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

Author: Arvind Krishnan

Associated Project: Yes
Associated Part Family: CY8C21x23/21x34/23x33/24x23A/24x94/27xxx/28xxx/29xxx, CY8CLED02/04/08/16, CY8CLED0xD, CY8CLED0xG, CY8CPLC20, CY8CLED16P01
Software Version: PSoC® Designer™ 5.4 SP1
Related Application Notes: For a complete list of the application notes, click here.

AN2163 demonstrates how a PSoC® 1 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 / DS18B20 (1-Wire digital temperature sensor) and a TMP75 (Two-Wire digital temperature sensor) along with the help of the attached example project.

Contents

1 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.

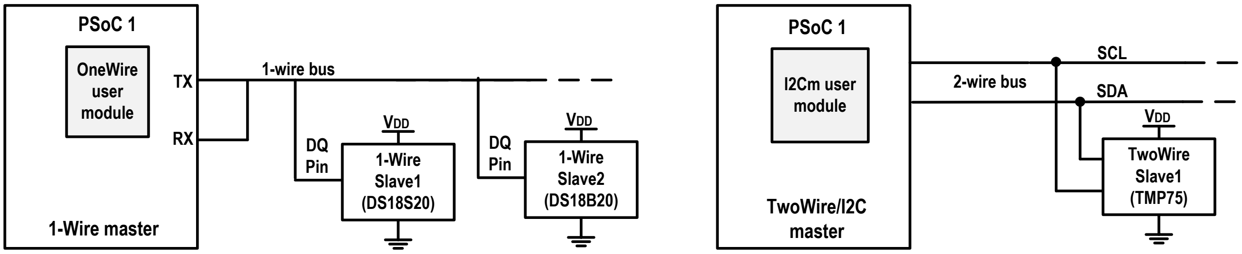
Digital temperature sensors are inexpensive, easily available, and widely used in microcontroller based temperature measurement systems. The use of digital temperature sensors frees the designer from worrying about digital noise coupling onto sensitive analog signals on the PCB layout because the digital 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 device can be interfaced to a 1-Wire based digital thermometer (DS18S20/DS18B20) from Maxim Integrated Products (earlier, Dallas Semiconductor) and Two-Wire or I^{2}C based digital temperature sensor (TMP75) from Texas Instruments. 1-Wire communication allows an interface that uses only one wire to retrieve data from devices on the bus. I^{2}C 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. The example projects attached to this application note will help you understand how to use the OneWire and I2Cm user modules to interface these sensors with a PSoC 1 device. Using the APIs provided in the example projects, you should be able to configure the digital interfaces with the sensor and read the temperature data (with a resolution of 0.5 °C with the DS18S20, 0.0625 °C with the DS18B20, and 0.0625 °C with the TMP75 digital sensor).

The following sections give a brief explanation of these sensors and the interfaces they follow. Note that all references to DS18X20 (where X = S or B) apply to both DS18S20 and DS18B20 unless specified otherwise.

Figure 1. Interfaces to 1-Wire and Two-Wire Slave Devices Using PSoC 1



2 PSoC Resources

Cypress provides a wealth of data at www.cypress.com to help you to select the right PSoC device for your design, and quickly and effectively integrate the device into your design. In this document, PSoC refers to the PSoC 1 family of devices. To learn more about PSoC 1, refer to the application note AN75320 - *Getting Started with PSoC 1*.

The following is an abbreviated list for PSoC 1:

- **Overview:** PSoC Portfolio, PSoC Roadmap
- **Product Selectors:** PSoC 1, PSoC 3, PSoC 4, or PSoC 5LP. In addition, PSoC Designer includes a device selection tool.
- **Datasheets:** Describe and provide electrical specifications for the PSoC 1 device family.
- **Application Notes and Code Examples:** Cover a broad range of topics, from basic to advanced level. Many of the application notes include code examples.
- **Technical Reference Manuals (TRM):** Provide detailed descriptions of the internal architecture of the PSoC 1 devices.
- **Development Kits:**
  - CY3215A-DK In-Circuit Emulation Lite Development Kit includes an in-circuit emulator (ICE). While the ICE-Cube is primarily used to debug PSoC 1 devices, it can also program PSoC 1 devices using ISSP.
  - CY3210-PSOCEVAL1 Kit enables you to evaluate and experiment Cypress's PSoC 1 programmable system-on-chip design methodology and architecture.
  - CY8CKIT-001 is a common development platform for all PSoC family devices.
- The MiniProg1 and MiniProg3 devices provide an interface for flash programming.

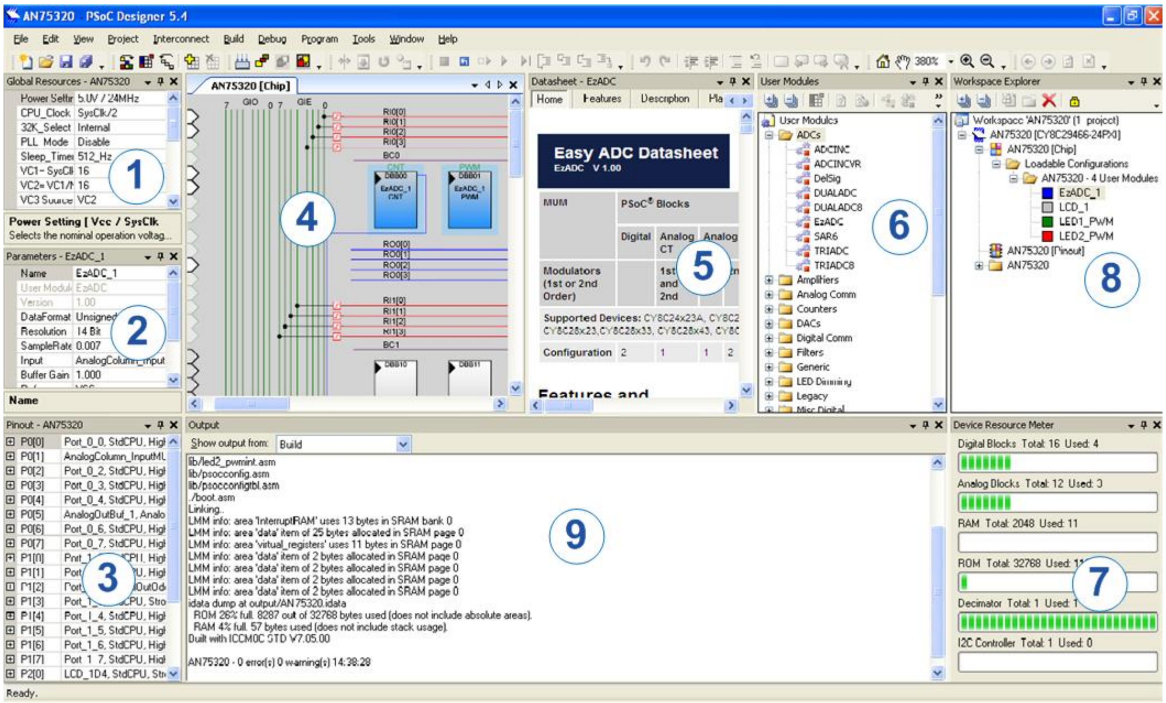
2.1 PSoC Designer

PSoC Designer is a free Windows-based Integrated Design Environment (IDE). Develop your applications using a library of pre-characterized analog and digital peripherals in a drag-and-drop design environment. Then, customize your design leveraging the dynamically generated API libraries of code. Figure 2 shows PSoC Designer windows. **Note:** This is not the default view.

1. **Global Resources –** all device hardware settings.
2. **Parameters –** the parameters of the currently selected User Modules.
3. **Pinout –** information related to device pins.
4. **Chip-Level Editor –** a diagram of the resources available on the selected chip.
5. **Datasheet –** the datasheet for the currently selected UM
6. **User Modules –** all available User Modules for the selected device.
7. **Device Resource Meter –** device resource usage for the current project configuration.
8. **Workspace –** a tree level diagram of files associated with the project.
9. **Output –** output from project build and debug operations.

Note: For detailed information on PSoC Designer, go to **PSoC® Designer** > **Help > Documentation** > **Designer Specific Documents** > **IDE User Guide**.

Figure 2. PSoC Designer Layout



Power Settir 5.0V / 24MHz
CPU_Clock SysClk/2
32K_Select Internal
PLL Mode Disable
Sleep_Timer 512_Hz
VC1= SysClk 16
VC2= VC1/N 16
VC3 Source VC2

Power Setting [Vcc / SysClk
Selects the nominal operation voltag...

Name EzADC_1
User Module EzADC
Version 1.00
DataFormat Unsigned
Resolution 14 Bit
SampleRate 0.007
Input AnalogColumn_Input
Buffer Gain 1.000

Show output from: Build

```
lib/led2_pwmint.asm
lib/psocconfig.asm
lib/psocconfigtbl.asm
./boot.asm
Linking..
LMM info: area 'InterruptRAM' uses 13 bytes in SRAM bank 0
LMM info: area 'data' item of 25 bytes allocated in SRAM page 0
LMM info: area 'virtual_registers' uses 11 bytes in SRAM page 0
LMM info: area 'data' item of 2 bytes allocated in SRAM page 0
LMM info: area 'data' item of 2 bytes allocated in SRAM page 0
LMM info: area 'data' item of 2 bytes allocated in SRAM page 0
LMM info: area 'data' item of 2 bytes allocated in SRAM page 0
idata dump at output/AN75320.idata
  ROM 26% full. 8287 out of 32768 bytes used (does not include absolute areas).
  RAM 4% full. 57 bytes used (does not include stack usage).
Built with ICCM8C STD V7.05.00

AN75320 - 0 error(s) 0 warning(s) 14:38:28
```

Digital Blocks Total: 16 Used: 4
Analog Blocks Total: 12 Used: 3
RAM Total: 2048 Used: 11
ROM Total: 32768 Used: 11
Decimator Total: 1 Used: 1
I2C Controller Total: 1 Used: 0

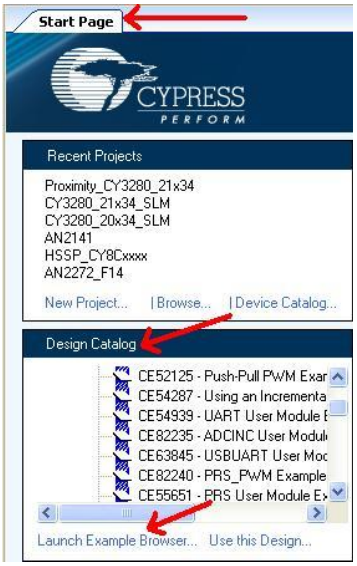
2.2 Code Examples

The following webpage lists the PSoC Designer based Code Examples. These Code Examples can speed up your design process by starting you off with a complete design, instead of a blank page and also show how PSoC Designer User modules can be used for various applications.

http://www.cypress.com/go/PSoC1Code Examples

To access the Code Examples integrated with PSoC Designer, follow the path **Start Page** > **Design Catalog** > **Launch Example Browser** as shown in Figure 3.

