SMBus communication with TSM

Application Note

1. Introduction

This application note introduces hardware and software designers to SMBus communication protocol and how it can be used to communicate with all versions of the Temperature Sensor Module (TSM) infrared thermometers. The TSM infrared thermometer is the low cost solution for non-contact temperature measurements. The sensor is integrated with signal conditioning that reliably produces a calibrated temperature output in a plug-and-play package having many options. Programmable outputs and flexible power requirements yield a turnkey solution for consumer, commercial, industrial and medical applications.

The operation of the TSM is controlled by an internal state machine, which controls the measurements and calculations of the object and ambient temperatures. Calibrated temperature measurements are then available through the SMBus compatible interface.

The processor supports 2 IR sensors (second one not implemented in the single zone units). The output of the IR sensors is amplified by a low noise low offset chopper amplifier with programmable gain, converted by a Sigma Delta analog to digital modulator to a single bit stream and fed to a powerful DSP for further processing. The signal is conditioned by programmable (by means of EEPROM) FIR and IIR low pass filters for noise reduction of the input signal to achieve the desired performance and refresh rate. The output of the IIR filter is the measurement result and is available in the internal RAM. 3 different cells are available: one for the on-board temperature sensor (on chip PTAT or PTC) and 2 for the IR sensors.

Based on results of the above measurements, the corresponding ambient temperature Tambient and object temperatures Tobject are calculated. Both calculated temperatures have an internal resolution of 0.01 ℃ ((-273.15℃ to 328.19℃) / 16 bit). The data for Ta and To can be read with a 0.02℃ resolution on the SMBus.

In 1995 Intel introduced the System Management Bus. It is used in personal computer for low-speed system management communication from motherboard to peripherals ICs. In February of 1995 the smart battery system appeared with SMBus to define the communication link between an intelligent battery, charger for the battery and a microcontroller that communicates with the rest of the system.

SMBus has also been used to connect a wide variety of devices including power-related devices, system sensors, inventory EEPROMs, communications devices and more. The original specification of the SMBus protocol can be found on http://www.smbus.org/specs/

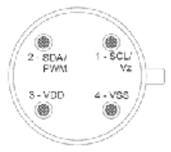


Figure 1.

Top View

Pin Name	Pin Description
1. SCL / Vz	Serial Clock (SCL) input for 2-wire communications protocol. For an external 8 -16V source to a MD-0003 or MD-005, a 5.7V zener (Vz) is integrated at this pin for connection of external bipolar transistor.
2. SDA / PWM	Serial Data (SDA) digital input / output. In SMBus compatible mode automatically configured as open drain NMOS. Optional mode: the measured object temperature is available at this pin Pulse Width Modulated (PWM).
3. VDD	External Voltage Supply. V_{DD}
4. VSS	Ground. V_{SS} The metal can is electrically connected to this pin.

SMBus communication with TSM

2. Table of Contents

3. Electrical Specifications

3.1 MD-0003 and MD-0005

All parameters are preliminary for T_A = 25 ℃, V_{DD} = 5 V (unless otherwise specified)

Parameter	Symbol	Test Conditions	Min	Typ	Max	Units
SMBus compatible 2-wire interface[2]						
Input High Voltage	V_{IH}(Ta, V)	Over temperature and supply	V_{DD} - 0.1			V
Input Low Voltage	V_{IL}(Ta, V)	Over temperature and supply			0.6	V
Output Low Voltage	V_{OL}	SDA pin in open drain mode, over temperature and supply, Isink = 2 mA			0.2	V
SCL Leakage	I_{SCL}, leak	V_{SCL} = 4 V, Ta = +85℃			30	µA
SDA Leakage	I_{SDA}, leak	V_{SDA} = 4 V, Ta = +85℃			0.3	µA
SCL Capacitance	C_{SCL}				10	pF
SDA Capacitance	C_{SDA}				10	pF
Slave Address	SA	Factory default		5A		hex
Wake up Request	t_{wake}	SDA low	33			ms
SMBus Request	t_{REQ}	SCL low	1.44			ms
Timeout, Low	$T_{imeout, L}$	SCL low	27		33	ms
Timeout, High	$T_{imeout, H}$	SCL high	45		55	µs
Acknowledge Setup Time	Tsuac(MD)	8-th SCL falling edge, Master	0.5		1.5	µs
Acknowledge Hold Time	Thdac(MD)	9-th SCL falling edge, Master	1.5		2.5	µs
Acknowledge Setup Time	Tsuac(SD)	8-th SCL falling edge, Slave	2.5			µs
Acknowledge Hold Time	Thdac(SD)	9-th SCL falling edge, Slave	1.5			µs
EEPROM						
Data Retention		Ta = +85℃	10			years
Erase / Write Cycles		Ta = +25℃	100,000			times
Erase / Write Cycles		Ta = +85℃	10,000			times
Erase Cell Time	Terase			5		ms
Write Cell Time	Twrite			5		ms

Notes: SMBus and refresh rate timings are given for the nominal calibrate HFO frequency and will vary with this frequency.

1. Power-up factory default is SMBus.
2. Maximum number of devices on one bus is 100; each device must have a unique slave address. Higher pull-up currents are recommended for higher number of devices, faster bus data transfer rates, and increased reactive loading of the bus.

   All voltages are referenced to VSS (ground) unless otherwise noted.
   Power savings mode is not available on the 5 Volt versions. (MD-0003, MD-0005)

3.2 MD-0006, MD-0007

All parameters are preliminary for T_A = 25 ℃, V_{DD} = 3 V (unless otherwise specified)

Parameter	Symbol	Test Conditions	Min	Typ	Max	Units
SMBus compatible 2-wire interface[2]						
Input High Voltage	V_{IH}(Ta, V)	Over temperature and supply	V_{DD} - 0.1			V
Input Low Voltage	V_{IL}(Ta, V)	Over temperature and supply			0.6	V
Output Low Voltage	V_{OL}	SDA pin in open drain mode, over temperature and supply, Isink = 2mA			0.25	V
SCL Leakage	I_{SCL}, leak	V_{SCL} = 3 V, Ta = +85℃			20	µA
SDA Leakage	I_{SDA} , leak	V_{SDA} = 3 V, Ta = +85℃			0.25	µA
SCL Capacitance	C_{SCL}				10	pF
SDA Capacitance	C_{SDA}				10	pF
Slave Address	SA	Factory default		5A		hex
Wake up Request	t_{wake}	SDA low	33			ms
SMBus Request	t_{REQ}	SCL low	1.44			ms
Timeout, Low	$T_{imeout, L}$	SCL low	27		33	ms
Timeout, High	$T_{imeout, H}$	SCL high	45		55	µs
Acknowledge Setup	Tsuac(MD)	8-th SCL falling edge, Master	0.5		1.5	µs
Acknowledge Hold	Thdac(MD)	9-th SCL falling edge, Master	1.5		2.5	µs
Acknowledge Setup	Tsuac(SD)	8-th SCL falling edge, Slave	2.5			µs
Acknowledge Hold	Thdac(SD)	9-th SCL falling edge, Slave	1.5			µs
EEPROM						
Data Retention		Ta = +85℃	10			years
Erase / Write Cycles		Ta = +25℃	100K			times
Erase / Write Cycles		Ta = +85℃	10K			times
Erase Cell Time	Terase			5		ms
Write Cell Time	Twrite			5		ms

Notes: SMBus and refresh rate timings are given for the nominal calibrate HFO frequency and will vary with this frequency.

1. Power-up factory default is SMBus.
2. Maximum number of devices on one bus is 100; each device must have a unique slave address. Higher pull-up currents are recommended for higher number of devices, faster bus data transfer rates, and increased reactive loading of the bus

   All voltages are referenced to V_{SS} (ground) unless otherwise noted.

4. SMBus Protocol Description

4.1 Glossary of Terms

ACK	Acknowledgement from receiver
Address Resolution Protocol	A protocol by which SMBus devices with assignable addresses on the bus are enumerated and assigned non-conflicting slave addresses.
ASSP	Application Specific Standard Product
Bus Master	Any device that initiates SMBus transactions and drives the clock.
Bus Slave	Target of a SMBus transaction which is driven by some master.
LSb	The Least Significant bit
Master-receiver	A bus master in a SMBus transaction while it is receiving data from a bus slave during a SMBus transaction.
Master-transmitter	A bus master in a SMBus transaction while it is transmitting data onto the bus during a SMBus transaction.
MSb	The Most Significant bit
NACK	Not Acknowledgement from receiver
OD	Open Drain
PEC	Packet Error Code
PP	Push Pull
Repeated Start	A repeated START is a START condition on the SMBus used to switch from write mode to read mode in a combined format protocol (e.g. Byte Read). The repeated START always follows an Acknowledge, and it always indicates that an address phase is beginning.
Slave-receiver	A Slave-receiver is a device that acts as a bus slave in a SMBus transaction while it is receiving address, command or other data from a device acting as a bus master in the transaction.
Slave-transmitter	A Slave-transmitter is a device acting as a bus slave in a SMBus transaction while it is transmitting data on the bus in response to a bus master's request.
TBD	To Be Defined

Note: *All versions of the Temperature sensor module are sometimes referred to as "the module" or just TSM.*

4.2 Overview

Only two bus lines are required to communicate to the TSM; a serial data line (SDA) and a serial clock line (SCL). Each TSM connected to the bus is software addressable by a unique address and a simple master/slave relationships exist at all times. Masters can operate as master-transmitters or as master-receivers. It’s a true multi-master bus including collision detection and arbitration to prevent data corruption if two or more masters simultaneously initiate data transfer. Serial, 8-bit oriented, bi-directional data transfers can be made up to 100 Kbit/s. It is based on the principles of operation of I2C protocol. Multiple devices, both bus masters and bus slaves, may be connected to a SMBus segment. Generally, a bus master device initiates a bus transfer between it and a single bus slave and provides the clock signals.

Only one device may master the bus at any time. Since more than one device may attempt to take control of the bus as a master, the SMBus protocol provides an arbitration mechanism that relies on the wired-AND connection of all SMBus device interfaces to the SMBus.

Devices may be powered by the bus VDD or by another power source Vbus (Figure.2).

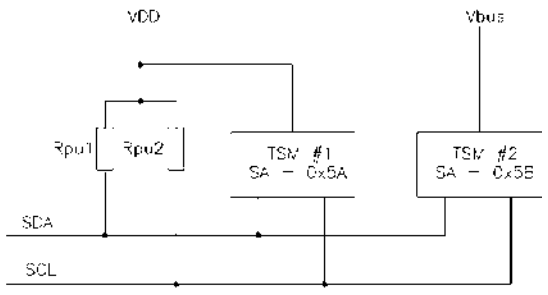


Figure 2 SMBus Topology

VDD may be 3 to 5 volts +/- 10% and there may be SMBus devices powered directly by the bus VDD. Both SDA and SCL lines are bi-directional, connected to a positive supply voltage through a pull-up resistor or a current source circuit. When the bus is free, both lines are high. The output stages of the devices connected to the bus must have an open drain or open collector in order to perform the wired-AND function. SMBus standard recommends for both the input and output stages of SMBus devices, not to load the bus when their power plane is turned off, i.e. powered-down devices should provide no leakage path to ground. A device that wants to place a ‘zero’ on the bus must drive the bus line to the defined logic low voltage level. In order to place a logic ‘one’ on the bus the device should release the bus line letting it be pulled high by the bus pull-up circuitry. The bus lines may be pulled high by a pull-up resistor or by a current source. In case this involves a higher bus capacitance, a more sophisticated circuit may be used that can limit the pull-down sink current while also providing enough current during the low - to high transition to maintain the rise time specifications of the SMBus.

