AN65977

PSoC® 3, PSoC 4, and PSoC 5LP – Temperature Measurement with a TMP05/TMP06 Digital Sensor

Author: Rodolfo Lossio
Associated Project: Yes
Associated Part Family: PSoC 3, PSoC 4, PSoC 5LP
Software Version: PSoC Creator™ v3.3

AN65977 describes the TMP05 Digital Temperature Sensor Interface Component, which is a building block for thermal management applications. It enables designers using PSoC to quickly and easily interface with multiple Analog Devices TMP05 or TMP06 digital temperature sensors through a simple, serial 2-wire digital interface.

Contents

1 Introduction....1
2 Typical System Architecture....2
3 Design Considerations when using TMP05 Sensors in Daisy-Chain Mode....3
4 Getting Started with the TMP05 Digital Temperature Sensor Interface Component....3
5 Example 1 – Firmware Details....5
6 Example 2 – Firmware Details....6
7 Example 3 – Firmware Details....8
8 Example 4 – Firmware Details....10
9 Evaluation Kits....13
10 Conclusion....13
Worldwide Sales and Design Support....15

1 Introduction

Digital temperature sensors are widely used in systems where multiple temperature measurements are required in specific locations on a large board (such as a line card in a switch or router) or in a remote location. The use of digital temperature sensors frees the designer from worrying about digital noise coupling on to sensitive analog signals on the PCB layout because the digital temperature sensor does the analog to digital conversion within its own package, at the location where the temperature measurement is needed.

The most common digital temperature sensors use industry-standard communications interfaces such as I^2C or SPI. These interfaces are well known and need no description here. For applications in which the thermal management controller has no I^2C or SPI interfaces available, an alternative is Analog Devices' TMP05 and TMP06 monolithic temperature sensors that generate a pulse-width modulated (PWM) serial digital output.

The duty cycle of the PWM output varies in direct proportion to the ambient temperature of the devices. The high period (T_H) of the PWM remains static over all temperatures, while the low period (T_L) varies. It offers a high temperature accuracy of ±1 °C from 0 °C to 70 °C with excellent transducer linearity. The digital output of the TMP05 is CMOS/TTL-compatible and, therefore, can be interfaced directly to PSoC. The digital output of the TMP06 is open-drain and requires a pull-up resistor for proper operation, which can be integrated inside PSoC. Throughout the rest of this document, any references to TMP05 also apply to the TMP06 sensor.

The **TMP05 Digital Temperature Sensor Interface** component makes it extremely easy for designers to develop thermal monitoring and management solutions using multiple TMP05 sensors. The component requires just one input and one output pin from PSoC, which enables the designer to implement many other system management functions in the same device.

The TMP05 Digital Temperature Sensor Interface component can be used in thermal management solutions for base stations, telecommunications, and in server and storage applications. Typical applications may include, but are not limited to, areas where remote temperature sensing, environmental control systems, computer thermal monitoring, thermal protection, industrial process control, and power-system monitoring and management are required.

2 Typical System Architecture

The TMP05 sensor has three modes of operation:

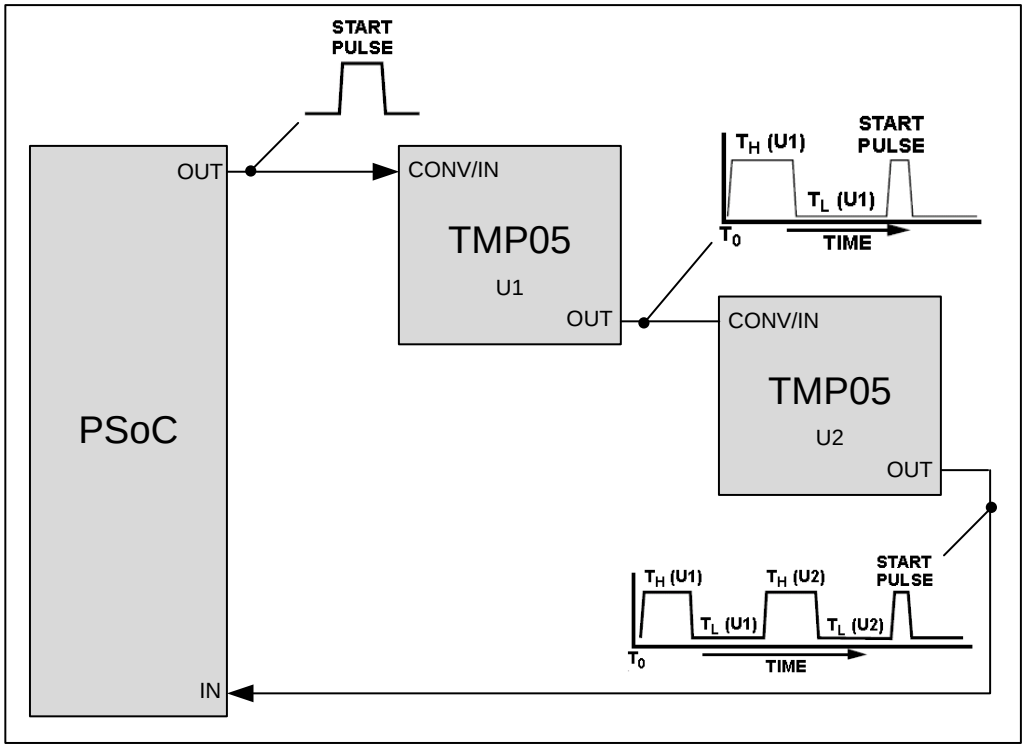
- Continuously converting mode
- Daisy-chained mode
- One-shot mode

