AN2163

Interfacing to 1-Wire/Two-Wire Digital Temperature Sensors using PSoC® 1

Author: Arvind Krishnan
Associated Project: Yes
Associated Part Family: CY8C28xxx
Software Version: PSoC® Designer™ 5.2
Related Application Notes: For a complete list of the application notes, click here.

If you have a question, or need help with this application note, contact the author at arvi@cypress.com.

AN2163 demonstrates how a PSoC® 1 microcontroller can be interfaced with 1-wire® and Two-Wire (I^2C) digital temperature sensors. This application note also illustrates how PSoC® 1 can be used for sensing temperature using a DS18S20 (1-wire digital temperature sensor) and a TMP75 (Two-Wire digital temperature sensor) along with the help of the attached example project.

Contents

Introduction

Monitoring temperature is a critical function in systems, such as Servers/CPUs, Battery charging/management, Consumer products, and Industrial controls, especially where there is a concern of loss in performance, reliability, and safety, due to variations in temperature. Several kinds of devices are available for sensing temperature such as thermistors, resistance temperature detectors (RTDs), thermocouples, and analog and digital output sensors.

Digital temperature sensors are inexpensive, easily available, and widely used in microcontroller based temperature management systems. The use of digital temperature sensors frees the designer from worrying about digital noise coupling onto 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. Due to the internal nature of conversion, the system requires minimum external hardware and the host to sensor interfaces can be kept simple.

This application note demonstrates how a PSoC 1 microcontroller can be interfaced to a 1-wire based digital thermometer (DS18S20) from Maxim Integrated Products (Dallas Semiconductor) and Two-Wire based digital temperature sensor (TMP75) from Texas Instruments. 1-wire communication allows an interface that uses only one pin to retrieve data from devices on the bus. I^2C is an industry standard communication interface that allows for easy and error free communication using just two pins and minimal external hardware.

After reading this application note, the reader should be able to implement a PSoC 1 based temperature sensing solution using the sensors mentioned above. With the help of the attached example project, you should be able to implement a system capable of measuring temperatures with a resolution of 0.25 °C with the DS18S20 and 0.0625 °C with the TMP75 digital sensor.

The following sections give a brief explanation of these sensors and the interfaces they follow.

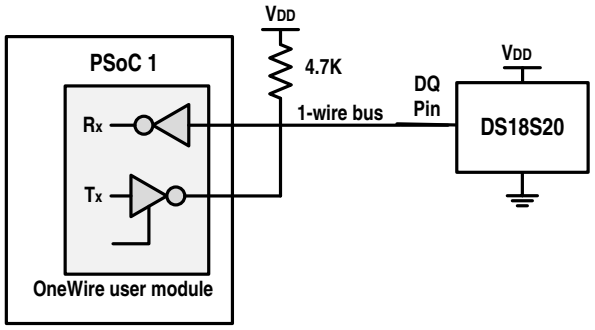
1-wire® Interface

The 1-Wire interface is a bidirectional, half duplex, serial signaling protocol designed by Dallas Semiconductor. 1-Wire is a compact communication interface for ICs that do not require high-speed communication. It uses a single wire for reading and writing, and has no clock signal. 1-wire devices have the ability to operate in **parasitic mode**, in which the connected devices can draw power from the 1-wire bus itself.

1-Wire is a relatively slow interface, with a typical data rate of 16 kbps. It is perfect for slow sensors such as thermometers, which do not need to be polled frequently.

Similar to I^2C, the 1-wire interface relies on the master-slave relationship. In this application, PSoC is the master and the DS18S20 sensor is the slave. Both devices use an open drain connection to the bus, which removes the possibility of a short due to bus contention. To achieve this, a pull-up resistor is attached to the bus, so that it is pulled to V_{DD} when released by both devices, as shown in Figure 1.

Figure 1. Hardware Configuration



A brief on the 1-Wire protocol can be found in the Appendix. A detailed tutorial on the 1-Wire protocol can be found on the Maxim site at http://www.maxim-ic.com/products/1-wire/flash/overview/index.cfm.

Example Projects

The DS18S20 Example Code and TMP75 Example Code sections give a step-by-step process on how to create a PSoC Designer project from scratch to measure temperature using the respective sensors. This application note also comes with a combined example project, which can be readily used to interface PSoC 1 to both the DS18S20 and TMP75 sensors using the CY8CKIT-001 Development Board and CY8CKIT-036 Thermal Management Expansion board (TM EBK).

Figure 2. CYC8CKIT-001 with CY8CKIT-036 Attached

Introduction to the DS18S20 Sensor

The DS18S20 is a 1-wire interfaced temperature sensor capable of 9-bit temperature measurements (±0.5 ℃ accuracy). The data pin (DQ) of the sensor is used as a 1-wire port for communication with PSoC 1 using the protocol explained later in this document. The sensor outputs temperature in degrees Celsius format. This data is stored in a 16-bit sign-extended 2's-complement format in the sensor's internal memory. For examples of digital output data and corresponding temperature readings see Table 1. Later sections explain sensor functions, such as user configurable alarm settings, slave addressing, and how PSoC 1 can be used to implement the same.

Table 1. Temperature or Data Relationships

Temperature	Digital Output (Binary)	Digital Output (HEX)
+85.0 ℃	0000 0000 1010 1010	00AAh
+25.0 ℃	0000 0000 0011 0010	0032h
+0.5 ℃	0000 0000 0000 0001	0001h
0.0 ℃	0000 0000 0000 0000	0000h
–0.5 ℃	1111 1111 1111 1111	FFFFh
–25.0 ℃	1111 1111 1100 1110	FFCEh
–55.0 ℃	1111 1111 1001 0010	FF92h

Hardware Configuration

The 1-wire bus has a single data line; the PSoC 1 and the sensor interface to the data line through a 3-state or open drain port. This allows each device to **release** the data line when the device is not transmitting so that the bus is available for use by the other device.

The 1-wire bus requires an external pull-up resistor of approximately 4.7 kΩ; thus, the idle state for the 1-wire bus is logic High. Figure 1 shows the hardware configuration used for interfacing the DS18S20 in non-parasitic mode.

Memory Structure

The DS18S20's internal memory consists of an SRAM *Scratchpad* that stores 9 bytes of temperature or configuration data and a nonvolatile EEPROM storage that holds 2 bytes of user defined alarm thresholds. The memory structure is listed in Table 2.

Table 2. Scratchpad Registers and EEPROM

Scratchpad (SRAM)			EEPROM
Byte 0	Temperature LSB (AAh)		
Byte 1	Temperature MSB (00h)		
Byte 2	TH Register or User Byte	→	TH alarm Register
Byte 3	TL Register or User Byte	→	TL alarm Register
Byte 4	Reserved (FFh)		
Byte 5	Reserved (FFh)		
Byte 6	COUNT REMAIN (0Ch)		
Byte 7	COUNT PER ℃ (10h)		
Byte 8	CRC*		

*Power-up state depends on value(s) stored in EEPROM.

The DS18S20 measures and stores temperature in 2's-complement format in the temperature registers (**Byte 0** and **Byte 1)**. As given in Table 3, the LSB (**Byte 0**) contains the value of the temperature and MSB (**Byte 1**) contains the sign of the temperature.

Table 3. Temperature Registers

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
LSB	26	25	24	23	22	21	20	2-1
	Bit15	**Bit14**	**Bit13**	**Bit12**	**Bit11**	**Bit10**	**Bit9**	**Bit8**
MSB	S	S	S	S	S	S	S	S

S = sign (1 = negative, 0 = positive)

The TH and TL registers (**Byte 2** and **Byte 3**) are used to configure the alarm thresholds. Later sections explain how these values can be copied to the nonvolatile memory to preserve them when the device is not powered. The COUNT_REMAIN and COUNT_PER_C registers (**Byte 6** and **Byte 7**) can be used by host MCU to obtain >9-bit resolution values. This is possible because the sensor internally performs a 12-bit measurement, which is rounded down to 9-bit. The sensor calculates an 8-bit cyclic redundancy check (CRC) value and stores in **Byte 8**. This can be used by the host MCU to verify integrity of data read from the 1-wire sensor.

Slave Addressing

Multiple DS18S20 sensors can be present on the same 1-wire bus. This is possible due to the presence of a 64-bit serial code, which is unique to each sensor hardcoded into the device ROM. This makes it possible for the bus master to address a specific slave device by reading the ROM serial. The 64-bit ROM has the following structure:

Table 4. DS18S20 ROM Structure

64 bit ROM code	
Byte 0	DS18S20 1-wire Family Code (10h)
Byte 1	48-bit Serial Number
Byte 2	
Byte 3	
Byte 4	
Byte 5	
Byte 6	
Byte 7	8-bit ROM CRC

PSoC 1 can address and fetch temperature data from a specific DS18S20 on the bus using the 48-bit identifier (**Byte 1 to 6**). Each 1-wire device has a family code, for example the DS18S20 has a family code = *10h* (**Byte 0**), which ensures that only DS18S20 devices are addressed. **Byte 7** of the ROM contains a device calculated 8-bit CRC value, which can be used by the host MCU for checking data integrity.