Figure 3. Code Examples in PSoC Designer

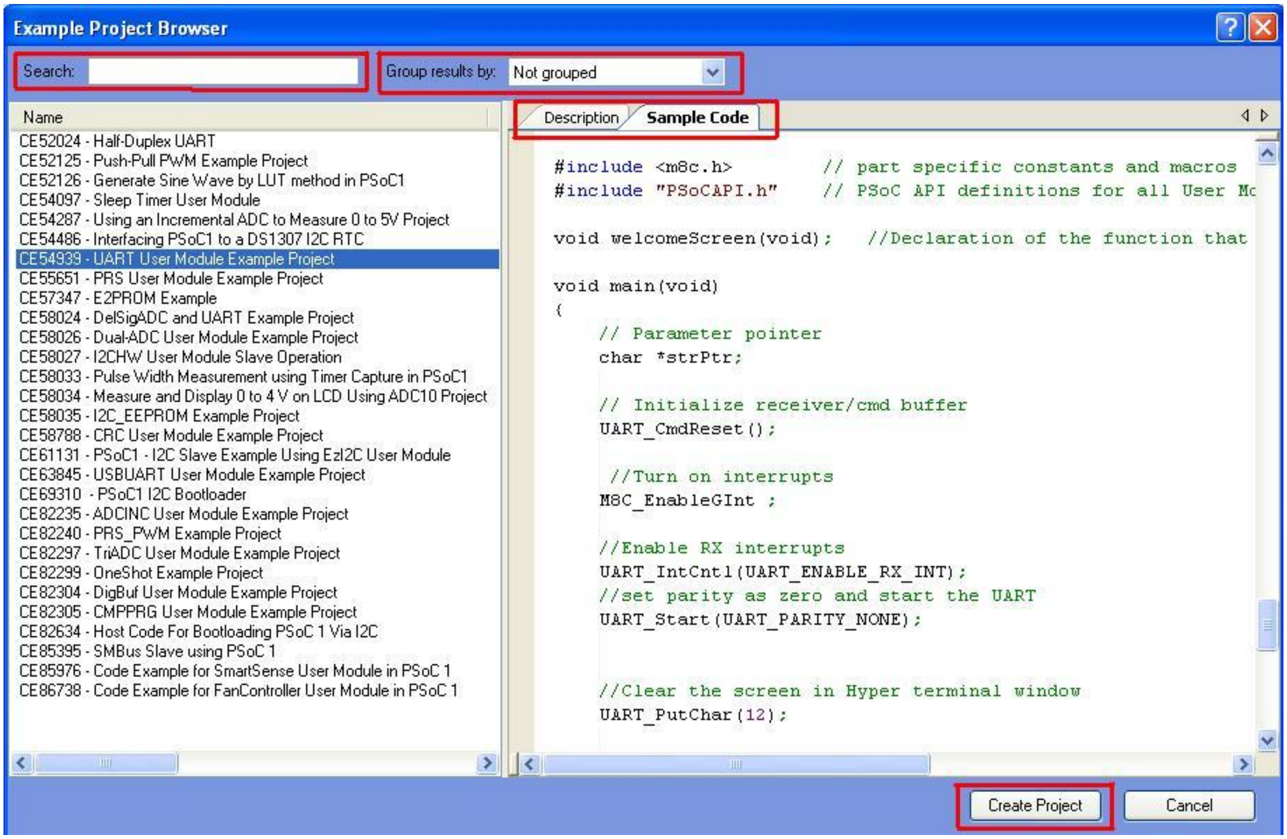


Start Page
CYPRESS
PERFORM
Recent Projects
Proximity_CY3280_21x34
CY3280_21x34_SLM
CY3280_20x34_SLM
AN2141
HSSP_CY8Cxxxx
AN2272_F14
New Project...
| Browse...
| Device Catalog...
Design Catalog
CE52125 - Push-Pull PWM Exar
CE54287 - Using an Incrementa
CE54939 - UART User Module E
CE82235 - ADCINC User Modul
CE63845 - USBUART User Moc
CE82240 - PRS_PWM Example
CE55651 - PRS User Module Ex
Launch Example Browser...
Use this Design...

In the Example Projects Browser shown in Figure 4, you have the following options.

- Keyword search to filter the projects.
- Listing the projects based on Category.
- Review the datasheet for the selection (on the Description tab).
- Review the code example for the selection. You can copy and paste code from this window to your project, which can help speed up code development, or
- Create a new project (and a new workspace if needed) based on the selection. This can speed up your design process by starting you off with a complete, basic design. You can then adapt that design to your application.

Figure 4. Code Example Projects, with Sample Codes



CE52024 - Half-Duplex UART
CE52125 - Push-Pull PWM Example Project
CE52126 - Generate Sine Wave by LUT method in PSoC1
CE54097 - Sleep Timer User Module
CE54287 - Using an Incremental ADC to Measure 0 to 5V Project
CE54486 - Interfacing PSoC1 to a DS1307 I2C RTC
CE54939 - UART User Module Example Project
CE55651 - PRS User Module Example Project
CE57347 - E2PROM Example
CE58024 - DelSigADC and UART Example Project
CE58026 - Dual-ADC User Module Example Project
CE58027 - I2CHW User Module Slave Operation
CE58033 - Pulse Width Measurement using Timer Capture in PSoC1
CE58034 - Measure and Display 0 to 4 V on LCD Using ADC10 Project
CE58035 - I2C_EEPROM Example Project
CE58788 - CRC User Module Example Project
CE61131 - PSoC1 - I2C Slave Example Using EzI2C User Module
CE63845 - USBUART User Module Example Project
CE69310 - PSoC1 I2C Bootloader
CE82235 - ADCINC User Module Example Project
CE82240 - PRS_PWM Example Project
CE82297 - TriADC User Module Example Project
CE82299 - OneShot Example Project
CE82304 - DigBuf User Module Example Project
CE82305 - CMPPRG User Module Example Project
CE82634 - Host Code For Bootloading PSoC 1 Via I2C
CE85395 - SMBus Slave using PSoC 1
CE85976 - Code Example for SmartSense User Module in PSoC 1
CE86738 - Code Example for FanController User Module in PSoC 1

```
#include <m8c.h>        // part specific constants and macros
#include "PSoCAPI.h"    // PSoC API definitions for all User Mc

void welcomeScreen(void);   //Declaration of the function that

void main(void)
{
    // Parameter pointer
    char *strPtr;

    // Initialize receiver/cmd buffer
    UART_CmdReset();

     //Turn on interrupts
    M8C_EnableGInt ;

    //Enable RX interrupts
    UART_IntCntl(UART_ENABLE_RX_INT);
    //set parity as zero and start the UART
    UART_Start(UART_PARITY_NONE);


    //Clear the screen in Hyper terminal window
    UART_PutChar(12);
```

2.3 Technical Support

If you have any questions, our technical support team is happy to assist you. You can create a support request on the Cypress Technical Support page.

You can also use the following support resources if you need quick assistance.

- Self-help
- Local Sales Office Locations

3 Example Projects

This application note includes example projects that can be readily evaluated on Cypress hardware and demonstrate temperature measurement using PSoC 1. The following example projects are available in the attachment from the application note's webpage-

Table 1. Example projects

Relative Project path	Description
\AN2163 projects\DS18X20 example	Demonstrates a 1-Wire interface using DS18S20/DS18B20 sensors
\AN2163 projects\TMP75 example	Demonstrates a Two-wire interface using TMP75 sensor
\AN2163 projects\TempSense_1Wire_2Wire example	Demonstrates both 1-Wire and two-wire interfaces in a single project using DS18S20/DS18B20 and TMP75 sensors

The DS18X20 Example Code and TMP75 Example Code sections give a step-by-step process to create a PSoC Designer project from scratch to measure the temperature using sensors.

3.1 Hardware Setup

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

- CY8CKIT-001 Development Board
- DS18B20 1-Wire temperature sensor (on CY8CKIT-036 Thermal Management Expansion board kit (TM EBK)
- TMP75 two-wire temperature sensor (on CY8CKIT-036 Thermal Management Expansion board kit (TM EBK)
- DS18B20 Series temperature sensor (module)
- CY8C28000-24PVXI on a CY8C28 family processor module.

Note 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.

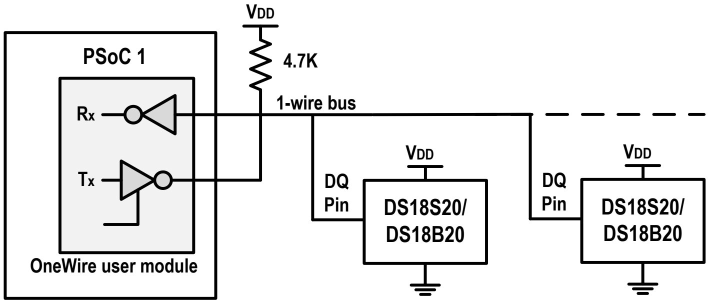
4 1-Wire® Interface

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 DS18X20 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 5.

Figure 5. Hardware Configuration



A brief on the 1-Wire protocol can be found in Appendix: 1-Wire Protocol - Read/Write Time Slots. 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.

5 DS18S20/DS18B20 Sensor

DS18S20 is a 1-Wire interfaced temperature sensor capable of 9-bit temperature measurements (±0.5 °C accuracy in normal mode and ±0.25 °C in extended resolution mode). The DS18B20 is capable of variable resolution (9-bit to 12-bit) temperature measurements (up to ±0.0625 °C 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 2 and Table 3. Later sections explain sensor functions such as user configurable alarm settings, slave addressing, and how PSoC 1 can be used to implement these.

Table 2. Temperature to Data Relationships – DS18S20

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

Table 3. Temperature to Data Relationships – DS18B20

Temperature	Digital Output (Binary)	Digital Output (HEX)
+125.0 °C	0000 0111 1101 0000	07D0h
+85.0 °C	0000 0101 0101 0000	0550h
+25.0625 °C	0000 0001 1001 0001	0191h
+10.125 °C	0000 0000 1010 0010	00A2h
+0.5 °C	0000 0000 0000 1000	0008h
0.0 °C	0000 0000 0000 0000	0000h
-0.5 °C	1111 1111 1111 1000	FFF8h
–10.125 °C	1111 1111 0101 1110	FF5Eh

Temperature	Digital Output (Binary)	Digital Output (HEX)
–25.0625 °C	1111 1111 0101 1110	FE6Fh
–55.0 °C	1111 1100 1001 0000	FC90h

5.1 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 1-Wire 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 5 shows the hardware configuration used for interfacing the DS18X20 in non-parasitic mode.