A three-state function control input pin (FUNC) sampled at power-up determines the mode in which the device operates. Setting the FUNC pin to a high state allows multiple TMP05s to be connected in daisy-chained mode. In that mode, multiple TMP05 temperature sensors can be connected, which enables PSoC to read all the sensors through the same two-wire interface.

In this configuration, PSoC generates a “Start” pulse to begin a new temperature-to-PWM conversion cycle. The output of the first TMP05 sensor begins with its own PWM output, followed by a new Start pulse that it generated internally. The output of the second TMP05 sensor begins with the PWM output of the previous sensor, followed by its own PWM output and terminated by a new Start pulse that it generated internally. When more sensors are daisy-chained in this fashion, the final return signal to PSoC contains the PWM outputs from all sensors starting with the output of the first sensor, followed by the output from the second sensor and so on until the terminating Start pulse is detected. Temperature-to-PWM conversion then stops until PSoC generates another Start pulse. The system-level connection scheme and expected waveforms are shown in Figure 1.

PSoC is an ideal choice for interfacing to TMP05 sensors because it has the programmable universal digital block resources available to implement the required custom interface logic with minimal firmware. This application note focuses on how to use the TMP05 Digital Temperature Sensor Interface component with PSoC by working through some example projects on the CY8CKIT-001 PSoC Development Kit (DVK). The examples will show how to operate the component and monitor the temperature of multiple TMP05 devices reliably.

Figure 1. Multiple TMP05 Sensors Daisy-Chained



3 Design Considerations when using TMP05 Sensors in Daisy-Chain Mode

Certain timing constraints must be observed when operating TMP05 sensors in daisy-chain mode. PSoC needs to generate the conversion start pulse on the CONV/IN pin of the first sensor. The start pulse lets the first TMP05 part know that it should now start a conversion and output its own temperature. The pulse width of the start pulse should be less than 25 µs but greater than 20 ns. These constraints are handled automatically by the component design and are mentioned in this application note for reference purposes only.

Figure 2 and Figure 3 show the input and output waveforms for the first sensor in the daisy-chain (taken from the TMP05 device datasheet):

Figure 2. TMP05 Start Pulse Waveform

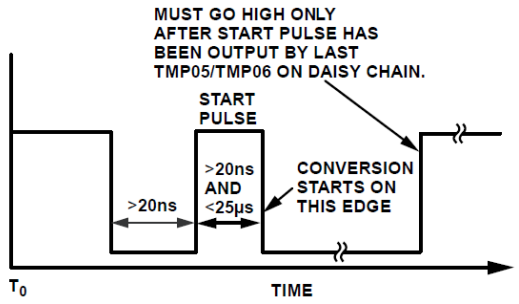
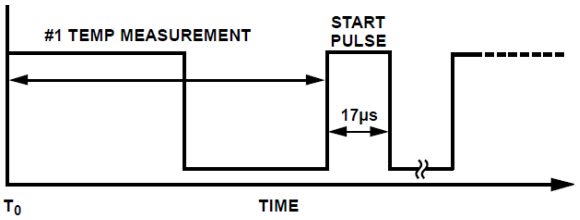