Communication Overview

PSoC 1 acts as the bus master for 1-wire communication and follows the 1-wire protocol to communicate with DS18S20. The 1-wire protocol specifies the following transaction sequence for accessing devices:

1. Initialization
2. ROM commands
3. Device specific function commands

Initialization

All transactions on the 1-wire bus begin with an initialization sequence. The initialization sequence consists of a reset pulse transmitted by PSoC followed by a presence pulse transmitted by the sensor. The presence pulse lets PSoC know that the sensor is on the bus and is ready to operate. Timing for the reset and presence pulses is detailed in Figure 32 in the Appendix.

ROM Commands

The DS18S20 ROM commands are used in relation with the 64-bit ROM code to identify and address specific devices on the 1-wire bus. These commands enable PSoC to send/receive isolated data to/from the sensor of interest in a system with multiple sensors. A list of ROM codes supported by DS18S20 is given below; their functions are explained in the ROM functions section. Information on the ROM codes can be found in greater detail in the DS18S20 datasheet.

Table 5. ROM Commands

ROM Command	Code
Search Rom	F0h
Read ROM	33h
Match ROM	55h
Skip ROM	CCh
Alarm Search	ECh

DS18S20 Function Commands

The sensor function commands allow the master to configure and communicate with the sensor. The sensor can transmit data to the PSoC only when it is requested. The sensor function commands related to temperature measurement are summarized in Table 6.

Table 6. DS18S20 Function Command Set (from DS18S20 Datasheet)

COMMAND	DESCRIPTION	PROTOCOL	1-Wire BUS ACTIVITY AFTER COMMAND IS ISSUED
TEMPERATURE CONVERSION COMMANDS			
Convert T	Initiates temperature conversion.	44h	DS18S20 transmits conversion status to master (not applicable for parasite-powered DS18S20s).
MEMORY COMMANDS			
Read Scratchpad	Reads the entire scratchpad including the CRC byte.	BEh	DS18S20 transmits up to 9 data bytes to master.
Write Scratchpad	Writes data into scratchpad bytes 2 and 3 (TH and TL).	4Eh	Master transmits 2 data bytes to DS18S20.
Copy Scratchpad	Copies TH and TL data from the scratchpad to EEPROM.	48h	None

COMMAND	DESCRIPTION	PROTOCOL	1-Wire BUS ACTIVITY AFTER COMMAND IS ISSUED
Recall E^2	Recalls TH and TL data from EEPROM to the scratchpad.	B8h	DS18S20 transmits recall status to master.
Read Power Supply	Signals DS18S20 power supply mode to the master.	B4h	DS18S20 transmits supply status to master.

DS18S20 Example Code

The following section walks-through an example implementation of the interface with a DS18S20 sensor. After following the steps given in this example, you should be able to interface a CY8C28xxx microcontroller to a DS18S20 sensor and display the measured temperature on the LCD. The DS18S20 library files used in this example can be found as an attachment to the application note.

Hardware Setup

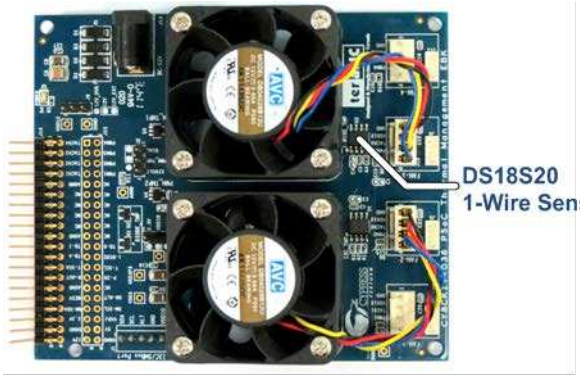
The examples in this application note are designed for use with the following hardware:

- CY8CKIT-001 Development Board
- CY8CKIT-036 Thermal Management Expansion board (TM EBK)
- CY8C28000-24PVXI on a CY8C28 family processor module.

It is to be noted that any PSoC 1 device that supports the OneWire user module can be used for this application. A list of such devices can be found in the OneWire user module datasheet.

Figure 3. DS18S20 - Hardware Setup

Figure 4. Location of DS18S20 on CY8CKIT-036

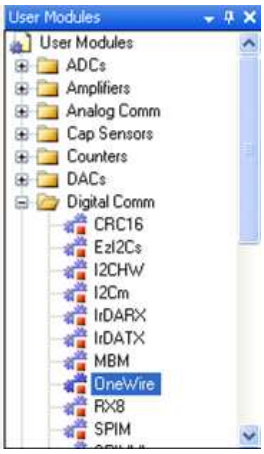


Creating a 1-Wire Project from Scratch

The following sections steps through the procedure to create a project to interface PSoC 1 to a DS18S20 sensor. To create a PSoC Designer project that configures a PSoC to work as a 1-wire master and read temperature and display temperature from DS18S20 sensor, follow the steps given below:

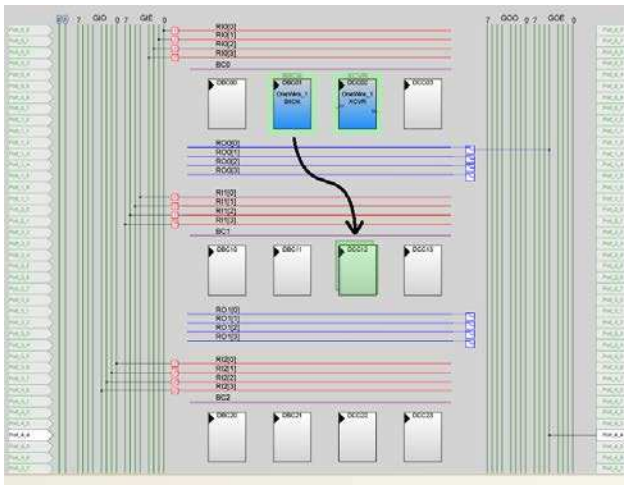
1. Open PSoC Designer software
2. Click the **File** menu and select **New Project**.
3. Choose any desired name for the project, for e.g., “OneWire_DS18S20” and click **OK**.
4. Select **View Catalog** to select the target device as **CY8C28645-24LTXI**. Click **Select** when the device has been selected.
5. Select the language as ‘C’ to generate the ‘Main’ file in C language and click **OK**. PSoC Designer will create the project’s base configuration.
6. Locate the “OneWire” user module in the user module catalog under “Digital Comm” sub-folder.

Figure 5. OneWire User Module Location



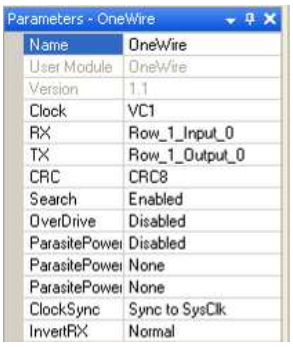
7. Right-click on the user module and select **Place**. This operation will place the OneWire BitCLK and XCVR modules in two digital blocks. To ease internal routing, re-place them by dragging and dropping the modules into the new slot at DCC12 as shown in the following figure.

Figure 6. Drag and Drop Modules into New Slot



8. Set the user module parameters as given in the following figure.

Figure 7. OneWire User Module Parameters



Note on Parasite Power mode: The CY8CKIT-036 provides power to the DS18S20 using the VDDIO line and hence parasitic power mode is not used. If this mode is to be used, the **ParasitePower** option should be enabled in the user module parameters.