In SMBus systems with higher bus capacitance (like wires) :
RPU = 1.5 K Ω (VDD = 5V, I_{PULLUP} = 3.3 mA) is suitable otherwise
RPU = 22 K Ω (VDD = 5V, I_{PULLUP} = 227 µA) can be used to meet SMBus low power DC specification (see low and high power DC specification below).

4.3 PEC

Version 1.1 of the SMBus specification introduced a Packet Error Checking mechanism to improve reliability and communication robustness. Implementation of Packet Error Checking by SMBus devices is optional for SMBus devices. Packet Error Checking, whenever applicable, is implemented by appending a Packet Error Code (PEC) at the end of each message transfer.

The PEC uses an 8-bit cyclic redundancy check (CRC-8) of each read or write bus transaction to calculate a Packet Error Code (PEC). The PEC may be calculated in any way that conforms to a CRC-8 represented by the polynomial, C(x) = x8 + x2 + x1 + 1 and must be calculated in the order of the bits as received. The PEC calculation includes all bytes in the transmission, including address, command and data. The PEC calculation does not include ACK, NACK, START, STOP nor Repeated START bits. This means that the PEC is computed over the entire message from the first START condition.

For the CRC calculation we use the following procedure:
In the case of the SMBus, the polynomial used is X8 + X2 + X + 1. The width of this polynomial is 8 (the highest power of X indicates the width) and it can be represented as 1 0000 0111. Since the width of the polynomial is 8 we refer to our CRC method as CRC-8. A message is represented as a bit-stream augmented with M = 8 zeroes at the end. The augmented bit-stream message is devised by the polynomial 1 0000 0111. The remainder will be the CRC-8 check byte.

To implement software code for calculating the CRC-8 directly, a simple procedure can be established and programmed accordingly:

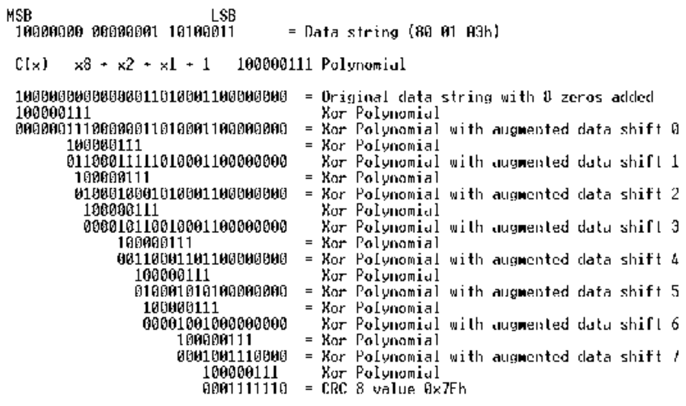
1. Concatenate the end of the data string (LSB) with eight zeros.
2. Perform the XOR operation on the data string against the binary version of the polynomial.
   a. First, shift the data string until a "1" appears at the MSB of the register.
   b. Line up the first "1" of the polynomial (1 X 8 part) so that it will logically operate against the first "1" of the data string when the XOR is performed.
   c. Perform the logical XOR of the 8 bits. There are actually 9 bits being operated on, but the first "1" bit will always result in a "0." This "0" is in the MSB place, and thus does not contribute to the magnitude of the final CRC value.
3. The result of the XOR operation should then be augmented by the "untouched" bits (those bits in the data string that are nine places to the right of the first "1" in the string). This augmented result is now saved in place of the data string.
4. Continue the process of shifting and XORing (from step 2) until the LSB of the polynomial does not line up with any of the added eight zeros (the end of the data string with eight zeros added has been shifted sufficiently so that all bits have been operated on). The result will be a completed CRC-8.

There are several ways to implement the CRC calculation used for PEC. The CRC can be calculated, using shifts and XOR operations, or a lookup table can be utilized. A full lookup table takes up 256 bytes of flash memory, but the CRC algorithm is executed in only a few clock cycles. The calculation method has very low memory requirements, but executes much slower and has a larger implementation than the lookup method.

For more information about calculation of CRC refer to the next document:
http://www.sbs-forum.org/marcom/dc2/20_crc-8_firmware_implementations.pdf
http://www.smbus.org/faq/faq.htm

SMBus communication with TSM

Calculation Method



```
MSB                      LSB
 10000000 00000001 10100011       = Data string (80 01 A3h)

 C(x)   x8 + x2 + x1 + 1   100000111 Polynomial

 10000000000000001101000110000000  = Original data string with 8 zeros added
 100000111                           Xor Polynomial
 00000011100000001101000110000000  = Xor Polynomial with augmented data shift 0
       100000111                   = Xor Polynomial
       01100011111010001100000000    Xor Polynomial with augmented data shift 1
        100000111                  = Xor Polynomial
        0100010001010001100000000  = Xor Polynomial with augmented data shift 2
         100000111                   Xor Polynomial
         000010110010001100000000    Xor Polynomial with augmented data shift 3
             100000111             = Xor Polynomial
             00110001101100000000  = Xor Polynomial with augmented data shift 4
               100000111             Xor Polynomial
               010001010100000000  = Xor Polynomial with augmented data shift 5
                100000111          = Xor Polynomial
                00001001000000000    Xor Polynomial with augmented data shift 6
                    100000111      = Xor Polynomial
                    0001001110000  = Xor Polynomial with augmented data shift 7
                       100000111     Xor Polynomial
                       000111110   = CRC 8 value 0x7Fh
```

4.4 *Electrical characteristics*

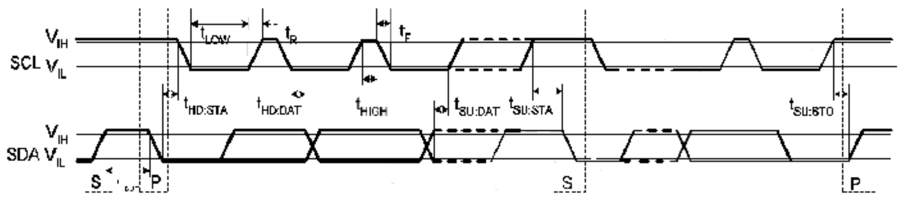


Figure 3 SMBus timing measurements

Symbol	Parameter	Min	Max	Units	Notes
f_{SMB}	SMBus Operation frequency	10	100	kHz	See note 1
t_{BUF}	Bus free time between Stop and Start condition	4.7	-	µs	
$t_{HD:STA}$	Hold time after(Repeated)Start Condition. After this period, the first clock is generated	4.0	-	µs	
$t_{SU:STA}$	Repeated Start Condition setup time	4.7	-	µs	
$t_{SU:STO}$	Stop Condition setup time	4.0	-	µs	See note 7
$t_{HD:DAT}$	Data hold time	300	-	ns	
$t_{SU:DAT}$	Data setup time	250	-	ns	
$t_{TIMEOUT}$	Detect clock low timeout	25	35	Ms	See note 2
t_{LOW}	Clock low period	4.7	-	Ms	
t_{HIGH}	Clock high period	4.0	50	Ms	See note 3
$t_{LOW:SEXT}$	Cumulative clock low extend time (slave device)	-	25	Ms	See note 4
$t_{LOW:MEXT}$	Cumulative clock low extend time (master device)	-	10	Ms	See note 5
t_F	Clock/Data Fall time	-	300	Ns	See note 6
t_R	Clock/Data Rise Time	-	1000	Ns	See note 6
T_{POR}	Time in which a device must be Operational after power-on reset	-	500	Ms	

Table 1

SMBus communication with TSM

Note 1: *A master shall not drive the clock at a frequency below the minimum f_{SMB}. Further, the operating clock frequency shall not be reduced below the minimum value of f_{SMB} due to periodic clock extending by slave devices This limit does not apply to the bus idle condition, and this limit is independent from the $t_{LOW: SEXT}$ and $t_{LOW: MEXT}$ limits. For example, if the SCL is high for $t_{HIGH,MAX}$, the clock must not be periodically stretched longer than $1/f_{SMB,MIN} - f_{HIGH,MAX}$. This requirement does not pertain to a device that extends the SCL low for data processing of a received byte, data buffering and so forth for longer than 100us in a non-periodic way.*

Note 2: *Devices participating in a transfer can abort the transfer in progress and release the bus when any single clock low interval exceeds the value of $t_{TIMEOUT,MIN}$. After the master in a transaction detects this condition, it must generate a stop condition within or after the current data byte in the transfer process. Devices that have detected this condition must reset their communication and be able to receive a new START condition no later than $t_{TIMEOUT,MAX}$. Typical device examples include the host controller, and embedded controller and most devices that can master the SMBus. Some simple devices do not contain a clock low drive circuit; this simple kind of device typically may reset its communications port after a start or a stop condition. A timeout condition can only be ensured if the device that is forcing the timeout holds the SCL low for $t_{TIMEOUT,MAX}$ or longer.*

Note 3: *$t_{HIGH,MAX}$ provides a simple guaranteed method for masters to detect bus idle conditions. A master can assume that the bus is free if it detects that the clock and data signals have been high for greater than $t_{HIGH,MAX}$.*

Note 4: *$t_{LOW:SEXT}$ is the cumulative time a given slave device is allowed to extend the clock cycles in one message from the initial START to the STOP. It is possible that, another slave device or the master will also extend the clock causing the combined clock low extend time to be greater than $t_{LOW:SEXT}$. Therefore, this parameter is measured with the slave device as the sole target of a full-speed master.*

Note 5: *$t_{LOW:MEXT}$ is the cumulative time a master device is allowed to extend its clock cycles within each byte of a message as defined from START-to-ACK, ACK-to-ACK, or ACK-to-STOP. It is possible that a slave device or another master will also extend the clock causing the combined clock low time to be greater than $t_{LOW:MEXT}$ on a given byte. Therefore, this parameter is measured with a full speed slave device as the sole target of the master.*

Note 6: *Rise and fall time is defined as follows:*

$t_R = (V_{IL,MAX} - 0.15)$ to $(V_{IH,MIN} + 0.15)$

$t_F = (V_{IH,MIN} + 0.15)$ to $(V_{IL,MAX} - 0.15)$

Note 7: *For the first silicon revision of a TSM module this value is above 500ns.*

4.5 Timeouts

Timeout measurement intervals illustrates the definition of the timeout intervals, $t_{LOW:SEXT}$ and $t_{LOW:MEXT}$.