Figure 3. TMP05 PWM Output Waveform



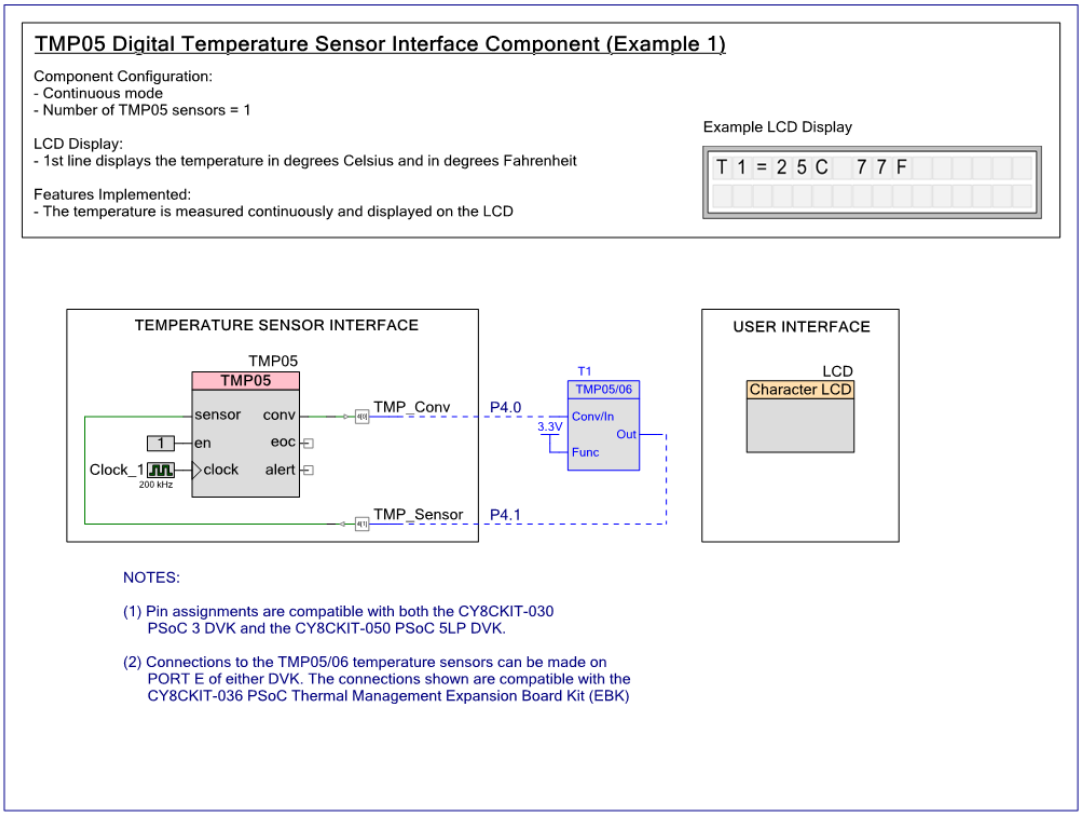
4 Getting Started with the TMP05 Digital Temperature Sensor Interface Component

Distributed with this application note is a PSoC Creator™ bundled project ZIP file that contains four example design projects and a library project. The library project contains the **TMP05** Creator component.

Save the ZIP file to a convenient location on your hard drive and extract the contents to a local folder.

To get started with the example projects, double-click on the *AN65977.cywrk* PSoC Creator workspace file.

Figure 4. Top-Level Schematic for TMP05 Example1



1. In the **Workspace Explorer** tab to the left of the screen, expand project **Example1** by clicking on the small '+' icon to the left of it.
2. Double-click on *TopDesign.cysch* to open the top-level design schematic, which shows the components used inside PSoC.

   To the right of the schematic is the PSoC Creator Component Catalog. The tab labeled **Cypress** contains the default library of Cypress-provided content. The Category labeled **Thermal Management** contains the TMP05 component. The provided library project can be re-used by importing into any PSoC Creator design project that requires the TMP05 interface component functionality.

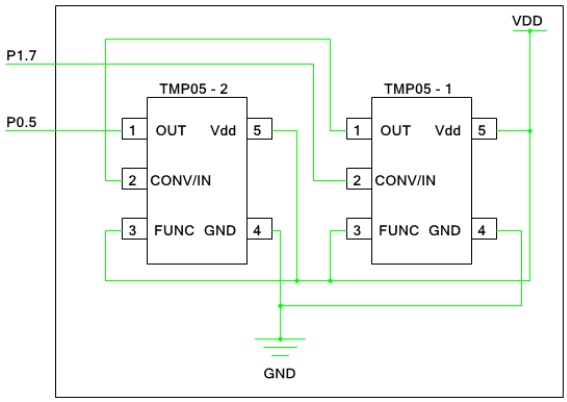
Figure 4 shows the top-level design schematic for Example Project 1.