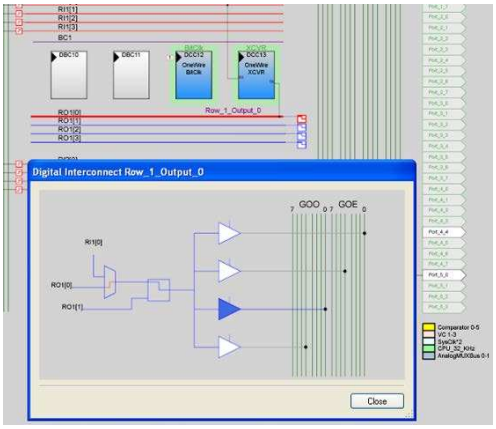
9. For debugging purposes and for displaying the temperature values, place an LCD user module from **"Misc Digital -> LCD"**. Set the following parameters:

Figure 8. LCD User Module Parameters

10. The DQ pin of the DS18S20 on the Thermal management EBK is connected to **Port_4_4** of PSoC on the 001 DVK. To route the OneWire module to the external pins, make the following connections:

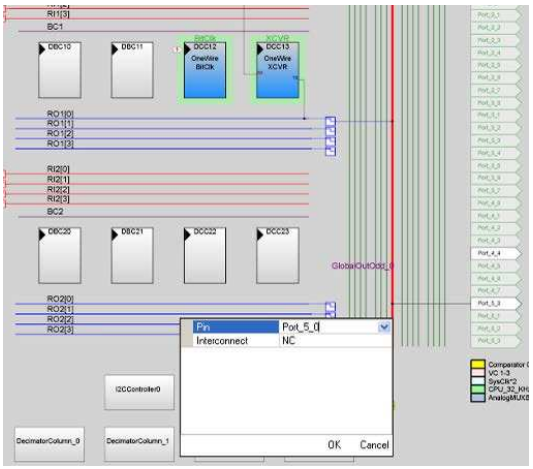
    a. Click on **Row_1_Output_0** line and select the connection to **GlobalOutOdd_0** as shown in the following figure.

Figure 9. Configuring Row_1_Output_0



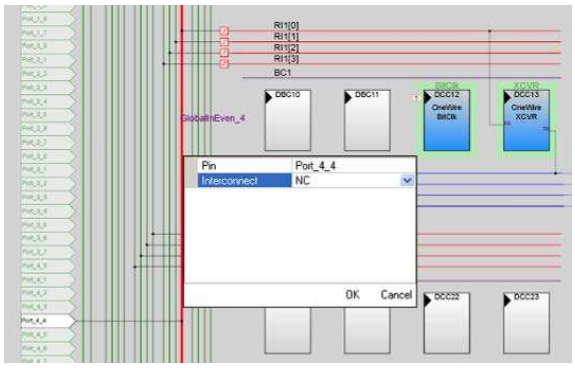
b. Click on the **GlobalOutOdd_0** line and select **Port_5_0** for the "Pin" option. This makes a connection from TX terminal of the module to Port_5_0.

Figure 10. Configuring GlobalOutOdd_0 Connection



11. Similarly, select the **GlobalInEven_4** line and select **Port_4_4** option. This connects Port_4_4 to the RX terminal of the module.

Figure 11. Routing Input to User Module



12. Even though the OneWire user module has two terminals (RX and TX), this is presented externally as a single port that connects to the 1-wire bus. Thus, the RX and TX pins (Port_4_4 and Port_5_0) have to be shorted. This can be done by externally shorting Port_4_4 and Port_5_0 pins. This short can also be made internally by using the AnalogMUXBus using the following method:

a. From the **Pinout editor** window, expand the entry for P4[4]

b. Set the **AnalogMUXBus** option to "AnalogMUXBus_1"

c. Repeat the above for P5[0].

Figure 12. Connecting Pins to the AnalogMUXBus

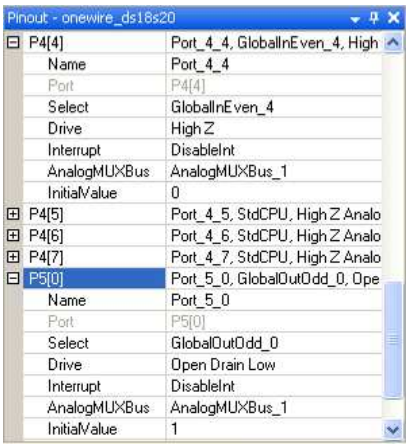
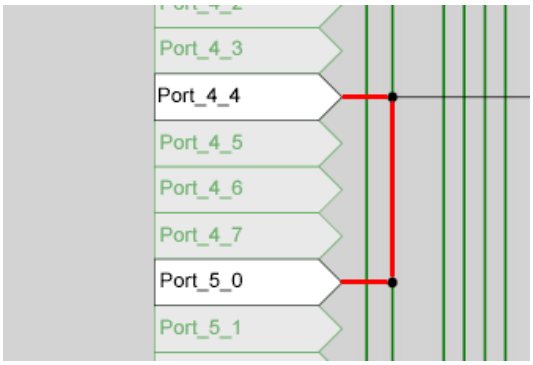


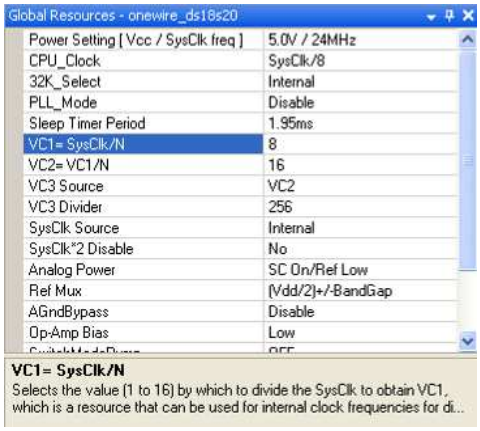
Figure 13. Internal Short Highlighted



Note that the drivemode of Port_4_4 is fixed to high impedance (HI-Z) as it is an input pin. The drivemode of Port_5_0 can be altered to open drain low (ODL) or pull-up, depending on the external circuitry. We leave it as ODL because we have an external pull-up resistor on the sensor board.

13. Notice that we have selected VC1 (Clock divider 1) as the clock source for OneWire timing generation. For proper operation, the value of clock needs to be **3 MHz**. To set VC1 = 3 MHz, change the VC1 clock division parameter in the Global Resources window from "16" to "8". This will result in a frequency of SysClk/8 or 24 MHz / 8 = 3 MHz.

Figure 14. Setting OneWire Timing Using Clock Dividers



14. Click on **Build -> Generate Configuration** files or Ctrl+F6 to populate the project with necessary APIs/libraries.
15. Right click “OneWire_DS18S20” (name of the project) folder and select “Add File”. Navigate to the folder with the DS18S20 libraries (under AN2163/DS18S20 library/) to import the *DS18S20.c* and *DS18S20.h* files into the project.
16. Open *main.c* from the Workplace Explorer window and replace it with the contents of file “*DS18S20_main.c*”
17. Click **Build project** (F7) from **Build** menu. Project should compile without any errors.
18. Configure the 001 DVK with the jumper settings as given in the following figure. The 036 kit does not need any jumpers to be populated and should be connected to PORT A (P7) of the 001 kit as shown in Figure 3. Connect a MiniProg1/3 programmer to the “PROG” header of the processor module.

    Figure 15. Jumper Settings

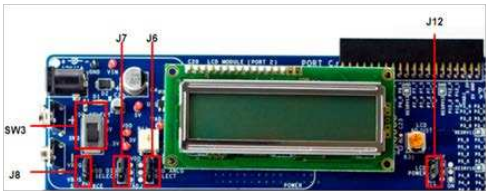


Table 7. Jumper Settings for 001 Kit

JUMPER	SETTING
J6	VDD_ANALOG to VDD
J7	VDD_DIG to VDD
J8	VDD to VREG
J12	LCD to ON
SW3	5 V Position

19. Select “Program part” from **Program** menu. Select “Power Cycle” for the Acquire mode and click on the Program icon. Once programming is complete, click on **Toggle Power** to power up the boards from the MiniProg1/3.
20. The LCD should display the ambient temperature as measured by DS18S20 in degree Celsius.

    Figure 16. LCD Showing DS18S20 Temperature

DS18S20 Advanced Functions

Overview of the DS18S20 Example Project

The previous section demonstrated how a PSoC Designer project can be setup to perform a simple temperature read operation from DS18S20. The DS18S20 provides certain advanced functions, such as CRC for ROM and Scratchpad, user configurable ALARM thresholds, and so on. The DS18S20 library provided with this application note provides API functions to simplify use of the advanced functions of the DS18S20.

The included example project “TempSense_1wire_2wire” integrates the DS18S20 library and provides usage examples of the APIs to access the advanced functions of the sensor.

The following flow charts show the routines used in the project.

