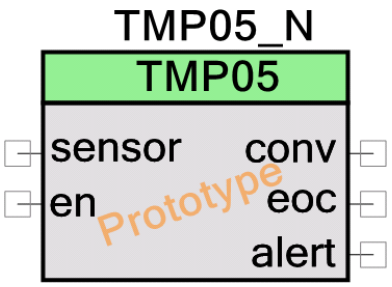
TMP05 Digital Temperature Sensor Interface Component

v1.20

Features



- Supports up to two TMP05 or TMP06 digital temperature sensors connected in daisy chain mode
- Continuous and one-shot modes of operation
- Temperature resolution down to 0.05°C
- ± 0.5°C typical accuracy at 25°C
- ±1.0°C accuracy from 0°C to 70°C

Contents

General Description
Input/Output Connections
Parameters
Placement
Resources
Application Programming Interface (APIs)
Sample Firmware Source Code
Interrupt Service Routine
Registers
DC and AC Electrical Characteristics
Reference Component Changes

General Description

The TMP05 Temp Sensor Interface component is a simple, easy to use component capable of interfacing with Analog Device's TMP05/06 digital temperature sensors in daisy chain mode. Designers can configure and monitor the temperature readings in one of two ways: 1) continuous monitoring option enables the designer to record temperatures in a continuous fashion, at a sample rate dictated by the temperature sensor(s), while 2) one-shot mode triggers the temperature measurement at a rate controllable by the user. The first mode is intended for use in an environment where temperature variations are abrupt and need to be monitored frequently, while the second option should be used when temperature measurements only need to be sampled once in a while or in applications where minimizing power consumption is important.

When to use a TMP05 Temp Sensor Interface Component

The TMP05 Temp Sensor Interface component should be used in Thermal Management applications that require monitoring of the temperatures of several remote subsystems. This component is designed to directly interface to TMP05/06 temperature sensors that produce a PWM output whose duty cycle is directly related to ambient temperature. Temperature readings can be reported in degrees Celsius or degrees Fahrenheit through easy to use firmware APIs.

Modes of Operation

Two modes of operation are supported in the TMP05 Temp Sensor Interface component: 1) continuous, and 2) one-shot.

1- In continuous mode, temperature measurements occur at the fastest possible rate, dictated by the conversion time of the sensors

2- In one-shot mode, temperature measurements occur at a rate dictated by application firmware. In this mode, the designer can set the rate of temperature monitoring

Input/Output Connections

This section describes the various input and output connections for the TMP05 Digital Temperature Sensor Interface component.

Input	May Be Hidden	Description
sensor	N	For single sensor applications, this input should be connected to the TMP05 temperature sensor output. That output is a pulse-width modulated digital signal whose duty cycle is proportional to temperature. For multi-sensor applications where the TMP05 temperature sensors are daisy chained together, this input should be connected to the output of the final TMP05 temperature sensor in the daisy chain. When interfacing to open-drain output TMP06 sensors, this signal requires a pull-up resistor for proper operation. This can be achieved using internal pull-up resistors on the selected PSoC GPIO pin or externally on the circuit board
en	N	Active high enable. Set this pin to a low level to disable the component.

Output	May Be Hidden	Description
conv	N	Conversion Start output. To start monitoring the temperatures, TMP05 sensors require a trigger input with specific timing constraints on low and high times. This output provides a trigger input to the first TMP05 sensor connected in daisy-chain mode.
eoc	N	End of conversion output. This signal pulses active high for approximately 5 µs when a new temperature measurement is available. When sensors are connected in daisy chain mode, this signal indicates a measurement has been made on all sensors.
alert	N	Temperature conversion error output. This signals pulses active high for approximately 5 µs when the temperature sensor measurement times out. This could occur due to a fault in one or more of the attached temperature sensors or in the case of connection failure to the sensor(s).

Parameters

Drag a TMP05 Digital Temperature Sensor Interface component onto your design and double-click it to open the Configure dialog.

Basic Tab

This tab is used to set the basic operational parameters of the component.

