Temperature Measurement with the PC87365/6

National Semiconductor
Application Note AN-1128
Ernest Bron
May 1999

The PC87365 and PC87366 are the latest additions to National Semiconductor's PC8736x family of LPC-based SuperI/O devices. These devices integrate most of the usual SuperI/O functions. Their prime feature, however, is the incorporation of analog functions such as temperature and voltage measurement.

This Application Note describes the new temperature measurement module. Both the PC87365 and the PC87366 include a diode-based temperature sensor. In addition, the PC87366 includes a thermistor-based temperature sensor.

DIODE-BASED TEMPERATURE MEASUREMENT SENSOR (TMS)

The PC87365 and PC87366 include sophisticated diode-based temperature measurement, using an on-chip multi-level current source. A Sigma-Delta A/D converter, which effectively reduces the circuit's susceptibility to noise, improves the accuracy of the temperature measurement. Automatic averaging ensures stable and reproducible readings.

Architecture

A separate logical device (number 0xE) is assigned to the TMS module. Figure 1 is a simplified block diagram of this module, showing one external diode, although two external diode connections are available. Once enabled, the TMS module continuously measures the temperature of up to two external diodes, and an internal diode (PC87366 only). The host system may read the measured temperature at any time. When the main system indicates it is in power-down mode (by inactivating SLPS3), the TMS module stops measuring temperature. Measurements resume once SLPS3 reactivates, and the internal wake-up sequence is completed. During this power-down period, the contents of the TMS registers are maintained by V_{SB}.

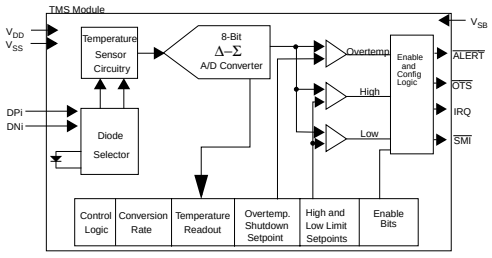

Figure 1. TMS Simplified Block Diagram

Figure 1 shows the basic operation of the TMS module. On the left side, the diode is connected to the DPi and DNi terminals. A diode selector module selects one of two diodes (PC87365), or one of three diodes (PC87366). The diode terminals are connected to the Temperature Sensor circuitry which forces two fixed currents through the diode. A Delta-Sigma Analog-to-Digital converter measures the voltage across the diode, while these currents are forced through it. These two voltages are used to calculate the temperature of the diode. The resulting 8-bit value is fed to the Read Channel Temperature register (RDCHT), and to the Enable and Configure Logic block. This block compares the measured temperature with previously programmed limit values, and generates alarms if the measured temperature exceeds the set limits. Three types of temperature alarms are available, High Limit, Low Limit and OverTemp. Each of these alarms has an associated status bit.

There are two independent alarm output terminals, OTS and ALERT:

- The OTS output is associated with an OverTemp event. If the measured temperature exceeds the OverTemp setting, the OTS output can be programmed to go active
- The ALERT output is associated with both High and Low Limit events. If the measured temperature falls outside the High or Low Limit temperature setting, the ALERT output can be programmed to go active. Note that the ALERT output is unconnected in the device.

Either ALERT or OTS can be enabled, or disabled, and can be routed to IRQ, SMI or both. Figure 2 shows the temperature alarm block diagram per channel. Figure 4 is an overview of the entire module and device.

Two OTS outputs can be activated: OTS1 and OTS2. Bits 0-2 of the SuperI/O Configuration A register, SIOCFA, control the OTS output number and OTS function enabling on pins.

Table 1 shows the programmed setting versus the value of bits 0-2 of SIOCFA. In Table 1 a 'Y' indicates that the respective OTS pin is activated for a particular channel, provided the OTS functionality is enabled for that channel. Typically the configuration of how many or which OTS pins to use is performed at the BIOS basis level.

This Application Note deals with the TMS module only, and assumes that one, or more, OTS outputs are enabled on one, or more pins. Throughout this Application Note, a single signal called OTS is used to reference the internal OverTemp event, regardless of the actual method programmed to output that event on one or more pins.

Table 1. OTS Pin Enabling and Functionality

	1 OTS Pin	**2 OTS Pins**	
SIOCFA	**xxxxxxx0b** **xxxxx001b** **xxxxx011b** **xxxxx111b**	**xxxxx101b**	
Source	**OTS1**	**OTS1**	**OTS2**
Remote 1	Y	Y	N
Remote 2	Y	N	Y
Local	Y	Y	N
VLM (PC87366 only)	Y	Y	N