Figure 17. main() Routine Flow Chart

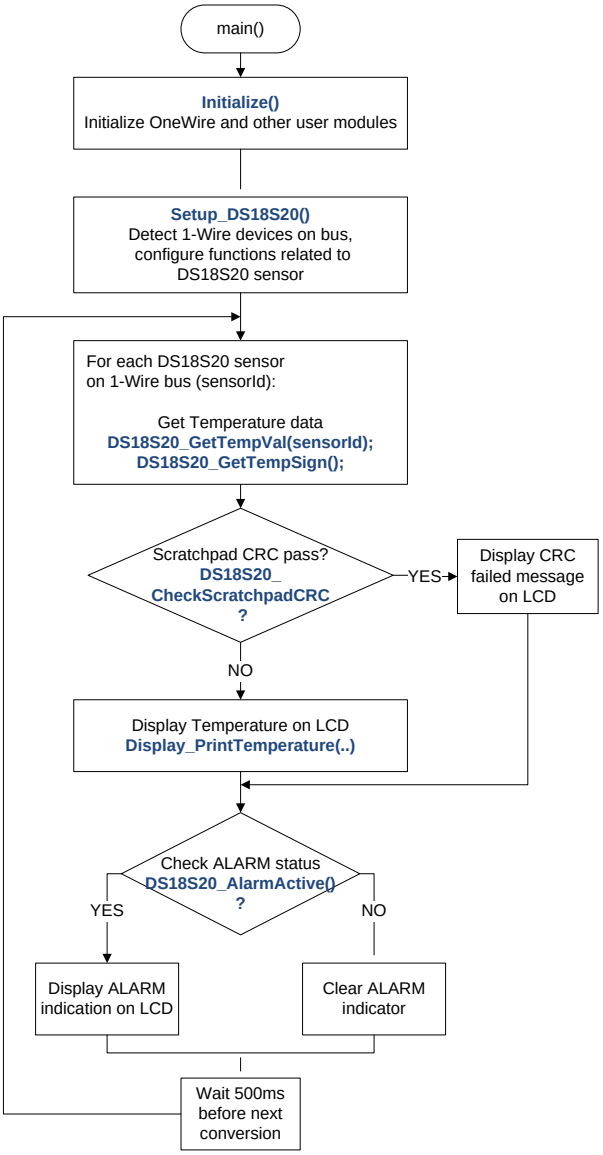


Figure 18. Setup_DS18S20() Routine Flow Chart

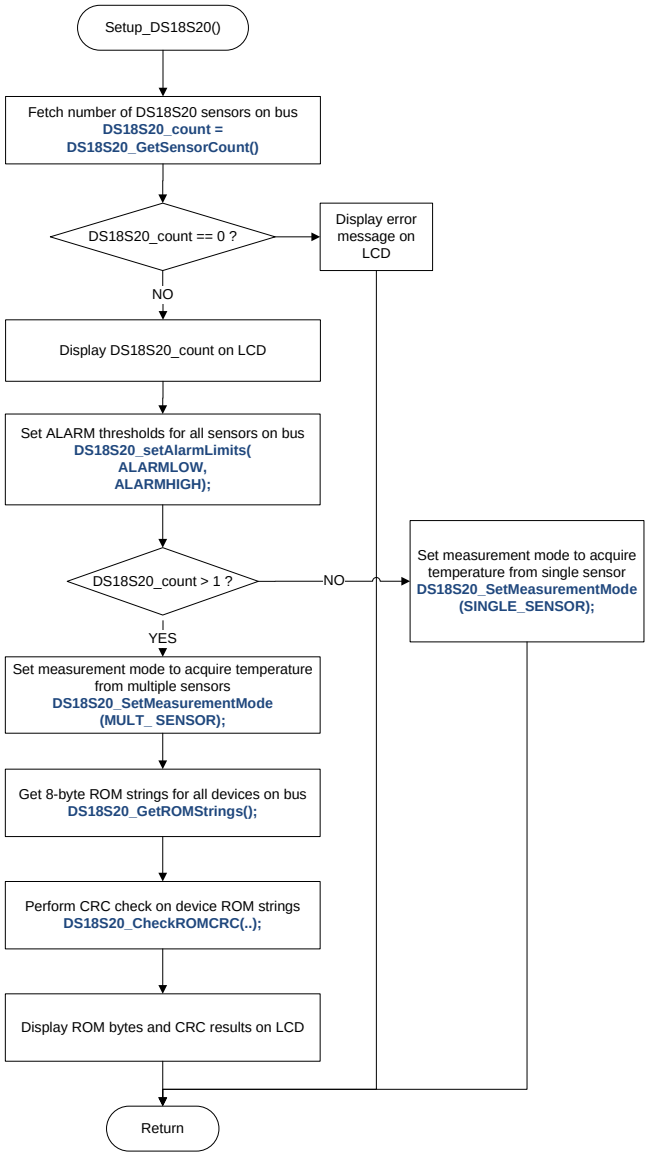
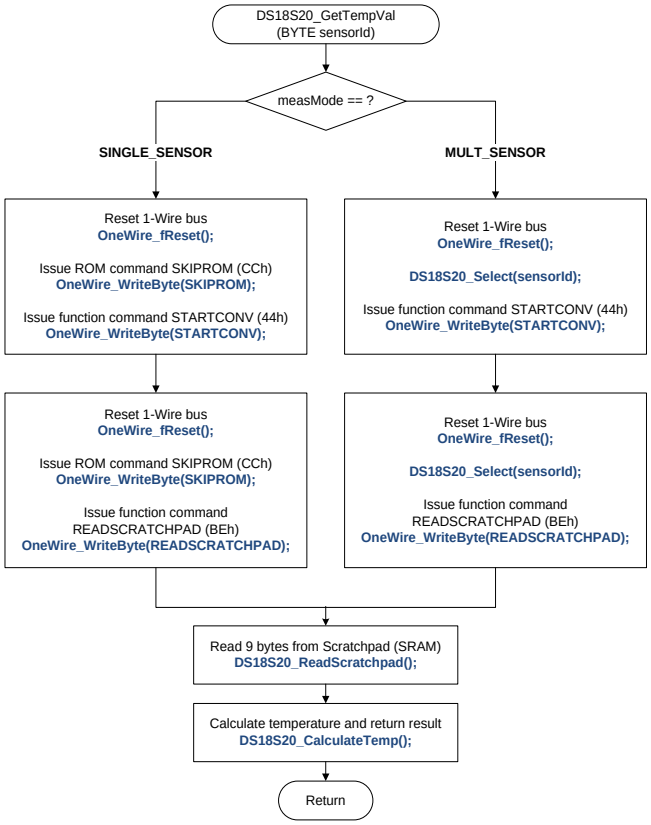


Figure 19. DS18S20_GetTempVal() Routine Flow Chart



Cyclic Redundancy Check

Both the ROM serial and the SRAM Scratchpad include an 8-bit CRC value for data integrity verification. The Master can make use of the CRC byte to make sure data read from the device is valid. The polynomial used by DS18S20/PSoC 1 for computing the CRC is:

```
CRC = X8 + X5 + X4 + 1
```

To check for a valid CRC, PSoC reads the entire content of the ROM/Scratchpad including the CRC byte generated by the device. PSoC computes an 8-bit CRC using the OneWire_bCRC8(BYTE bData) function. First, a OneWire_ClearCRC8() command is issued to reset the internal shift register. The data bytes read from the sensor (from array *baScratchpad[]*) are then fed byte-wise to the OneWire_bCRC8 function. After the last byte (CRC byte generated by device) is passed, the OneWire_bCRC8 function should return a 0, indicating valid data.

```
BYTE DS18S20_CheckScratchpadCRC(void)
{
   BYTE loopVar;

   /* Clear CRC before computing */
   OneWire_ClearCRC8();

   /* Sequentially calculate CRC8 on the first 8
bytes
      of Scratchpad (TEMP_LSB to COUNT_PER_C) */
   for (loopVar = 0; loopVar < 8; loopVar++)
   {
      OneWire_bCRC8(scratchpad[loopVar]);
   }

   /* Result after calculating CRC (including
the CRC byte from sensor)
      should be equal to 0 for valid data */
   if (OneWire_bCRC8(scratchpad[CRC]))
   {
      /* CRC failed. Invalid data! */
      return 0;
   }
   else
   {
      /* CRC passed. Data is valid */
      return 1;
   }
}
```

Extended Resolution

The DS18S20 returns a 9-bit temperature output in the Temperature registers (**Byte0** and **Byte1** in Scratchpad). However the sensor internally measures temperature with greater than 9-bit accuracy before it is rounded down. This **extended resolution** value can be accessed using the COUNT_REMAIN and COUNT_PER_C registers using the equation given in the datasheet. In the example project, measurements with an extended resolution of 0.25 °C is possible by setting the macro EXTENDED_RESMODE to TRUE in the header file "*DS18S20.h*".

```
#define EXTENDED_RESMODE   TRUE
```

ROM Functions

As mentioned earlier, the 64-bit ROM serial provides a way to address 1-wire devices. Addressing is performed by sending the ROM commands (Table 5) after a Reset command. The *DS18S20_GetROMStrings* function provides a way to programmatically fetch the device ROM bytes from all sensors on the bus:

```
void DS18S20_GetROMStrings(void)
{
   BYTE currDevice;

   /* Initiate reset */
   OneWire_fReset();

   /* Address the first device */
   OneWire_fFindFirst();

   /* Fetch ROM bytes and address next device on
bus */
   for (currDevice = 0; currDevice <
sensorCount; currDevice++)
   {
```

```
      OneWire_GetROM(romPointer[currDevice]);
      OneWire_fFindNext();
   }
}
```

When there is just one device on the bus, the SKIPROM command can be used; no addressing protocol is followed. In this mode, all 1-wire devices on the bus will respond to subsequent commands and send data. This is to be avoided when multiple devices are present, as every one of them will respond, causing data collision on the bus, resulting in a CRC error.