All of the example projects provided with this application note are designed to run on either the CY8CKIT-030 PSoC 3 Development Kit (DVK) or the CY8CKIT-050 PSoC 5LP DVK, supporting up to four TMP05 or TMP06 temperature sensors. The projects are easily modified to support more TMP05 or TMP06 sensors for further development and testing.

Each example project consists of monitoring the ambient temperature from a specific number of TMP05 sensors. To replicate the environment for each project properly, some wire connections are required on the PSoC DVK. The *Top Design* of each project in PSoC Creator illustrates a circuit shown in blue. You can also refer to Figure 5 for the example using two sensors in daisy-chain.

Double-click the **TMP05** component to open the component customizer as show in Figure 6.

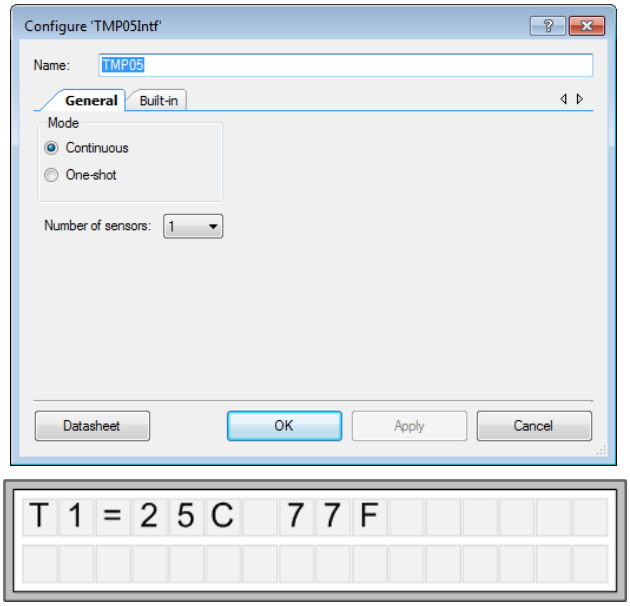
Figure 5. Wire Connections for two daisy-chained TMP05 sensors



For the first example, the **Mode** parameter is set to **Continious** and the **Number of Sensors** parameter is set to **1**. This will enable continuous monitoring of one TMP05 sensor. Click **OK** to close the customizer.

Program the Example1 project into the DVK by selecting **Debug > Program** from the pull-down menus. If the project is running correctly, the text displayed on the debug LCD should look similar to the following picture.

Figure 6. TMP05 Component Customizer



Note that the temperature is displayed in both degrees Celsius and Fahrenheit.

5 Example 1 – Firmware Details

In the **Workspace Explorer**, double-click *main.c* to examine the firmware used in this example project. The code is shown in the Code 1 excerpt. As shown, all lines of code that begin with the prefix **TMP05** reference APIs provided by Cypress for this component. Refer to the component datasheet for a full list of the provided APIs.

In Example1, PSoC is monitoring the temperature of one TMP05 sensor continuously. The monitored temperature is displayed on the LCD.

Congratulations! You have just completed your first **TMP05** temperature sensor interface design with PSoC.

Code 1. Example1 *main.c* Firmware code

```
#include <device.h>

#define TEMP_SCALE            (100u)

int main(void)
{
    /* Enable global interrupts to the CPU core */
    CyGlobalIntEnable;

    /* Start the hardware components */
    LCD_Start();
    LCD_ClearDisplay();
    TMP05_Start();

    /* Trigger the first TMP05 sensor measurement */
    TMP05_Trigger();

    /* Display the temperature continuously */
    while (1)
    {
        /* Only display the temperature when end-of-conversion is detected */
```

```
        if (TMP05_ConversionStatus() & TMP05_STATUS_COMPLETE)
        {
            LCD_Position(0,0);
            LCD_PrintString("T1=");
            LCD_PrintNumber((uint16)TMP05_GetTemperature(0) / TEMP_SCALE);
            LCD_PrintString("C ");
            LCD_PrintNumber((((uint16)TMP05_GetTemperature(0) / TEMP_SCALE) * 1.8) + 32);
            LCD_PrintString("F ");
        }
    }

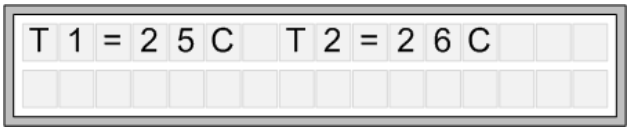
    return(0);
}
```

6 Example 2 – Firmware Details

The firmware of Example2 also operates in continuous mode; however, it can sample one or two sensors depending on how the hardware is configured. You should set the number of sensors in the TMP05 component to be equal to the real number of sensors installed in the hardware. This example also supports daisy-chain error detection.

