1 Scope

This document introduces the users in SMBus communication protocol and especially how it can be used to communicate with MLX90614 infrared thermometers. The MLX90614 is an Infra Red thermometer for non contact temperature measurements. Both the IR sensitive thermopile detector chip and the signal conditioning ASSP are integrated in the same TO-39 can. Thanks to its low noise amplifier, 17-bit ADC and powerful DSP unit, a high accuracy and resolution of the thermometer is achieved. The thermometer comes factory calibrated with a digital PWM and SMBus output. As a standard, the 10-bit PWM is configured to continuously transmit the measured temperature in range of -20 to 120 °C, with an output resolution of 0.14 °C and the POR default is SMBus.

The original purpose of the SMBus was to define the communication link between an intelligent battery, charger for the battery and a microcontroller that communicates with the rest of the system. However, SMBus can also be 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/

2 Related Melexis Products

EVB90614 is the evaluation board which supports the MLX90614 devices.

3 Table of contents

4 General SMBus protocol discription

4.1 Definitions 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 Last 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.

4.2 SMBus overview

Only two bus lines are required; a serial data line (SDA) and a serial clock line (SCL). Each device 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 at up to 100 kbit/s. The System Management Bus (SMBus) is a two-wire interface through which various system component chips can communicate with each other and with the rest of the system. 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. The one exception to this rule is during initial bus setup when a single master may initiate transactions with multiple slaves simultaneously. A bus slave device can receive data provided by the master or it can provide data to the master. 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 (Fig.1).

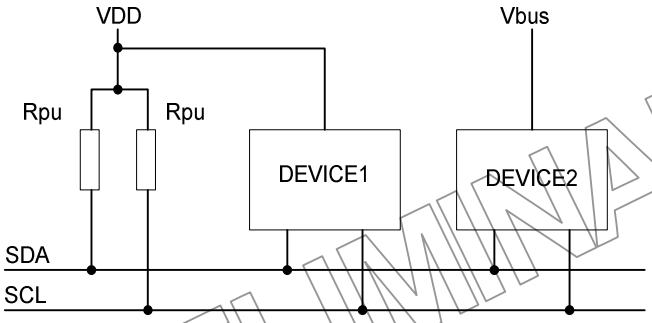


Fig.1: SMBus Topology

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 or other similar 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) R_{PU}=1.5kΩ (V_{DD}=5V,I_{PULLUP}=3.3mA) is suitable otherwise R_{PU}=22kΩ(V_{DD}=5V,I_{PULLUP}=227µA) can be used to meet SMBus low power DC specification (see low and high power DC specification below).

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.

Fig.2 shows a CRC calculation example.