Sequence for communicating with a single sensor:

```
OneWire_fReset();
OneWire_WriteByte(SKIPROM);
OneWire_WriteByte(STARTCONV);

OneWire_fReset();
OneWire_WriteByte(SKIPROM);

<followed by commands to read Scratchpad>
```

If multiple sensors are present, the MATCHROM command followed by 8 bytes of ROM data can be used to address a particular sensor with that ROM serial. Any subsequent function commands will be responded to only by the selected sensor. *DS18S20_Select* implements this protocol:

```
void DS18S20_Select(BYTE sensorId)
{
   BYTE loopVar;
   OneWire_fReset();
   /* Issue MATCHROM command followed by sending
8 bytes of ROM data to select device on bus */
   OneWire_WriteByte(MATCHROM);

   /* Send 8-byte ROM string of selected device
*/
   for (loopVar = 0; loopVar < 8; loopVar++)
   {
      OneWire_WriteByte(
         baRomPointer[sensorId][loopVar]);
   }
}

<followed by commands to read Scratchpad>
```

A list of all the functions in DS18S20 library is given in the following table.

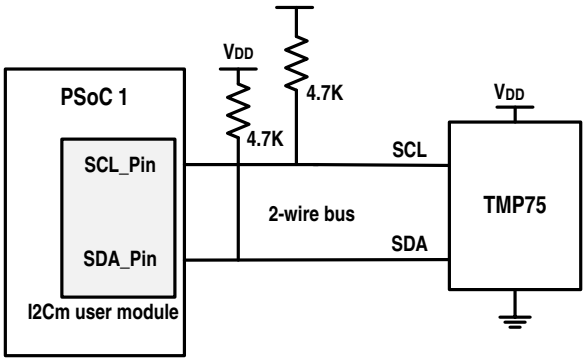
Table 8. DS18S20 Library Functions

Function Prototype	Description
`BYTE DS18S20_GetSensorCount(void);`	Returns the number of DS18S20 devices on 1-wire bus
`WORD DS18S20_GetTempVal(BYTE sensorId);`	Get value of temperature from sensor by reading Scratchpad of specific sensor
`BOOL DS18S20_GetTempSign(void);`	Get sign of temperature from the Scratchpad data
`void DS18S20_ReadScratchpad(void);`	Read 9-bytes from device Scratchpad into internal memory
`WORD DS18S20_CalculateTemp(void);`	Calculate temperature value from the read Scratchpad data
`BYTE DS18S20_CheckScratchpadCRC(void);`	Validate CRC for Scratchpad data
`void DS18S20_setAlarmLimits(CHAR alarmLowVal, CHAR alarmHighVal);`	Sets the low and high temperature limit to define an ALARM condition
`BOOL DS18S20_AlarmActive(void);`	Query devices on the bus for an ALARM signal
`BYTE * DS18S20_GetLCDString(WORD sensorVal, BOOL sign);`	Convert temperature into an ASCII string which can be displayed on the LCD
`BOOL DS18S20_FamilyCheck(BYTE * deviceROM);`	Indicates whether specific 1-wire device family is present on bus
`BOOL DS18S20_CheckROMCRC(BYTE * deviceROM);`	Validate CRC for ROM data
`void DS18S20_GetROMStrings(void);`	Scans bus for devices and fetches ROM strings from each device sequentially. DS18S20_GetSensorCount() has to be called atleast once before calling this function
`void DS18S20_SetMeasurementMode(BYTE mode);`	Update local variable used to indicate measurement mode
`BYTE DS18S20_TwosComplement(BYTE data);`	Perform and return Two's complement of an 8-bit number
`void DS18S20_Select(BYTE sensorId);`	Address a specific sensor on the bus using its ROM string

Two-Wire Interface

The Two-Wire or inter-integrated circuit (I^2C) interface is a chip-to-chip serial communications standard developed by Phillips Semiconductor. Data transfer between devices is made possible using the I^2C bus, which consists of two physical lines: serial data (SDA), and serial clock (SCL). I^2C is an industry standard interface and needs no introduction here. Detailed information about the I^2C protocol and the related user modules in PSoC can be found in the application note "AN50987 - Getting Started with I^2C in PSoC 1".

Figure 20. Hardware Connections



TMP75 Sensor

The TMP75 is a Two-Wire (I^2C) serial output sensor, capable of temperature measurements with a resolution of 0.0625 °C. TMP75 allows for 8 slave devices on the bus, while the TMP175, which is identical in operation, allows up to 27 devices. The TMP175 and TMP75 are specified for operation over a temperature range of −40 ℃ to +125 ℃.

The TMP75 uses 7-bit slave addressing with one direction bit (R/W). For an overview of the basics of the I^2C protocol, see the application note, AN50987 - Getting Started with I2C in PSoC 1.

The temperature to data output relation is given in the following table.

Table 9. Temperature/Data Relationships

Temperature	Digital Output (Binary)	Digital Output (HEX)
+128 ℃	0111 1111 1111	7FFh
+127.9375 ℃	0111 1111 1111	7FFh
+100 ℃	0110 0100 0000	640h
+80 ℃	0101 0000 0000	500h
+75 ℃	0100 1011 0000	4B0h
+50 ℃	0011 0010 0000	320h
+25 ℃	0001 1001 0000	190h
+0.25 ℃	0000 0000 0100	004h
0 ℃	0000 0000 0000	000h
-0.25 ℃	1111 1111 1100	FFCh
-25 ℃	1110 0111 0000	E70h
-55 ℃	1100 1001 0000	C90h

TMP75 Registers

The TMP75 internal register structure consists of four 8-bit registers as given in Table 11. These registers are selected using a **Pointer Register**. **Bit1** and **Bit0** of the Pointer Register are used to address one of the four TMP75 registers.

Table 10. Pointer Register

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
0	0	0	0	0	0	**Pointer Value**	

Table 11. TMP75 Registers

Pointer Value (Bit1: Bit0)	Register Selected
00h	Temperature Register (Read-only)
01h	Configuration Register (Read/Write)
10h	T_{LOW} Register (Read/Write)
11h	T_{HIGH} Register (Read/Write)

To read/write to a particular register, the I^2C Master has to first address it by writing one byte of data containing the **Pointer Value** to the device. This is equivalent to using **Pointer Value** as the I^2C sub-address before reading/writing data bytes to the subsequent memory locations.

A brief description of the four TMP75 registers is given below.

Temperature Registers

The 12-bit temperature value measured by the sensor is stored in two bytes in the Temperature Registers in the format given in the following table.

Table 12. Temperature Register – 12-bit Resolution

Byte 1 (MSB)							
Bit7	**Bit6**	**Bit5**	**Bit4**	**Bit3**	**Bit2**	**Bit1**	**Bit0**
T11	T10	T9	T8	T7	T6	T5	T4

Byte 2 (LSB)							
Bit7	**Bit6**	**Bit5**	**Bit4**	**Bit3**	**Bit2**	**Bit1**	**Bit0**
T3	T2	T1	T0	0	0	0	0

When a 9-bit conversion is performed, the register format is as given below:

Table 13. Temperature Register – 9-bit Resolution

Byte 1 (MSB)							
Bit7	**Bit6**	**Bit5**	**Bit4**	**Bit3**	**Bit2**	**Bit1**	**Bit0**
T8	T7	T6	T5	T4	T3	T2	T1

Byte 2 (LSB)							
Bit7	**Bit6**	**Bit5**	**Bit4**	**Bit3**	**Bit2**	**Bit1**	**Bit0**
T0	0	0	0	0	0	0	0

Configuration Register

The Configuration Register is an 8-bit read/write register used to store bits that control the operational modes of the temperature sensor. The format of the Configuration Register is given in Table 14.

Table 14. Configuration Register

Bit7	**Bit6**	**Bit5**	**Bit4**	**Bit3**	**Bit2**	**Bit1**	**Bit0**
T8	T7	T6	T5	T4	T3	T2	T1

The configuration register is used to configure the following functions:

- Shutdown Mode (SD)
- Thermostat Mode (TM)
- Polarity (POL)
- Fault Queue (F1/F0)
- Converter Resolution (R1/R0)
- One-Shot (OS)

See the TMP75 Datasheet for a detailed explanation of these functions.