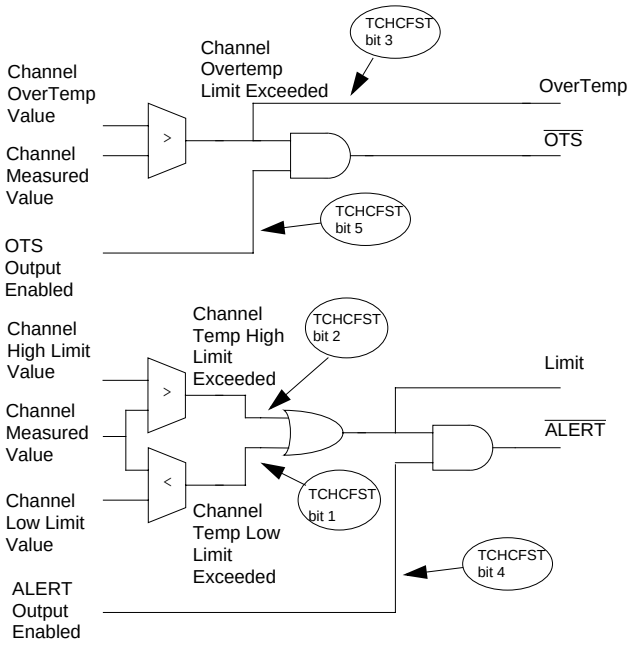

Figure 2. TMS Channel OverTemp and Limit Detection

Figure 3 shows a simplified diagram where the PC87365, or PC87366, is connected to a chipset that provides a thermal alarm input. The OTS output of the TMS module can be connected to this THRM input. There are various ways a chipset may handle a thermal alarm event from the TMS module. In the example shown in Figure 3, the chipset provides WakeOnLAN support, which allows automatic error reporting to a Network Controller. In addition, or alternatively, OTS or ALERT events can be reported to the chipset via the SMI or SERIRQ signals.

A pull-up may be required on OTS, since this pin is open drain. The pull-up can be tied to V_{SB} or V_{DD}, depending on the application, however in the example of Figure 3 it is assumed that the THRM input requires a low level on its THRM input when V_{DD} is not applied. Such system configurations require the pull-up to be tied to V_{DD}.

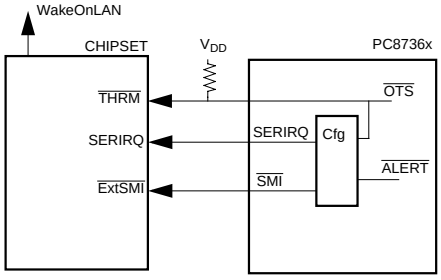

Figure 3. OTS and ALERT System Connection Scheme

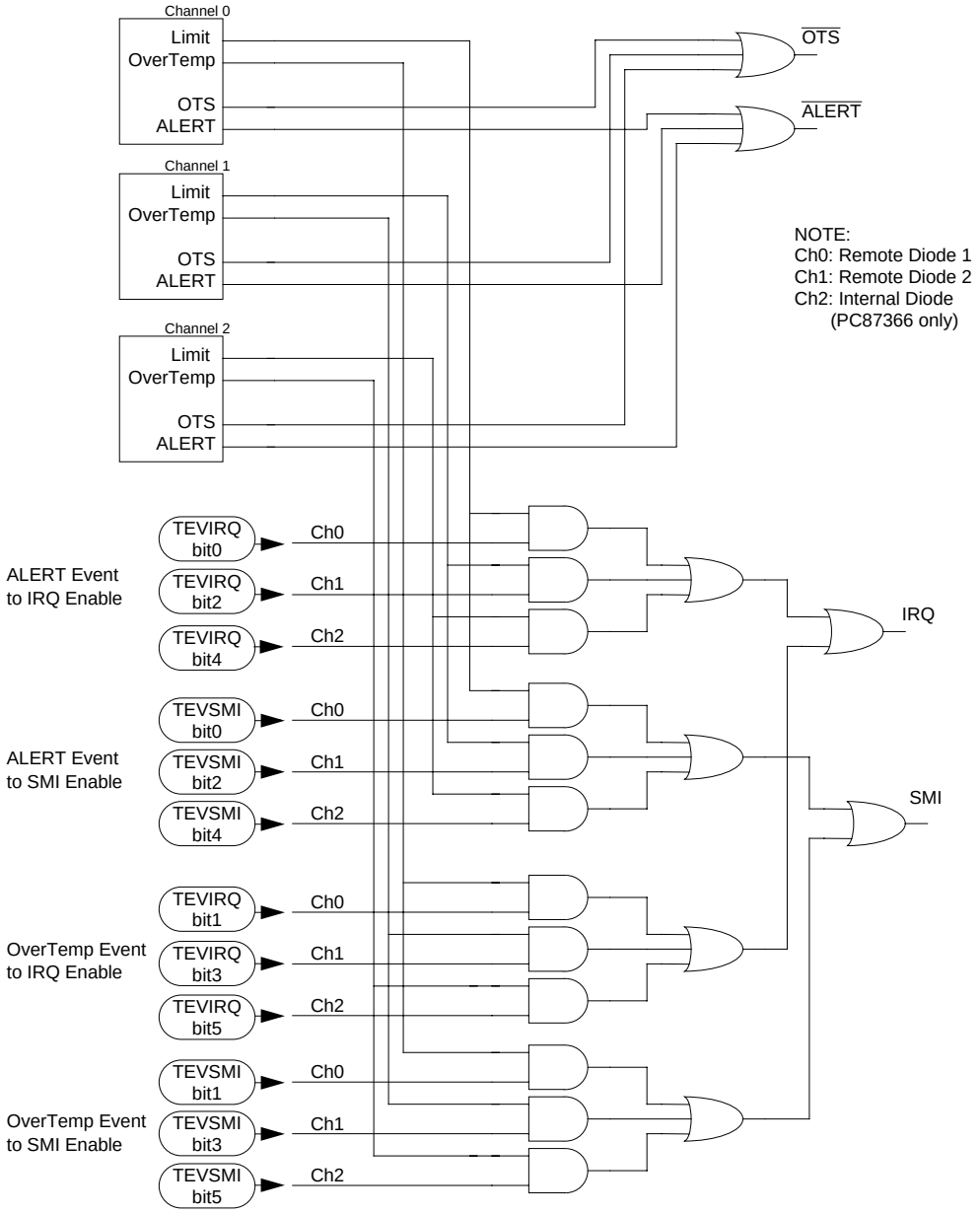

Figure 4. TMS Status Mapping

Temperature Data Format and Conversion

Temperature data is stored in the RDCHT, CHTH, CHTL and CHOTL registers as an 8-bit, 2's complement word with a Least Significant Bit (LSB) equal to 1˚ C.