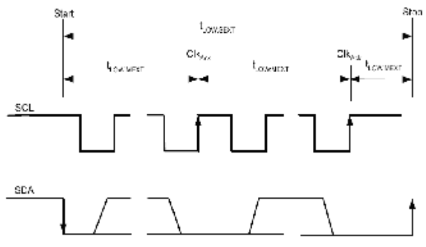


Figure 4 Timeout measurement intervals

SMBus communication with TSM

4.6 Slave device timeout definitions and conditions

The $t_{TIMEOUT,MIN}$ parameter allows a master or slave to conclude that a defective device is holding the clock low indefinitely or a master is intentionally trying to drive devices off the bus. It is highly recommended that a slave device release the bus (stop driving the bus and let SCL and SDA float high) when it detects any single clock held low longer than $_{tTIMEOUT,MIN}$. Devices that have detected this condition must reset their communication and be able to receive a new START condition in no later than $_{tTIMEOUT,MAX}$. Slave devices that violate $_{tLOW:SEXT}$ are not conformant with this specification. A Master is allowed to abort the transaction in progress to any slave that violates the $t_{LOW:SEXT}$ or $t_{TIMEOUT,MIN}$ specifications.

4.7 Master device timeout definitions and conditions

$t_{LOW:\ MEXT}$ is defined as the cumulative time a master device is allowed to extend its clock cycles within one byte in a message as measured from:

START to ACK
ACK to ACK
ACK to STOP

A system host may not violate $t_{LOW:MEXT}$ except when caused by the combination of its clock extension with the clock extension from a slave device or another master. A Master is allowed to abort the transaction in progress to any slave that violates the $t_{LOW:SEXT}$ or $t_{TIMEOUT,MIN}$ specifications. This can be accomplished by the Master issuing a STOP condition at the conclusion of the byte transfer in progress.

Note: ***A Master should take care when evaluating $t_{LOW:SEXT}$ violation during arbitration since the clock may be held low by multiple slave devices simultaneously. The arbitration interval may be extended for several bytes in the case of devices that respond to commands to the SMBus ARP address. If timeouts are handled at the driver level, the software may need to allow timeouts to be configured or disabled by applications that use the driver in order to support older devices that do not fully meet the SMBus timeout specifications. Devices that implement 'shared' slave addresses may also violate this specification due to combined clock stretching by the different devices sharing the address. $T_{TIMEOUT,MIN}$, however, does not increase due to combined clock stretching. Therefore, this is a safer timeout parameter for a Master to use when it knows it's accessing SMBus 2.0 devices.***

4.8 Low-power DC specifications

In the table bellow are given low power DC parameters of the SMBus specification.

Symbol	Parameter	Min	Max	Units	Notes
V_{IL}	Data, Clock Input Low Voltage	-	0.8	V	
V_{IH}	Data, Clock Input High Voltage	2.1	VDD	V	
V_{OL}	Data, Clock Output Low Voltage	-	0.4	V	
I_{LEAK}	Input Leakage	-	±5	µA	Note 1
I_{PULLUP}	Current trough pull-up resistor or current source	100	350	µA	Note 2
V_{DD}	Nominal bus voltage	2.7	5.5	V	3V to 5V ±10%

Note 1: ***Devices must meet this specification whether powered or un-powered. However, a microcontroller acting as a SMBus host may exceed ILEAK by no more than 10 µA. Note 2: The $I_{PULLUP,MAX}$ specification is determined primarily by the need to accommodate a maximum of 1.1K equivalent series resistor of removable SMBus devices, such as the Smart Battery, while maintaining the $V_{OL,MAX}$ of the bus.***

Table 2

Because of the relatively low pull-up current, the system designer must ensure that the loading on the bus remains within acceptable limits. Additionally, to prevent bus loading, any devices that remain connected to the active bus while un-powered (that is, their Vcc lowered to zero), must also meet the leakage current specification.

4.9 High-power DC specifications

High-power SMBus is specified below. These higher power specifications provide the robustness necessary, for example, to allow SMBus to cross the PCI connector, thus allowing SMBus devices on PCI add-in cards to communicate with other devices on both the system board and other PCI add-in cards in the same system. These higher power electrical specifications are an alternative to the lower power specifications stated above and may be used in environments where necessary.

Symbol	Parameter	Min	Max	Units	Notes
V_{IL}	SMBus signal Input low voltage	-	0.8	V	
V_{IH}	SMBus signal Input high voltage	2.1	VDD	V	
V_{OL}	SMBus signal Output low voltage	-	0.4	V	@ I_{PULLUP}
$I_{LEAK-BUS}$	Input Leakage per bus segment		±200	µA	
$I_{LEAK-PIN}$	Input Leakage per device pin		±10	µA	
V_{DD}	Nominal bus voltage	2.7	5.5	V	3V to 5V ±10%
I_{PULLUP}	Current sinking, VOL = 0.4V	4		mA	
C_{BUS}	Capacitive load per bus segment		400	pF	Note 1
C_I	Capacitance for SDA or SCL pin		10	pF	Note 2
V_{NOISE}	Signal noise immunity from 10 MHz to 100 MHz	300	-	mVp-p	This AC item applies To the high-power DC Specification only

Note 1: ***Capacitive load for each bus line includes all pin, wire and connector capacitances. The maximum capacitive load affects the selection of the R_{PU} pull-up resistor or the current source in order to meet the rise time specifications of SMBus.***

Note 2: ***Pin capacitance (C_I) is defined as the total capacitive load of one SMBus device as seen in a typical manufacturer's data sheet.***

Table 3

While SMBus devices used in low-power segments have practically no minimum current sinking requirements due to the low pull-up current specified for low-power segments, devices in high power segments are required to sink a minimum current of 4 mA while maintaining the $V_{OL, MAX}$ of 0.4 Volts. The requirement for 4 mA sink current determines the minimum value of the pull-up resistor R_{PU} that can be used in SMBus systems. Un-powered devices connected to either a low-power or high-power SMBus segment must provide, either within the device or through the interface circuitry, protection against "back powering" the SMBus.

4.10 Bit transfer

In accordance to the SMBus specification, the MSb is transferred first. SMBus uses fixed voltage levels to define the logic "ZERO" and logic "ONE" on the bus respectively. The data on SDA must be stable during the "HIGH" period of the clock. Data can change state only when SCL is low. Each transfer begins with START bit and finishes with STOP bit (Figure 5).

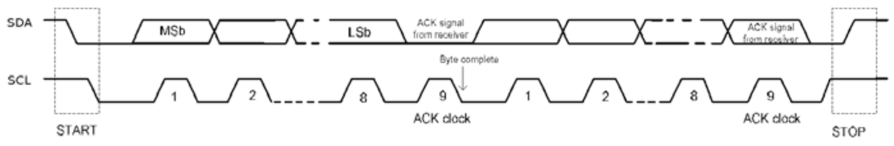


Figure 5 SMBus byte format

START bit is defined by HIGH to LOW transition of the SDA line while SCL is HIGH.
STOP bit is defined by LOW to HIGH transition of the SDA line while SCL is HIGH.

Every byte consists of 8 bits. Each byte transferred on the bus must be followed by an acknowledge bit. The acknowledge-related clock pulse is generated by the master (ACK clock). The transmitter, master or slave, releases the SDA line (HIGH) during the acknowledge clock cycle. In order to acknowledge a byte, the receiver must pull the SDA line LOW during the HIGH period of the clock pulse according to the SMBus timing specifications. A receiver that wishes to NACK a byte must let the SDA line remain HIGH during the acknowledge clock pulse. A SMBus device must always acknowledge (ACK) its own address.

A SMBus slave device may decide to NACK a byte other than the address byte in the following situations:

> The slave device is busy performing a real time task, or data requested are not available. The master upon detection of the NACK condition must generate a STOP condition to abort the transfer. Note that as an alternative, the slave device can extend the clock LOW period within the limits of the specifications in order to complete its tasks and continue the transfer.
> The slave device detects an invalid command or invalid data. In this case the slave device must NACK the received byte. The master upon detection of this condition must generate a STOP condition and retry the transaction. If a master-receiver is involved in the transaction it must signal the end of data to the slave-transmitter by generating an NACK on the last byte that was clocked out by the slave. The slave-transmitter must release the data line to allow the master to generate a STOP condition.

5. Comparing the I2C Bus to the SMBus

The I2C bus and the SMBus are popular 2-wire buses that are essentially compatible with each other. Normally devices, both masters and slaves, are freely interchangeable between both buses. Both buses feature addressable slaves (although specific address allocations can vary between the two).

The buses operate at the same speed, up to 100 kHz, but the I2C bus has both 400 kHz and 2 MHz versions. Complete compatibility between both buses is ensured only below 100 kHz. Here are explored the significant differences between I2C and SMB.

SMBus communication with TSM

5.1 Timeout and Clock Speed differences

Timeout and (as a consequence of timeout) minimum clock speed are the most important differences between the I2C bus and the SMBus.

I2C Bus = DC (no timeout)
SMBus = 10 kHz (> 35 mSec timeout)

Timeout is where a slave device resets its interface whenever SCL goes low for longer than the timeout, typically 35 mSec. Use of a timeout also dictates a minimum speed for the clock, because it can never go static. Thus, the SMBus has a minimum-clock-speed specification. By comparison, the I2C bus can go static indefinitely. In the I2C bus, either a master or a slave can hold the clock low as long as necessary to process data. In the I2C bus, if the slave locks up and holds either SCL or SDA low, error recovery is impossible. Very few slave devices actually have the ability to hold SCL. As a result, the most common bus error is slave devices that have ended up in a state where SDA is low. In the I2C bus, a master accomplishes error recovery by clocking SCL until SDA is high and then issuing a Start followed by a Stop. In contrast to the I2C bus, SMBus slaves are expected to reset their interface whenever SCL is low for longer than the timeout specified in the SMBus specification of 35 mSec. SMBus specifies $t_{LOW:\,SEXT}$ as the cumulative clock low extend time for a slave device. I2C does not have a similar specification. SMBus specifies $t_{LOW:\,MEXT}$ as the cumulative clock low extend time for a master device. Again I2C does not have a similar specification

5.2 DC specifications differences

Both I2C and SMBus are capable of operating with mixed devices that have either fixed input levels (such as Smart Batteries) or input levels related to VDD. When mixing devices, the I2C specification defines the VDD to be 5.0 Volt +/- 10% and the fixed input levels to be 1.5 and 3.0 Volts. Instead of relating the bus input levels to VDD, SMBus defines them to be fixed at 0.8 and 2.1 Volts. This SMBus specification allows for bus implementations with VDD ranging from 3 to 5 Volts +/- 10%. I2C specifies the maximum leakage current to be 10 µA while SMBus version 1.0 specified maximum leakage current of 1 uA. Version 1.1 of the SMBus specification relaxes the leakage requirements to 5 µA, in order to reduce the cost of testing of SMBus devices.

While I2C defines maximum bus capacitance 400 pF SMBus does not specify a maximum bus capacitance. Instead it specifies the I_{PULLUP} maximum of 350 µA in Low-power DC specification and minimum 4 mA in High-power DC specification. Bus capacitance can be calculated taking into consideration the maximum rise time and I_{PULLUP}.

High	I2C VDD Dependent	0.7 * V_{DD}
	I2C Fixed	3.0V
	SMBus	2.1V
Low	I2C VDD Dependent	0.3 * V_{DD}
	I2C Fixed	1.5V
	SMBus	0.8V

5.3 Other differences

ACK and **NACK** usage:

The differences in the use of the NACK bus signaling follow:

> In I2C, a slave receiver is allowed not to acknowledge the slave address, if for example is unable to receive because it's performing some real time task. SMBus requires devices to acknowledge their own address always, as a mechanism to detect a removable device's presence on the bus (battery, docking station, etc.). I2C specifies that a slave device, although it may acknowledge its own address, some time later in the transfer it may decide that it cannot receive any more data bytes. The I2C specifies, that the device may indicate this by generating the not acknowledge on the first byte to follow. Besides to indicate a slave device busy condition, SMBus is using the NACK mechanism also to indicate the reception of an invalid command or data. Since such a condition may occur on the last byte of the transfer, it is required that SMBus devices have the ability to generate the not acknowledge after the transfer of each byte and before the completion of the transaction. This is important because SMBus does not provide any other resend signaling.