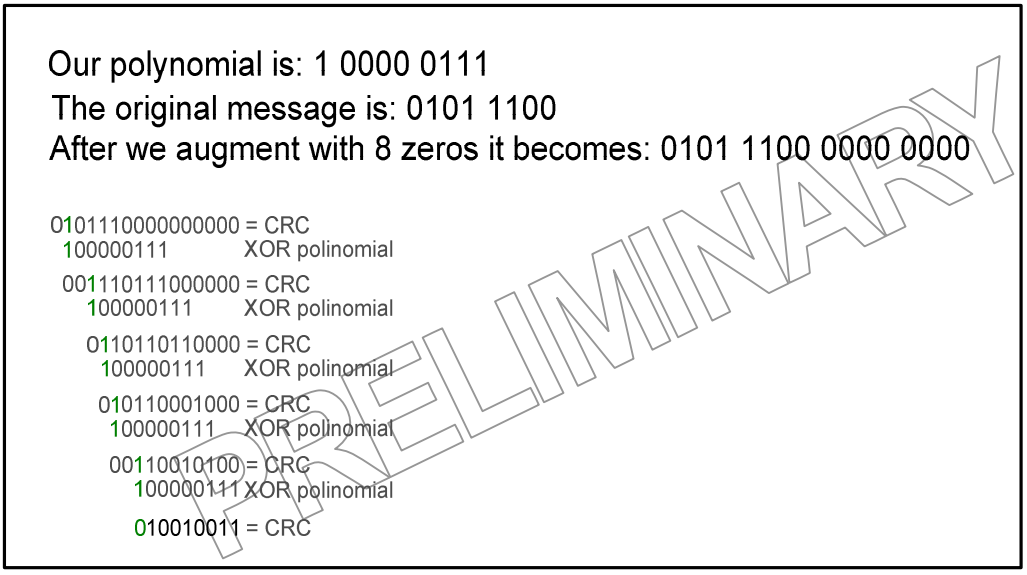


Fig.2

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

Fig.3 shows the common structure of an SMBus transaction

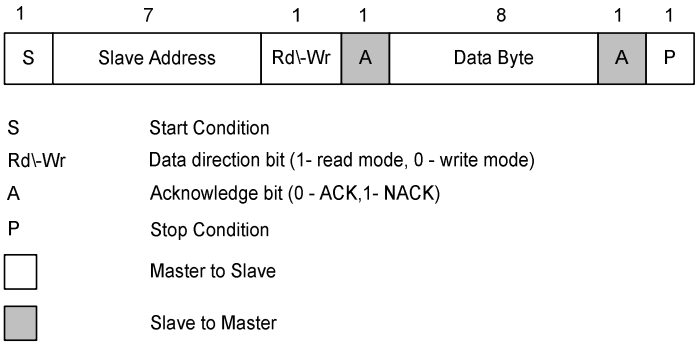


Fig.3: SMBus Transaction

4.3 Electrical characteristics of SMBus devices

The diagram bellow illustrates the various SMBus timings

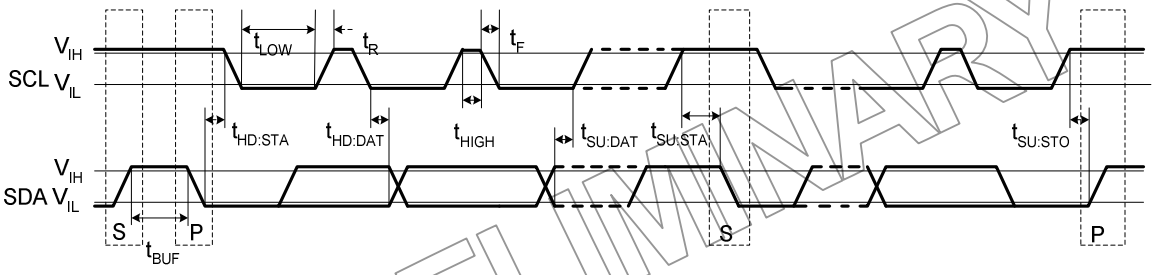


Fig.4: SMBus timing measurements

The table below describes all timings.

Table 1

Symbol	Parameter	Min	Max	Units	Comments
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	
$t_{HD:DAT}$	Data hold time	300	-	ns	See note 7
$t_{SU:DAT}$	Data setup time	250	-	ns	
$t_{TIMEOUT}$	Detect clock low timeout	25	35	ms	See note 2
t_{LOW}	Clock low period	4.7	-	µs	
t_{HIGH}	Clock high period	4.0	50	µs	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	

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 nonperiodic 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 MLX90614 module this value is above 500ns

4.4 Timeouts

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

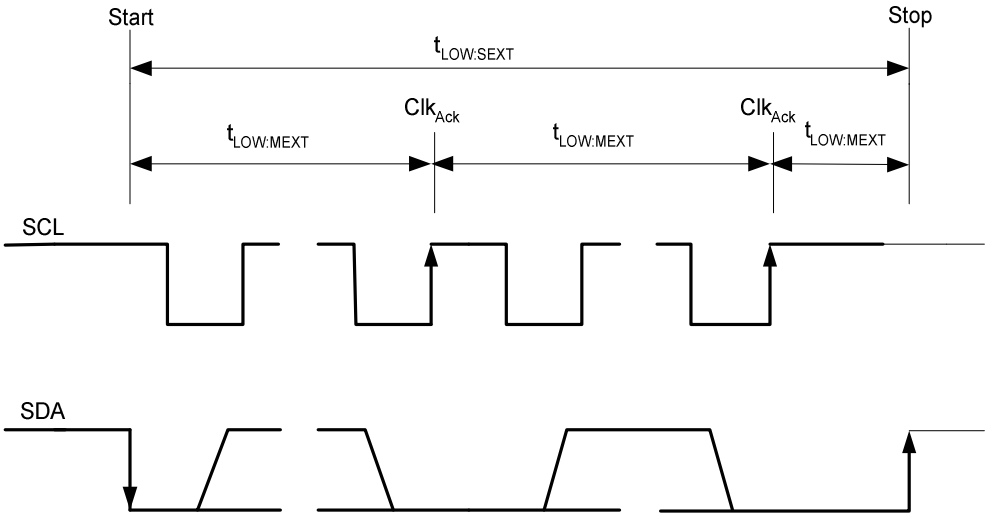


Fig.5: Timeout measurement intervals

4.5 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 $t_{TIMEOUT,MIN}$. Devices that have detected this condition must reset their communication and be able to receive a new START condition in no later than $t_{TIMEOUT,MAX}$. Slave devices that violate $t_{LOW:SEXT}$ 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.

4.6 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.

4.7 Low-power DC specifications

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

Table 2

Symbol	Parameter	Min	Max	Units	Comments
V_{IL}	Data, Clock Input Low Voltage	-	0.8	V	
V_{IH}	Data, Clock Input High Voltage	2.1	V_{DD}	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 unpowered. However, a microcontroller acting as an SMBus host may exceed I_{LEAK} 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.

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 unpowered (that is, their Vcc lowered to zero), must also meet the leakage current specification.

4.8 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.

Table 3

Symbol	Parameter	Min	Max	Units	Comments
V_{IL}	SMBus signal Input low voltage	-	0.8	V	
V_{IH}	SMBus signal Input high voltage	2.1	V_{DD}	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, V_{OL}=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 10MHz to 100MHz	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.

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.
Unpowered 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.

4.9 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 (Fig.6).

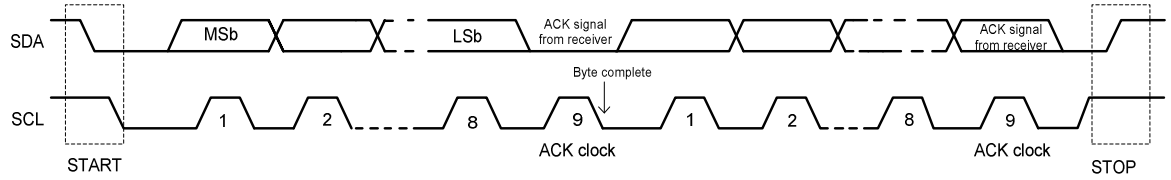


Fig.6: SMBus byte format

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

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 = 10kHz (35mS timeout)
Timeout is where a slave device resets its interface whenever SCL goes low for longer than the timeout, typically 35mSec. 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 35mS.
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 I^2C defines maximum bus capacitance 400pF SMBus does not specify a maximum bus capacitance. Instead it specifies the I_{PULLUP} maximum of 350µA in Low-power DC specification and minimum 4mA in High-power DC specification. Bus capacitance can be calculated taking into consideration the maximum rise time and I_{PULLUP}.
In the table below are given a summery of level specifications for the I2C Bus and the SMBus.