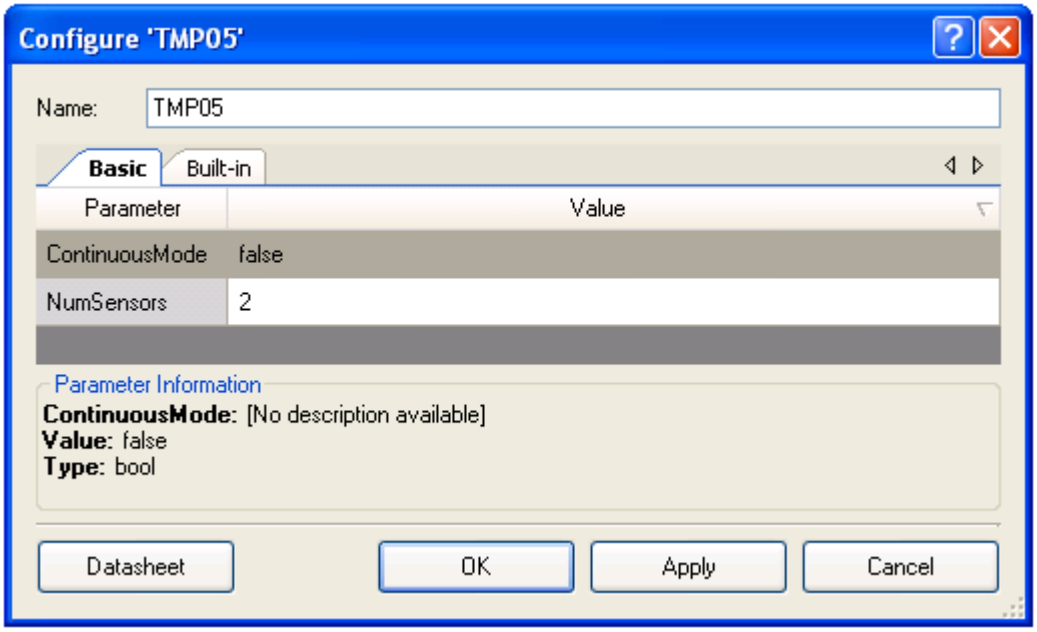


Figure 1 – TMP05 Temp Sensor Interface Basic Tab

ContinuousMode

This parameter is used to specify the operating mode of the component. This parameter is a Boolean value to select between continuous and one-shot modes of operation. Valid values for this parameter are true and false. Default setting is true, enabling a continuous mode of operation.

NumSensors

This parameter is used to specify the number of TMP05 temperature sensors that are connected in daisy-chain mode. Currently the component can support up to four TMP05 temperature sensors. Valid range for this parameter is 1-4. Default setting is 1.

Placement

There are no specific placement requirements for this component.

Resources

The TMP05 Digital Temperature Sensor Interface component uses the following device resources:

Configuration	Digital Blocks				API Memory (Bytes)	
	Datapath (s)	Macro cell(s)	Status Register(s)	Control Register(s)	Flash	SRAM
All	2	9	1	1	2532	8

Application Programming Interface

Application Programming Interface (API) routines allow you to configure the component using firmware. 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 TMP05_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 TMP05.

Function	Description
TMP05_Start()	Starts the component
TMP05_Stop()	Stops the component
TMP05_Init()	Initializes the component
TMP05_Enable()	Enables the component
TMP05_Trigger()	Triggers the interfaced TMP05 sensors to start temperature measurement depending upon the mode of operation
TMP05_ CalcTempC()	Calculates the temperature(s) in degrees Celsius
TMP05_ CalcTempF()	Calculates the temperature(s) in degrees Fahrenheit
TMP05_DiscoverSensors()	Automatically detects how many temperature sensors are daisy-chained to the component
TMP05_ConversionStatus()	Returns current state of temperature conversion (busy, completed or error)
TMP05_SaveConfig()	Saves the current state of the component before entering low power mode
TMP05_RestoreConfig()	Restores the previous state of the component after waking from low power mode
TMP05_Sleep()	Puts the component into low power mode
TMP05_Wakeup()	Wakes the component up from low power mode

void TMP05_Start(void)

Description: Starts the component. Calls the TMP05_Init() API if the component has not been initialized before. Calls the enable API.

Parameters: None

Return Value: None

Side Effects: None

void TMP05_Stop (void)

Description: Disables and stops the component

Parameters: None

Return Value: None

Side Effects: None

void TMP05_Init(void)

Description: Initializes the component

Parameters: None

Return Value: None

Side Effects: None

void TMP05_Enable(void)

Description: Enables the component

Parameters: None

Return Value: None

Side Effects: None

void TMP05_Trigger (void)

Description: Provides a valid strobe/trigger input to the first TMP05 sensor connected to the component.

Parameters: None

Return Value: None

Side Effects: None

int8 TMP05_CalcTempC (uint8 SensorNum)

Description: Calculates the temperature(s) in degrees Celsius

Parameters: uint8 SensorNum. The TMP05 sensor number from 0..3

Return Value: int8

Side Effects: None

int8 TMP05_CalcTempF (uint8 SensorNum)

Description: Calculates the temperature(s) in degrees Fahrenheit

Parameters: uint8 SensorNum. The TMP05 sensor number from 0..3

Return Value: int8

Side Effects: None

uint8 TMP05_DiscoverSensors (void)

Description: This API is provided for applications that might have a variable number of temperature sensors connected. It automatically detects how many temperature sensors are daisy-chained to the component. The algorithm starts by checking to see if the number of sensors actually connected matches the ***NumSensors*** parameter setting in the Basic Tab of the component customizer. If not, it will retry assuming 1 less sensor is connected. This process will repeat until the actual number of sensors connected is known.

Confirming whether or not a sensor is attached or not takes a few hundred milliseconds per sensor per iteration of the algorithm. To limit the sensing time, reduce the ***NumSensors*** parameter setting in the Basic Tab of the component customizer to the maximum number of possible sensors in the system.