More information about the differences between I2C and SMBus can be found on:

http://www.maxim-ic.com/appnotes_frame.cfm/appnote_number/476
http://www.smbus.org/specs/

6. SMBus communications with the TSM

6.1 Overview

The TSM can only be used as a slave device. Generally, the master initiates the start of data transfer by selecting a slave through the Slave Address (SA). Each TSM on the SMBus must have a unique SA, which must be preprogrammed before insertion on the SMBus. Dexter has a TSM evaluation kit MD-0004 that can be used to preprogram new Slave Addresses into each TSM.

The TSM meets all the timing specifications of the SMBus (refer to Electrical characteristics of SMBus devices above). Information from the TSM is via its 32 x 17 RAM locations. It is not possible to write into the RAM memory. It can only be read and only a limited number of RAM registers are of interest to the customer (see Table 4 below). RAM readings format is described in more details below as well as in the TSM data sheet. 32 x 16 EEPROM is available for keeping the calibration data, chip configuration and chip ID. Entire EEPROM can be read via the SMBus compatible interface. Some EEPROM locations are write protected (access is possible with entry to Calibration mode). Before writing to EEPROM an erase has to take place. Erase is simply writing zero into EEPROM. Erase operations have the same access constrains as the write operations.

Note that changes in EEPROM will result in reconfiguration of the TSM after POR (also includes enter and exit Sleep mode).

For example, Slave Address can be changed in EEPROM, but the TSM will respond to the old SA until POR is exited. If the TSM is configured in PWM output mode, a SMBus request condition is needed. SMBus request overrides the OD / PP bit that configures the SDA / PWM pin into Open Drain NMOS or Push-Pull CMOS. For example, a TSM configured for PP PWM will switch to OD SMBus upon SMBus request condition. The diagram below illustrates the way of switching to SMBus if PWM is enabled. PWM output can be the POR default if configured in EEPROM.

SMBus communication with TSM

The diagram below shows the way of switching to SMBus if PWM is enabled (factory programmed POR default for TSM is SMBus, PWM disabled). Note that the SCL pin needs to be kept high in order to use PWM.

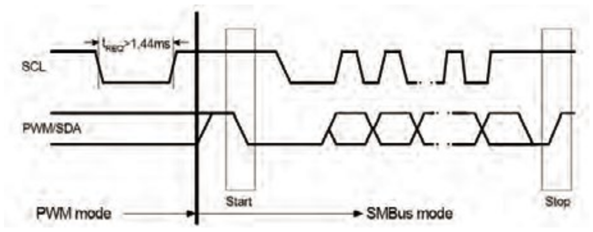


Figure 6 Switching from PWM mode to SMBus mode.

The TSM's SMBus request condition requires forcing LOW the SCL pad for period longer than the request time t_{REQ}. The Data line value is ignored in this case. Once disabled PWM, it can be only enabled by switching Off-On of the supply or exit from Sleep Mode.

6.2 Timing

The specific timings in TSM's SMBus are: SMBus Request (t_{REQ}) is the time that the SCL should be forced low in order to switch the TSM from PWM mode to SMBus mode; Tsuac (SD) is the time after the eighth falling edge of SCL that TSM will force PWM / SDA low to acknowledge the last received byte. Thdac (SD) is the time after the ninth falling edge of SCL that TSM will release the PWM / SDA so the MD could continue with the communication. Tsuac (MD) is the time after the eighth falling edge of SCL that TSM will release PWM / SDA so that the MD could acknowledge the last received byte. Thdac (MD) is the time after the ninth falling edge of SCL that TSM will take control over the PWM / SDA so the it could continue with the next byte to transmit. (The indexes MD and SD for the latest timings are used – MD when the master device is making acknowledge; SD when the slave device is making acknowledge).

For other timings see Electrical characteristics of SMBus devices above.

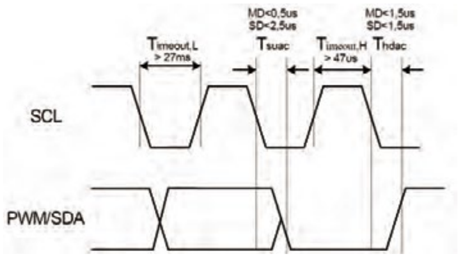


Figure 7 SMBus timing.

6.3 Detailed Communication description

RAM and EEPROM can be read both with 32 x 16 sizes. If the RAM is read, the data is divided by two, due to a sign bit in RAM (for example**,** Tobj1 - RAM address 0x07h will sweep between 0x27ADh to 0x7FFF as the object temperature rises from -70.01℃ to +382.19℃). The MSB read from RAM is an error flag (active high) for the linearized temperatures (Tobj1, Tobj2 and Ta). The MSB for the raw data (e.g. IR sensor1 data) is a sign bit (sign and magnitude format). A write of 0x0000 must be done prior to writing in EEPROM in order to erase the EEPROM cell content. Refer to EEPROM detailed description for factory calibration EEPROM locations that need to be kept unaltered.

Opcode	Command
000x xxxx*	RAM Access
001x xxxx*	EEPROM Access
1111_0000**	Read Flags
1111_1111	Enter Sleep Mode

Note: The xxxxx represent the 5 LSBits of the memory map address to be read / written.*
*Note**: Behaves like read command. The TSM returns PEC after 16 bits data of which only 4 are meaningful and if the MD wants it, it can stop the communication after the first byte.*
The difference between (RAM or EEPROM) read to Read Flags is that the latter does not have a repeated start bit. *Flags read are:*
Data[15] – EEBUSY – the previous write/erase EEPROM access is still in progress. High active.
Data[14] – unused
Data[13] - EE_DEAD – EEPROM double error has occurred. High active.
Data[12] – INIT – POR initialization routine is still ongoing. Low active.
Data[11] – not implemented.
Data[10..0] – all zeros.

Flags read is a diagnostic feature. The TSM can be used regardless of these flags.

RAM (32 x 16)		
Name	Opcode and Address	Read access
Factory reserved	0x00h	Yes
…	…	…
Ambient sensor data	0x03h	Yes
IR 1 sensor data	0x04h	Yes
IR 2 sensor data	0x05h	Yes
Ta calibrated data	0x06h	Yes
Tobj1 calibrated data	0x07h	Yes
Tobj2 calibrated data	0x08h	Yes
Factory reserved	0x09h	Yes
…	…	…
Factory reserved	0x1Fh	Yes

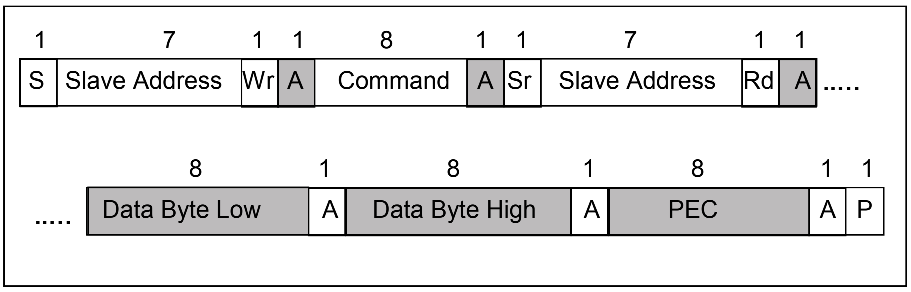
Table 4

EEPROM (32 x 16)		
Name	Opcode and Address	Write access
To_{max}	0x20h	Yes
To_{min}	0x21h	Yes
PWMCTRL	0x22h	Yes
Ta range	0x23h	Yes
Emissivity correction coefficient (Ke)	0x24h	Yes
Config Register1	0x25h	Yes
Factory reserved	0x26h	No
…	…	…
Factory reserved	0x2Dh	No
SMBus address	0x2Eh	Yes
Factory reserved	0x2Fh	Yes
Factory reserved	0x30h	No
…	…	…
Factory reserved	0x38h	No
Factory reserved	0x39h	Yes
Factory reserved	0x3Ah	No
Factory reserved	0x3Bh	No
ID number	0x3Ch	No
ID number	0x3Dh	No
ID number	0x3Eh	No
ID number	0x3Fh	No

Table 5

SMBus communication with TSM

All bytes are sent and received with MSb first.
The format of SMBus reading from RAM is:



Read Word (depending on command – RAM or EEPROM).

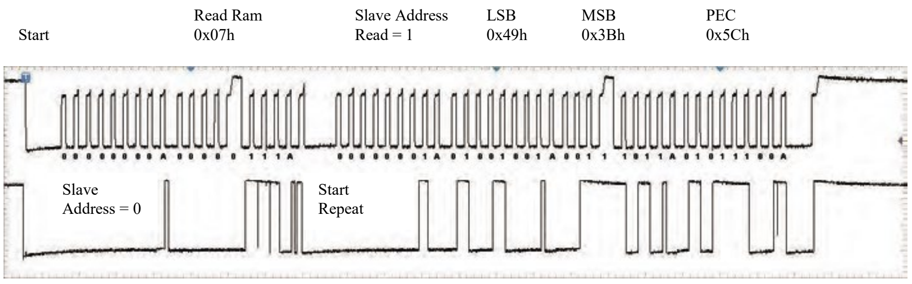


Figure 8 Read Tobj1.

Read RAM 0x07h, results = 0x3B49h, PEC = 0x5Ch.

RAM memory is read only via SMBus. The reading data are divided by two, due to a sign bit (Sn) in RAM (for example, TOBJ1 - RAM address 0x07h will sweep between 0x27ADh to 0x7FFF as the object temperature rises from -70.01°C to +382.19℃). The MSb read from RAM is an error flag (active high) for the linearized temperatures (T_{OBJ1}, T_{OBJ2} and Ta). The MSb for the raw data (e.g. IR sensor1 data) is a sign bit (sign and magnitude format).