Close *main.c* and *TopDesign.cysch* for Example1. Go back to the Workspace Explorer and right-click on **Project 'Example2'**. Select **Set as Active Project.**

Open *TopDesign.cysch* for Example2 to start working on this project. Program the Example2 project into the DVK by selecting **Debug > Program** from the pull-down menus. If the project is running correctly, the text displayed on the debug LCD should look similar to the following picture.



If a sensing error is detected due to an open circuit anywhere in the daisy-chain, a message is displayed and sampling stops. In that case, the error message displayed on the LCD should look like the following picture:



To generate an error condition, you can disconnect one of the cables connected to P4.0 or P4.1.

The code for Example2 is shown in the Code 2 excerpt.

Code 2. Example2 *main.c* Firmware code

```
#include <device.h>

#define TEMP_SCALE            (100u)

static void DisplayTemp(void);        /* Display temperatures on the LCD */
static void DisplayErrorMsg(void);    /* Display error messages on the LCD */

int main(void)
{
    uint8 status;

    /* Enable global interrupts to the CPU core */
    CyGlobalIntEnable;

    /* Start the hardware components */
    LCD_Start();
    LCD_ClearDisplay();
    TMP05_Start();

    /* Trigger the first TMP05 sensor measurement */
    TMP05_Trigger();

    /* Display the temperature continuously */
    while (1)
    {
        status = TMP05_ConversionStatus();

        /* Error detected, display error message */
        if (status & TMP05_STATUS_ERROR)
        {
            DisplayErrorMsg();
```

```
            /* Stop the component */
            TMP05_Stop();

            while (1)
            {
                /* Stop here */
            }
        }

        /* Display the temperature(s) measured from the sensor(s) */
        else if (status & TMP05_STATUS_COMPLETE)
        {
            DisplayTemp();
        }
    }

    return(0);
}

/* Function to display temperatures on the character LCD */
static void DisplayTemp(void)
{
    uint8 I;

        LCD_Position(0,0);

    /* Display the temperature(s) on the LCD */
    for (i=0; I < TMP05_CUSTOM_NUM_SENSORS; i++)
    {
        /* Print the following string on LCD: T1=xxC */
        LCD_PrintString("T");
        LCD_PrintNumber(i+1);
        LCD_PrintString("=");
        LCD_PrintNumber((uint16)TMP05_GetTemperature(i)/TEMP_SCALE);
        LCD_PrintString("C ");
    }
}

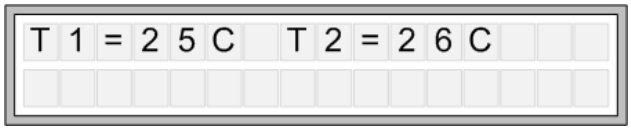
/* Function to display error messages */
void displayErrorMsg(void)
{
        LCD_ClearDisplay();
        LCD_Position(0,0);
        LCD_PrintString("Error Detected! ");
        LCD_Position(1,0);
        LCD_PrintString("Reset to Retry..");
}
```

7 Example 3 – Firmware Details

While the **TMP05** component operates in continuous mode in the first two examples, it may be useful to operate it in the one-shot mode to set a deterministic sensor sample rate or to save power. This example also supports error detection, however, unlike Example2, sampling is not suspended when an error is detected. Instead, the firmware will continuously retry sampling.

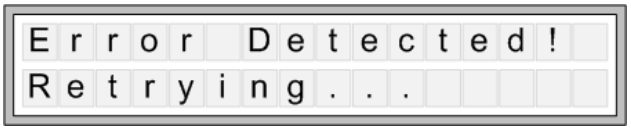
Close *main.c* and *TopDesign.cysch* for Example2. Go back to the Workspace Explorer and right-click on **Project 'Example3'**. Select **Set as Active Project.**

Open *TopDesign.cysch* for Example3 to start working on this project. Program the Example3 project into the DVK by selecting **Debug > Program** from the pull-down menus. If the project is running correctly, the text displayed on the debug LCD should look similar to the following picture:



In this example, a timer is added to control the temperature sensor sample rate. By default, the timer is configured to generate an interrupt every 1 second. This can be changed by changing the **period** parameter of the Timer component placed on the schematic sheet.