T_{LOW} and T_{HIGH} Register

The TMP75 sensor has an ability to produce an ALERT signal (on its ALERT pin) whenever the measured temperature exceeds certain predefined limits. The T_{LOW} and T_{HIGH} registers are used to configure the temperature limits for the ALERT indication function. The behavior of the ALERT also depends on the Fault Queue and Polarity settings. The T_{LOW} and T_{HIGH} registers have the same format as the Temperature registers (12-bit values in two bytes).

Table 15. T_{LOW} / T_{HIGH} Register Format

Byte 1 (MSB)							
Bit7	**Bit6**	**Bit5**	**Bit4**	**Bit3**	**Bit2**	**Bit1**	**Bit0**
T11	T10	T9	T8	T7	T6	T5	T4

Byte 2 (LSB)							
Bit7	**Bit6**	**Bit5**	**Bit4**	**Bit3**	**Bit2**	**Bit1**	**Bit0**
T3	T2	T1	T0	0	0	0	0

TMP75 Example Code

The following section walks-through an example implementation of the interface with a TMP75 sensor. After following the steps given in this example, you should be able to interface a CY8C28xxx microcontroller to a TMP75 sensor and display the measured temperature on the LCD. The TMP75 library files used in this example can be found as an attachment to the application note.

Hardware Setup

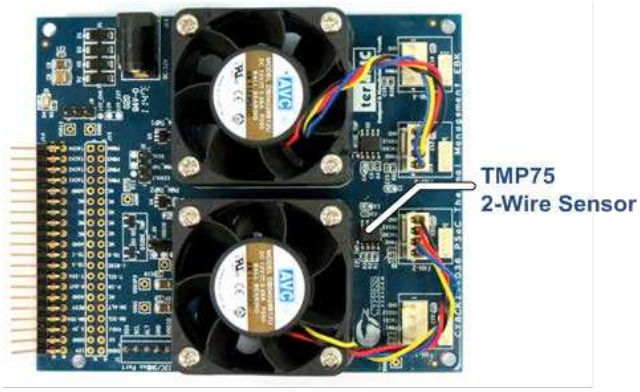
The examples in this application note are designed for use with the following hardware:

- CY8CKIT-001 Development Board
- CY8CKIT-036 Thermal Management Expansion board (TM EBK)
- CY8C28000-24PVXI on a CY8C28 family processor module.

The device used is a CY8C28000-24PVXI on a CY8C28 family processor module. However, any PSoC 1 device that supports the I2Cm user module can be used for this application.

Figure 21. TMP75 - Hardware Setup

Figure 22. TMP75 Sensor Location

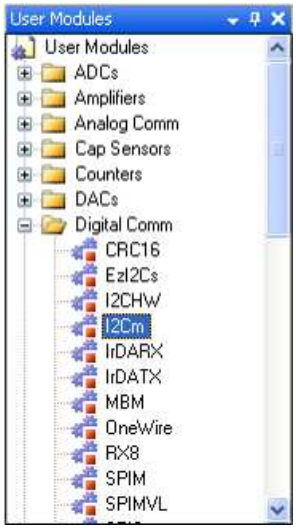


Creating a Two-Wire Project from Scratch

To create a PSoC Designer project that configures a PSoC to work as Two-Wire (I^2C) master and read temperature from TMP75 sensor, follow the steps given below:

1. Open PSoC Designer software
2. Click the **File** menu and select **New Project**.
3. Choose any desired name for the project, for e.g., "TwoWire_TMP75" and click **OK**.
4. Select **View Catalog** to select the target device as **CY8C28645-24LTXI**. Click **Select** when the device has been selected.
5. Select the language as 'C' to generate the 'Main' file in C language and click **OK**. PSoC Designer will create the project's base configuration.
6. Locate the "I2Cm" user module in the **User modules** catalog under "Digital Comm" sub-folder.

Figure 23. Select the I2Cm User Module



Note This project uses an I2Cm (I^2C Master user module) for I^2C communication. The I2Cm is implemented in firmware (consumes more CPU processing), but provides more flexibility in terms of choice of SDA, SCL pins and does not consume any digital blocks.

The I2CHW user module can also be used for this purpose. This provides hardware buffers for I^2C data, but places a restriction on the PSoC pins, which can be used for I^2C (P1[0] - P1[1] or P1[5] - P1[7]).

7. Right-click on the I2Cm user module and select **Place**.
8. Set the user module parameters as given below. The Ports are selected to match with the connections (T-SDA and T-SCL) to the TM EBK.

   Figure 24. I2Cm Parameters

9. For debugging purposes and for displaying the temperature values, place an LCD user module from "**Misc Digital -> LCD**". Set the parameters as shown in the following figure:

Figure 25. LCD Parameters

10. Click on **Build -> Generate Configuration** files or Ctrl+F6 to populate the project with necessary APIs/libraries.
11. Right click “TwoWire_TMP75” (name of the project) folder and select “Add File”. Navigate to the folder with the TMP75 libraries (under AN2163/TMP75 library) to import the *TMP75.c* and *TMP75.h* files into the project.
12. Open *main.c* from the Workplace Explorer window and replace with the contents of file “*TMP75_main.c*”
13. Click **Build project** (F7) from **Build** menu. Project should compile without any errors.
14. Configure the 001 DVK with the jumper settings as given in the following figure. The 036 kit does not need any jumpers to be populated and should be connected to PORT A (P7) of the 001 kit as shown in Figure 21. Connect a MiniProg1/3 programmer to the “PROG” header of the processor module.

Figure 26. Jumper Settings

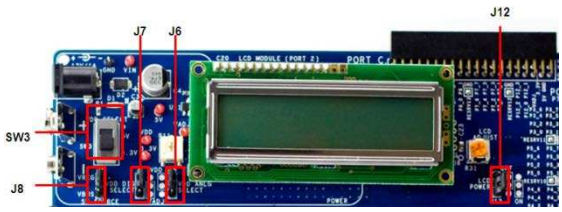
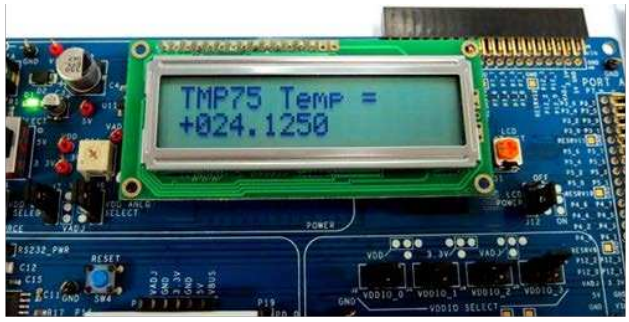


Table 16. Jumper Settings for 001 Kit

JUMPER	SETTING
J6	VDD_ANALOG to VDD
J7	VDD_DIG to VDD
J8	VDD to VREG
J12	LCD to ON
SW3	5 V Position

15. Select “Program part” from **Program** menu. Select “Power Cycle” for the Acquire mode and click on the Program icon. Once programming is complete, click on **Toggle Power** to power-up the boards from the MiniProg1/3.
16. The LCD should display the ambient temperature as measured by TMP75 in degree Celsius

Figure 27. LCD Showing TMP75 Temperature



Overview of the TMP75 Example Project

The previous section demonstrated how a PSoC Designer project can be setup to perform a simple temperature read operation from TMP75. In that case, the TMP75 was operating with the default/power-up configuration in which all the bits in the configuration registers are loaded with a '0' value. This is the simplest configuration to quickly fetch temperature data. The included example project "TempSense_1wire_2wire" integrates the TMP75 library and provides usage examples for the provided library's APIs to configure the different operating modes on the TMP75. The following flow charts illustrate this project's code.