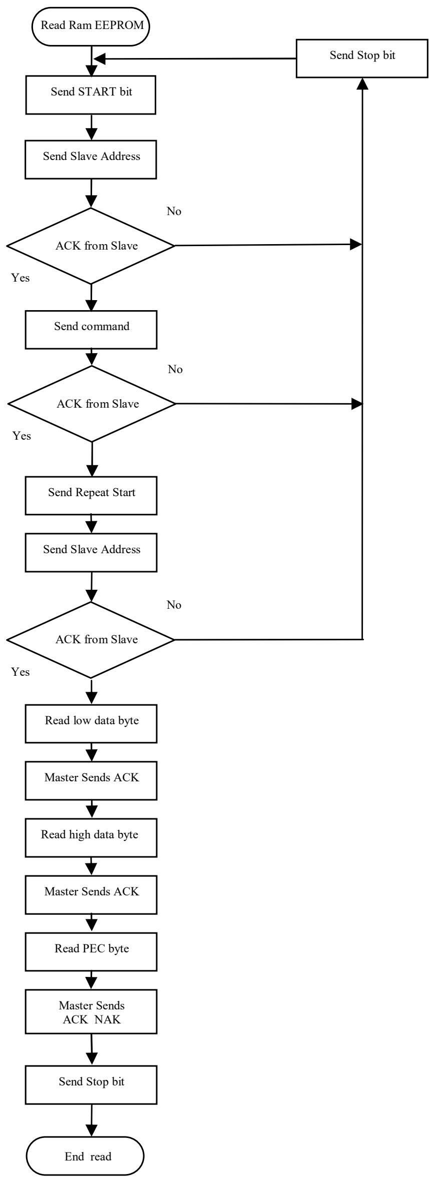
Pseudo code example: Reading RAM address 0x07 (Tobj1)

```
1. Send START bit
2. Send Slave Address (0x00* for example) + Rd\-Wr bit**
3. Send Command (0b000x_xxxx + 0b0000_0111 -> 0b0000_0111)
4. Send Repeated START bit
5. Send Slave Address +Rd \ -Wr bit**
6. Read Data Byte Low (master must send ACK bit)
7. Read Data Byte High (master must send ACK bit)
8. Read PEC (master can send ACK or NACK)
9. Send STOP bit
```

Note : Any TSM will respond to address 0x00 Note**: Bit Rd\-Wr has no meaning for TSM*

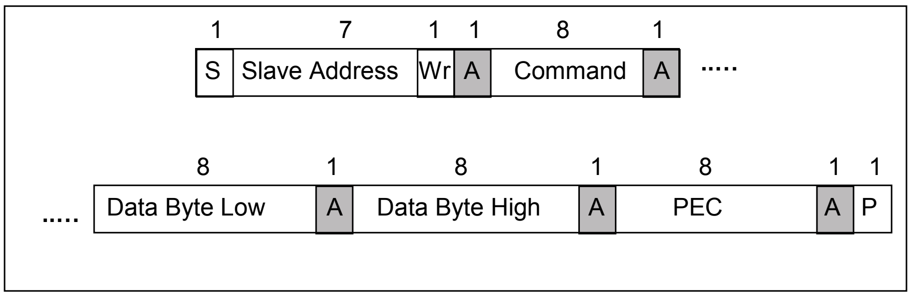
SMBus communication with TSM

Read RAM or EEPROM Block Diagram



SMBus communication with TSM

The format of SMBus writing in EEPROM is:



Write Word (depending on command – RAM or EEPROM).

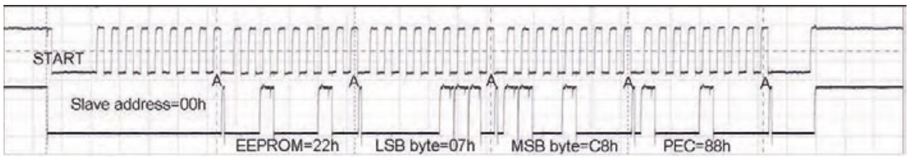


Figure 9 Write PWMCTRL after writing zero's to erase EEPROM cell.

Write 0xC807 to EEPROM address 0x22h, PEC = 0x88h.

In Application mode only 9 cells are accessible for writing. An attempt to write not accessible EEPROM cell results in no change. Before writing an erasing operation must be done. An erasing operation is just a writing of zeros in an EEPROM cell. After writing / erasing 5ms is need for the new value to be written / erased. After writing it is strongly recommended that the device is restarted by turning off / on the power supply or by putting the sensor in / out sleep.

```
Pseudo code example:
An Erasing of the EEPROM address 0x0E (SMBus Address)
1. Send START bit
2. Send Slave Address (0x00* for example) + Rd\-Wr bit**
3. Send Command (0b001x_xxxx + 0b0000_1110 -> 0b0010_1110)
4. Send Low data 0x00
5. Send High data 0x00
6. Send PEC 0x6F
7. Send STOP bit
8. Wait 5ms (this time is need the cell to be erased)

A writing of 0x5A in EEPROM address 0x0E (SMBus Address)

1. Send START bit
2. Send Slave Address (0x00* for example) + Rd\-Wr bit**
3. Send Command (0b001x_xxxx + 0b0000_1110 -> 0b0010_1110)
4. Send Low Byte 0x5A
5. Send High Byte 0x00 (the high byte of the EEPROM address 0x0E has no meaning)
6. Send PEC 0xE1
7. Send STOP bit
8. Wait 5ms (this time is need the cell to be written)
9. Turn off / Turn on module power supply to reset TSM (After this TSM will respond to the new slave
   address 0x5A)***
```

Germany & Other Countries
Laser Components GmbH
Tel: +49 8142 2864 – 0
Fax: +49 8142 2864 – 11
info@lasercomponents.com
www.lasercomponents.com

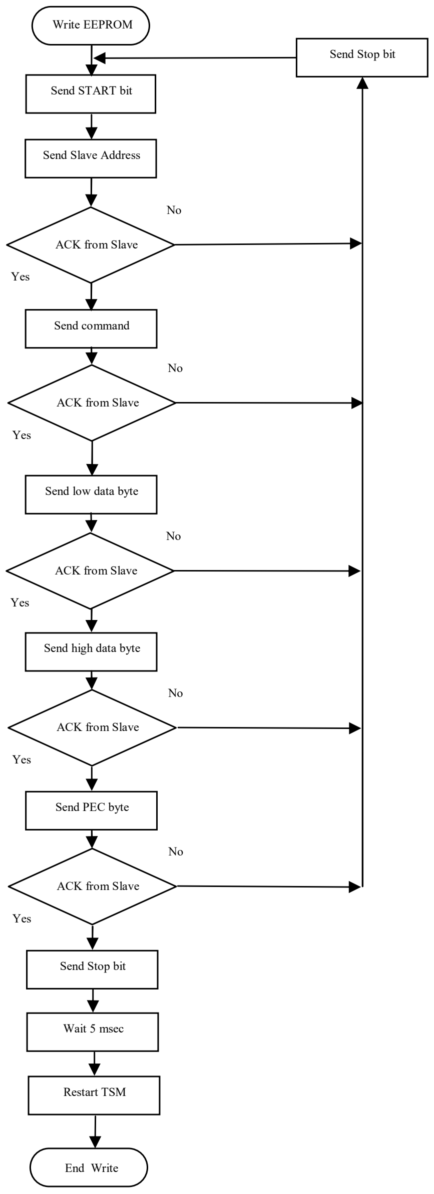
France
Laser Components S.A.S.
Tel: +33 1 39 59 52 25
Fax: +33 1 39 59 53 50
info@lasercomponents.fr
www.lasercomponents.fr

United Kingdom
Laser Components (UK) Ltd.
Tel: +44 1245 491 499
Fax: +44 1245 491 801
info@lasercomponents.co.uk
www.lasercomponents.co.uk

Nordic Countries
Laser Components Nordic AB
Tel: +46 31 703 71 73
Fax: +46 31 703 71 01
info@lasercomponents.se
www.lasercomponents.se

SMBus communication with TSM

Write / Erase EEPROM Block Diagram



7. Sleep Mode

The format of SMBus transaction which enters Sleep mode is:

1	7	1	1	8	1	8	1	1
S	Slave Address	Wr	A	Command	A	PEC	A	P

```
Pseudo code example: Put TSM in Sleep mode
1. Send START bit
2. Send Slave Address (0x00* for example) + Rd\-Wr bit**
3. Send command 0xFF
4. Send PEC byte 0xF3
5. Send STOP bit
6. Put the SCL line in low level
```

The TSM can enter in Sleep Mode via the command “Enter SLEEP mode” sent via the SMBus interface. This mode is not available for the 5 Volt supply version. To limit the current consumption to 2.5 µA (typical), the SCL pin should be kept low during sleep. TSM goes back into power-up default mode (via POR reset) by setting SCL pin high and then PWM / SDA pin low for at least t_{DDq} > 33 ms.

If EEPROM is configured for PWM (EN_PWM is high), the PWM interface will be selected after awakening and if PWM control [2], PPODB is 1 the TSM will output a PWM pulse train with push-pull output.

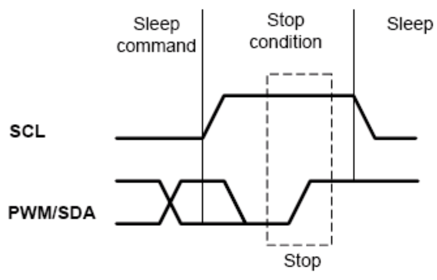


Figure 10 Entering Sleep Mode.

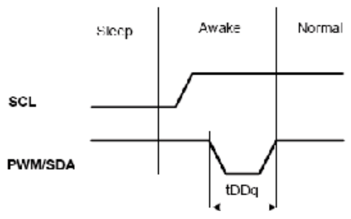


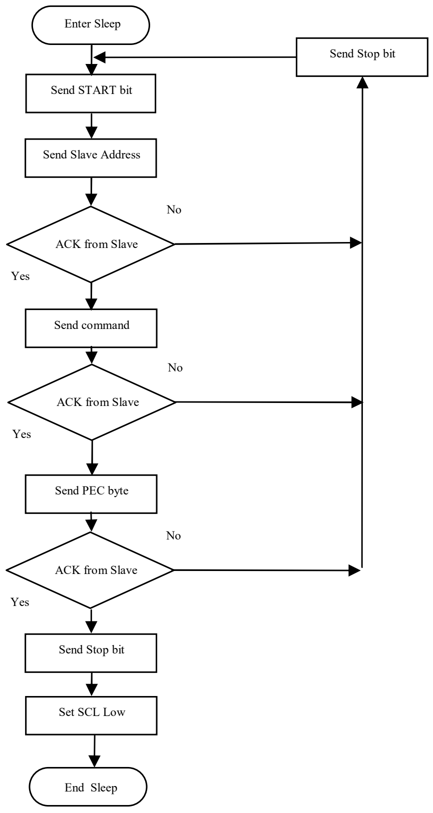
Figure 11 Exiting Sleep Mode.

After exit from Sleep Mode data is available after 0.25 seconds (typ). For the first measurement the IIR filter is skipped, all measurements afterwards pass the digital filtering as configured in EEPROM.

LASER COMPONENTS

SMBus communication with TSM

Enter in Sleep Mode Block Diagram



8. Electrical considerations of SMBus applications with TSM

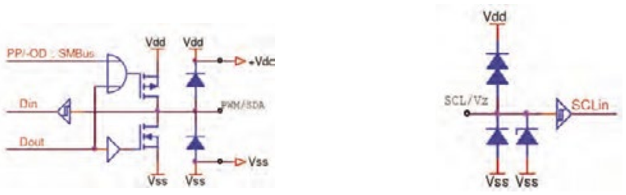


Figure.12 Input / Output pin schematics of TSM

For reliability reasons, the TSM incorporates ESD clamp diodes to Vdd and from Vss. Therefore a powered-down TSM will load the SMBus. This differs from the SMBus specification. In power-managed systems it is therefore needed to keep the TSM powered when the SMBus is needed. This is no issue with sleep mode.