If an error is detected, a message is displayed and the firmware tries again to sample from the sensors. The error message displayed on the LCD should look like the following picture:



To generate an error condition, you can disconnect one of the wires connected to P4.0 or P4.1.

The code of Example3 is shown in the Code 3 excerpt.

Code 3. Example3 *main.c* Firmware code

```
#include <device.h>

#define TEMP_SCALE            (100u)

static void DisplayTemp(void);          /* Display temperatures on the LCD */
static void DisplayErrorMsg(void);      /* Display error messages on the LCD */

static CYBIT errorFlag;                 /* Error condition flag */
static CYBIT timerFlag;                 /* Timer expired flag */

/* Interrupt handler for the error scenarios */
CY_ISR(AlertInterruptHandler)
{
    /* Set the error flag for the main loop to process */
    errorFlag = 1;
}

/* Interrupt handler for the temperature sensor sample rate timer */
CY_ISR(TimerInterruptHandler)
{
    /* Clear the timer registers */
    Timer_ReadStatusRegister();

    /* Set the timer flag for the main loop to process */
    timerFlag = 1;
}

int main(void)
{
    uint8 status;

    /* Clear flags */
    errorFlag = 0;
    timerFlag = 0;

    /* Enable global interrupts to the CPU core */
    CyGlobalIntEnable;

    /* Start the hardware components */
    LCD_Start();
    LCD_ClearDisplay();
```

```
    TMP05_Start();
    Timer_Start();

    /* Start and configure the Alert interrupt handler */
    AlertISR_Start();
    AlertISR_Disable();
    AlertISR_SetVector(AlertInterruptHandler);
    AlertISR_Enable();

    /* Start and configure the Timer interrupt handler */
    TimerISR_Start();
    TimerISR_Disable();
    TimerISR_SetVector(TimerInterruptHandler);
    TimerISR_Enable();

    /* Trigger the first TMP05 sensor measurement */
    TMP05_Trigger();

    while (1)
    {
        /* Error handling */
        if (errorFlag)
        {
            errorFlag = 0;

            /* Display error message */
            DisplayErrorMsg();

            /* Restart the component */
            TMP05_Stop();
            TMP05_Start();

            /* Trigger again to re-try the measurements */
            TMP05_Trigger();
        }

        /* Check for new temperature conversion result */
        else
        {
            /* Read the status of the conversion */
            status = TMP05_ConversionStatus();

            if ((status & TMP05_STATUS_COMPLETE) && !(status & TMP05_STATUS_ERROR))
            {
                /* Display the temperature(s) measured from the sensor(s) */
                DisplayTemp();
            }
        }

        /* Timer expired, initiate a new measurement */
        if (timerFlag)
        {
            timerFlag = 0;
            TMP05_Trigger();
        }
    }

    return(0);
}

/* Function to display temperatures on the character LCD */
static void DisplayTemp(void)
{
    uint8 I;

    LCD_ClearDisplay();
    LCD_Position(0,0);

    /* Display the temperature in a specific location on the LCD */
    for (i=0; I < TMP05_CUSTOM_NUM_SENSORS; i++)
```

```
    {
        /* Print the following string on LCD: T1=xxC */
        LCD_PrintString("T");
        LCD_PrintNumber(i+1);
        LCD_PrintString("=");
        LCD_PrintNumber((uint16)TMP05_GetTemperature(i)/TEMP_SCALE);
        LCD_PrintString("C ");
    }
}

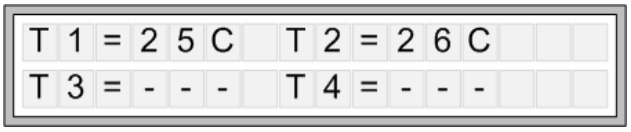
/* Function to display an error message */
static void DisplayErrorMsg(void)
{
    LCD_ClearDisplay();
    LCD_Position(0,0);
    LCD_PrintString("Error Detected! ");
    LCD_Position(1,0);
    LCD_PrintString("Retrying...     ");
}
```

8 Example 4 – Firmware Details

In this example, the **TMP05** component is configured in one-shot mode. The main difference in this example is the addition of a discovery mechanism that automatically determines the number of sensors in the daisy-chain. This feature is useful in cases where the same PSoC design will be used across several different variations of the same product, but each one supports a different number of TMP05 sensors.

Close *main.c* and *TopDesign.cysch* for Example3. Go back to the Workspace Explorer and right-click on **Project 'Example4'**. Select **Set as Active Project.**

Open *TopDesign.cysch* for Example4 to start working on this project. Program the Example4 project into the DVK by selecting **Debug > Program** from the pull-down menus. If the project is running correctly, the text displayed on the debug LCD should look similar to the following picture.