Figure 28. main() Routine Flow Chart

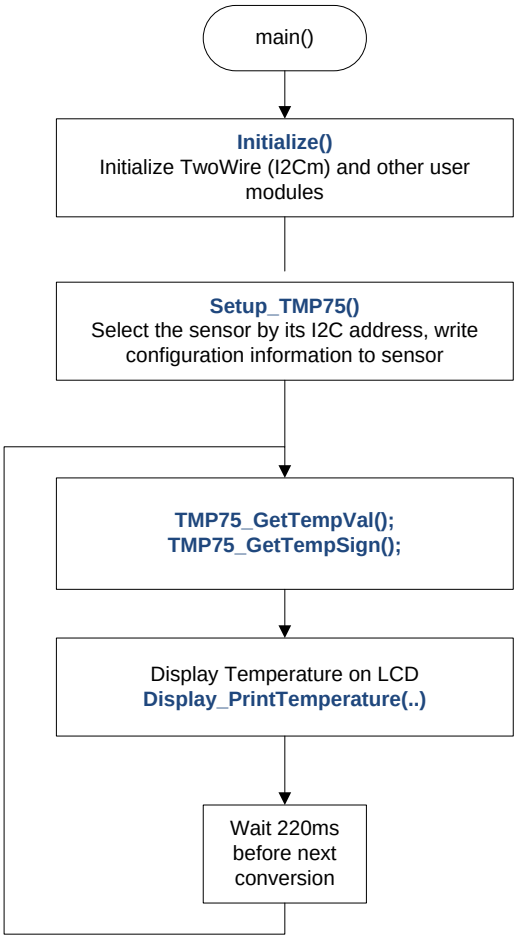


Figure 29. Setup_TMP75() Routine Flow Chart

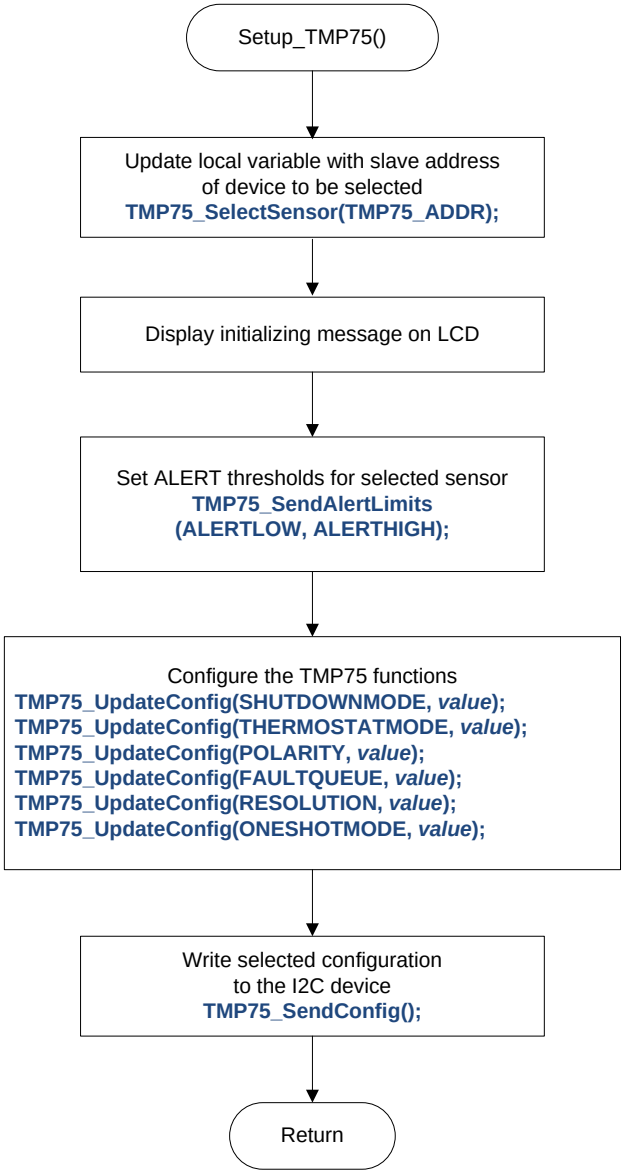
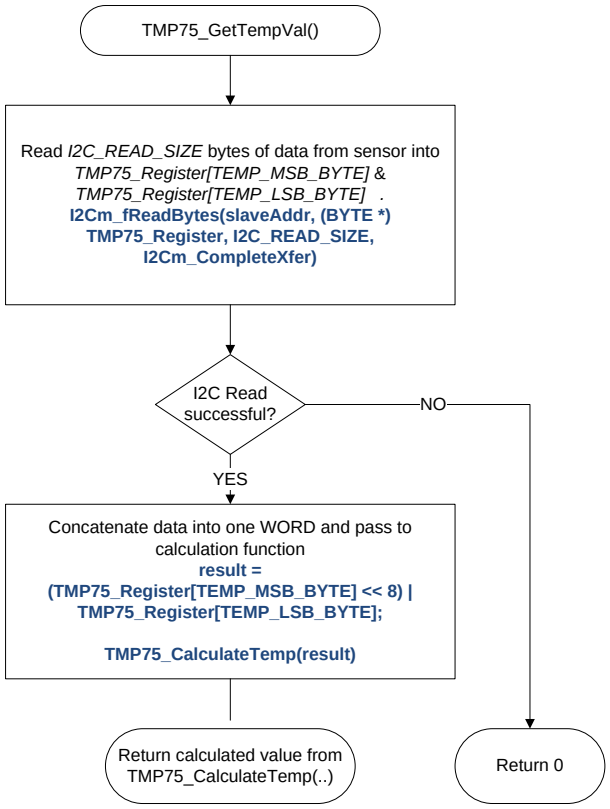


Figure 30. TMP75_GetTempVal() Routine Flow Chart



Configuring the TMP75

The TMP75 has six features that can be configured using the internal configuration register. The power-up/reset value of the configuration register is all bits set to 0.

The TMP75 library maintains a configuration variable that holds the value of the TMP75 configuration register. The following sub routine can be used to update this variable to configure a specific feature.

```
void TMP75_UpdateConfig(BYTE param, BYTE data)
```

Where `bParam` is the feature to be configured and `bData` is the setting for that feature. The following table shows the macros in "*TMP75.h*" that can be used as attributes for this sub routine.

Table 17. MACROs for TMP75_UpdateConfig() Parameters

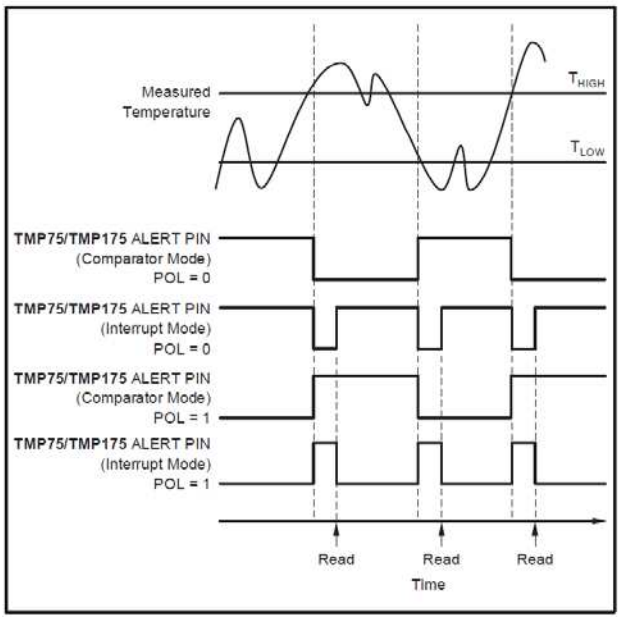
bParam	bData
SHUTDOWNMODE	SHUTDOWN_DIS
	SHUTDOWN_EN
THERMOSTATMODE	COMPARATOR
	INTERRUPT
POLARITY	ACTIVELOW
	ACTIVEHIGH
FAULTQUEUE	FAULTS_1
	FAULTS_2
	FAULTS_4
	FAULTS_6
RESOLUTION	RES_9_BITS
	RES_10_BITS
	RES_11_BITS
	RES_12_BITS
ONESHOTMODE	ONESHOT_DIS

After the configuration for the respective features are updated, the configuration byte is sent to the device using the function:

```
BOOL TMP75_SendConfig(void);
```

The ALERT Function

Figure 31. ALERT Pin Behavior (from TMP75 Datasheet)



The ALERT function is configured using the POLARITY and FAULT_QUEUE parameters in the Configuration register. The ALERT temperature thresholds are set using the T_{LOW} and T_{HIGH} registers. The following function can be used to change the temperature limits:

```
TMP75_SendAlertLimits(CHAR tLow, CHAR tHigh)
```

Note that the function parameters are 8-bit signed numerals (-127 °C to +127 °C). If a better resolution is needed for the ALERT limit values, the function definition should be modified to accept **float** values.