The auxiliary functions of the SCL pin adds undershoot to the clock pulse (5 Volt versions only) as shown in the picture below (see Figure 13). This undershoot is caused by the transient response of the on-chip synthesized Zener diode. Typical duration of undershoot is approximately 15 µs. An increased reactance of the SCL line is likely to increase this effect. Undershoot does not affect the recognition of the SCL rising edge by the TSM, but may affect proper operation of non TSM slaves on the same bus.

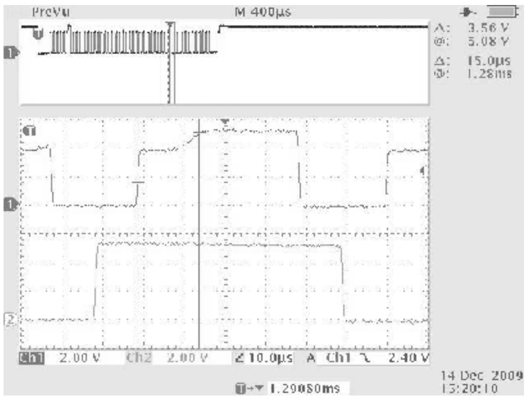


Figure 13 Undershoot of SMBUS SCL line due to the on chip zener diode.
(5 Volt versions only)

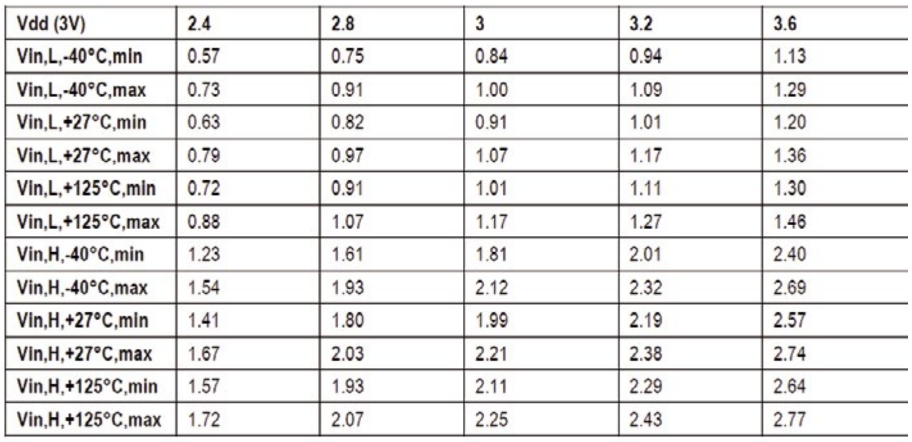
LASER COMPONENTS

SMBus communication with TSM

The input levels of the TSM are not 100% compliant with the SMBus specification. The SMBus specification states an input low voltage maximum value of 0.8V and a minimum high voltage of 2.1V. For the TSM (refer to the data sheet) the specifications differs. When 5V (also applies for > 5V applications) is used, the TSM uses an on-chip voltage regulator (5V – to – 3V ± 10%). With the 3V version, the power supply is used directly. Then, at 5V (as well as at > 5V) the internal circuitry of TSM operates at 3V ± 10%, while at 3V the power supply specification covers 3V ± 20%. The higher tolerance of this power supply results in a higher tolerance of the input levels. Worst-case values for 5V TSMs are Vin,L = 0.5…1.5V and Vin,H = 1.6…2.4V (over all temperatures and supply voltages), and for 3V TSMs are 0.5…1,5V and 1.2…2.8V (idem).

However, this does not mean, that 5V TSMs are likely to have Vin,L = 1.5V and Vin,H = 1.6V at the same time; also TSMs will not have Vin,L = 1.5V and Vin,H = 1.2V at the same time. Both thresholds decrease as the power supply voltage decreases. The two thresholds are also affected by temperature in the same direction. A hysteresis is provided on both SDA and SCL inputs for noise immunity.

As a summary, keeping the logic levels on the bus Vlow < 0.5V and Vhigh > 2.8V will certainly cover all operational cases with the TSM, but is not likely to be really necessary. Detailed values (guaranteed by design, not test limits) are given below:



Vdd (3V)	2.4	2.8	3	3.2	3.6
Vin,L,-40°C,min	0.57	0.75	0.84	0.94	1.13
Vin,L,-40°C,max	0.73	0.91	1.00	1.09	1.29
Vin,L,+27°C,min	0.63	0.82	0.91	1.01	1.20
Vin,L,+27°C,max	0.79	0.97	1.07	1.17	1.36
Vin,L,+125°C,min	0.72	0.91	1.01	1.11	1.30
Vin,L,+125°C,max	0.88	1.07	1.17	1.27	1.46
Vin,H,-40°C,min	1.23	1.61	1.81	2.01	2.40
Vin,H,-40°C,max	1.54	1.93	2.12	2.32	2.69
Vin,H,+27°C,min	1.41	1.80	1.99	2.19	2.57
Vin,H,+27°C,max	1.67	2.03	2.21	2.38	2.74
Vin,H,+125°C,min	1.57	1.93	2.11	2.29	2.64
Vin,H,+125°C,max	1.72	2.07	2.25	2.43	2.77

Table 6

SMBus communication with TSM

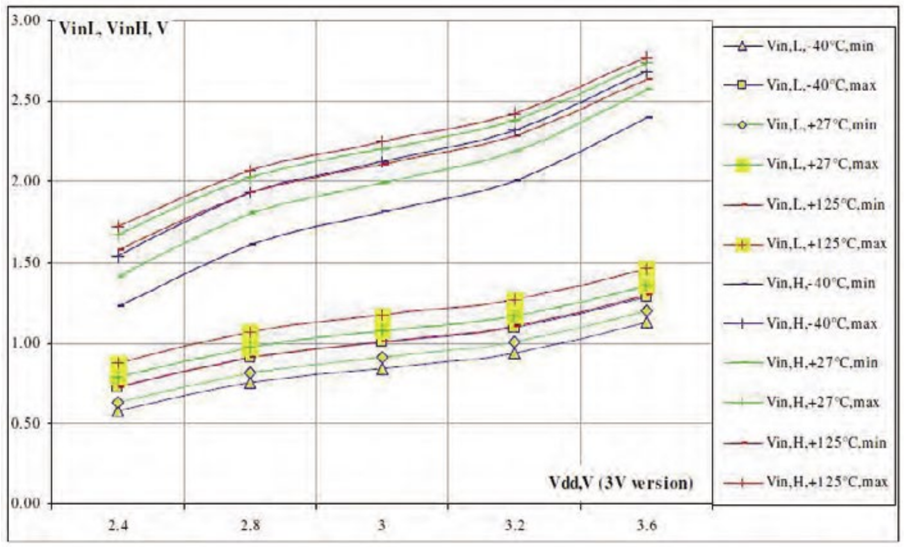


Figure 14 Input voltage levels versus power supply voltage and temperature

9. Applications Information

Use of the TSM Thermometer in SMBus Configuration

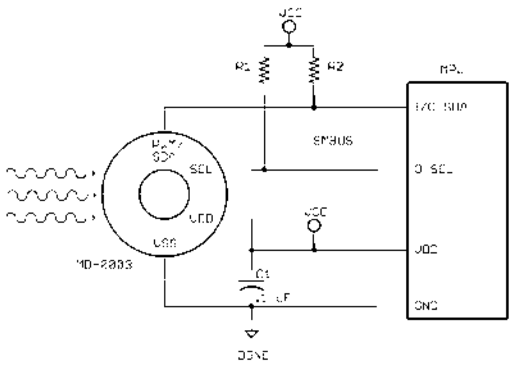


Figure 15 TSM SMBus connection.

Germany & Other Countries
Laser Components GmbH
Tel: +49 8142 2864 – 0
Fax: +49 8142 2864 – 11
info@lasercomponents.com
www.lasercomponents.com

France
Laser Components S.A.S.
Tel: +33 1 39 59 52 25
Fax: +33 1 39 59 53 50
info@lasercomponents.fr
www.lasercomponents.fr

United Kingdom
Laser Components (UK) Ltd.
Tel: +44 1245 491 499
Fax: +44 1245 491 801
info@lasercomponents.co.uk
www.lasercomponents.co.uk

Nordic Countries
Laser Components Nordic AB
Tel: +46 31 703 71 73
Fax: +46 31 703 71 01
info@lasercomponents.se
www.lasercomponents.se

SMBus communication with TSM

Connection of TSM to SMBus with 3.3 V power supply. The TSM has diode clamps SDA / SCL to Vdd so it is necessary to provide TSM with power in order not to load the SMBus lines.

Use of multiple TSMs in SMBus configuration

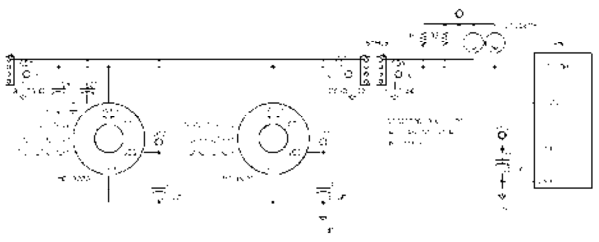


Figure 16 Use of multiple TSM devices in SMBus network.

The TSM supports a 7-bit slave address in EEPROM, thus allowing up to 100 devices to be read via two common wires. Current source pull-ups (such as LTC1694) may be preferred with higher capacitive loading on the bus (C3 and C4 represent the lines' parasitics), while simple resistive pull-ups (R1 & R2) provide the obvious low cost advantage.

High Voltage Source Operation

As a standard, the module MD-0003 and MD-0005 work with a supply voltage of 5 Volt. In addition, thanks to the integrated internal reference regulator available at pin SCL/Vz, these modules can easily be powered from higher voltage source (like VDD = 8 V…16 V). Only a few external components as depicted in the diagram below are required to achieve this.

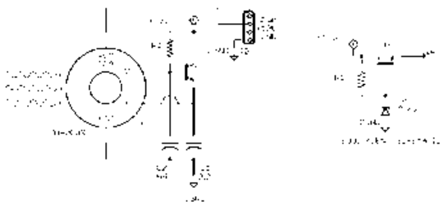


Figure 17 12 V regulator implementation.

With the second (synthesized Zener diode) function of the SCL/Vz pin used, the 2-wire interface function is available only if the voltage regulator is overdriven (5 V regulated power is forced to Vdd pin).

Severe noise can also be coupled within the package from the SCL (in worst cases also from the SDA) pin. This issue can be solved by using PWM output. Also the PWM output can pass additional filtering (at lower PWM frequency settings). With a simple LPF RC network added also increase of the ESD rating is possible.