Table 4

High	I^2C VDD Dependent	$0.7*V_{DD}$
	I^2C Fixed	3.0V
	SMBus	2.1V

Low	I^2C VDD Dependent	$0.3*V_{DD}$
	I^2C 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 I^2C and SMBus can be found on:
http://www.maxim-ic.com/appnotes_frame.cfm/appnote_number/476
http://www.smbus.org/specs/

6 SMBus comunication with MLX90614

6.1 Overview

The MLX90614 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). The MLX90614 meets all the timing specifications of the SMBus (refer to Electrical characteristics of SMBus devices above). MLX90614 has 32x17 RAM. 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 6 below). (RAM readings format is described in more details below as well as in the MLX90614 data sheet.) 32x16 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 ASIC after POR (also includes enter and exit Sleep mode). For example, Slave Address can be changed in EEPROM, but the ASIC will respond to the old SA until POR is exited.

If MLX90614 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, MLX90614 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.

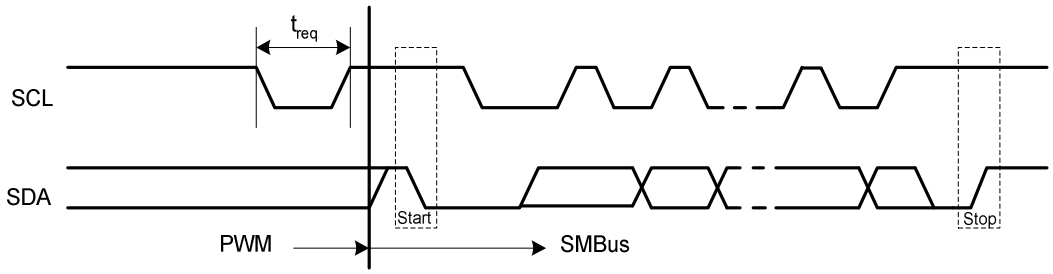


Fig.7 SMBus request

The MLX90614's SMBus request condition requires forcing LOW the SCL pad for period longer than the request time t_{REQ} (see Table 5 below). 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 Timings

The specific timings in MLX90614's SMBus are: SMBus Request (t_{REQ}) is the time that the SCL should be forced low in order to switch the MLX90614 from PWM mode to SMBus mode; Tsuac (SD) is the time after the eighth falling edge of SCL that MLX90614 will force PWM/SDA low to acknowledge the last received byte. Thdac (SD) is the time after the ninth falling edge of SCL that MLX90614 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 MLX90614 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 MLX90614 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.

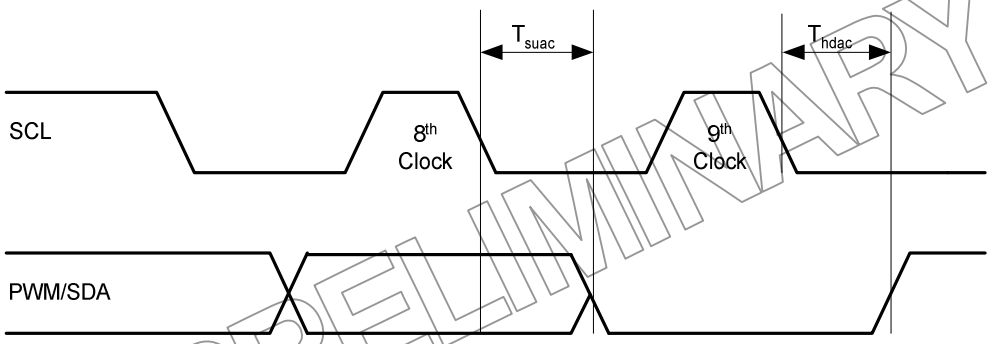


Fig.8:MLX90614 specific timings

In Table 5 are given the values of the specific timings.

Table 5

Symbol	Parameter	Min	Max	Units	Comments
t_{REQ}	SMBus Request	2		ms	
t_{suac} (MD)		0.5	1	µs	
t_{suac}(SD)		1.5	2	µs	
t_{hdac}(MD)		1.5	2	µs	
t_{hdac}(SD)		0.5	1	µs	

6.3 Detailed Communication description

Table 6 describes commands needed for communication with MLX90614.

Table 6

Command	Description
000x xxxx*	RAM Access
001x xxxx*	EEPROM Access
1111 0000**	Read Flags
1111 1111	Enter SLEEP mode

*Note *: The xxxxx is the address of the cell that has to be accessed. Read/Write is selected via the Read/Write bit (refer to Fig. 3).*
*Note**: Behaves like read command. The MLX90614 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 read and read flags is that the latter does not have a repeated start bit.*

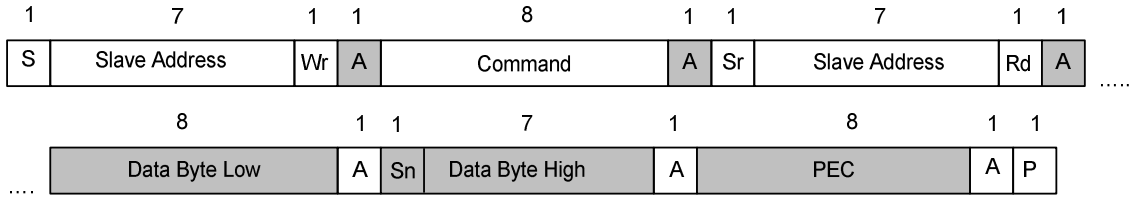
Table 7 describes most important RAM registers. For other registers in RAM and EEPROM memory refer to MLX90614 datasheet.