6 Communication Overview

PSoC 1 acts as the bus master for 1-Wire communication and follows the 1-Wire protocol to communicate with DS18X20. The protocol is used to write and read data from the sensor’s internal memory structure. Please refer to Appendix: DS18X20 1-Wire Sensor for more information on the memory structure. The 1-Wire protocol specifies the following transaction sequence for accessing devices:

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

6.1 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 35 in Appendix: 1-Wire Protocol - Read/Write Time Slots.

6.2 ROM Commands

The DS18X20 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 DS18X20 is given below; their functions are explained in the ROM section. Information on the ROM codes can be found in greater detail in the DS18S20 datasheet / DS18B20 datasheet.

Table 4. ROM Commands

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

6.3 DS18X20 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 5.

Table 5. DS18X20 Function Command Set (from DS18X20 Datasheet)

Command	Description	Protocol	1-Wire Bus Activity After Command Is Issued
Temperature conversion commands			

Command	Description	Protocol	1-Wire Bus Activity After Command Is Issued
Convert T	Initiates temperature conversion	44h	DS18X20 transmits conversion status to master (not applicable for parasite-powered DS18X20s).
Memory Commands			
Read Scratchpad	Reads the entire scratchpad including the CRC byte	BEh	DS18X20 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 DS18X20.
Copy Scratchpad	Copies TH and TL data from the scratchpad to EEPROM	48h	None
Recall E^2	Recalls TH and TL data from EEPROM to the scratchpad	B8h	DS18X20 transmits recall status to master.
Read Power Supply	Signals DS18X20 power supply mode to the master	B4h	DS18X20 transmits supply status to master.

7 DS18X20 Example Code

The following section walks-through an example implementation of the interface with a DS18X20 sensor. For a ready-made example project to interface to both the DS18S20 and the DS18B20, refer to the "*TempSense_1wire_2wire*" example project provided with this application note. To port the example project attached with this application note to a PSoC1 device other than CY8C28xxx, refer to "Cloning a Project" section in PSoC Designer Help Topics available from the Designer IDE (Help menu > Help Topics > Project Manager > Cloning a Project).

After following the steps given in this section, you should be able to interface a CY8C28xxx microcontroller to the DS18S20 / DS18B20 sensors connected either separately or together and display the measured temperatures on the LCD. The DS18X20 library files used in this example can be found as an attachment to the application note (see *DS18X20 library* folder).

The hardware setup to evaluate this example is shown in Figure 6 and Figure 7. Figure 8 shows the block diagram of the hardware setup used for this example project.

Figure 6. 1-Wire Temperature Measurement - Hardware Setup

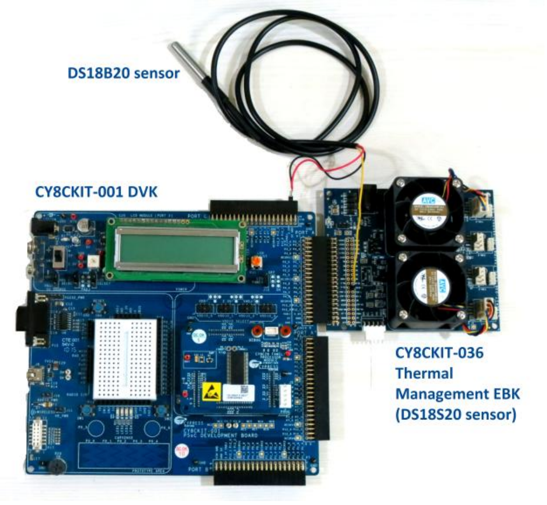


Figure 7. Location of DS18S20 on CY8CKIT-036

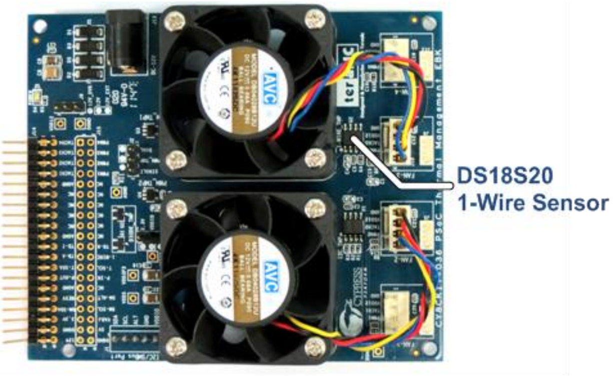
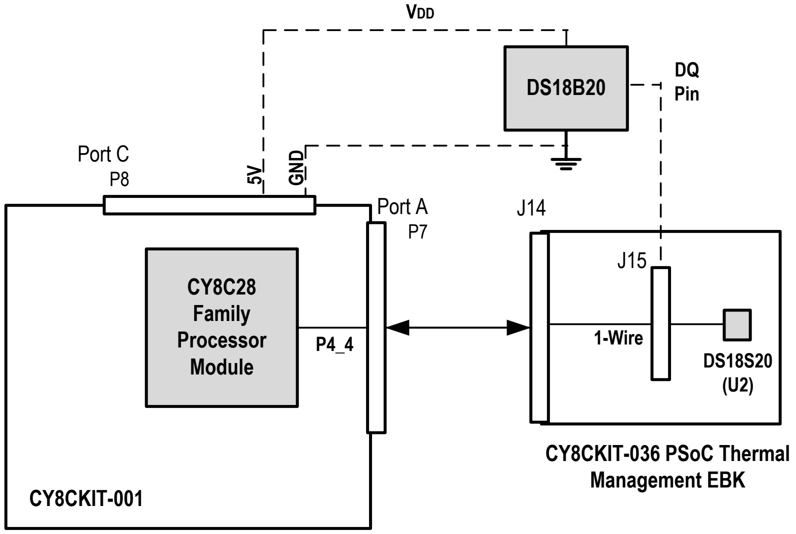


Figure 8. Block Diagram Showing Hardware Connections



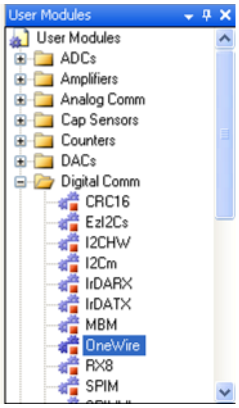
8 Creating a 1-Wire Project from Scratch

The following sections steps through the procedure to create a project to interface PSoC 1 to a DS18X20 sensor using the DS18X20 library. This library contains API functions that are common to both DS18S20 and DS18B20 sensors. To create a PSoC Designer project that configures a PSoC to work as a 1-Wire master to read temperature and display temperature from DS18S20 sensor, follow the steps given below:

1. Make the hardware connections:
   a. Connect the CY8CKIT-036 EBK to Port A of the CY8CKIT-001 development kit. This connects the DS18S20 sensor on the EBK to PSoC device.
   b. Connect a DS18B20 sensor to J15 on the EBK. Connect the sensor's data pin (DQ) to **1-Wire** pin on J15. Connect the VDD and GND pins of the sensor to any 5V/GND pins on either the EBK or the DVK.

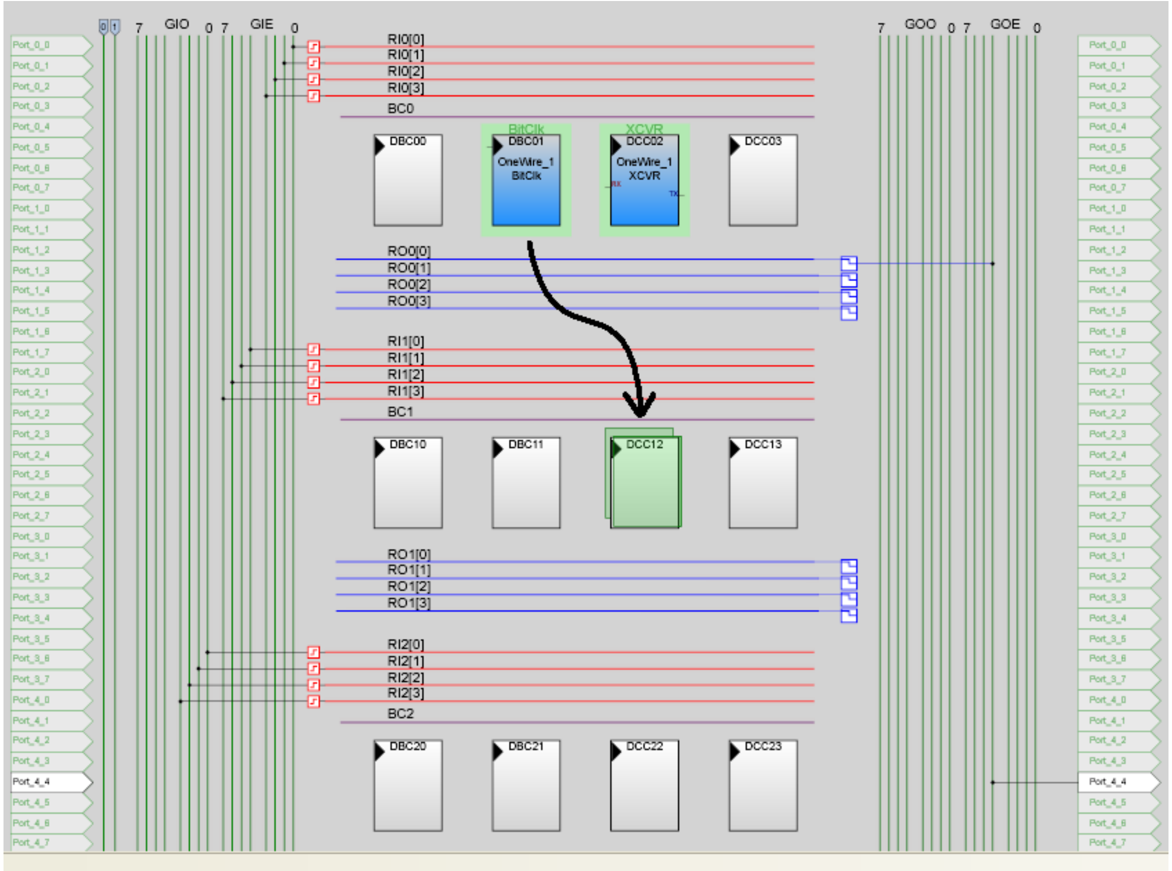
      If the CY8CKIT-036 EBK / DS18S20 is not required, the DS18B20 sensor (DQ) can directly be connected to **P4_4** pin on Port A of the DVK.
2. Open PSoC Designer software
3. Click the **File** menu and select **New Project**.
4. Choose any desired name for the project, for e.g., "OneWire_DS18X20" and click **OK**.
5. Select **View Catalog** to select the target device as **CY8C28645-24LTXI**. Click **Select** when the device has been selected.
6. 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.
7. Locate the "OneWire" user module in the user module catalog under "Digital Comm" sub-folder.