♦ APPENDIX – SMBus exemplary functions for PIC18 using microchip MCC18 compiler

```
                                                  //SMBus control signals
#define _SCL_IO TRISCbits.TRISC3                  // Pin RC3 direction control bit
#define _SDA_IO TRISCbits.TRISC4                  // Pin RC4 direction control bit
#define _SCL PORTCbits.RC3                        // Assigns pin RC3 for SLC line
#define _SDA PORTCbits.RC4                        // Assigns pin RC4 for SDA line
#define mSDA_HIGH() _SDA_IO=1;                    // Sets SDA line
#define mSDA_LOW() _SDA=0;_SDA_IO=0;              // Clears SDA line
#define mSCL_HIGH() _SCL=1;_SCL_IO=0;             // Sets SCL line
#define mSCL_LOW() _SCL=0;_SCL_IO=0;              // Clears SCL line

#define ACK              0
#define NACK             1

#define SA               0x00                     // Slave address
#define DEFAULT_SA       0x5A                     // Default Slave address
#define RAM_Access       0x00                     // RAM access command
#define EEPROM_Access    0x20                     // EEPROM access command
#define RAM_Tobj1        0x07                     // To1 address in the eeprom

//Delay constants
#define DEL1SEC    100000
#define DEL80ms    7400
#define DEL200ms 18500

//High and Low level of clock @ Fosc=11.0592MHz, Tcy=362ns
#define HIGHLEV 3
#define LOWLEV 1
//Delay constants @ Fosc=11.0592MHz, Tcy=362ns
#define TBUF 2

//*****************************************************************************************************
// START CONDITION ON SMBus
//*****************************************************************************************************
//Name: START_bit
//Function: Generates START condition on SMBus
//Parameters: No
//Return: No
//Comments: Refer to "System Management BUS (SMBus) specification Version 2.0"
//*****************************************************************************************************
void START_bit(void)
{
        mSDA_HIGH();                              // Set SDA line
        Delay10TCYx( TBUF );                      // Wait a few microseconds
        mSCL_HIGH();                              // Set SCL line
        Delay10TCYx( TBUF );                      // Generate bus free time between Stop
                                                  // and Start condition (Tbuf=4.7us min)
        mSDA_LOW();                               // Clear SDA line
        Delay10TCYx( TBUF );                      // Hold time after (Repeated) Start
                                                  // Condition. After this period, the first clock is generated.
                                                  //(Thd:sta=4.0us min)
        mSCL_LOW();                               // Clear SCL line
        Delay10TCYx( TBUF );                      // Wait a few microseconds
}
```

SMBus communication with TSM

```
//*****************************************************************************************************
// STOP CONDITION ON SMBus
//*****************************************************************************************************
//Name: STOP_bit
//Function: Generates STOP condition on SMBus
//Parameters: No
//Return: No
//Comments: Refer to "System Management BUS (SMBus) specification Version 2.0"
//*****************************************************************************************************
void STOP_bit(void)
{
            mSCL_LOW();                             // Clear SCL line
            Delay10TCYx( TBUF );                    // Wait a few microseconds
            mSDA_LOW();                             // Clear SDA line
            Delay10TCYx( TBUF );                    // Wait a few microseconds
            mSCL_HIGH();                            // Set SCL line
            Delay10TCYx( TBUF );                    // Stop condition setup time(Tsu:sto=4.0us min)
            mSDA_HIGH();                            // Set SDA line
}

//*****************************************************************************************************
// TRANSMIT DATA ON SMBus
//*****************************************************************************************************
//Name: TX_byte
//Function: Sends a byte on SMBus
//Parameters: unsigned char TX_buffer ( the byte which will be send on the SMBus )
//Return: unsigned char Ack_bit (acknowledgment bit)
// 0 - ACK
// 1 - NACK
//Comments: Sends MSbit first
//*****************************************************************************************************
unsigned char TX_byte(unsigned char Tx_buffer)
{
            unsigned char Bit_counter;
            unsigned char Ack_bit;
            unsigned char bit_out;
            for(Bit_counter=8; Bit_counter; Bit_counter--)
            {
                        if(Tx_buffer&0x80)
                                    bit_out=1;              // If the current bit of Tx_buffer is 1 set bit_out
                        else
                                    bit_out=0;              // else clear bit_out
                        send_bit(bit_out);                  // Send the current bit on SDA
                        Tx_buffer<<=1;                      // Get next bit for checking
            }
            Ack_bit=Receive_bit();                  // Get acknowledgment bit
            return Ack_bit;
}                                                   // End of TX_byte()

//*****************************************************************************************************
//-----------------------------------------------------------------------------------------------------
void send_bit(unsigned char bit_out)
{
            if(bit_out==0)                          // Check bit
                        mSDA_LOW();                 // Set SDA if bit_out=1
            else
                        mSDA_HIGH();                // Clear SDA if bit_out=0
            Nop();                                  //
            Nop();                                  // Tsu:dat = 250ns minimum
            Nop();                                  //
            mSCL_HIGH();                            // Set SCL line
            Delay10TCYx( HIGHLEV );                 // High Level of Clock Pulse
            mSCL_LOW();                             // Clear SCL line
            Delay10TCYx( LOWLEV );                  // Low Level of Clock Pulse
            // mSDA_HIGH();                         // Master release SDA line ,
            return;
}                                                   //End of send_bit()
```

LASER COMPONENTS

SMBus communication with TSM

```
//*******************************************************************************************************
// RECEIVE DATA ON SMBus
//*******************************************************************************************************
//Name: RX_byte
//Function: Receives a byte on SMBus
//Parameters: unsigned char ack_nack (acknowledgment bit)
// 0 - master sends ACK
// 1 - master sends NACK
//Return: unsigned char RX_buffer (Received byte)
//Comments: MSbit is received first
//*******************************************************************************************************
unsigned char RX_byte(unsigned char ack_nack)
{
            unsigned char RX_buffer;
            unsigned char Bit_Counter;
            for(Bit_Counter=8; Bit_Counter; Bit_Counter--)
            {
                        if(Receive_bit())                       // Read a bit from the SDA line
                        {
                                    RX_buffer <<= 1;            // If the bit is HIGH save 1 in RX_buffer
                                    RX_buffer |=0b00000001;
                        }
                        else
                        {
                        RX_buffer <<= 1;                        // If the bit is LOW save 0 in RX_buffer
                        RX_buffer &=0b11111110;
                        }
            }
            send_bit(ack_nack);                                 // Send acknowledgment bit
            return RX_buffer;
}                                                               // End of RX_byte()


//*******************************************************************************************************
//*******************************************************************************************************

unsigned char Receive_bit(void)
{
            unsigned char bit;
            _SDA_IO=1;                                          // SDA-input
            mSCL_HIGH();                                        // Set SCL line
            Delay10TCYx( HIGHLEV );                             // High Level of Clock Pulse
            if(_SDA)                                            // Read bit, save it in bit
                        bit=1;
            else
                        bit=0;
            mSCL_LOW();                                         // Clear SCL line
            Delay10TCYx( LOWLEV );                              // Low Level of Clock Pulse
            return bit;
}                                                               //End of Receive_bit()
//-----------------------------------------------------------------------------------------------------------------------
```

LASER COMPONENTS

SMBus communication with TSM

```
//*********************************************************************************************
//                                  CALCULATION PEC PACKET
//*********************************************************************************************
//Name:        PEC_calculation
//Function:    Calculates the PEC of received bytes
//Parameters:  unsigned char pec[]
//Return:      pec[0]-this byte contains calculated crc value
//Comments:    Refer to "System Managment BUS(SMBus) specification Version 2.0"
//*********************************************************************************************
#define CRCTABLE
#ifdef CRCTABLE
#include "crc8Table.h"
#endif

unsigned char PEC_calculation(unsigned char pec[])
{
#ifndef CRCTABLE
    unsigned char   crc[6];
    unsigned char   BitPosition=47;
    unsigned char   shift;
    unsigned char   i;
    unsigned char   j;
    unsigned char   temp;

    do{
        crc[5]=0;                           /* Load CRC value 0x000000000107 */
        crc[4]=0;
        crc[3]=0;
        crc[2]=0;
        crc[1]=0x01;                        /* the polynomial used is X8 + X2 + X + 1. */
        crc[0]=0x07;                        /* The width of this polynomial is 8 */
                                            /* and it can be represented as 1 0000 0111 */
        BitPosition=47;                     /* Set maximum bit position at 47 */
        shift=0;

        //Find first 1 in the transmitted message

        i=5;                                /* Set highest index */
        j=0;
                while((pec[i]&(0x80>>j))==0 && i>0)
                {
                        BitPosition--;
                        if(j<7)
                        {
                                j++;
                        }
                        else
                        {
                                j=0x00;
                                i--;
                        }
                }       /*End of while */

                shift=BitPosition-8;        /*Get shift value for crc value*/
```

Germany & Other Countries
Laser Components GmbH
Tel: +49 8142 2864 – 0
Fax: +49 8142 2864 – 11
info@lasercomponents.com
www.lasercomponents.com

France
Laser Components S.A.S.
Tel: +33 1 39 59 52 25
Fax: +33 1 39 59 53 50
info@lasercomponents.fr
www.lasercomponents.fr

United Kingdom
Laser Components (UK) Ltd.
Tel: +44 1245 491 499
Fax: +44 1245 491 801
info@lasercomponents.co.uk
www.lasercomponents.co.uk

Nordic Countries
Laser Components Nordic AB
Tel: +46 31 703 71 73
Fax: +46 31 703 71 01
info@lasercomponents.se
www.lasercomponents.se

LASER COMPONENTS

SMBus communication with TSM

```
                //Shift crc value
                while(shift)
                {
                        for(i=5; i<0xFF; i--)
                        {
                                if((crc[i-1]&0x80) && (i>0))
                                {
                                        temp=1;
                                }
                                else
                                {
                                        temp=0;
                                }
                                crc[i]<<=1;
                                crc[i]+=temp;

                        }       /*End of for*/
                        shift--;

                }       /*End of while*/


            //Exclusive OR between pec and crc
                for(i=0; i<=5; i++)
                {
                        pec[i] ^=crc[i];
                }       /*End of for*/

    } while(BitPosition>8);         /*End of do-while*/

    return pec[0];

#else

    unsigned char TablePecReg;          // Calculated PEC byte storage

        TablePecReg = 0;
        TablePecReg = CRC_8_TABLE[TablePecReg ^ pec[5]];
        TablePecReg = CRC_8_TABLE[TablePecReg ^ pec[4]];
        TablePecReg = CRC_8_TABLE[TablePecReg ^ pec[3]];
        TablePecReg = CRC_8_TABLE[TablePecReg ^ pec[2]];
        TablePecReg = CRC_8_TABLE[TablePecReg ^ pec[1]];


    return TablePecReg;


#endif

}       /*End of PEC_calculation*/
```