Table 7

Name	Address (hexadecimal)
Ambient sensor data	0x03
IR sensor 1 data	0x04
IR sensor 2 data	0x05
Linearized ambient temperature T_A	0x06
Linearized object temperature T_{obj1}	0x07
Linearized object temperature T_{obj2}	0x08

All bytes are sent and received with MSb first.

The format of SMBus reading from RAM is:



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°C). 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).

<u>Pseudo code example:</u> Reading RAM address 0x07 (Tobj1)

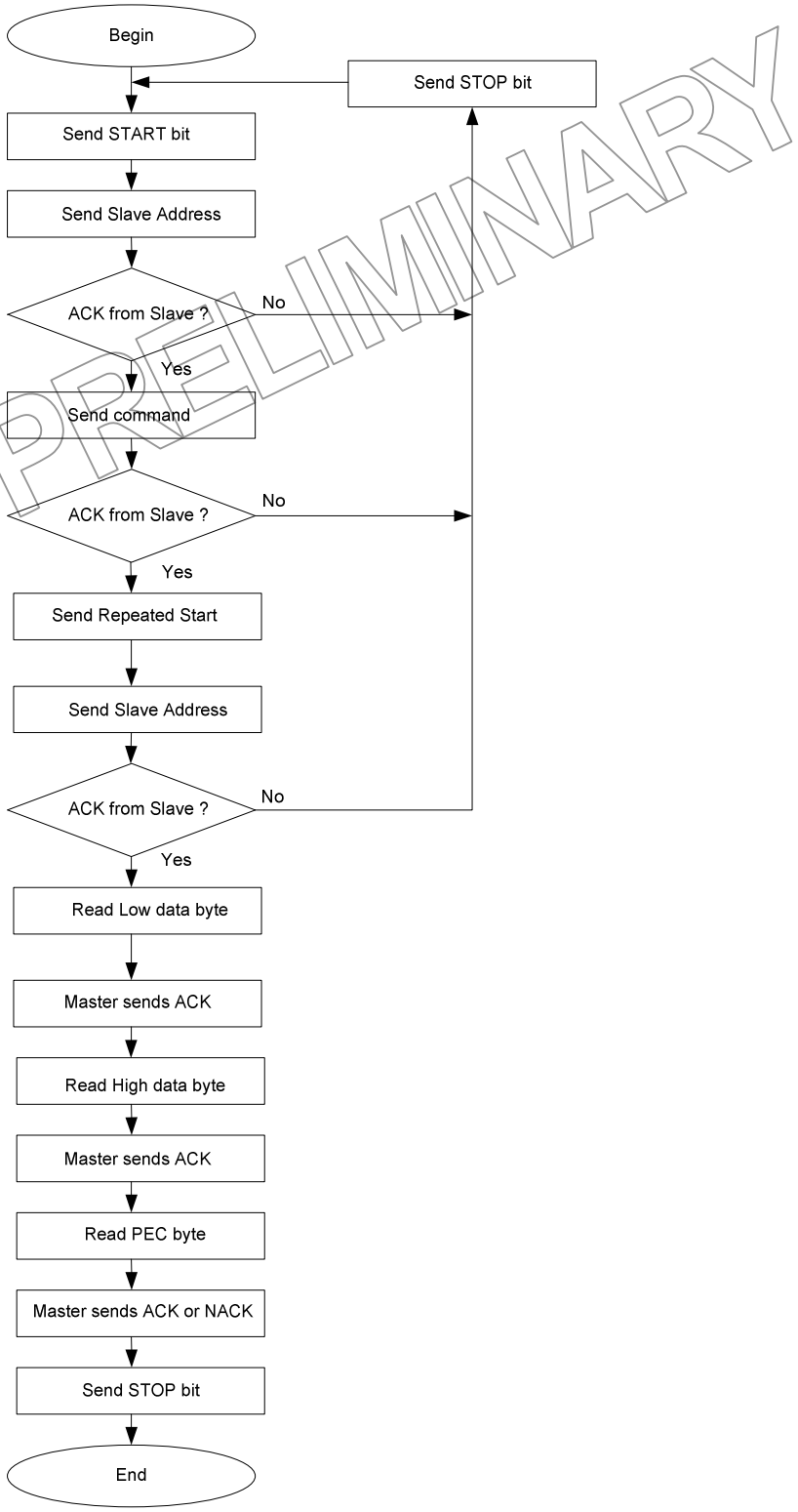
1. Send START bit
2. Send Slave Address (0x00* for example) + Rd\-Wr bit**
3. Send Command (0b000x_xxxx + 0b0000_0111 -> 0b0000_0111)
4. Send Repeated START_bit
5. Send Slave Address + Rd\-Wr bit**
6. Read Data Byte Low (master must send ACK bit)
7. Read Data Byte High (master must send ACK bit)
8. Read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

Note : Any MLX90614 will respond to address 0x00*
*Note**: Bit Rd\-Wr has no meaning for MLX90614*

