermistor based temperature monitor system



A constant voltage Vhi is applied to the resistor and thermistor combination shown in Figure 1.

The resistance of the thermistor changes with change in temperature, thus the voltage drop across the thermistor changes with temperature. The voltage drop across the thermistor and resistor are measured, and the resistance of the thermistor is found using the following equation

$$R_T = R_{ref}\left(\frac{V_T - V_{low}}{V_{hi} - V_T}\right)$$

The temperature is then determined based on the resistance, either through the Equation or LUT method. The temperature is obtained in Equation method by directly using the Steinhart-Hart Equation shown below:

$$\frac{1}{T_K} = A + B * ln(R_{Thermistor}) + C * (ln(R_{Thermistor}))^3$$

Where A, B and C are Steinhart-Hart coefficients that are calculated by the component. The coefficients are obtained by solving three simultaneous equations formed by substituting the “Thermistor Parameters" in the component.

In case of the LUT method, the table relating the temperatures and resistances is generated by the component and stored in the program memory. To generate the LUT, the resistances for temperatures within the range are calculated using the following equation:

$$R_{Thermistor} = e^{\left[(\beta - (\alpha/2))^{\frac{1}{3}} - (\beta + (\alpha/2))^{\frac{1}{3}}\right]}$$

The resistances are calculated starting with “Min Temp” and increments of “Accuracy” up to “Max Temp”, specified by the user. A LUT is formed with these resistances by the component and

stored in the program memory. The temperature corresponding to the specific resistance measured is then obtained from the table, during runtime.

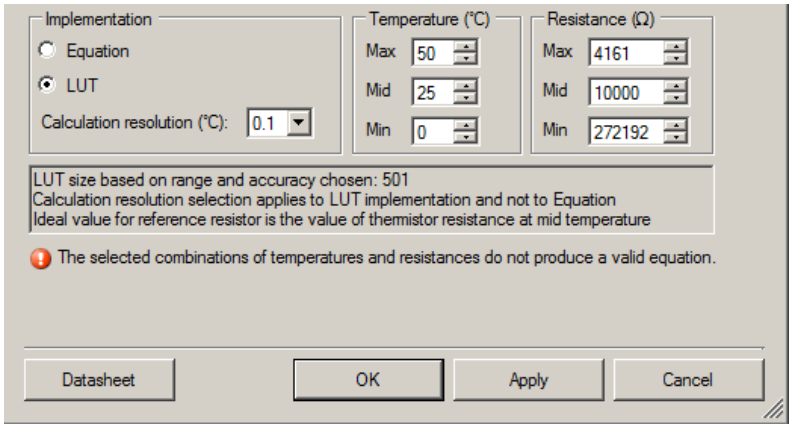
Where $\alpha = \frac{A - \frac{1}{T}}{C}, \beta = \sqrt{\left(\frac{B}{3C}\right)^3 + \frac{\alpha^2}{4}}$

These equations are provided in this document for user reference. The component makes all these calculations and provides the required temperature based on the selections in the Configure dialog.

The comparison between the two implementation methods is provided in the following table.