Parameters: none

Return Value: uint8 representing the number of sensors actually connected (0..4)

Side Effects:	None

uint8 TMP05_ConversionStatus (void)

Description:	Enables firmware to synchronize with the hardware
Parameters:	none
Return Value:	uint8 status code: 0=conversion in progress, 1=conversion complete, 2=sensor error
Side Effects:	None

Sample Firmware Source Code

Some C language examples demonstrating the basic functionality of the TMP05 reference component are shown below. These examples assume that the component has been placed in a design with the default name " TMP05 ".

Note: If you rename your component you must also edit the example code as appropriate to match the API names to the component name you specify.

Example 1: Using parameter values ContinuousMode=true, and NumSensors=2.

```
#include <device.h>
int8 temp[TMP05COUNT];

void main()
{
      CYGlobalIntEnable;                    // Enable global interrupts
      TMP05_Start();                        // Start the TMP05 component
      TMP05_Trigger();                      // Trigger TMP05 to output temperature

    for(;;)
    {
            temp[0]=TMP05_CalcTempC(0);     // Calculate sensor 1 temperature in C
            temp[1]=TMP05_CalcTempF(1);     // Calculate sensor 2 temperature in F
    }

}
```

Example 2: Using parameter values ContinuousMode=false, and NumSensors=2.

```
#include <device.h>
int8 temp[TMP05COUNT];

void main()
{
      CYGlobalIntEnable;                  // Enable global interrupts
      TMP05_Start();                      // Start the TMP05 component
      TMP05_Trigger();                    // Trigger TMP05 to output temperature

    for(;;)
    {
            CyDelay(1000);                // Initiate the next reading after 1 sec
            temp[0]=TMP05_CalcTempC(0);   // Calculate sensor 1 temperature in C
            temp[1]=TMP05_CalcTempF(1);   // Calculate sensor 2 temperature in F
            TMP05_Trigger();              // Trigger another measurement
    }
}
```

Example 3: Using the DiscoverSensors() API. Using parameter values ContinuousMode=false, and NumSensors=4.

```
#include <device.h>
uint8 sensorCt;
uint8 index;

void main()
{
      CYGlobalIntEnable;                  // Enable global interrupts
      TMP05_Start();                      // Start the TMP05 component
      sensorCt = TMP05_DiscoverSensors(); // Determine how many sensors are connected
      if (sensorCt == 0)
      {
            TMP05_Stop();
      }
      else
      {
            TMP05_Trigger();              // Trigger TMP05 to output temperature
      }

      for(;;)
      {
            CyDelay(1000);
            for(index=0;index<sensorCt; index++)
            {
                  Temp[index]=TMP05_CalcTempC(index);
            }
                  TMP05_Trigger();
      }
}
```

Interrupt Service Routine

The TMP05 Temp Sensor Interface component uses a buried interrupt service routine that is triggered after the pulse width modulated waveform(s) has(ve) been correctly measured and captured. Once the component is instantiated in the project, the interrupt service routine APIs will automatically be created in a filename that matches the name of the component with "_INT.c" appended. For example, if the component has been named "TMP05", the interrupt service routine APIs can be found in the file "TMP05_INT.c" with an associated "TMP05_INT.h" header file.

The following is a C language example of the generated Interrupt Service Routine locations where the user can enter custom ISR code, if required:

```
CY_ISR(TMP05_ISR_Interrupt)
{
    uint8 index;

    for(index=0; index<TMP05_NUMSENSORS; index++)
    {
        /* Store high time of sensor PWM */
        TMP05_1_Hi[index] = CY_GET_REG16(TMP05_1_HI_CNT_PTR);

        /* Store low time of sensor PWM */
        TMP05_1_Lo[index] = CY_GET_REG16(TMP05_1_LO_CNT_PTR);
    }

    /* In continuous mode, trigger next measurement, otherwise clear busy flag */
    #if (TMP05_1_CONTINUOUS == 1)
        TMP05_1_CONTROL_REG |= 0x01u;  /* Generate 2.5 us CONV strobe */
    #else
        TMP05_1_busyFlag = 0u;
    #endif

    /* `ENTER YOUR CUSTOM CODE HERE` */

    /* `#END` */
```

Registers

The TMP05 Temp Sensor Interface component has several control and status registers that are used by the firmware APIs to control operation and monitor status. None of these registers are accessible directly by user firmware.

DC and AC Electrical Characteristics

The following values are indicative of expected performance and based on initial characterization data.

TMP05 DC Specifications

Parameter	Description	Conditions	Min	Typ	Max	Units
Idd	Block current consumption	<tbd>	<tbd>	<tbd>	<tbd>	µA

TMP05 AC Specifications

Parameter	Description	Conditions	Min	Typ	Max	Units
Measurement Time	Time to take sensor measurements	Continuous mode (measurement time per sensor)	105	116	151	ms

Reference Component Changes

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

Version	Description of Changes
v1.0	First release
v1.10	Component re-designed to support up to 4 TMP05/06 temperature sensors
v1.20	Added DiscoverSensors API to auto-detect the number of sensors connected

© Cypress Semiconductor Corporation, 2011. 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.