Figure 9. OneWire User Module Location



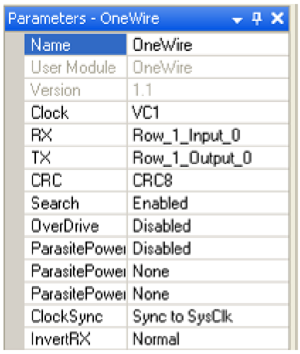
8. Right-click on the user module and select **Place**. This operation will place the OneWire BitCLK and XCVR modules in two digital blocks. To ease internal routing, re-place them by dragging and dropping the modules into the new slot at DCC12 as shown in the following figure.

Figure 10. Drag and Drop Modules into New Slot



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

Figure 11. OneWire User Module Parameters



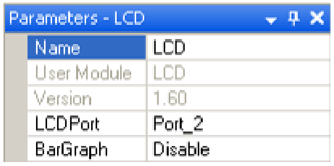
Parameters - OneWire

Name	OneWire
User Module	OneWire
Version	1.1
Clock	VC1
RX	Row_1_Input_0
TX	Row_1_Output_0
CRC	CRC8
Search	Enabled
OverDrive	Disabled
ParasitePower	Disabled
ParasitePower	None
ParasitePower	None
ClockSync	Sync to SysClk
InvertRX	Normal

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.

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

Figure 12. LCD User Module Parameters



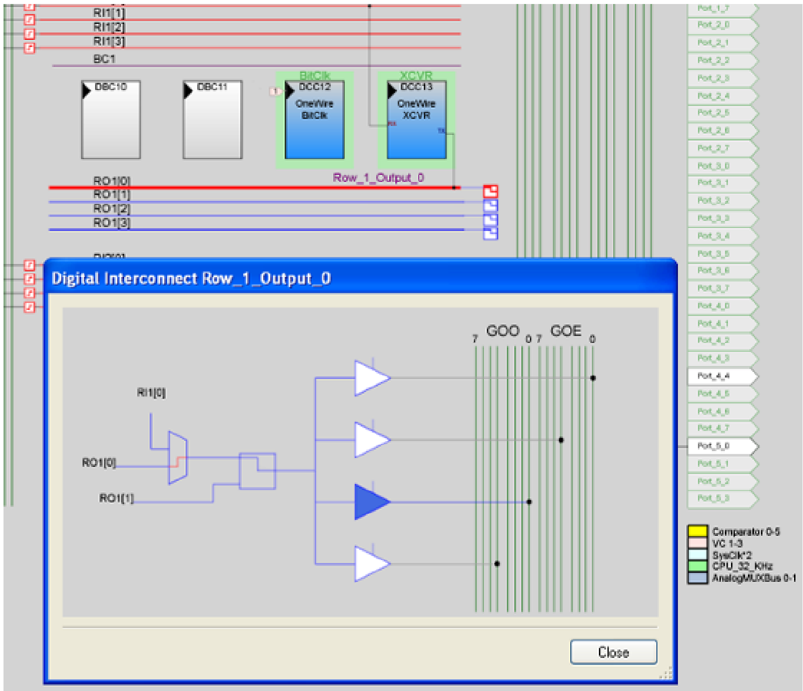
Parameters - LCD

Name	LCD
User Module	LCD
Version	1.60
LCDPort	Port_2
BarGraph	Disable

11. 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:

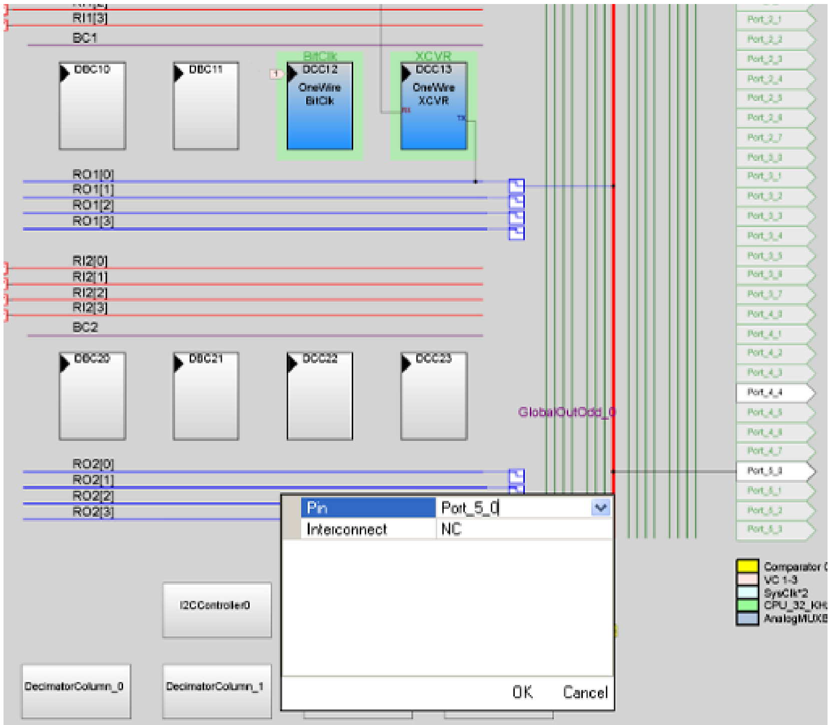
    a. Click on **Row_1_Output_0** line and select the connection to **GlobalOutOdd_0** as shown in the following figure.

Figure 13. 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 14. Configuring GlobalOutOdd_0 Connection

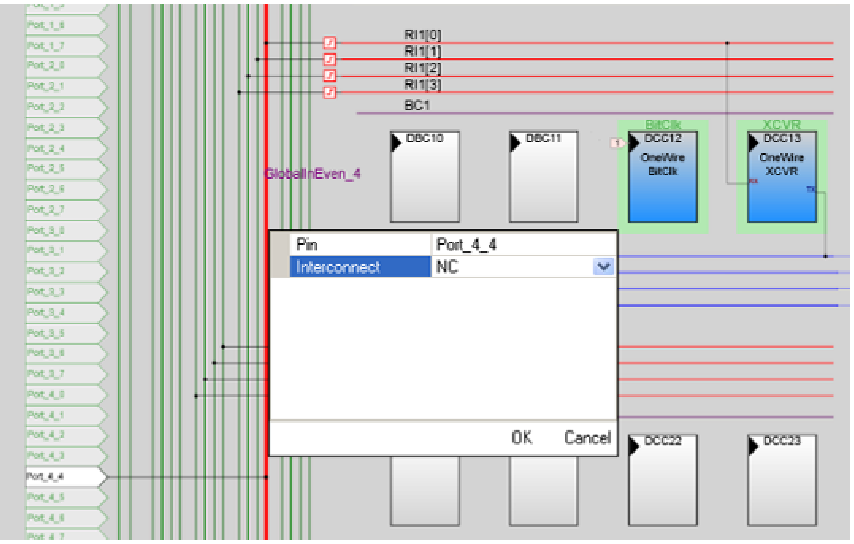


Pin: Port_5_0

Interconnect: NC

12. 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 15. Routing Input to User Module



Pin: Port_4_4

Interconnect: NC

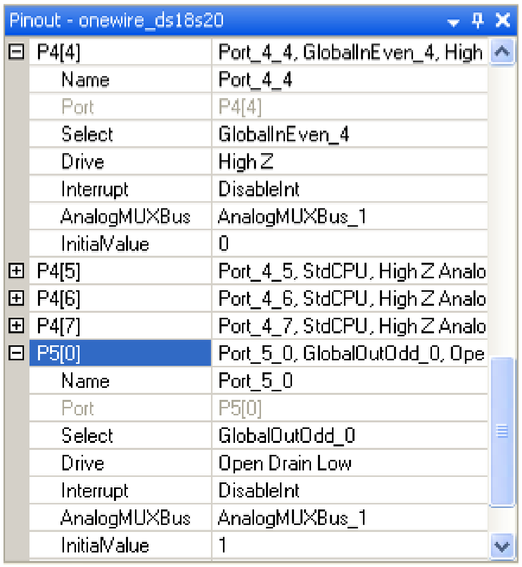
13. Even though the OneWire user module has two terminals (RX and TX), this is presented externally as a single port that connects to the 1-Wire bus. Thus, the RX and TX pins (Port_4_4 and Port_5_0) have to be shorted. This can be done by externally shorting Port_4_4 and Port_5_0 pins. However, for devices which have an AnalogMUXBus, such as CY8C20xx6, 21x34, 21x45, 24x94, 24x93 and 28xxx, the 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].

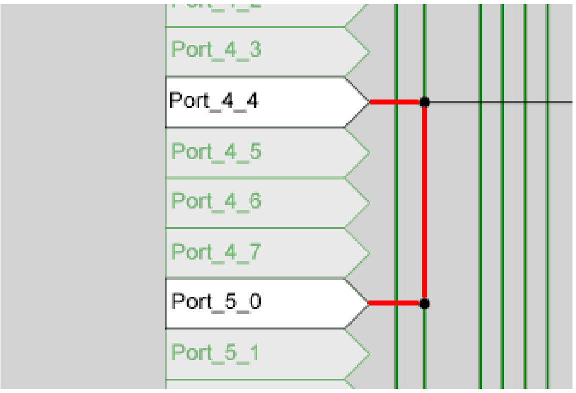
Figure 16. Connecting Pins to the AnalogMUXBus



Pinout - onewire_ds18s20

⊟ P4[4]	Port_4_4, GlobalInEven_4, High
Name	Port_4_4
Port	P4[4]
Select	GlobalInEven_4
Drive	High Z
Interrupt	DisableInt
AnalogMUXBus	AnalogMUXBus_1
InitialValue	0
⊞ P4[5]	Port_4_5, StdCPU, High Z Analo
⊞ P4[6]	Port_4_6, StdCPU, High Z Analo
⊞ P4[7]	Port_4_7, StdCPU, High Z Analo
⊟ P5[0]	Port_5_0, GlobalOutOdd_0, Ope
Name	Port_5_0
Port	P5[0]
Select	GlobalOutOdd_0
Drive	Open Drain Low
Interrupt	DisableInt
AnalogMUXBus	AnalogMUXBus_1
InitialValue	1

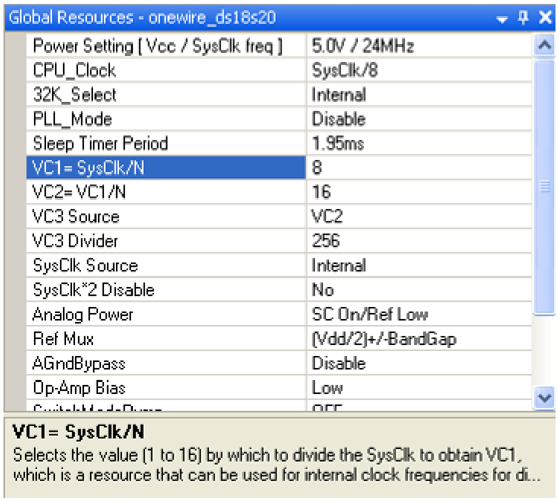
Figure 17. Internal Short Highlighted



Note that the drive mode of Port_4_4 is fixed to high impedance (HI-Z) as it is an input pin. The drive mode 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.