To convert read temperatures, that are in 2's complement, software should check bit 7. If set, the measured temperature is below zero.

To convert the number to a real temperature, subtract the value 256 from it. If bit 7 is clear, the value read is the real temperature.

Diode Fault Protection

To protect against invalid temperature readings, the TMS module contains specialized protection circuitry that enables software to detect potential hardware problems and decide on the appropriate course of action. If the D1P or the D2P line is shorted to V_{DD} or floating, the temperature reading is 127˚ C and the Open bit in TCHCFST is set. Upon completion of a conversion, the setting of the Open bit in TCHCFST generates both an ALERT and an OTS event.

If the D1P or the D2P line is shorted to GND or D1N or D2N respectively, the temperature reading is 0˚ C and the Temperature Low Limit bit of TCHCFST is **not** set.

Programming Model

The registers that control the TMS are partially organized in banks. Figure 5 and Figure 6 show all the registers. The lower 10 offsets are common to all three banks (Figure 5). The upper six offsets are unique registers per bank (Figure 6). A bank effectively controls the measurements on a single diode. One bank is available per diode, which implies that two banks are available in the PC87365, and three banks are available for the PC87366. Bank 0 is for remote diode 1, bank 1 for remote diode 2, and bank 2 for the internal diode (PC87366 only).

To access a specific register in a bank, write the bank number into the TMSBS register. Subsequent read or write accesses to offsets 0xA to 0xE access the registers of that particular bank. The following pseudo C-code example demonstrates a read access of RDCHT of bank 1, effectively reading the measured temperature on remote diode 2.

```
outp(TMS_BASE + TMSBS,0x01h);
Temp = inp(TMS_BASE + RDCHT);
```

This example, as all following examples, assumes that the base address of the TMS module is **TMS_BASE**. This value must be assigned.

Register		Bits							
Offset	Mnemonic	7	6	5	4	3	2	1	0
00h	TEVSTS	Reserved				Remote 2 Overtemp Event Status	Remote 2 $\overline{\text{ALERT}}$ Event Status	Remote 1 Overtemp Event Status	Remote 1 $\overline{\text{ALERT}}$ Event Status
01h	Reserved								
02h	TEVSMI	Reserved				Remote 2 Overtemp Event to $\overline{\text{SMI}}$ Enable	Remote 2 $\overline{\text{ALERT}}$ Event to $\overline{\text{SMI}}$ Enable	Remote 1 Overtemp Event to $\overline{\text{SMI}}$ Enable	Remote 1 $\overline{\text{ALERT}}$ Event to $\overline{\text{SMI}}$ Enable
03h	Reserved								
04h	TEVIRQ	Reserved				Remote 2 Overtemp Event to IRQ Enable	Remote 2 $\overline{\text{ALERT}}$ Event to IRQ Enable	Remote 1 Overtemp Event to IRQ Enable	Remote 1 $\overline{\text{ALERT}}$ Event to IRQ Enable
05h-07h	Reserved								
08h	TMSCFG	Reserved						External V_{REF}	Standby Mode
09h	TMSBS	Reserved				Bank Select			

Figure 5. TMS Control and Status Registers

Register		Bits							
Offset	Mnemonic	7	6	5	4	3	2	1	0
0Ah	TCHCFST	End of Conversion	Open	$\overline{\text{OTS}}$ Output Enable	$\overline{\text{ALERT}}$ Output Enable	Channel OverTemp Limit Exceeded	Channel Temp High Limit Exceeded	Channel Temp Low Limit Exceeded	Channel Enable
0Bh	RDCHT	Channel Temperature Value							
0Ch	CHTH	Channel Temperature High Limit Value							
0Dh	CHTL	Channel Temperature Low Limit Value							
0Eh	CHOTL	Channel OverTemp Value							
0Fh	Reserved								

Figure 6. TMS Channel Registers

Measuring Temperature

As mentioned above, if enabled, the TMS module continuously measures temperature. Therefore, the first step towards measuring temperatures is to enable the module itself. This is accomplished by clearing bit 0 of TMSCFG. In addition, the designer must decide what reference voltage to use in the system.

To operate, the TMS module requires a reference voltage (V_{REF}). This voltage can be applied from either an internal, or an external, source. Bit 1 of TMSCFG selects the source. It is recommended to use the internal V_{REF}. This ensures that the V_{REF} level is optimal, and saves the cost of an additional, external, reference voltage device. This Application Note assumes the use of the internally generated V_{REF}. To enable the TMS module, and to select the internal V_{REF}:

```
outp(TMS_BASE + TMSCFG, 0x00)      // Select internal VREF, out of standby
```

The TMS block is now enabled. Each channel also has its own channel enable bit. To enable temperature measurements on any particular channel, set this bit. For example, to enable channel 1:

```
outp(TMS_BASE + TMSBS, x01)        // Select bank 1
outp(TMS_BASE + TCHCFST, 0x01)     // Enable channel
```

The TMS module can now make some basic temperature measurements. Bit 7 of TCHCFST (Valid bit) indicates if a new measurement was made since the last time this bit was cleared. However, since reading the RDCHT register always returns the last temperature measured, it is not essential to read this bit. To simply read temperature:

```
inp(TMS_BASE + RDCHT)              // Read temperature
```