LASER COMPONENTS

SMBus communication with TSM

```
// crc8Table.c
#include "crc8Table.h"

//      Direct-Calculation Method
//      The TSM uses a CRC polynomial of type C(x) = X8 + X² + X + 1. The width of the polynomial is 8 (hence CRC-8) and
//      can be represented as 1 0000 0111b. To implement software code for calculating the CRC-8 directly, a simple procedure
//      can be established and programmed accordingly:
//      1. Concatenate the end of the data string (LSB) with eight zeros.
//      2. Perform the XOR operation on the data string against the binary version of the polynomial.
//              1. First, shift the data string until a "1" appears at the MSB of the register.
//              2. Line up the first "1" of the polynomial (1X8 part) so that it will logically operate against the first "1" of the
//                data string when the XOR is performed.
//              3. Perform the logical XOR of the 8 bits. There are actually 9 bits being operated on, but the first "1" bit will
//                always result in a "0." This "0" is in the MSB place, and thus does not contribute to the magnitude of the
//                final CRC value.
//      3. The result of the XOR operation should then be augmented by the "untouched" bits (those bits in the data string
//        that are nine places to the right of the first "1" in the string). This augmented result is now saved in place of the
//        data string.
//      4. Continue the process of shifting and XORing (from step 2) until the LSB of the polynomial does not line up with
//        any of the added eight zeros (the end of the data string with eight zeros added has been shifted sufficiently so
//        that all bits have been operated on). The result will be a completed CRC-8.
//      Table-Driven Method
//      Some host systems might not be able to directly calculate the CRC-8 in a reasonable amount of time. This is especially
//      likely if the worst-case communication string is read: a long string with 128 data bytes. In this case, a table-driven
//      method can be used to speed up the process. The table method is more efficient because it effectively processes a
//      number of bits at a time. Consequently, the larger the table, the faster the CRC is calculated. Because the TSM
//      communicates eight bits at a time, the most reasonable choice is to use a 256-location table capable of processing one
//      byte at a time.
//      To use this method, a static 256-byte lookup table must first be available in memory and preloaded with the CRC values.
//      Then during the calculation, the table has to be indexed and the result returned.
//      The procedure follows:
//      1. Initialize Register (variable) with zeros.
//      2. Perform the XOR operation on the data string, eight bits at a time.
//              1. XOR 8 bits of the data string with the contents of Register (variable)
//              2. The 8-bit result of the XOR operation is now used as a pointer into the 256-byte lookup table.
//              3. Return the correct table entry.
//              4. Continue the XOR operation on the 8-bit result of the lookup table query against the remaining bytes in
//                the data string, one byte at a time, until the end of data string is reached.
//      3. The last value in the (variable) register is the CRC-8.

   rom char    CRC_8_TABLE[] = { 0x00, 0x07, 0x0e, 0x09, 0x1c, 0x1b, 0x12, 0x15,
                                 0x38, 0x3f, 0x36, 0x31, 0x24, 0x23, 0x2a, 0x2d,
                                 0x70, 0x77, 0x7E, 0x79, 0x6C, 0x6B, 0x62, 0x65,
                                 0x48, 0x4F, 0x46, 0x41, 0x54, 0x53, 0x5A, 0x5D,
                                 0xE0, 0xE7, 0xEE, 0xE9, 0xFC, 0xFB, 0xF2, 0xF5,
                                 0xD8, 0xDF, 0xD6, 0xD1, 0xC4, 0xC3, 0xCA, 0xCD,
                                 0x90, 0x97, 0x9E, 0x99, 0x8C, 0x8B, 0x82, 0x85,
                                 0xA8, 0xAF, 0xA6, 0xA1, 0xB4, 0xB3, 0xBA, 0xBD,
                                 0xC7, 0xC0, 0xC9, 0xCE, 0xDB, 0xDC, 0xD5, 0xD2,
                                 0xFF, 0xF8, 0xF1, 0xF6, 0xE3, 0xE4, 0xED, 0xEA,
                                 0xB7, 0xB0, 0xB9, 0xBE, 0xAB, 0xAC, 0xA5, 0xA2,
                                 0x8F, 0x88, 0x81, 0x86, 0x93, 0x94, 0x9D, 0x9A,
                                 0x27, 0x20, 0x29, 0x2E, 0x3B, 0x3C, 0x35, 0x32,
                                 0x1F, 0x18, 0x11, 0x16, 0x03, 0x04, 0x0D, 0x0A,
                                 0x57, 0x50, 0x59, 0x5E, 0x4B, 0x4C, 0x45, 0x42,
                                 0x6F, 0x68, 0x61, 0x66, 0x73, 0x74, 0x7D, 0x7A,
                                 0x89, 0x8E, 0x87, 0x80, 0x95, 0x92, 0x9B, 0x9C,
                                 0xB1, 0xB6, 0xBF, 0xB8, 0xAD, 0xAA, 0xA3, 0xA4,
                                 0xF9, 0xFE, 0xF7, 0xF0, 0xE5, 0xE2, 0xEB, 0xEC,
                                 0xC1, 0xC6, 0xCF, 0xC8, 0xDD, 0xDA, 0xD3, 0xD4,
                                 0x69, 0x6E, 0x67, 0x60, 0x75, 0x72, 0x7B, 0x7C,
                                 0x51, 0x56, 0x5F, 0x58, 0x4D, 0x4A, 0x43, 0x44,
                                 0x19, 0x1E, 0x17, 0x10, 0x05, 0x02, 0x0B, 0x0C,
                                 0x21, 0x26, 0x2F, 0x28, 0x3D, 0x3A, 0x33, 0x34,
                                 0x4E, 0x49, 0x40, 0x47, 0x52, 0x55, 0x5C, 0x5B,
                                 0x76, 0x71, 0x78, 0x7F, 0x6A, 0x6D, 0x64, 0x63,
                                 0x3E, 0x39, 0x30, 0x37, 0x22, 0x25, 0x2C, 0x2B,
                                 0x06, 0x01, 0x08, 0x0F, 0x1A, 0x1D, 0x14, 0x13,
                                 0xAE, 0xA9, 0xA0, 0xA7, 0xB2, 0xB5, 0xBC, 0xBB,
                                 0x96, 0x91, 0x98, 0x9F, 0x8A, 0x8D, 0x84, 0x83,
                                 0xDE, 0xD9, 0xD0, 0xD7, 0xC2, 0xC5, 0xCC, 0xCB,
                                 0xE6, 0xE1, 0xE8, 0xEF, 0xFA, 0xFD, 0xF4, 0xF3};
```

LASER COMPONENTS

SMBus communication with TSM

```
//***************************************************************************
//                    BASIC INFORMATION ABOUT THIS PROGRAM
//***************************************************************************
//File name:    main.c
//Details:            SMBus comunication with TSM using PIC18F4320
//                        Language C: MCC18 compiler
//                        Fosc=11.0592MHz, Tcy=362ns
//***************************************************************************
//                                    HEADER FILES
//***************************************************************************
#include <p18F4320.h>

//***************************************************************************
//                              CONFIGURATION BITS
//***************************************************************************
#pragma config  OSC=HS,FSCM=OFF,IESO=ON
#pragma config  PWRT=ON,BOR=OFF,BORV=42
#pragma config  WDT=OFF
#pragma config  MCLRE=ON,PBAD=DIG,CCP2MX=C1
#pragma config  STVR=ON,LVP=OFF,DEBUG=OFF
#pragma config  CP0=OFF,CP1=OFF,CP2=OFF,CP3=OFF
#pragma config  CPB=ON,CPD=OFF
#pragma config  WRT0=OFF,WRT1=OFF,WRT2=OFF,WRT3=OFF
#pragma config  WRTC=OFF,WRTB=ON,WRTD=OFF
#pragma config  EBTR0=OFF,EBTR1=OFF,EBTR2=OFF,EBTR3=OFF
#pragma config  EBTRB=ON

#pragma interrupt high_isr
void high_isr(void)
{
// User code...
}


#pragma interruptlow low_isr
void low_isr(void)
{
// User code...
}

#pragma code

//*******************************************************************************
//                            MAIN PROGRAM
//*******************************************************************************
void main(void)
{
      unsigned char   SlaveAddress;                    //Contains device address
      unsigned char   command;                         //Contains the access command
      unsigned int    data;                            //Contains data value
      float               t;                           //Contains calculated temperature in degrees Celsius

      MCUinit();                                       //MCU initialization
      SlaveAddress=SA<<1;                              //Set device address
      command=RAM_Access|RAM_Tobj1;                    //Form RAM access command + RAM address

//    SendRequest();                                   //Switch to SMBus mode - this is need if module is in PWM mode only
//    DummyCommand(SlaveAddress);                      //This is need if Request Command is sent even when the module is in SMBus mode
      delay(DEL200ms);                                 //Wait after POR,Tvalid=0.15s

      while(1)
      {
            data=MemRead(SlaveAddress,command);        //Read memory
            t=CalcTemp(data);                          //Calculate temperature
            delay(DEL1SEC);                            //Wait 1 second
      }

}                                                      //End of main()
```

LASER COMPONENTS

SMBus communication with TSM

```
//*****************************************************************************************************************
//                                        FUNCTION'S DEFINITIONS
//*****************************************************************************************************************
void MCUinit(void)
{
                                                  //IO setting-up
     ADCON1=0b00001111;                           //All channels are digital I/O
                                                  //SMBus setting-up
     mSDA_HIGH();                                 //The bus is in idle state
     mSCL_HIGH();                                 //SDA and SCL are in high level from pull up resistors
}                                                 //End of init()

//-----------------------------------------------------------------------------------------------------------------

unsigned int MemRead(unsigned char SlaveAddress,unsigned char command)
{
     unsigned int  data;                          // Data storage (DataH:DataL)
     unsigned char Pec;                           // PEC byte storage
     unsigned char DataL;                         // Low data byte storage
     unsigned char DataH;                         // High data byte storage
     unsigned char arr[6];                        // Buffer for the sent bytes
     unsigned char PecReg;                        // Calculated PEC byte storage
     unsigned char ErrorCounter;                   // Defines the number of the attempts for communication with TSM

     ErrorCounter=0x00;                            // Initialising of ErrorCounter

     do{
     repeat:

          STOP_bit();                             //If slave send NACK stop comunication
          --ErrorCounter;                         //Pre-decrement ErrorCounter

          if(!ErrorCounter){                      //ErrorCounter=0?
               break;                             //Yes,go out from do-while{}
          }

          START_bit();                            //Start condition

          if(TX_byte(SlaveAddress)){              //Send SlaveAddress
               goto    repeat;                    //Repeat comunication again
          }

          if(TX_byte(command)){                   //Send command
               goto    repeat;                    //Repeat comunication again
          }

          START_bit();                            //Repeated Start condition

          if(TX_byte(SlaveAddress)){              //Send SlaveAddress
               goto    repeat;                    //Repeat comunication again
          }

          DataL=RX_byte(ACK);                    //Read low data,master must send ACK
          DataH=RX_byte(ACK);                     //Read high data,master must send ACK
          Pec=RX_byte(NACK);                     //Read PEC byte, master must send NACK
          STOP_bit();                             //Stop condition

          arr[5]=SlaveAddress;                    //
          arr[4]=command;                          //
          arr[3]=SlaveAddress;                     //Load array arr
          arr[2]=DataL;                           //
          arr[1]=DataH;                           //
          arr[0]=0;                                //
          PecReg=PEC_calculation(arr);            //Calculate CRC

     }while(PecReg != Pec);                       //If received and calculated CRC are equal go out from do-while{}

     *((unsigned char *)(&data))=DataL;           //
     *((unsigned char *)(&data)+1)=DataH ;        //data=DataH:DataL

     return data;
}
```

```
//------------------------------------------------------------------------------------------------
void SendRequest(void)
{
      mSCL_LOW();                              //SCL 1 _____________|<---------80ms---------->|_______________
      delay(DEL80ms);                          //       0           |____________________|
      mSCL_HIGH();
}
//------------------------------------------------------------------------------------------------


void DummyCommand(unsigned char byte)
{
      START_bit();                             //Start condition
      TX_byte(byte);                           //Send Slave Address or whatever,no need ACK checking
      STOP_bit();                              //Stop condition
}
//------------------------------------------------------------------------------------------------
float CalcTemp(unsigned int value)
{
      float temp;

      temp=(value*0.02)-273.15;

      return temp;
}
```

10. Revision Table

Version	Modifications	Comment	Date
1		Release	July 2010