14. 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 18. Setting OneWire Timing Using Clock Dividers



Global Resources - onewire_ds18s20

Power Setting [Vcc / SysClk freq]	5.0V / 24MHz
CPU_Clock	SysClk/8
32K_Select	Internal
PLL_Mode	Disable
Sleep Timer Period	1.95ms
VC1= SysClk/N	8
VC2= VC1/N	16
VC3 Source	VC2
VC3 Divider	256
SysClk Source	Internal
SysClk*2 Disable	No
Analog Power	SC On/Ref Low
Ref Mux	(Vdd/2)+/-BandGap
AGndBypass	Disable
Op-Amp Bias	Low

VC1= SysClk/N
Selects the value (1 to 16) by which to divide the SysClk to obtain VC1, which is a resource that can be used for internal clock frequencies for di...

15. Click on **Build -> Generate Configuration** files or Ctrl+F6 to populate the project with necessary APIs/libraries.

16. Right-click “OneWire_DS18X20” (name of the project) folder and select “Add File”. Navigate to the folder with the DS18S20 libraries (under .../DS18X20 library/) to import the *DS18X20.c* and *DS18X20.h* files into the project.

17. Open *main.c* from the Workplace Explorer window and replace it with the contents of file “*DS18X20_main.c*”

18. Click **Build project** (F7) from **Build** menu. Project should compile without any errors.

19. 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 19. Connect a MiniProg1/3 programmer to the “PROG” header of the processor module.

Figure 19. Jumper Settings

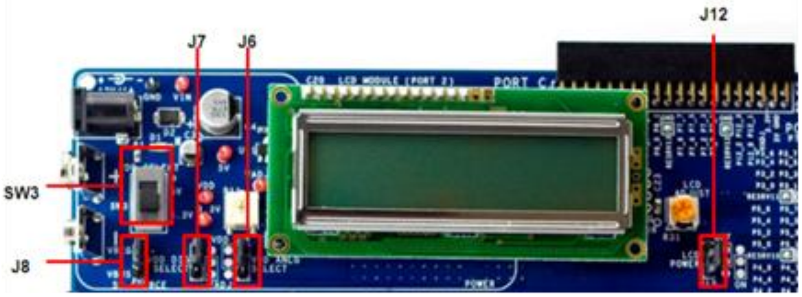


Table 6. Jumper Settings for 001 Kit

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

20. 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.

21. The LCD should display the ambient temperature as measured by DS18X20 in degree Celsius.

9 DS18X20 Advanced Functions

9.1 Overview of the 1-Wire Example Project

The previous section demonstrated how a PSoC Designer project can be setup to perform a simple temperature read operation from DS18X20. The DS18X20 provides certain advanced functions, such as CRC for ROM and Scratchpad, user configurable ALARM thresholds, and so on. The DS18X20 library provided with this application note provides API functions to simplify use of these advanced functions.

The included example project “TempSense_1wire_2wire” integrates the DS18X20 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 20. main() Routine Flow Chart

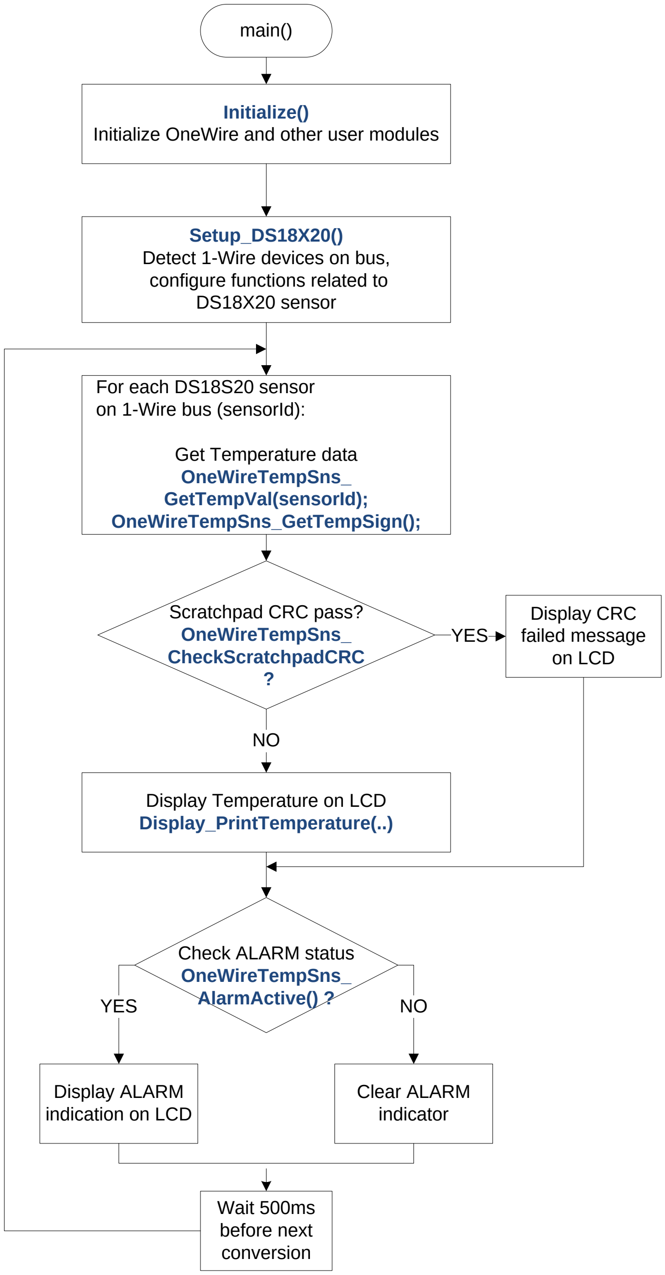

main()
Initialize()
Initialize OneWire and other user modules
Setup_DS18X20()
Detect 1-Wire devices on bus, configure functions related to DS18X20 sensor
For each DS18S20 sensor on 1-Wire bus (sensorId):
Get Temperature data
OneWireTempSns_ GetTempVal(sensorId);
OneWireTempSns_GetTempSign();
Scratchpad CRC pass? OneWireTempSns_ CheckScratchpadCRC ?
YES
Display CRC failed message on LCD
NO
Display Temperature on LCD
Display_PrintTemperature(..)
Check ALARM status OneWireTempSns_ AlarmActive() ?
YES
NO
Display ALARM indication on LCD
Clear ALARM indicator
Wait 500ms before next conversion