In some special cases, temperatures must be read only when a new measurement has been made, i.e. the valid bit has been set. To use the valid bit, check its status before reading the temperature:

```
Status = inp(TMS_BASE + TCHCFST)       // Get current Status
if (Status&0x80) == 0x80               // Valid (i.e. New) data?
{    outp(TMS_BASE + TCHCFST, Status)  // Write back, clear all pending flags
     inp(TMS_BASE + RDCHT)             // Read temperature
}    // if
```

Diode Fault Detection

An even more advanced temperature reading routine would also include open/short circuit checks when reading the temperature. The following is a representative routine that checks for open, or short, circuits. Note the check for the programmed lower limit before verifying that the actual temperature read is 0˚ C, and that there is no lower limit event detected. This ensures that the non-occurrence of a lower limit event is due to a diode short, and not simply because the lower limit is set below 0˚ C:

```
Status = inp(TMS_BASE + TCHCFST)       // Get current Status
if (Status&0x40) == 0x40               // Diode Open??
     return(OPEN_FAULT);               //
LowLimit = inp(TMS_BASE+CHTL);         // Read Lower limit
if (LowerLimit)&0x80) == 0x00)         // Lower limit >= 0?
{    if ((Temperature == 0x00) &&      // If Temperature is 0 and no low limit temp detected
          (Status &0x02)==0x00))
     return(SHORT_ERROR);
}    // if
return(NO_ERROR);                      // No error detected
```

Using ALERT and/or OverTemp

To detect OverTemp and/or an out-of-limit temperature, the required OverTemp and limits, must first be defined, and programmed into the relevant registers. To program OverTemp, and limits, for channel 1 (remote diode 2):

```
outp(TMS_BASE+TMSBS,0x01);             // Select bank/channel 1
outp(TMS_BASE+CHOTL,OverTemp);         // Program OverTemp
outp(TMS_BASE+CHTH,HighLimit);         // Program High Limit
outp(TMS_BASE+CHTL,LowLimit);          // Program Low Limit
```

Once these values are initialized, the TMS module automatically updates the relevant flags in the TCHCFST and TEVSTS registers. To enable the OTS output, set bit 5 of TCHCFST.

```
Status = inp(TMS_BASE + TCHCFST)       // Get current Status
Status |= 0x20;                        // Set bit 5
outp(TMS_BASE + TCHCFST, Status)       // Write back
```

To enable either an OverTemp or ALERT event to SMI or IRQ, the relevant bit in either TMSSMI or TMSIRQ must be set. For example, to enable ALERT event of channel 1 (remote diode 2) onto SMI:

```
SMIConf = inp(TMS_BASE + TMSSMI)       // Get current Configuration
Status |= 0x04;                        // Set bit 2
outp(TMS_BASE + TMSSMI, SMIConf)       // Write back
```

Before enabling any status onto IRQ or SMI it is always good practice to clear all pending flags. To clear all pending flags of channel 1 (remote diode 2):

```
     outp(TMS_BASE+TMSBS,0x01);            // Select bank/channel 1
     Status = inp(TMS_BASE+TCHCFST);       // Get current status
     outp(TMS_BASE+TCHCFST,Status);        // Clear all flags
```

Generating Functions

Using the code examples shown above, general-purpose functions that operate the TMS module can be generated.

Initialize the TMS

```
// ***************************************************************************
// * TMSInit   Initializes TMS Module
// * Inputs:   Vref: Selects VREF source, 0 for internal, 1 for external
// *           SMIConf: Alert or Overtemp status to SMI Mapping
// *                Bit 0 Set to enable Remote Diode 1 Alert Event to SMI
// *                Bit 1 Set to enable Remote Diode 1 OverTemp Event to SMI
// *                Bit 2 Set to enable Remote Diode 2 Alert Event to SMI
// *                Bit 3 Set to enable Remote Diode 2 OverTemp Event to SMI
// *                Bit 4 Set to enable Local Diode Alert Event to SMI
// *                Bit 5 Set to enable Local Diode OverTemp Event to SMI
// *           IRQConf: Alert or Overtemp status to IRQ Mapping
// *                Bit 0 Set to enable Remote Diode 1 Alert Event to IRQ
// *                Bit 1 Set to enable Remote Diode 1 OverTemp Event to IRQ
// *                Bit 2 Set to enable Remote Diode 2 Alert Event to IRQ
// *                Bit 3 Set to enable Remote Diode 2 OverTemp Event to IRQ
// *                Bit 4 Set to enable Local Diode Alert Event to IRQ
// *                Bit 5 Set to enable Local Diode OverTemp Event to IRQ
// * Outputs:  None
// ***************************************************************************
void TMSInit(int Vref, int SMIConf, int IRQConf)
{    int  TMSConf;
     TMSConf = inp(TMS_BASE+TMSCFG);
     if (Vref) outp(TMS_BASE+TMSCFG.(TMSConf|2));// External V_REF
     else outp(TMS_BASE+TMSCFG.(TMSConf&0xFD));// Internal V_REF
     outp(TMS_BASE+TEVSMI,SMIConf);
     outp(TMS_BASE+TEVIRQ,IRQConf);
}
```