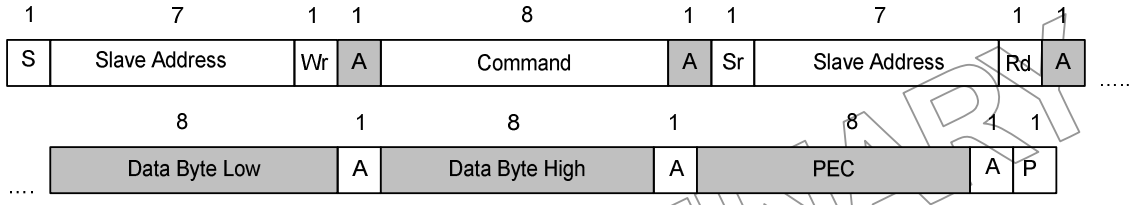
After POR before to read the temperature additional delay is needed or the first measurements will not be correct. This additional delay depends on the FIR filter (After POR the digital part of the MLX90614 is restarted, IIR =100% for the first temperature calculation flow from the module,

see Fig.11. After this first temperature calculation flow from the module, IIR refreshes its value from EEPROM ConfigRegister<2..0>).
See AppNote "Understanding MLX90614 on-chip digital signal filters" on http://www.melexis.com/Asset.aspx?nID=5272 to understand how to calculate this delay time.

Read RAM Block Diagram



The format of SMBus reading from EEPROM is:



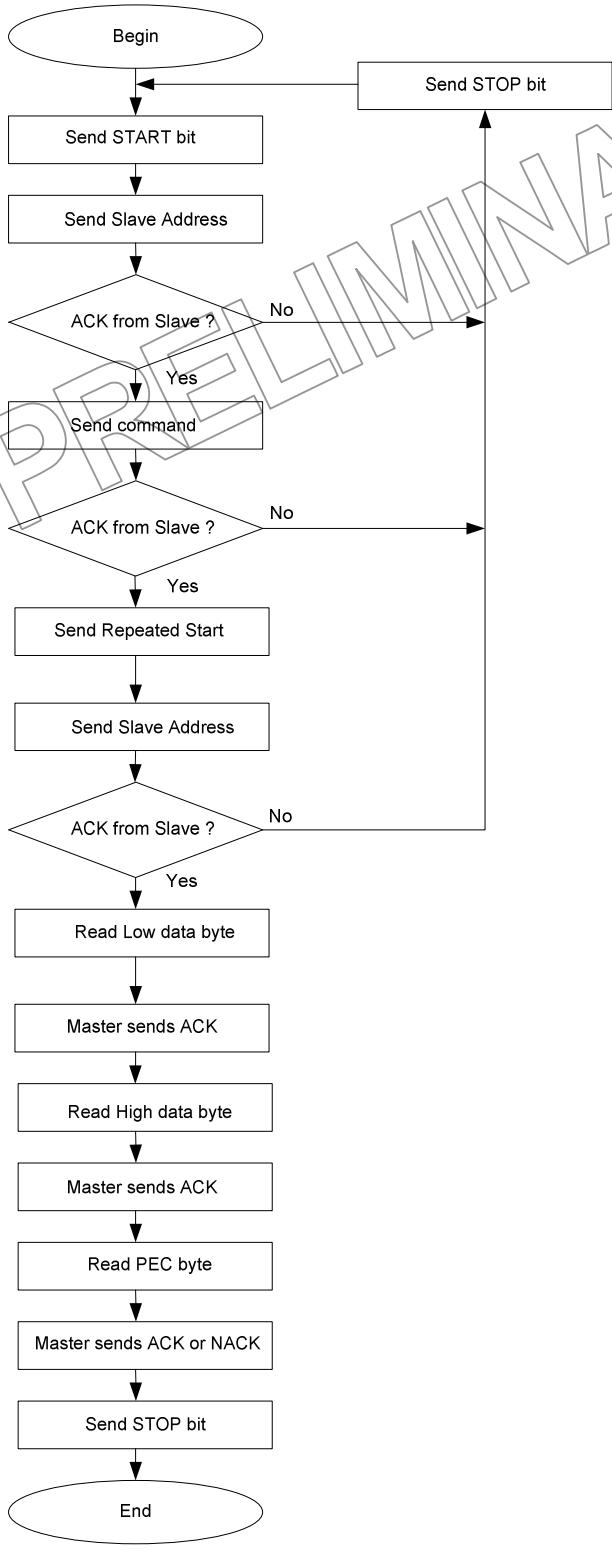
EEPROM memory is accessible for reading without restriction.

<u>Pseudo code example:</u> Reading EEPROM address 0x0E (SMBus Address)

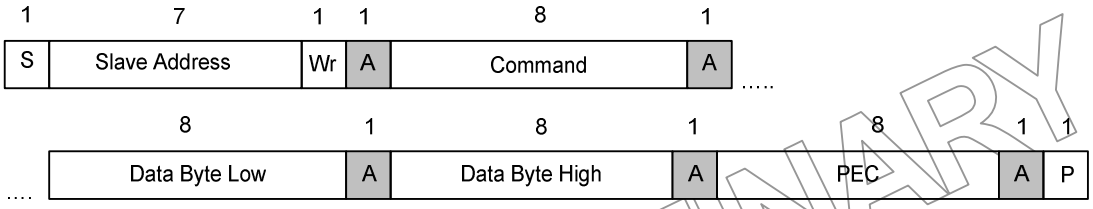
1. Send START bit
2. Send Slave Address (0x00* for example) + Rd\-Wr bit**
3. Send Command (0b001x_xxxx + 0b0000_1110 -> 0b0010_1110)
4. Send Repeated START_bit
5. Send Slave Address + Rd\-Wr bit**
6. Read Data Byte Low (master must send ACK bit)
7. Read Data Byte High (master must send ACK bit)
8. Read PEC (master can send ACK or NACK)
9. Send STOP bit