This example also uses a 1-second timer to control the temperature sensor sample rate.

If an error is detected, the temperature will be displayed as '- - -'. In the above LCD picture, the firmware only discovered two sensors in the daisy-chain, therefore only the measurements of the first two temperatures are displayed. You could manually change the number of sensors in the daisy-chain of your hardware and test the discovery feature. The code of Example4 is shown in the following excerpt.

Code 4. Example4 *main.c* Firmware code

```
#include <device.h>

#define MAX_NUM_SENSORS    (4u)     /* Max number of sensors in the daisy-chain */
#define TEMP_SCALE             (100u)

static void DisplayTemp(void);  /* Display temperatures on the LCD */

static CYBIT errorFlag;          /* Error condition flag */
static CYBIT timerFlag;          /* Timer expired flag */
static uint8 sensorCount;        /* Number of TMP05 sensors connected to PSoC */

/* Interrupt handler for the error scenarios */
CY_ISR(AlertInterruptHandler)
{
    /* Set the error flag for the main loop to process */
    errorFlag = 1;
}

/* Interrupt handler for the temperature sensor sample rate timer */
CY_ISR(TimerInterruptHandler)
{
    /* Clear the timer registers */
    Timer_ReadStatusRegister();

    /* Set the timer flag for the main loop to process */
```

```
    timerFlag = 1;
}

int main(void)
{
    uint8 status;

    /* Clear flags */
    errorFlag = 0;
    timerFlag = 0;

    /* Enable global interrupts to the CPU core */
    CyGlobalIntEnable;

    /* Start the hardware components */
    LCD_Start();
    LCD_ClearDisplay();
    TMP05_Start();

    /* Start and configure the Alert interrupt handler */
    AlertISR_Start();
    AlertISR_Disable();
    AlertISR_SetVector(AlertInterruptHandler);
    AlertISR_Enable();

    /* Discover the number of TMP05 temperature sensors in the daisy-chain */
    sensorCount = TMP05_DiscoverSensors();
    errorFlag = 0;

    /* Trigger the first TMP05 sensor measurement */
    TMP05_Trigger();

    /* Start and configure the Timer interrupt handler */
    TimerISR_Start();
    TimerISR_Disable();
    TimerISR_SetVector(TimerInterruptHandler);
    TimerISR_Enable();
    Timer_Start();

    while (1)
    {
        /* Error handling */
        if (errorFlag)
        {
            /* Restart the component */
            TMP05_Stop();
            TMP05_Start();

            /* Clear the temperatures on LCD */
            sensorCount = 0;
            DisplayTemp();

            /* Re-discover the number of sensors in the daisy-chain */
            sensorCount = TMP05_DiscoverSensors();
            errorFlag = 0;

            /* Trigger again to re-try the measurements */
            TMP05_Trigger();
        }

        /* Check for new temperature conversion result */
        else
        {
            /* Read the status of the conversion */
            status = TMP05_ConversionStatus();

            if ((status & TMP05_STATUS_COMPLETE) && !(status & TMP05_STATUS_ERROR))
            {
                /* Display the temperature(s) measured from the sensor(s) */
                DisplayTemp();
```

```
            }
        }

        /* Timer expired, initiate a new measurement */
        if (timerFlag)
        {
            timerFlag = 0;
            TMP05_Trigger();
        }
    }

    return(0);
}

/* Function to display temperatures on the character LCD */
static void DisplayTemp(void)
{
    uint8 I;

    LCD_ClearDisplay();
    LCD_Position(0,0);

    /* Display the temperature in a specific location on the LCD */
    for (i=0; i<MAX_NUM_SENSORS; i++)
    {
        /* Print the following string on LCD: T1=xxC */
        LCD_PrintString("T");
        LCD_PrintNumber(i+1);
        LCD_PrintString("=");
        if (I < sensorCount)
        {
            LCD_PrintNumber((uint16)TMP05_GetTemperature(i)/TEMP_SCALE);
            LCD_PrintString("C ");
        }
        else
        {
            LCD_PrintString("--- ");
        }
        if (I == 1)
        {
            /* Line break */
            LCD_Position(1,0);
        }
    }
}
```

9 Evaluation Kits

If you would like to continue working with the TMP05 component, consider purchasing the PSoC Thermal Management Expansion Board Kit (CY8CKIT-036) shown in Figure 7. The CY8CKIT-036 consists of two TMP05 devices and can be used to implement a complete thermal management design.