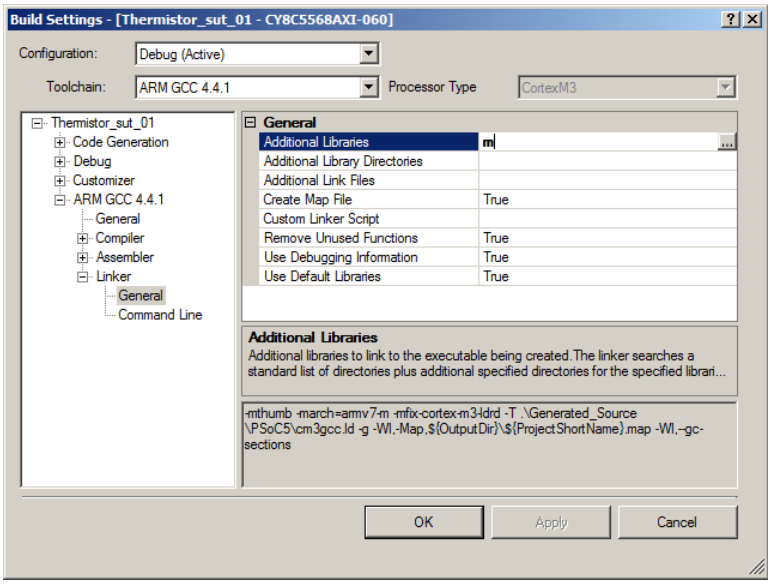
	Equation	LUT	Comments
Accuracy (+/-)	0.01	≥ 0.01	Accuracy shown is the accuracy of calculation alone. This does not include the accuracy of thermistor, reference resistor, voltage reference or ADC. The accuracy of equation method is better than ± 0.01 °C, but the output is limited to resolution of ± 0.01 °C because the function returns 1/100ths °C.
Memory usage	Higher if floating point not already included	Lower if floating point not already included	The memory usage of Equation is fixed and is due to the floating point library. If the floating point library is already used by other components or functions, then the Equation method is efficient. The memory usage of LUT depends on the range and accuracy chosen.
	Lower if floating point already included	Higher if floating point already included	
Range	Wider than specified	Limited to specified	In Equation the temperature can be measured outside of the range specified (lower accuracy). The LUT is restricted to the specified range.
Speed	Slower	Faster	The Equation method uses floating point math library functions, which are computation intensive. The LUT method only requires a binary search of the LUT.

Not all combinations of Temperature and Resistance produce a valid Steinhart-Hart equation. If the entered values would produce an invalid equation, the following error is generated:



This should not occur when using reference values from a thermistor datasheet or when using accurately pre-measured values.

When using the equation implementation method with the GCC compiler, you must specify inclusion of the math library in the **Linker** options of the Build Settings dialog by entering “m” in **Additional libraries** field, as shown in the following figure:



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 Thermistor Calculator component does not have any specific deviations.

Resources

Component is implemented entirely in firmware. It does not consume any other PSoC resources.

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 4 (GCC)		PSoC 5LP (GCC)	
	Flash Bytes	SRAM Bytes	Flash Bytes	SRAM Bytes	Flash Bytes	SRAM Bytes
Equation	329	0	252	0	236	0
LUT	764 + (LUT size * 4)	0	168 + (LUT size * 4)	0	188 + (LUT size * 4)	0

Performance

The performance of the component depends on the implementation method chosen in the customizer. The measurements below have been gathered using a CPU speed of 24 MHz, with the associated compiler configured in Release mode. These numbers should be treated as approximations and used to determine necessary trade-offs.

Device	Equation method	101-entry LUT	2001-entry LUT
PSoC 3 (Keil PK51)	27,000 cycles	4000 cycles	6000 cycles
PSoC 4 (GCC)	30,000 cycles	280 cycles	650 cycles
PSoC 5LP (GCC)	20,000 cycles	250 cycles	600 cycles

Component Changes

Version	Description of Changes	Reason for Changes / Impact
1.20.a	Updated datasheet.	Removed references to obsolete PSoC 5 device.
1.20	Removed arguments check from Thermistor_GetResistance() function.	To prevent CPU halt if thermistor is disconnected.
	Updated MISRA Compliance section.	The component does not have any specific deviations.
	Updated API Memory Usage section.	
	Updated datasheet with memory usage and performance of the component for PSoC 4.	
1.10	Updated Sample Firmware Source Code section with description how to find PSoC Creator example projects	
	Updated Block Diagram of the thermistor based temperature monitor system.	Old diagram was done with poor quality.
	Updated API Memory Usage and Performance data.	
	Changed API's parameters names of Thermistor_GetResistance() from Vreference, VThermistor to vReference, vThermistor; and of Thermistor_GetTemperature() from ResT to resT.	According to Cypress coding standards parameters should be lower camel case.
	Added MISRA Compliance section.	The component was not verified for MISRA compliance.
1.0	Version 1.0 is the first release of the Thermistor Calculator component	

© Cypress Semiconductor Corporation, 2013-2014. 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 HALFICULAR 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.