Figure 21. Setup_DS18X20() Routine Flow Chart

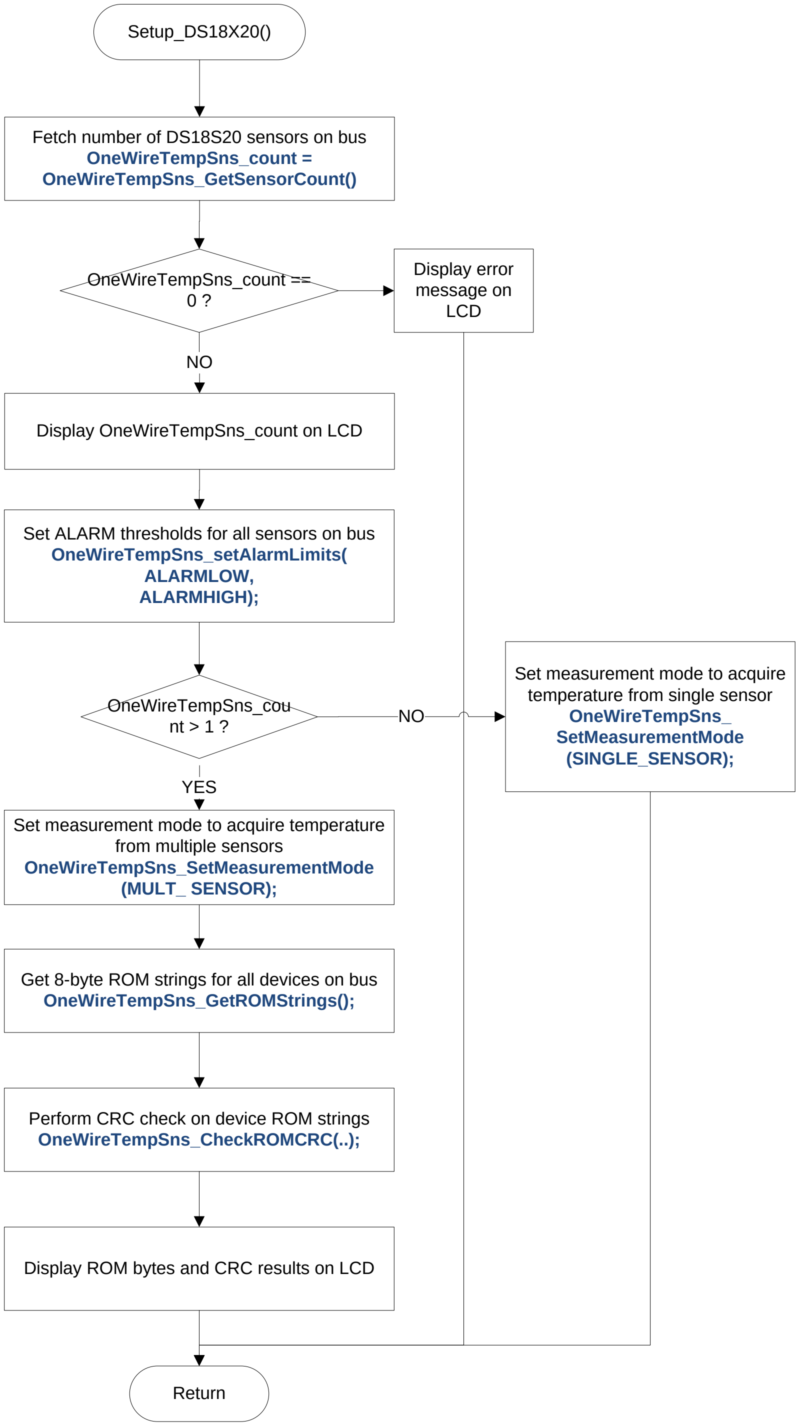
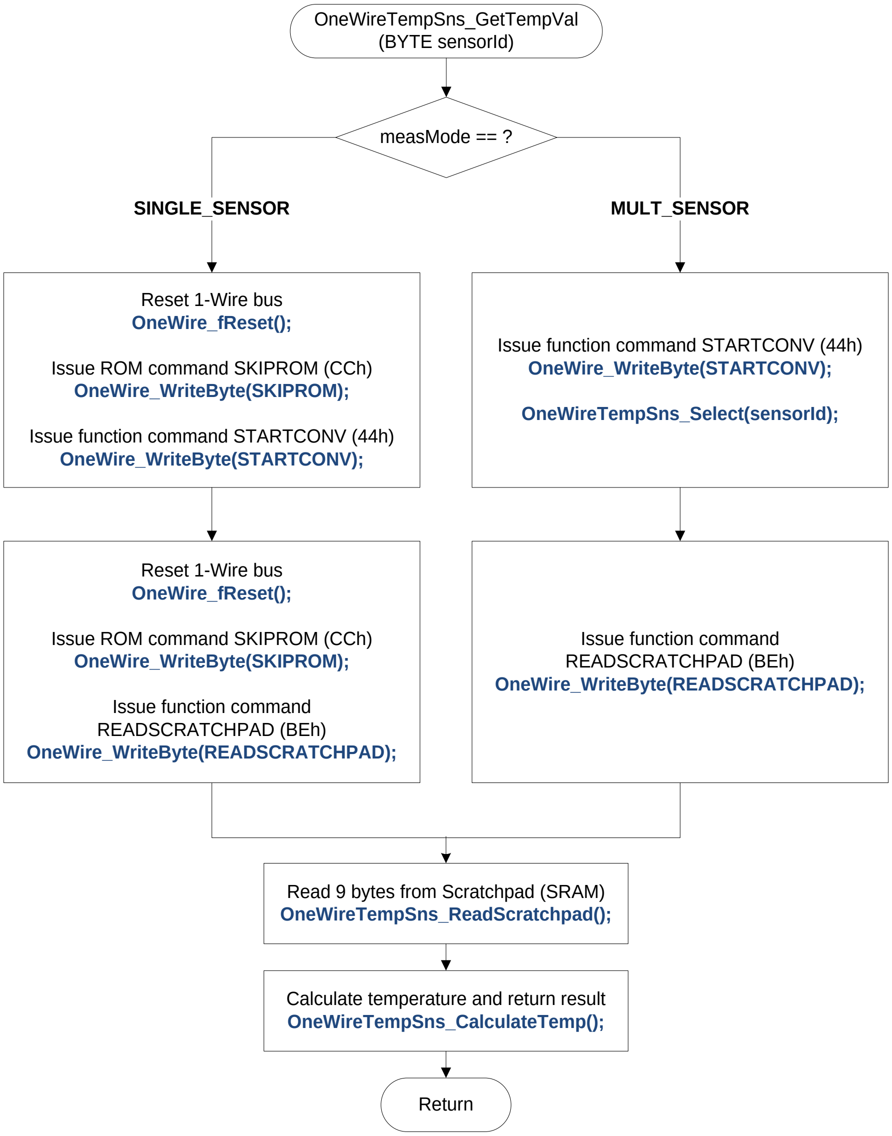

Setup_DS18X20()
Fetch number of DS18S20 sensors on bus
OneWireTempSns_count =
OneWireTempSns_GetSensorCount()
OneWireTempSns_count == 0 ?
Display error message on LCD
NO
Display OneWireTempSns_count on LCD
Set ALARM thresholds for all sensors on bus
OneWireTempSns_setAlarmLimits(
ALARMLOW,
ALARMHIGH);
OneWireTempSns_count > 1 ?
NO
Set measurement mode to acquire temperature from single sensor
OneWireTempSns_
SetMeasurementMode
(SINGLE_SENSOR);
YES
Set measurement mode to acquire temperature from multiple sensors
OneWireTempSns_SetMeasurementMode
(MULT_ SENSOR);
Get 8-byte ROM strings for all devices on bus
OneWireTempSns_GetROMStrings();
Perform CRC check on device ROM strings
OneWireTempSns_CheckROMCRC(..);
Display ROM bytes and CRC results on LCD
Return

Figure 22. OneWireTempSns_GetTempVal() Routine Flow Chart


OneWireTempSns_GetTempVal
(BYTE sensorId)
measMode == ?
SINGLE_SENSOR
MULT_SENSOR
Reset 1-Wire bus
OneWire_fReset();
Issue ROM command SKIPROM (CCh)
OneWire_WriteByte(SKIPROM);
Issue function command STARTCONV (44h)
OneWire_WriteByte(STARTCONV);
Issue function command STARTCONV (44h)
OneWire_WriteByte(STARTCONV);
OneWireTempSns_Select(sensorId);
Reset 1-Wire bus
OneWire_fReset();
Issue ROM command SKIPROM (CCh)
OneWire_WriteByte(SKIPROM);
Issue function command
READSCRATCHPAD (BEh)
OneWire_WriteByte(READSCRATCHPAD);
Issue function command
READSCRATCHPAD (BEh)
OneWire_WriteByte(READSCRATCHPAD);
Read 9 bytes from Scratchpad (SRAM)
OneWireTempSns_ReadScratchpad();
Calculate temperature and return result
OneWireTempSns_CalculateTemp();
Return

9.2 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 OneWireTempSns_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;
   }
      }
```

9.3 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 defining/uncommenting the macro EXTENDED_RESMODE in the header file "*DS18X20.h*".

```
#define EXTENDED_RESMODE
```

9.4 ROM Functions

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 *OneWireTempSns_GetSensorCount* function embeds the OneWire_GetROM function, using which the ROM bytes from a sensor can be acquired:

```
BYTE OneWireTempSns_GetSensorCount(void)
{
        sensorCount = 0;

        /* Uncomment the code below to restrict the search to specific sensor family */
//      BYTE targetFamilyCode;
//              targetFamilyCode = (DS18S20_FAMILYCODE);
//              targetFamilyCode = (DS18B20_FAMILYCODE);
//      OneWire_FamilyTargetSetup(targetFamilyCode);

        /* Poll the bus for devices */
        if (OneWire_fFindFirst())
        {
                OneWire_GetROM(romPointer[sensorCount]);

                if (OneWireTempSns_FamilyCheck(romPointer[sensorCount]))
                {
                        /* Atleast one sensor has been found */
                        sensorCount = 1;

                        /* Search for other 1-wire devices on bus */
                        while (OneWire_fFindNext())
                        {
                                OneWire_GetROM(romPointer[sensorCount]);
                                if (OneWireTempSns_FamilyCheck(romPointer[sensorCount]))
                                {
                                        sensorCount++;
                                }
                        }
                }
        }
        return sensorCount;
}
```

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. *OneWireTempSns_Select* implements this protocol:

```
void OneWireTempSns_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(romPointer[sensorId][loopVar]);
   }
}