A list of functions used to configure TMP75 is given in the following table:

Table 18. TMP75 Functions

Function prototype	Description
`void TMP75_SelectSensor(BYTE addr);`	Update variable with slave address of device to be selected
`void TMP75_UpdateConfig(BYTE param, BYTE data);`	Update configuration variable
`BOOL TMP75_SendConfig(void);`	Write configuration data to the sensor
`DWORD TMP75_GetTempVal(void);`	Fetch temperature data using I^2C Read operation and return calculated temperature value
`BOOL TMP75_GetTempSign(void);`	Get sign of temperature from the readback data
`DWORD TMP75_CalculateTemp(WORD tempData);`	Calculate temperature value from the readback data
`BYTE* TMP75_GetLCDString(DWORD sensorVal, BOOL sign);`	Convert temperature into an ASCII string which can be displayed on the LCD
`BOOL TMP75_SendAlertLimits(CHAR tLow, CHAR tHigh);`	Sets the low and high temperature limit to define an ALERT condition
`WORD TMP75_TwosComplement(WORD data);`	Perform Two's complement operation on a 12-bit number

Summary

Thermal monitoring and management solutions based on 1-Wire and Two-Wire digital temperature sensors can be quickly and easily designed using PSoC 1. The DS18S20/TMP75 libraries provided with this application note enables easy access to the sensor functions and simplifies their configuration process.

PSoC's unique ability 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.

Related Application Notes

- AN78920 - PSoC® 1 Temperature Measurement Using Diode
- AN78737 - PSoC® 1 - Temperature Sensing Solution using a TMP05/TMP06 Digital Temperature Sensor
- AN78692 - PSoC® 1 - Intelligent Fan Controller

About the Author

Name:	Arvind Krishnan
Title:	Applications Engineer
Background:	M.Sc. (Hons) Physics, Bachelors in Electrical and Electronics Engineering from Birla Institute of Technology and Sciences, Pilani (Goa Campus), India
Contact:	arvi@cypress.com

Appendix

1-Wire Protocol - Read/Write Time Slots

The PSoC writes data to the 1-Wire digital sensor during write time slots and reads data from the digital sensor during read time slots. One bit of data is transmitted over the 1-wire bus per time slot.

Write Time Slots

There are two types of write time slots: “Write 1” time slots and “Write 0.” time slots. These correspond to logic 1 and logic 0, respectively. All write time slots must be a minimum of 60 µs duration with a minimum 1 µs recovery time between individual write slots. The PSoC initiates both types of write time slot by pulling the 1-wire bus low.

To generate a Write 1 time slot the PSoC must release the 1-wire bus within 15 µs of pulling it low. When the bus is released, the external pull-up resistor pulls the bus high. To generate a Write 0 time slot the PSoC must continue to hold the bus low for the duration of the time slot, at least 60 µs.

The digital sensor samples the 1-wire bus during a window that lasts from 15 µs to 60 µs after PSoC initiates the write time slot. If the bus is high during the sampling time, a 1 is written to the sensor. If the line is low, a 0 is written to the sensor.

Read Time Slots

The digital sensor can only transmit data to PSoC when PSoC is issuing a read command. All read time slots must be a minimum of 60 µs in duration with a minimum of a 1 µs recovery time between slots.

The PSoC initiates the read time slot by pulling the 1-wire bus low for a minimum of 1 µs and then releasing the bus (Figure 14). After the PSoC initiates the read time slot, the digital sensor begins transmitting a 1 or 0 on the bus. The sensor transmits a 1 by leaving the bus high and then transmits a 0 by pulling the bus low. All output data from the sensor is valid for 15 µs after the falling edge that initiated the read time slot. Therefore, PSoC must release the bus and then sample the bus state within 15 µs from the start of the slot.

Figure 32. Timing Diagram for Initialization (from the DS18S20 Datasheet)

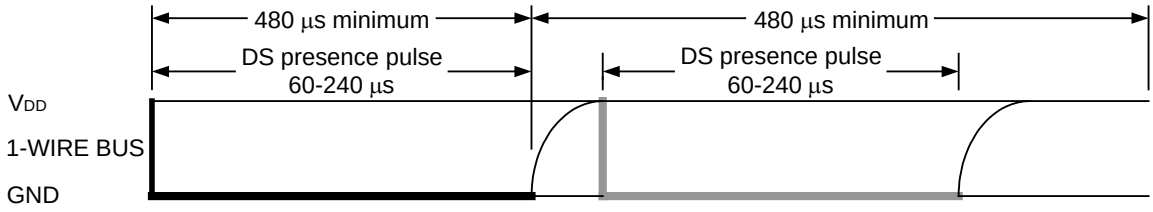
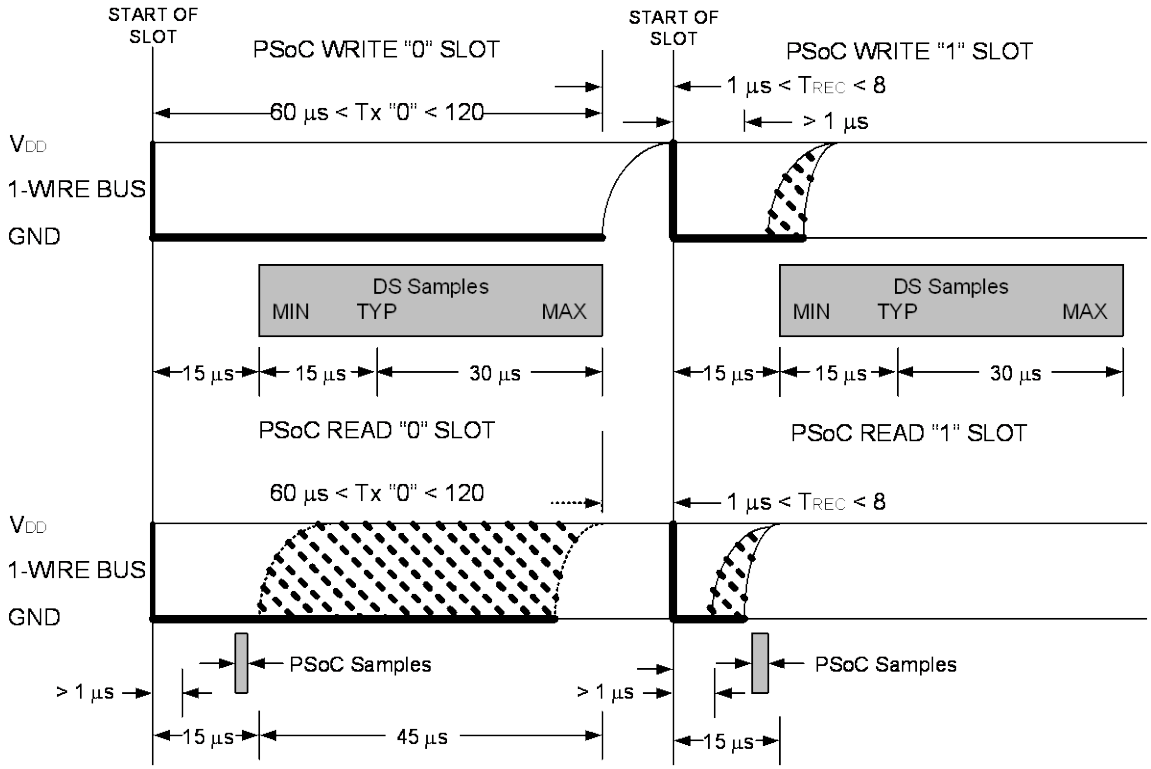


Figure 33. Timing Diagram for Read/Write Slots (from the DS18S20 Datasheet)



Document History

Document Title: AN2163 - Interfacing to 1-Wire/Two-Wire Digital Temperature Sensors using PSoC® 1

Document Number: 001-35340

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	1541809	SWU	10/04/07	New Application Note.
*A	3088605	OWEN	11/17/10	Software version updated to PSoC® Designer™ 5.1. Updated associated project.
*B	3172767	OWEN	02/14/11	Project source file updated and made minor edits in the document. No technical updates. Copyright year changed and document version revised.
*C	3306526	OWEN	07/08/11	Updated associated project.
*D	3432646	OWEN	11/09/2011	Obsolete spec.
*E	3666042	ARVI	07/04/2012	Rewrite of document to include Two-Wire sensor support, added new example project. Updated template. The document is activated again.
*F	3751286	ARVI	09/21/2012	Updated associated project files.

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

Automotive	cypress.com/go/automotive
Clocks & Buffers	cypress.com/go/clocks
Interface	cypress.com/go/interface
Lighting & Power Control	cypress.com/go/powerpsoc cypress.com/go/plc
Memory	cypress.com/go/memory
Optical Navigation Sensors	cypress.com/go/ons
PSoC	cypress.com/go/psoc
Touch Sensing	cypress.com/go/touch
USB Controllers	cypress.com/go/usb
Wireless/RF	cypress.com/go/wireless

PSoC® Solutions

psoc.cypress.com/solutions

PSoC 1 | PSoC 3 | PSoC 5

Cypress Developer Community

Community | Forums | Blogs | Video | Training

Technical Support

cypress.com/go/support

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

Phone : 408-943-2600
Fax : 408-943-4730
Website : www.cypress.com