Figure 7. CY8CKIT-036 PSoC Thermal Management Expansion Board Kit (EBK)



10 Conclusion

The TMP05 Digital Temperature Sensor Interface component enables designers to quickly and easily design thermal monitoring and management solutions providing support of multiple TMP05 sensors in daisy-chain applications.

The unique ability of the PSoC architecture to combine custom digital logic, analog signal chain processing, and an MCU in a single device enables system designers to integrate many external fixed-function ASSPs. This powerful integration capability not only reduces BOM cost, but also results in PCB board layouts that are less congested and more reliable.

Document History

Document Title: AN65977 - PSoC® 3, PSoC 4, and PSoC 5LP – Temperature Measurement with a TMP05/TMP06 Digital Sensor

Document Number: 001-65977

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	3123189	AMKH	12/29/2010	New Spec – Initial Release
*A	3571727	RLOS	04/05/2012	Updated the document using the new template. Updated the example projects to use the TMP05 Component v1.20. Added support for PSoC 5.
*B	3819560	JK	11/22/2012	Updated the example projects to use the new TMP05 Component v1.30 revised to work with PSoC Creator v2.1 SP1 and PSoC 5LP. Updated the example project pinouts to work with the CY8CKIT-030 PSoC 3 Development Kit (DVK) and CY8CKIT-050 PSoC 5LP DVK.
*C	4680806	RLRM	03/16/2015	Project files updated to PSoC Creator 3.1 Updated example code shown in application note Updated screenshots Cleaned up grammar and general text in application note.
*D	5100103	RLRM	01/27/2016	Updated example project to PSoC Creator 3.3 and added PSoC 4 support. Updated template Updated title
*E	5713254	AESATMP9	04/26/2017	Updated logo and copyright.

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

ARM® Cortex® Microcontrollers	cypress.com/arm
Automotive	cypress.com/automotive
Clocks & Buffers	cypress.com/clocks
Interface	cypress.com/interface
Internet of Things	cypress.com/iot
Memory	cypress.com/memory
Microcontrollers	cypress.com/mcu
PSoC	cypress.com/psoc
Power Management ICs	cypress.com/pmic
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless Connectivity	cypress.com/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6

Cypress Developer Community

Forums | WICED IOT Forums | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

All other trademarks or registered trademarks referenced herein are the property of their respective owners.

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709

©Cypress Semiconductor Corporation, 2010-2017. This document is the property of Cypress Semiconductor Corporation and its subsidiaries, including Spansion LLC ("Cypress"). This document, including any software or firmware included or referenced in this document ("Software"), is owned by Cypress under the intellectual property laws and treaties of the United States and other countries worldwide. Cypress reserves all rights under such laws and treaties and does not, except as specifically stated in this paragraph, grant any license under its patents, copyrights, trademarks, or other intellectual property rights. If the Software is not accompanied by a license agreement and you do not otherwise have a written agreement with Cypress governing the use of the Software, then Cypress hereby grants you a personal, non-exclusive, nontransferable license (without the right to sublicense) (1) under its copyright rights in the Software (a) for Software provided in source code form, to modify and reproduce the Software solely for use with Cypress hardware products, only internally within your organization, and (b) to distribute the Software in binary code form externally to end users (either directly or indirectly through resellers and distributors), solely for use on Cypress hardware product units, and (2) under those claims of Cypress's patents that are infringed by the Software (as provided by Cypress, unmodified) to make, use, distribute, and import the Software solely for use with Cypress hardware products. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS DOCUMENT OR ANY SOFTWARE OR ACCOMPANYING HARDWARE, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. To the extent permitted by applicable law, Cypress reserves the right to make changes to this document without further notice. Cypress does not assume any liability arising out of the application or use of any product or circuit described in this document. Any information provided in this document, including any sample design information or programming code, is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Cypress products are not designed, intended, or authorized for use as critical components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or system could cause personal injury, death, or property damage ("Unintended Uses"). A critical component is any component of a device or system whose failure to perform can be reasonably expected to cause the failure of the device or system, or to affect its safety or effectiveness. Cypress is not liable, in whole or in part, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from or related to all Unintended Uses of Cypress products. You shall indemnify and hold Cypress harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of Cypress products.

Cypress, the Cypress logo, Spansion, the Spansion logo, and combinations thereof, WICED, PSoC, CapSense, EZ-USB, F-RAM, and Traveo are trademarks or registered trademarks of Cypress in the United States and other countries. For a more complete list of Cypress trademarks, visit cypress.com. Other names and brands may be claimed as property of their respective owners.