        <followed by commands to read Scratchpad>
```

A list of all the functions in DS18X20 library is given in the following table.

Table 7. DS18S20 Library Functions

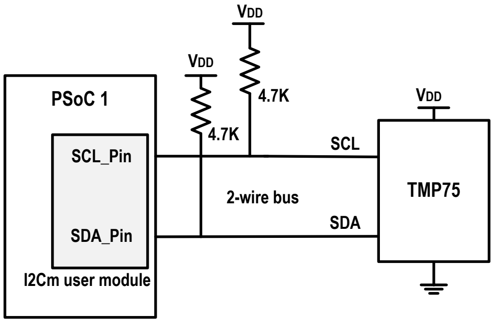
Function Prototype	Description
`BYTE OneWireTempSns_GetSensorCount(void);`	Returns the number of DS18S20 devices on 1-Wire bus. Scans bus for devices and fetches ROM strings from each device sequentially.
`WORD OneWireTempSns_GetTempVal(BYTE sensorId);`	Get value of temperature from sensor by reading Scratchpad of specific sensor
`BOOL OneWireTempSns_GetTempSign(void);`	Get sign of temperature from the Scratchpad data
`void OneWireTempSns_ReadScratchpad(void);`	Read 9-bytes from device Scratchpad into internal memory
`WORD OneWireTempSns_CalculateTemp(BYTE sensorId);`	Calculate temperature value from the read Scratchpad data
`BYTE OneWireTempSns_CheckScratchpadCRC(void);`	Validate CRC for Scratchpad data
`void OneWireTempSns_Configure(CHAR alarmLowVal, CHAR alarmHighVal, BYTE tmpResolution);`	Sets the low and high temperature limit to define an ALARM condition Sets the resolution for DS18B20 sensor
`BOOL OneWireTempSns_AlarmActive(void);`	Query devices on the bus for an ALARM signal

Function Prototype	Description
`BYTE * OneWireTempSns_GetLCDString(DWORD sensorVal, BOOL sign);`	Convert temperature into an ASCII string which can be displayed on the LCD
`BOOL OneWireTempSns_FamilyCheck(BYTE * deviceROM);`	Indicates whether specific 1-Wire device family is present on bus
`BOOL OneWireTempSns_CheckROMCRC(BYTE * deviceROM);`	Validate CRC for ROM data
`void OneWireTempSns_SetMeasurementMode(BYTE mode);`	Update local variable used to indicate measurement mode
`BYTE OneWireTempSns_TwosComplement(BYTE data);`	Perform and return Two's complement of an 8-bit number
`void OneWireTempSns_Select(BYTE sensorId);`	Address a specific sensor on the bus using its ROM string
`WORD DS18S20_GetScaleFactor(BYTE tmpResolution1);`	Get value of scalar used to translate temperature value to string
`WORD DS18B20_GetScaleFactor(BYTE tmpResolution1);`	

10 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 23. Hardware Connections



11 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 °C to +125 °C.

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. More information on the internal memory structure of the sensor is provided in TMP75 Memory Structure.

Table 8. Temperature/Data Relationships

Temperature	Digital Output (Binary)	Digital Output (HEX)
+128 °C	0111 1111 1111	7FFh
+127.9375 °C	0111 1111 1111	7FFh
+100 °C	0110 0100 0000	640h
+80 °C	0101 0000 0000	500h

Temperature	Digital Output (Binary)	Digital Output (HEX)
+75 °C	0100 1011 0000	4B0h
+50 °C	0011 0010 0000	320h
+25 °C	0001 1001 0000	190h
+0.25 °C	0000 0000 0100	004h
0 °C	0000 0000 0000	000h
-0.25 °C	1111 1111 1100	FFCh
-25 °C	1110 0111 0000	E70h
-55 °C	1100 1001 0000	C90h

12 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. A ready-made example to interface a TMP75 sensor with CY8C28xxx device is available in the “TMP75 example” and “TempSense_1Wire_2Wire example” projects attached with this application note. To port the example code attached with this application note on a PSoC1 device other than CY8C28xxx, please refer to “Cloning a Project” section in PSoC Designer Help Topics available from the Designer IDE (Help menu -> Help Topics -> Project Manager -> Cloning a Project).

12.1 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. Note that any PSoC 1 device that supports the I2Cm user module can be used for this application. A list of such devices can be found in the I2Cm user module.

Figure 24. TMP75 - Hardware Setup

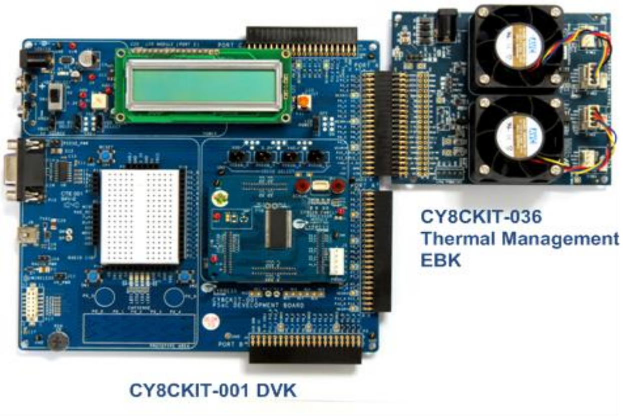
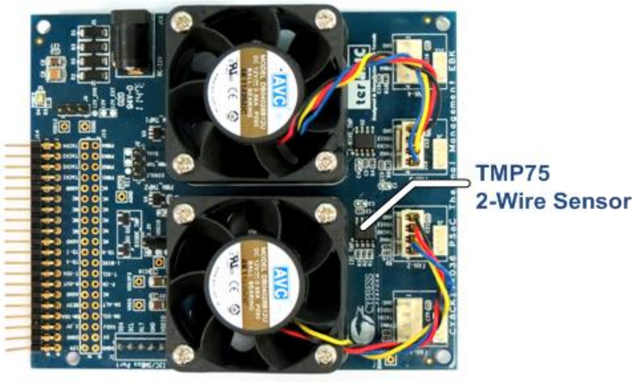


Figure 25. TMP75 Sensor Location

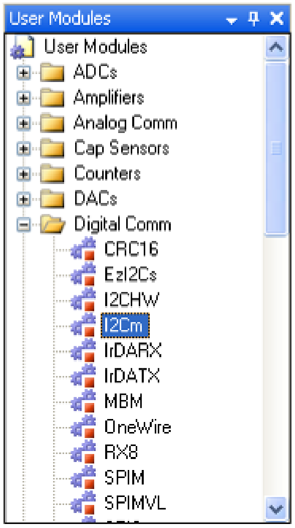


13 Creating a Two-Wire Project from Scratch

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 26. Select the I2Cm User Module

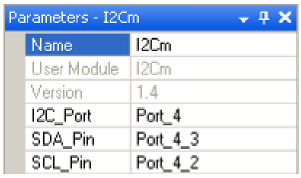


Note This project uses an I2Cm (I^{2}C Master User module) for I^{2}C communication. The I2Cm is implemented in firmware (consumes more CPU processing), but provides more flexibility in terms of choice of SDA, SCL pins and does not consume any digital blocks.

The I2CHW user module can also be used for this purpose. This provides hardware buffers for I^{2}C data, but places a restriction on the PSoC pins, which can be used for I^{2}C (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 27. I2Cm Parameters

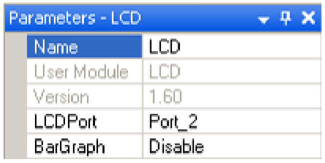


Parameters - I2Cm

Name	I2Cm
User Module	I2Cm
Version	1.4
I2C_Port	Port_4
SDA_Pin	Port_4_3
SCL_Pin	Port_4_2

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 28. LCD Parameters



Parameters - LCD

Name	LCD
User Module	LCD
Version	1.60
LCDPort	Port_2
BarGraph	Disable

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 29. Connect a MiniProg1/3 programmer to the "PROG" header of the processor module.

Figure 29. Jumper Settings

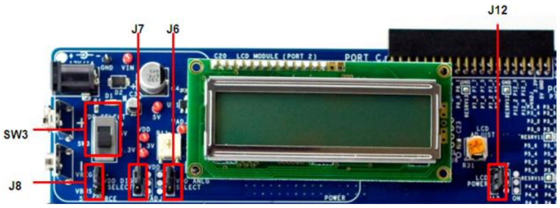


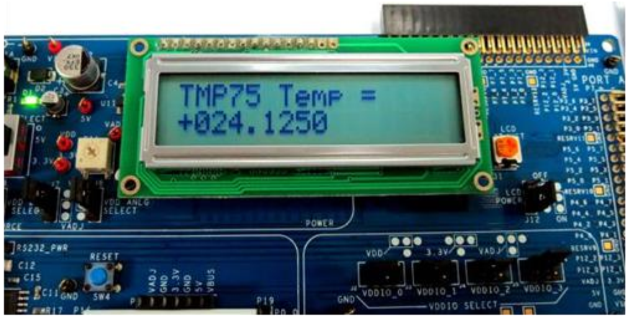
Table 9. Jumper Settings for 001 Kit

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.

The LCD should display the ambient temperature as measured by TMP75 in degree Celsius.

Figure 30. LCD Showing TMP75 Temperature



14 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 31. main() Routine Flow Chart

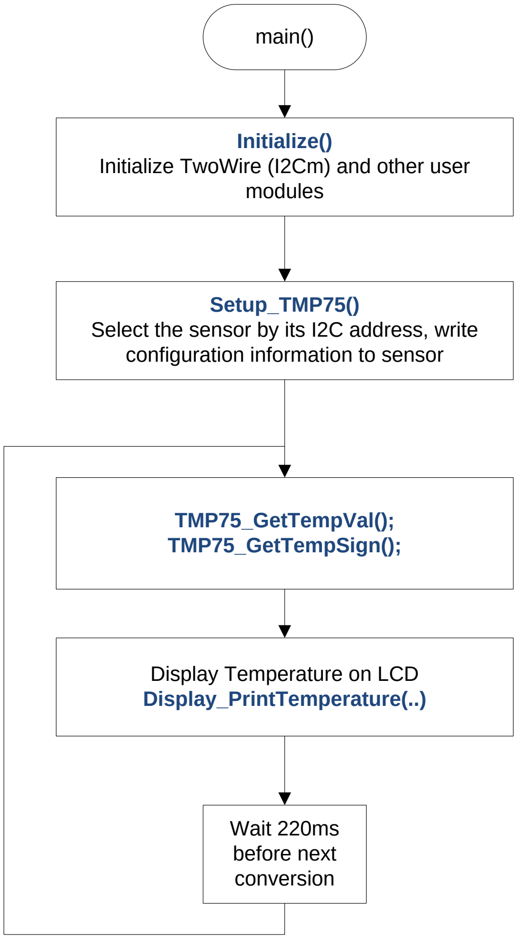

main()
Initialize()
Initialize TwoWire (I2Cm) and other user modules
Setup_TMP75()
Select the sensor by its I2C address, write configuration information to sensor
TMP75_GetTempVal();
TMP75_GetTempSign();
Display Temperature on LCD
Display_PrintTemperature(..)
Wait 220ms before next conversion

Figure 32. Setup_TMP75() Routine Flow Chart

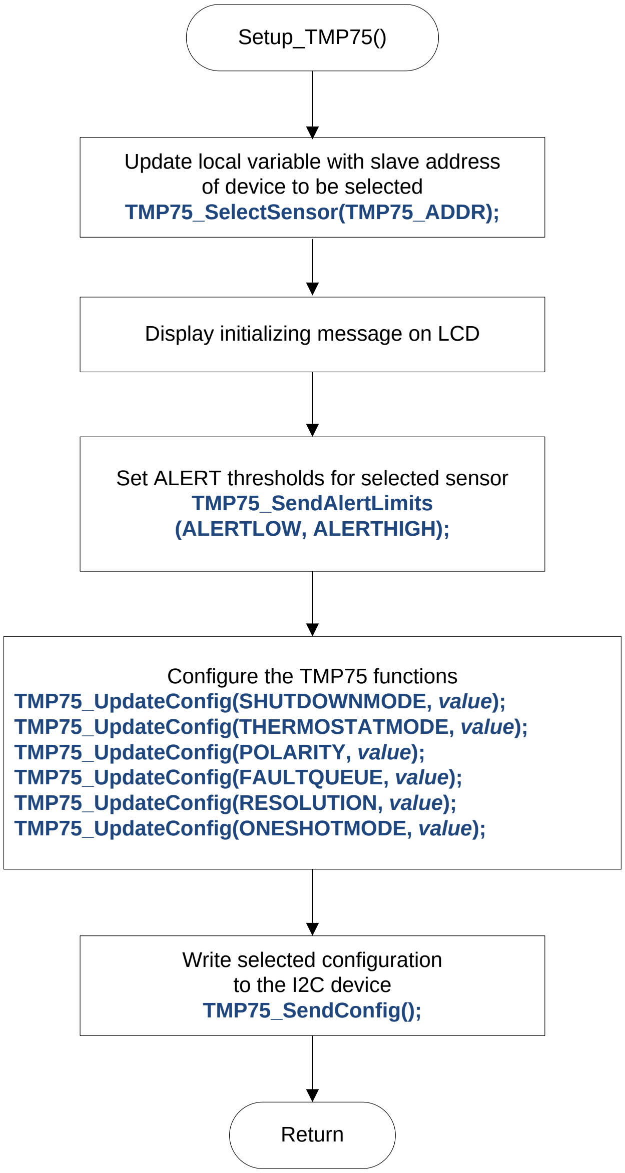