Enabling or Disabling TMS Altogether

```
// ***************************************************************************
// * TMSStandby     Places TMS module into standby, or takes it out of it
// * Inputs:        Enable: 1 to put into standby, 0 to take out of standby
// * Outputs:       None
// ***************************************************************************
void TMSStandby(int Enable)
{    int  TMSConf;
     TMSConf = inp(TMS_BASE+TMSCFG);
     if (Enable) outp(TMS_BASE+TMSCFG.(TMSConf|0x01));// Enable standby mode
     else outp(TMS_BASE+TMSCFG.(TMSConf&0xFE));// Disable standby mode
```

Enabling or Disabling Measurements of a Particular Channel

```
// ***************************************************************************
// * EnableChannel  Enables or disables single channels in TMS module
// * Inputs:        Channel: Selects the channel, must be between 0 and 2
// *                Enable: 1 to enable, 0 to disable
// * Outputs:       None
// ***************************************************************************
void EnableChannel(int Channel, int Enable)
{    int  Status;
     outp(TMS_BASE+TMSBS,Channel);                    // Select bank #
     Status = inp(TMS_BASE+TCHCFST);
     if (Enable)    {    outp(TMS_BASE+TCHCFST, Status); // Clear all flags
                         Status |= 0x01;                 // Set enable bit
                         outp(TMS_BASE+TCHCFST, Status); // Enable the channel
                    }    // if
     else {    Status &= 0x30;                          // Keep bits 4 and 5
               outp(TMS_BASE+TCHCFST, Status);          // Disable the channel, retain flags
          }    // else
}
```

Enabling or Disabling OTS Output of a Particular Channel

```
// ************************************************************************
// * OTSEnable      Enables or disables OTS output for a single channel
// * Inputs:        Channel: Selects the channel, must be between 0 and 2
// *                Enable: 1 to enable OTS, 0 to disable OTS
// * Outputs:       None
// ************************************************************************
void OTSEnable(int Channel, int Enable)
{    int  Status;
     outp(TMS_BASE+TMSBS,Channel);         // Select bank #
     Status = inp(TMS_BASE+TCHCFST);
     if (Enable)    {    outp(TMS_BASE+TCHCFST, Status);// Clear all flags
                         Status |= 0x020;    // Set enable bit
                         outp(TMS_BASE+TCHCFST, Status);// Enable OTS
                    }    // if
     else {    Status &= 0x11;              // Keep bit 4 and 0
               outp(TMS_BASE+TCHCFST, Status);// Disable OTS, retain flags
          }    // else
}
```

Basic Reading Temperature of a Particular Channel

```
// ************************************************************************
// * ReadTemp       Read back latest temperature read from channel, no attempt is
// *                made to verify that any new measurement has been made since the
// *                last time the temperature was read (i.e. Valid bit is not checked)
// * Inputs:        Channel: Selects the channel, must be between 0 and 2
// * Outputs:       Integer value of measured temperature. Value is between
                   0 and 255
// ************************************************************************
int ReadTemp(int Channel)
{    outp(TMS_BASE+TMSBS,Channel);         // Select bank #
     return(inp(TMS_BASE+RDCHT);
}
```

Diode Fault Verification

```
// ************************************************************************
// * ChannelCheck   Check the selected channel for diode faults
// * Inputs:        Channel: Selects the channel, must be between 0 and 2
// *                Temperature: The previously read temperature
// * Outputs:       SHORT_ERROR if diode Short was detected
// *                OPEN_FAULT if diode Open was detected
// *                NO_ERROR if No Error was detected
// ************************************************************************
int ChannelCheck(int Channel, int Temperature)
{    int  Status, LowLimit;
     outp(TMS_BASE+TMSBS,Channel);         // Select bank #
     Status = inp(TMS_BASE + TCHCFST)      // Get current Status
     if (Status&0x40) == 0x40)             // Diode Open??
          return(OPEN_FAULT);              // Yes, return Error
     LowLimit = inp(TMS_BASE+CHTL);        // Read Lower limit
     if (LowerLimit)&0x80) == 0x00)        // Lower limit >= 0?
     {    if ((Temperature == 0x00) &&     // If Temperature is 0 and no low limit
               (Status &0x02)==0x00))      // temperature detected
          return(SHORT_ERROR);             // return error
     }    // if
     return(NO_ERROR);                     // no error detected
}
```

Initializing OverTemp and Limiting Temperature Values

```
// ************************************************************************
// * InitLimits     Initializes the Values set to OverTemp, lower limit
// *                and upper limit
// * Inputs:        Channel: Selects the channel, must be between 0 and 2
// *                OverTemp: OverTemp value
// *                HighLimit: High Limit value
// *                LowLimit: Low Limit value
// * Outputs:       None
// ************************************************************************
void InitLimits(int Channel, int OverTemp, int HighLimit, int LowLimit)
```

```
{    int  Status
     outp(TMS_BASE+TMSBS,Channel);         // Select bank/channel #
     outp(TMS_BASE+CHOTL,OverTemp);        // Set OverTemp
     outp(TMS_BASE+CHTH,HighLimit);        // Set Higher limit
     outp(TMS_BASE+CHTL,LowLimit);         // Set Lower limit
}
```

Converting Temperature Values (0 to 255) to Real Temperature and Vice Versa

```
// ************************************************************************
// * Val2Temp       Converts 2's complement temperature value to corresponding
// *                temperature
// * Inputs:        Value: Byte value (MUST between 0 and 255)
// * Outputs:       Real temperature
// ************************************************************************
int Val2Temp(int Value)
{    int  Temperature
     if (Value&0x80)    Temperature = (Value-0x100);// Negative temperature
     else Temperature = Value;              // Positive
     return(Temperature);
}

// ************************************************************************
// * Temp2Val       Converts real temperature to 2's complement temperature value
// *                that can be used for the TMS
// * Inputs:        Temperature: Real temperature (MUST between -128 and 127)
// * Outputs:       Temperature: 2's complement number
// ************************************************************************
int Temp2Val(int Temperature)
{    int  Value
     if (Temperature <0) Value = Temperature + 0x100;// If below zero, simply add
     else Value = Temperature;              // Else NOP
     return(Value);
}
```

Combining

To operate the TMS module using some of the functions described here, perform all initializations while the channels are disabled. Only after all initializations are finished should the channels be enabled. The recommended order of initialization is: disable channel(s), initialize, enable channel(s).