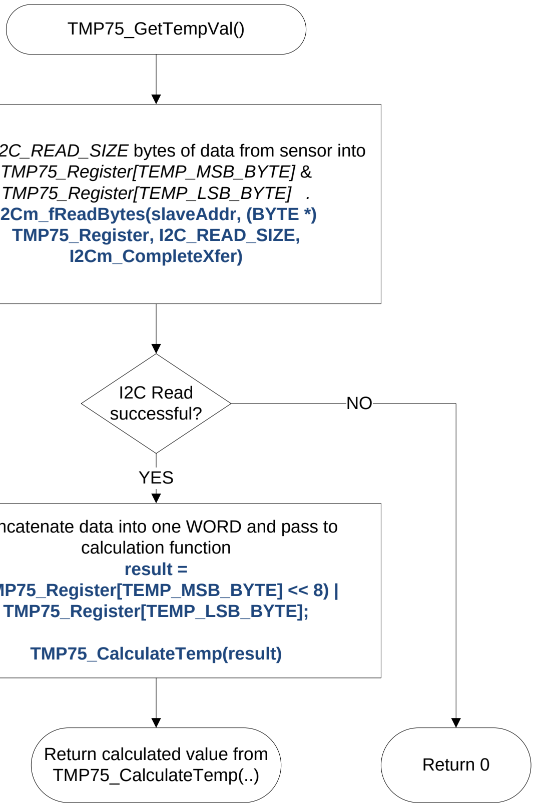

Setup_TMP75()
Update local variable with slave address of device to be selected
TMP75_SelectSensor(TMP75_ADDR);
Display initializing message on LCD
Set ALERT thresholds for selected sensor
TMP75_SendAlertLimits (ALERTLOW, ALERTHIGH);
Configure the TMP75 functions
TMP75_UpdateConfig(SHUTDOWNMODE, value);
TMP75_UpdateConfig(THERMOSTATMODE, value);
TMP75_UpdateConfig(POLARITY, value);
TMP75_UpdateConfig(FAULTQUEUE, value);
TMP75_UpdateConfig(RESOLUTION, value);
TMP75_UpdateConfig(ONESHOTMODE, value);
Write selected configuration to the I2C device
TMP75_SendConfig();
Return

Figure 33. TMP75_GetTempVal() Routine Flow Chart


TMP75_GetTempVal()
Read *I2C_READ_SIZE* bytes of data from sensor into *TMP75_Register[TEMP_MSB_BYTE]* & *TMP75_Register[TEMP_LSB_BYTE]* .
I2Cm_fReadBytes(slaveAddr, (BYTE *) TMP75_Register, I2C_READ_SIZE, I2Cm_CompleteXfer)
I2C Read successful?
NO
YES
Concatenate data into one WORD and pass to calculation function
result = (TMP75_Register[TEMP_MSB_BYTE] << 8) | TMP75_Register[TEMP_LSB_BYTE];
TMP75_CalculateTemp(result)
Return calculated value from TMP75_CalculateTemp(..)
Return 0

14.1 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 10. 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

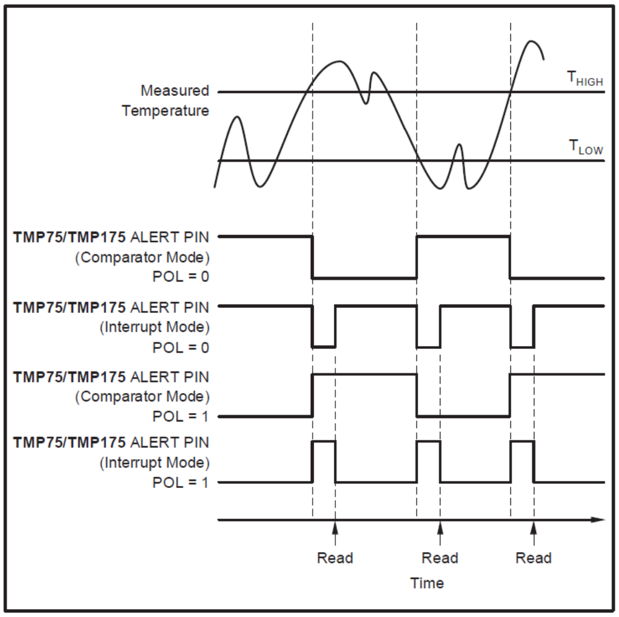
bParam	bData
	RES_10_BITS
	RES_11_BITS
	RES_12_BITS
ONESHOTMODE	ONESHOT_DIS

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

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

14.2 The ALERT Function

Figure 34. 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 11. 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

Function prototype	Description
`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

15 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 DS18X20/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.

16 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

17 Appendix: DS18X20 1-Wire Sensor

17.1 Memory Structure

The DS18X20'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 (3 bytes for the DS18B20) bytes of user defined alarm thresholds.

Table 12. Scratchpad Registers and EEPROM - DS18S20

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

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

Table 13. Scratchpad Registers and EEPROM - DS18B20

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

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

The DS18X20 measures and stores temperature in 2's-complement format in the temperature registers (**Byte 0** and **Byte 1). Table 14** and Table 15 give the LSB (**Byte 0**) and MSB (**Byte 1**) contents that are used to obtain the value and sign of temperature.

Table 14. Temperature Registers - DS18S20

	Bit7	**Bit6**	**Bit5**	**Bit4**	**Bit3**	**Bit2**	**Bit1**	**Bit0**
LSB	2^6	2^5	2^4	2^3	2^2	2^1	2^0	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)

Table 15. Temperature Registers - DS18B20

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
LSB	2^{3}	2^{2}	2^{1}	2^{0}	2^{-1}	2^{-2}	2^{-3}	2^{-4}
	Bit15	**Bit14**	**Bit13**	**Bit12**	**Bit11**	**Bit10**	**Bit9**	**Bit8**
MSB	S	S	S	S	S	2^{6}	2^{5}	2^{4}

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

The TH and TL registers (**Byte 2** and **Byte 3**) are used to configure the alarm thresholds. The configuration register (**Byte 4)** is used to configure the conversion resolution for the DS18B20 sensor. 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 for the DS18S20 sensor. This is possible because the DS18S20 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.

17.2 Slave Addressing

Multiple DS18X20 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 16. DS18X20 ROM Structure

64 bit ROM code	
Byte 0	1-Wire Family Code (10h = DS18S20) , (28h = DS18B20)
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 DS18X20 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. The family code of DS18B20 = *28h*. **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.

18 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.

18.1 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.

18.2 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. 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 35. Timing Diagram for Initialization (from the DS18S20 Datasheet)

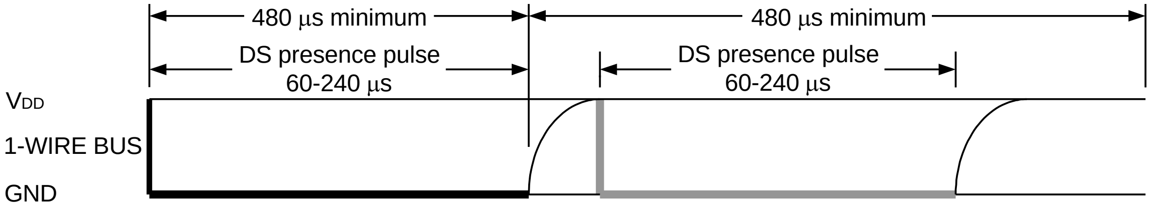
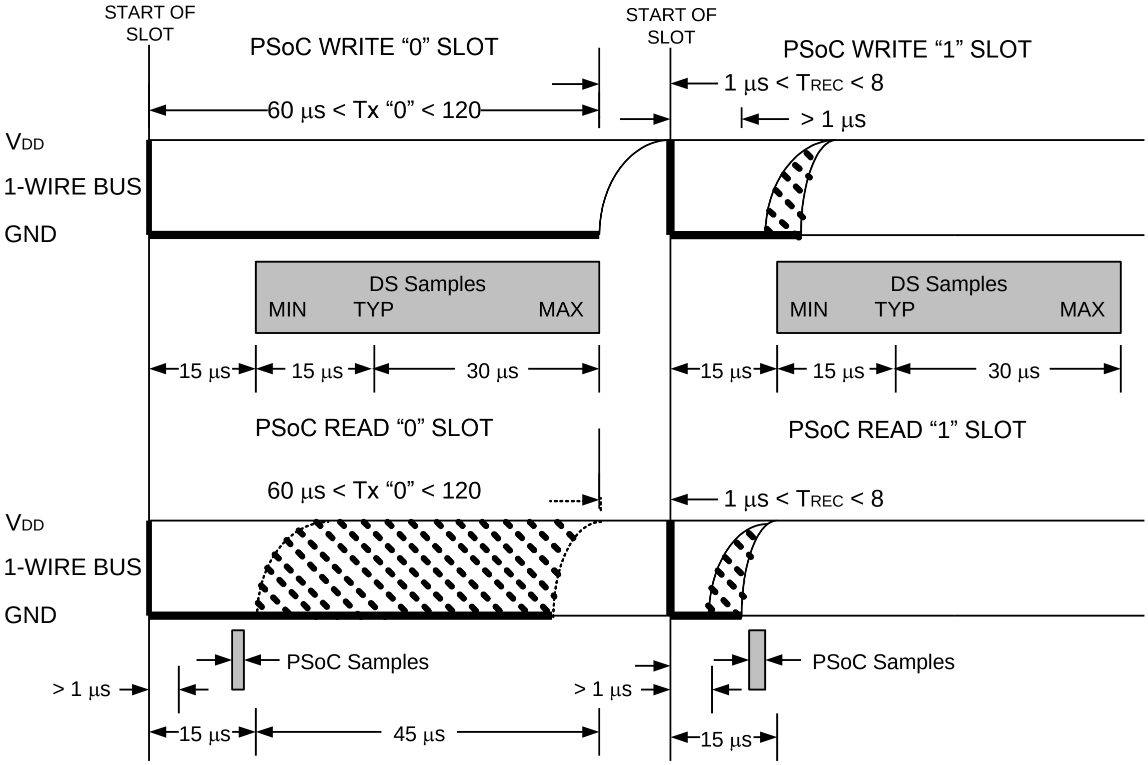


Figure 36. Timing Diagram for Read/Write Slots (from the DS18S20 Datasheet)



19 Appendix: TMP75 Two-Wire Sensor

19.1 TMP75 Memory Structure

The TMP75 internal memory structure consists of four 8-bit registers as given in Table 18. 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 17. Pointer Register

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

Table 18. 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^{2}C 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^{2}C sub-address before reading/writing data bytes to the subsequent memory locations.

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

19.1.1 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 19. 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 20. 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

19.1.2 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 21.

Table 21. 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.

19.1.3 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 22. 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

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.
*G	4763074	ARVI	05/21/2015	Added information on interfacing with DS18B20 1-Wire sensor Updated associated part family list Updated project files to PSoC Designer 5.4 SP1 Updated template
*H	5688155	AESATMP8	04/07/2017	Updated logo, Copyright, and Sales page.

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