See the example code below that initializes the TMS to operate from an internal V_{REF}, and maps all possible events to IRQ. Channels 0 to 2 are enabled with no OTS outputs. Limits are set to −20° C to +75° C. OverTemp is set at +90° C. On entry, it is assumed that the TMS is in its default state, i.e. in standby, and all individual channels are disabled.

```
TMSInit(0,0x00,0x3F);                                    // Int VREF, map all events to IRQ;
SetLimits(0,Temp2Val(90),Temp2Val(75),Temp2Val(-20));    // Set channel 0 limit values
SetLimits(1,Temp2Val(90),Temp2Val(75),Temp2Val(-20));    // Set channel 1 limit values
SetLimits(2,Temp2Val(90),Temp2Val(75),Temp2Val(-20));    // Set channel 2 limit values
TMSStandby(0);                                           // Take TMS out of standby
EnableChannel(0,1);                                      // Enable channel 0
EnableChannel(1,1);                                      // Enable channel 1
EnableChannel(2,1);                                      // Enable channel 2
     .
     .
     .
CPUTemp = Val2Temp(ReadTemp(0));                         // Get channel 0 temperature
Error = ChannelCheck(0,CPUTemp);                         // Make Sure No Diode Fault
```

Using the functions described above, a basic temperature measurement system can be implemented. The functions can easily be extended to add additional functionality.

THERMISTOR-BASED TEMPERATURE MEASUREMENT

The PC87366 can measure temperatures on channels 11 to 13 of its internal Voltage Level Measurement module (VLM). The measured signal is a voltage, but by using a thermistor as part of the resistive voltage divider on the voltage input, the temperature can be measured. Figure 7 shows examples of connecting thermistors to the VLM. In these examples, a higher temperature results in a lower measured voltage.

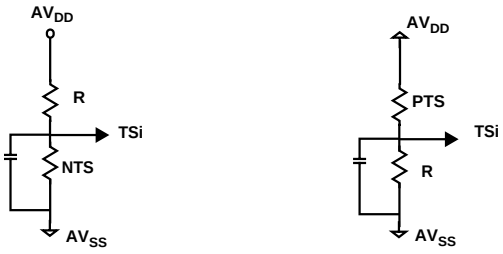

Figure 7. Thermistor-based Temperature Measurement

The channels of the VLM that support temperature measurement include standard OverTemp and ALERT event notification schemes.

Principle of Operation

There are two types of thermistors, Negative Temperature Coefficient (NTC) and Positive Temperature Coefficient (PTC). The resistance of an NTC decreases with increasing temperature. The resistance of a PTC increases with increasing temperature. Usually, NTC thermistors are used. This Application Note assumes the use of NTC thermistors for temperature measurement purposes. Most NTCs are specified by a resistance value at 25˚C, and a constant, B, valid for temperatures between 25°C and 85°C. Figure 8 shows typical curves for NTC resistance versus temperature.

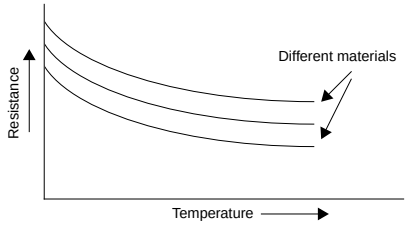

Figure 8. NTC Resistance vs. Temperature

The theoretical resistance of an NTC can be expressed as:

$$R_T = A \bullet e^{B/T}$$

Where R_T is the resistance of the NTC, A and B are constants, and T is the temperature in degrees Kelvin. B depends on the material, and typically lies between 2000 and 5500˚K. This implies that a temperature change of 1˚C would result in a resistance change of 2-6% (at a nominal temperature of 25˚C).

The VLM measures voltages using the following equation:

$$Vi = 2.45 \times Vref \times RDCHV_i/256$$

Using the NTC thermistor-based voltage divider, as shown in Figure 7, the input voltage to the VLM is:

$$V_i = AV_{DD} \times \left(\frac{R_T}{R + R_T}\right)$$

Using the above equations, system performance and measurements can be evaluated. For example, assume an NTC with nominal resistance of 10K at 25˚C and B of 3820 between 28 and 85˚C. From these values A can be calculated, and a table of temperature versus Vi and measured voltage can be generated. Table 2 shows the numbers generated using an NTC with parameters as described above and a 10K pull-up resistor (R in Figure 7). Actual system temperature can be measured using such a table, and implementing a lookup table to determine temperature from measured voltage.

T (K)	T (C)	Rt	Vi	RDCHV
298	25	10000	1.65	142
300	27	9181	1.58	136
302	29	8438	1.51	130
304	31	7765	1.442	124
306	33	7152	1.376	118
308	35	6596	1.312	113
310	37	6088	1.249	107
312	39	5626	1.188	102
314	41	5204	1.129	97
316	43	4818	1.073	92
318	45	4465	1.019	87
320	47	4142	0.967	83
322	49	3846	0.917	79
324	51	3575	0.869	74
326	53	3325	0.824	71
328	55	3096	0.78	67
330	57	2885	0.739	63
332	59	2691	0.7	60
334	61	2512	0.662	57
336	63	2346	0.627	54
338	65	2194	0.594	51
340	67	2053	0.562	48
342	69	1922	0.532	45
344	71	1801	0.504	43
346	73	1689	0.477	41
348	75	1585	0.452	38
350	77	1489	0.428	36
352	79	1399	0.405	34
354	81	1316	0.384	33
356	83	1239	0.364	31
358	85	1167	0.345	29

Where:
T (K) = Temperature in degrees Kelvin
T (C) = Temperature in degrees Centigrade
Rt = NTC Resistance
Vi = Voltage applied to VLM input
RDCHV = Measured number by VLM (decimal)

Table 2. Lookup Table Example

Other Resistance Versus Temperature Profiles

The method described in the previous section assumes that the resistance of the thermistor follows the theoretical formula. For all practical purposes, the system designer must always verify the real life resistance/temperature curve. Although some thermistor vendors specify the temperature/resistance characteristics of their devices using the parameters described above, other vendors supply different data including a complete temperature vs. resistance table and/or figures, or other formulas and parameters.

Lookup Table Implementation

In practise it is hard to implement a thermistor-based temperature measurement system without the use of a lookup table:

- The wide variety of methods used by thermistor vendors to specify the resistance vs. temperature characteristics of their thermistors, prohibits the use of a standard (universal) Readout-to-Temperature conversion function. Such a function could be implemented, but any change in the type of thermistor used could require significant changes in this function.
- Thermistor vendors supply resistance vs. temperature data using either formulas, curves or tables. In all cases, the behavior is an approximation and only provides conversion from thermistor resistance to thermistor temperature.
  - — If a logarithmic formula is provided, a higher level language, such as C, could do the conversion easily. However, most BIOS is written in assembly, and the overhead of doing these kind of calculations at the assembly level is significant.
  - — If a polynomial approximation formula is provided, higher level languages can be used without problems, but implementation in assembly is a problem for both code size and speed. An example of a polynomial approximation is:

$$T = \sum_{k=1}^{n} a_k \times R_T^{\,k}$$

Where:
T = Temperature
R_T = Thermistor Resistance
a_k = Coefficient
n = Number of iterations

— If a graph is provided, its behavior must be approximated which results in the same practical problem as described above. Alternatively, a table can be built from the figure. In this case, a lookup table is required to convert temperature to resistance.

In conclusion, it is strongly recommended to implement a lookup table to convert read values into real temperatures. An example of such a function is:

```
; ********************************************************
; * _GetTemp:  Procedure to get the temperate based on
; *            read value. Table lookup is implemented
; * Input:     AX - Value read from PC87366 (index into table)
; * Output:    AL - Actual Thermistor temperature, based
; *                 on value read

; * Registers Used: BX, DI, AX
; ********************************************************
;
          PUBLIC        _GetTemp
_GetTemp  PROC     near
          mov  bx,  offset _TABLE
          mov  di, ax
          mov  al,  [bx+di]
          ret
_GetTemp  ENDP

     .
     .
     .

;***** Here we define the table that translates the value read from the
;***** PC87366 into the temperature of the thermistor.
;***** NOTE:   THIS TABLE MUST BE MODIFIED ACCORDING TO THE
;*****         SYSTEM EXTERNAL RESISTORS AND THE THERMISTOR TYPE.
;*****         THE DATA IN THIS TABLE IS PROVIDED AS AN EXAMPLE
;*****         AND SHOULD NOT BE USED FOR REAL APPLICATIONS.

_TABLE    db   60h,5Fh,5Eh,5Ch,5Ah,59h,57h,56h
          db   51h,50h,49h,48h,47h,45h,44h,43h
                    .
                    .
                    .
          db   07h,07h,06h,06h,06h,06h,06h,06h
          db   05h,05h,05h,05h,05h,05h,05h,05h
```

Architecture

Thermistor-based temperature measurement is performed using the VLM module. For channels 11 to 13, OverTemp and limit settings, similar to those of the TMS, are provided. OverTemp, or ALERT, events can be enabled to generate IRQ and/or SMI similar to the TMS module. The major difference is that all configuration must be done from within the VLM module. For details on voltage measurement techniques using the PC87366, refer to the datasheet.

Additional Thermistor-based Temperature Measurement Information

To ensure the best thermistor-based temperature measurement performance, it is strongly recommended to initially test the design using standard resistors instead of a resistor/thermistor combination. This can be done by replacing the thermistor with a resistor that has a resistance equal to the nominal resistance of the thermistor to be used. Only if these measurements are stable, and AV_{CC} is observed clean and free of noise, should the thermistor be replaced in the circuit.

Take care to avoid self-heating of the thermistor. If the bias current of the thermistor is too high, the device heats itself, thereby reducing the resistance and inadvertently causing error to the measurement. Consult your thermistor vendor for specific data of self-heating versus power.

Thermistors are used in a wide variety of applications. This Application Note is not intended to describe all possible ways and methods to implement thermistor-based temperature measurement. For further, detailed, information on using thermistors and their temperature curves contact the following suppliers:

- Quality Thermistor Inc. www.thermistor.com
- Iksan Technology Co. Ltd. www.iksan.co.kr
- Alpha Sensors, Inc. www.alphasensors.com
- BetaTHERM Corp. www.betatherm.com
- US Sensors Corp. www.ussensor.com

or any other sensor and/or thermistor vendor.

Noise and Layout Issues

The TMS and VLM modules are essentially analog functions. Separate Analog V_{SS} and V_{DD} planes are provided inside the PC87365 and PC87366. To minimize noise that could cause invalid temperature and/or voltage readings, adhere to the following rules and guidelines.

TMS Related Guidelines

- High frequency EMI is best filtered at DXP and DXN with an external 2200 pF capacitor. This value can be increased to about 3300 pF, including cable capacitance. Capacitance higher than 3300 pF introduces errors due to the rise time of the switched current source. Adding a 0.1 µF filter at DXP and DXN may improve noise performance in some systems.
- Place the PC87365/6 as close as practical to the remote diode. In a noisy environment, such as a computer motherboard, this distance can be 4 to 8 inches (typical) or more, as long as the worst noise sources (such as CRTs, clock generators, memory buses, and ISA/PCI buses) are avoided.
- Do not route the DXP and DXN lines next to high inductance signals. Do not route the traces across a fast memory bus, which can easily introduce 30˚C error, even with good filtering. Otherwise, most noise sources are fairly benign.
- Route the DXP and DXN traces in parallel, and in close proximity to each other, away from any high-voltage traces such as +12VDC. Beware of leakage currents from PC board contamination; e.g., a 20 MΩ leakage path from DXP to ground causes about 1˚C error.
- Connect guard traces to GND on either side of the DXP and DXN traces (Figure 9). With guard traces in place, routing near high-voltage traces is not a problem. Provide a GND plane under the traces, if possible.
- Route traces to remote diodes through as few vias and cross-unders as possible, to minimize copper/solder thermocouple effects.
- When introducing a thermocouple in traces to remote diodes, make sure that both the DXP and the DXN paths have matching thermocouples. In general, PC board-induced thermocouples are not a serious problem. A copper-solder thermocouple generates 3V/˚C, and it takes about 200V of voltage error at DXP and DXN to cause a 1˚C measurement error. Thus most parasitic thermocouple errors are negligible.
- Use wide traces for routing to remote diodes. Narrow traces are more inductive, and tend to pick up radiated noise. The 10 mil widths and spacings recommended in Figure 9 are not absolutely necessary (as they offer only a minor improvement in leakage and noise), but try to use them where practical.
- For remote sensor distances longer than 8 in., or in particularly noisy environments, a twisted pair is recommended. Its practical length is 6 to 12 feet (typical) before noise becomes a problem, as tested in a noisy electronics laboratory. For longer distances, the best solution is a shielded twisted pair, like that used for audio microphones. Connect the twisted pair to DXP and DXN, and the shield to GND, and leave the shield's remote end unterminated.
- Excess capacitance at DXP or DXN limits practical remote sensor distances. For very long cable runs, the cable's parasitic capacitance often provides noise filtering, and the 2200 pF capacitor can be removed, or reduced in value. Cable resistance also affects remote sensor accuracy; a 1Ω series resistance introduces about 1/2˚C error.

General Guidelines for TMS and VLM

- Supply analog AV_{DD} though an external RC or LC filter.
- Analog AV_{DD}. Place a 0.1 µF capacitor, and a 10-47 µF tantalum capacitor, on the AV_{DD} pins, as close as possible to the pin.
- A separate, low-impedance, ground plane for analog ground is recommended. This plane provides the ground point for the voltage dividers. Although such a plane provides the best performance, it is not mandatory, and systems with a sufficiently low noise level can be designed without it. Figure 10 shows an example plane layout.
- For thermistor-based temperature measurements, the voltage dividers should be physically located as close as possible. This may be problematic, but every effort should be made to reduce the distance between the thermistor and the PC87366.

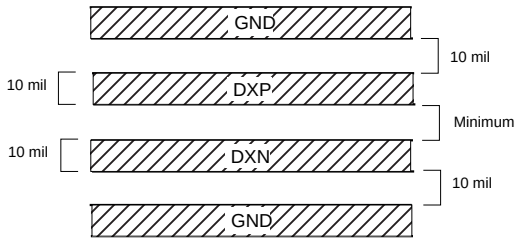

Figure 9. DXP/DXN PC Traces

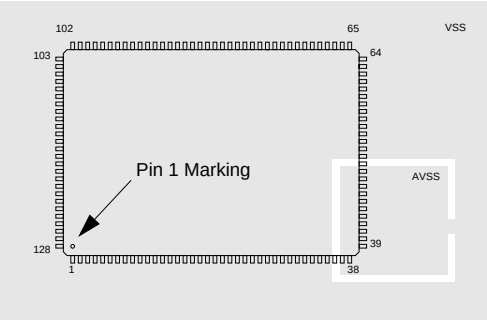

Figure 10. V_{SS} Plane Layout Example

National Semiconductor Corporation
Americas
Tel: 1-800-272-9959
Fax: 1-800-737-7018
Email: support@nsc.com

National Semiconductor Europe
Fax: +49 (0) 1 80-530 85 86
Email: europe.support@nsc.com
Deutsch Tel: +49 (0) 69 9508 6208
English Tel: +44 (0) 870 24 0 2171
Français Tel: +33 (0) 1 41 91 8790

National Semiconductor Asia Pacific Response Group
Tel: 65-2544466
Fax: 65-2504466
Email: sea.support@nsc.com

National Semiconductor Japan Ltd.
Tel: 81-3-5639-7560
Fax: 81-3-5639-7507

www.national.com