Note : Any MLX90614 will respond to address 0x00*
*Note**: Bit Rd\-Wrhas no meaning for MLX90614*

Read EEPROM Block Diagram



The format of SMBus writing in EEPROM is:



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 a/an writing/erasing 5ms are need 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)

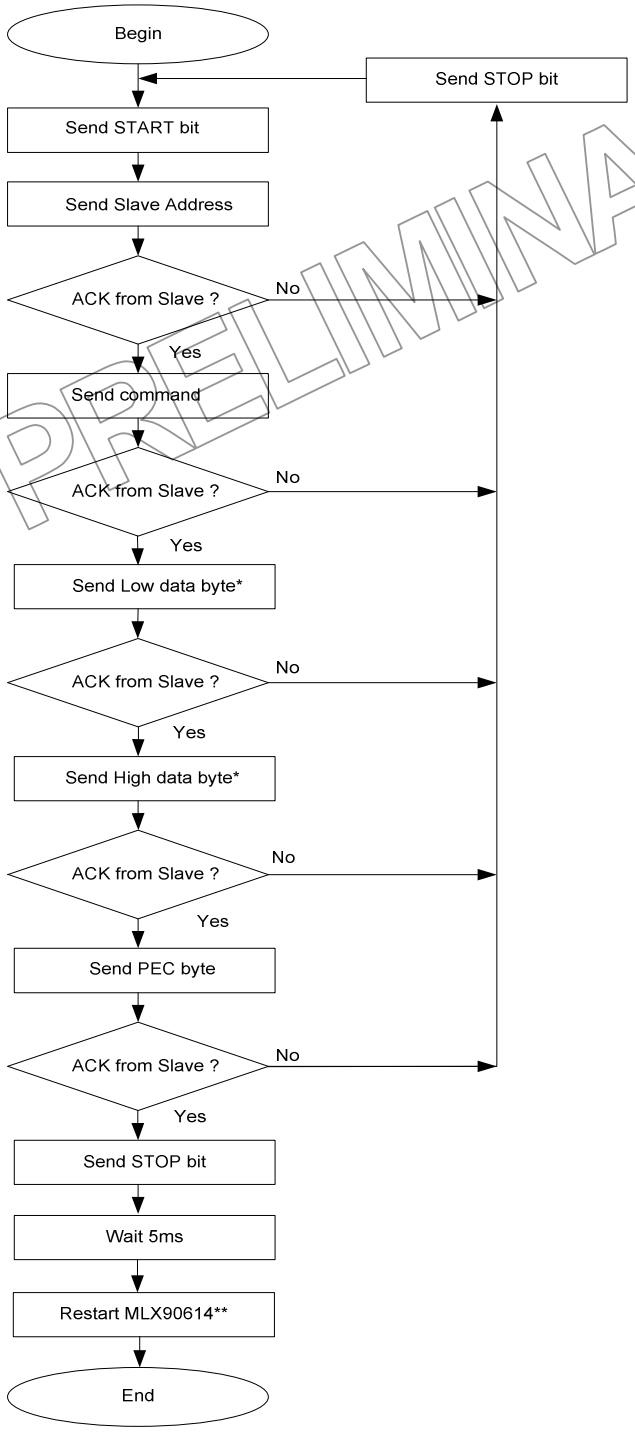
1. Send START bit
2. Send Slave Address (0x00* for example) + Rd\-Wr bit**
3. Send Command (0b001x_xxxx + 0b0000_1110 -> 0b0010_1110)
4. Send Low data 0x00
5. Send High data 0x00
6. Send PEC 0x6F
7. Send STOP bit
8. Wait 5ms (this time is need the cell to be erased)

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 MLX90614 (After this MLX90614 will respond to the new slave address 0x5A)***

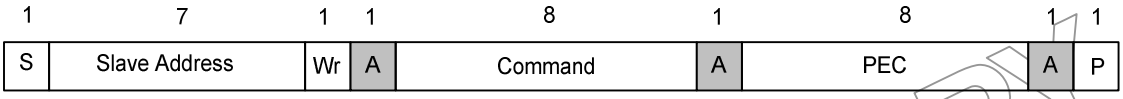
Note : Any MLX90614 will respond to address 0x00*
*Note**: Bit Rd\-Wr has no meaning for MLX90614*
*Note***: Put in/Put out a MLX90614 in/from Sleep Mode also resets MLX90614*

Write/Erase EEPROM Block Diagram



Note : For an erasing operation Low and High data byte must be zero*
*Note**: For an erasing operation no need to restart MLX90614*

The format of SMBus transaction which enters Sleep mode is:

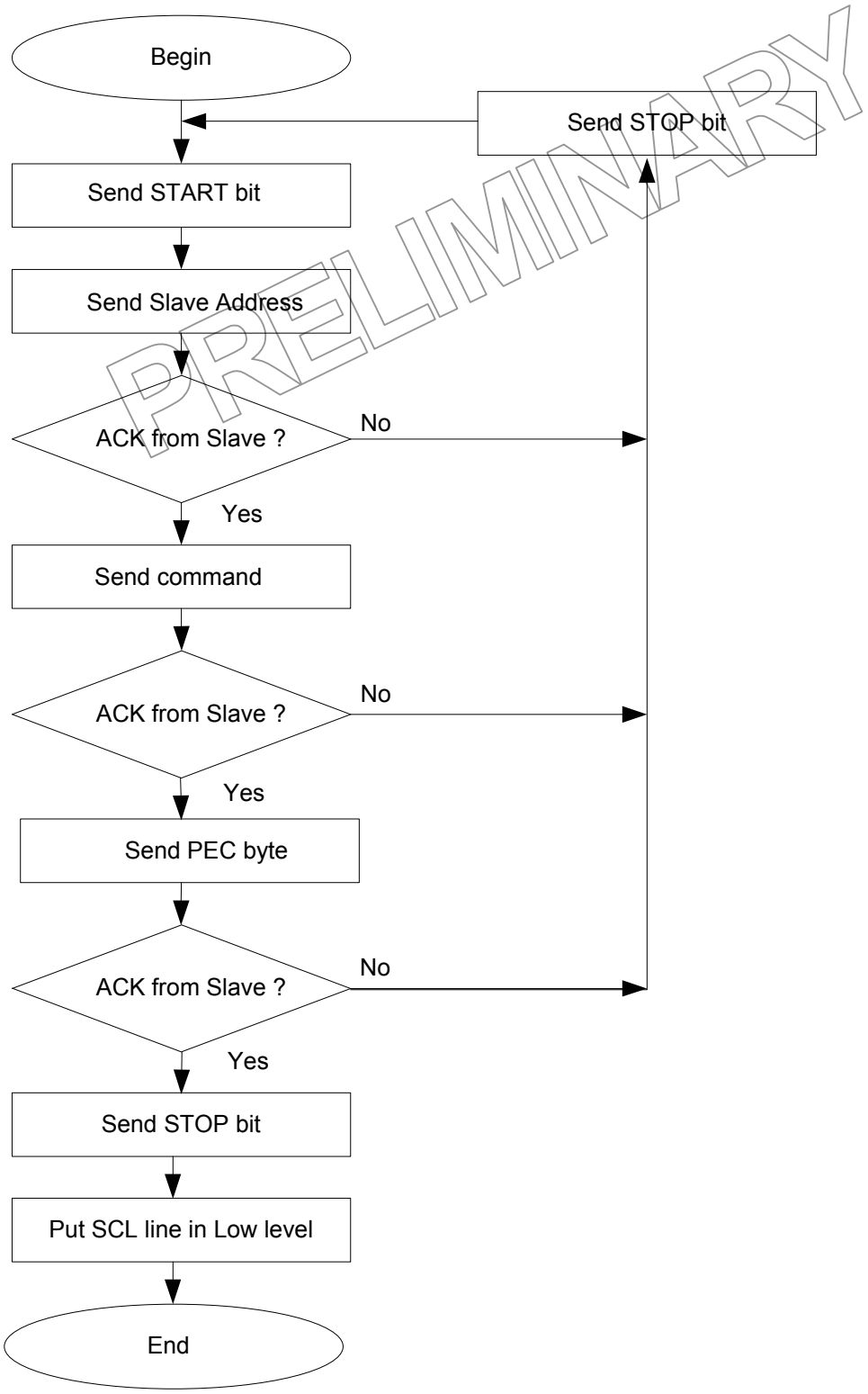


Pseudo code example: Put MLX90614 in Sleep mode

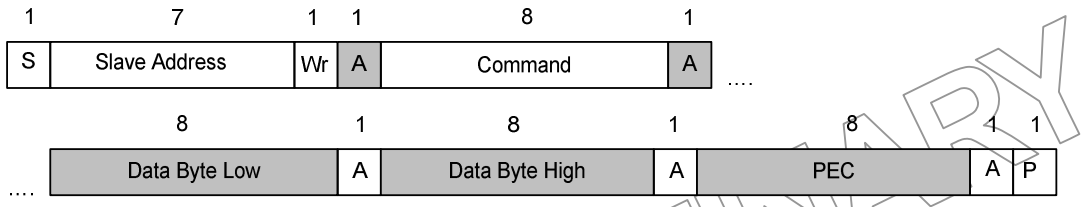
1. Send START bit
2. Send Slave Address (0x00* for example) + Rd\-Wr bit**
3. Send command 0xFF
4. Send PEC byte 0xF3
5. Send STOP bit
6. Put the SCL line in low level

Note : Any MLX90614 will respond to address 0x00*
*Note**: Bit Rd\-Wr has no meaning for MLX90614*

Enter in Sleep Mode Block Diagram



The format of a SMBus transaction which reads flags is:



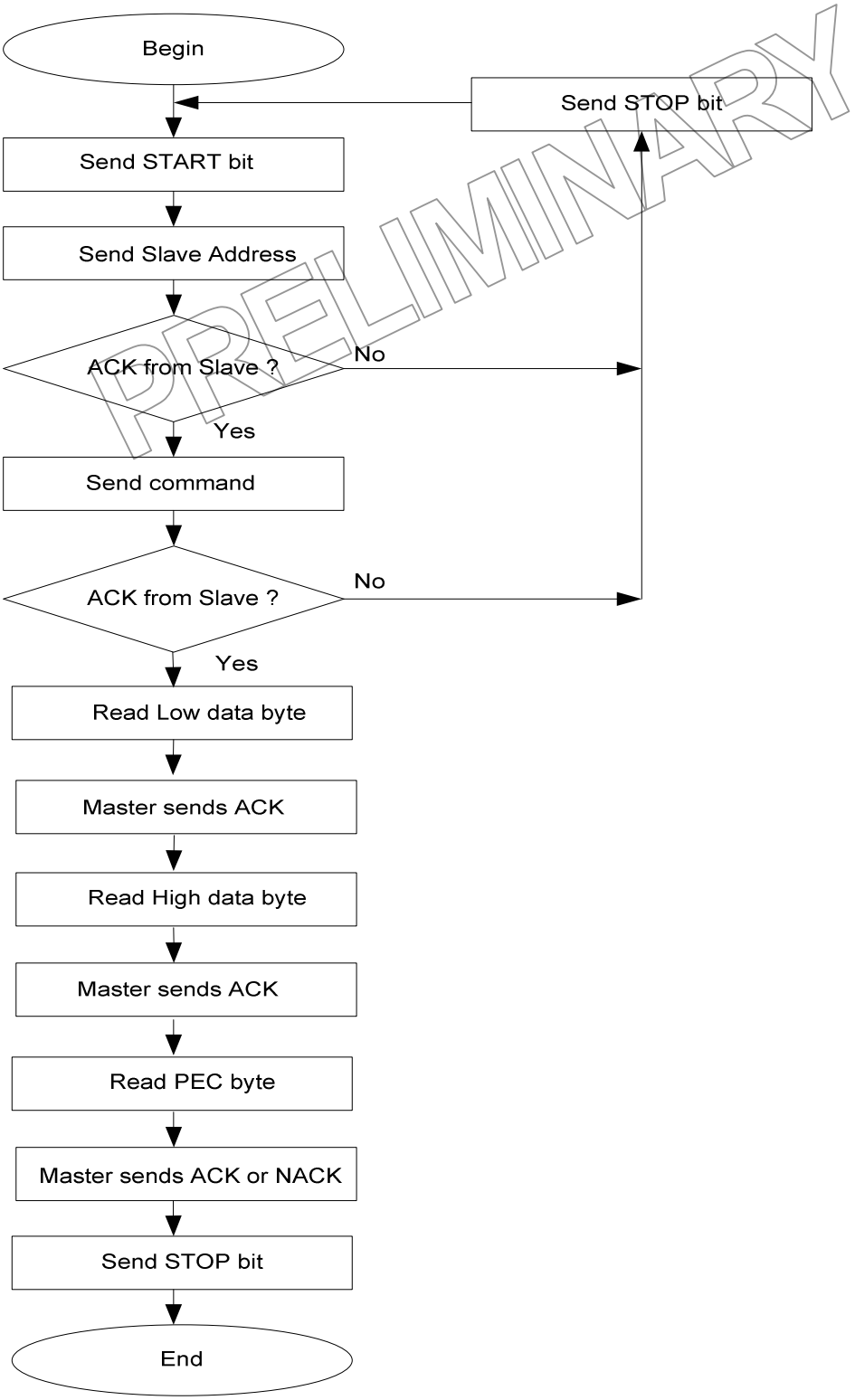
Flags read are:
Data[7] - EEBUSY – the previous write/erase EEPROM access is still in progress. High active.
Data[6] - Unused
Data[5] - EE_DEAD – EEPROM double error has occurred. High active.
Data[4] - INIT – POR initialization routine is still ongoing. Low active.
Data[3] - Not implemented.
Data[2..0] - All zeros.
Data[8..15] - All zeros.
Flags read is a diagnostic feature. The MLX90614 can be used regardless of these flags.

Pseudo code example: Flags reading

1. Send START bit
2. Send Slave Address (0x00* for example) + Rd\-Wr bit**
3. Send Command 0xF0
4. Read Data Byte Low (master must send ACK bit)
5. Read Data Byte High (master must send ACK bit)
6. Read PEC (master can send ACK or NACK)
7. Send STOP bit

Note : Any MLX90614 will respond to address 0x00*
*Note**: Bit Rd\-Wr has no meaning for MLX90614*

Read Flags Block Diagram



7 Sleep Mode

MLX90614 can enter Sleep Mode via command “Enter SLEEP mode” sent via the SMBus interface. This mode is not available for the 5V supply version. To limit the current consumption to 2.5uA (typ), the SCL pin should be kept low during sleep (Fig.9). MLX90614 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}=14ms (Fig.10).

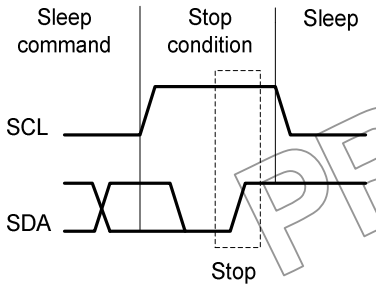


Fig.9

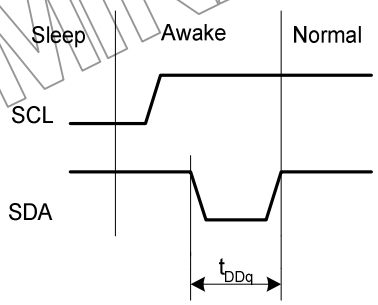


Fig.10

After the module is awakened and before reading the temperature, an additional delay is required or the first measurements will not be correct. This additional delay depends on the FIR filter.
(After wake-up the digital part of the MLX90614 is restarted, IIR =100% for the first temperature calculation flow from the module, see Fig.11. After this first temperature calculation flow from the module, IIR restores its value to the value before the module was put in sleep mode).
See AppNote ”Understanding MLX90614 on-chip digital signal filters” on http://www.melexis.com/Asset.aspx?nID=5272 to understand how to calculate this delay time.

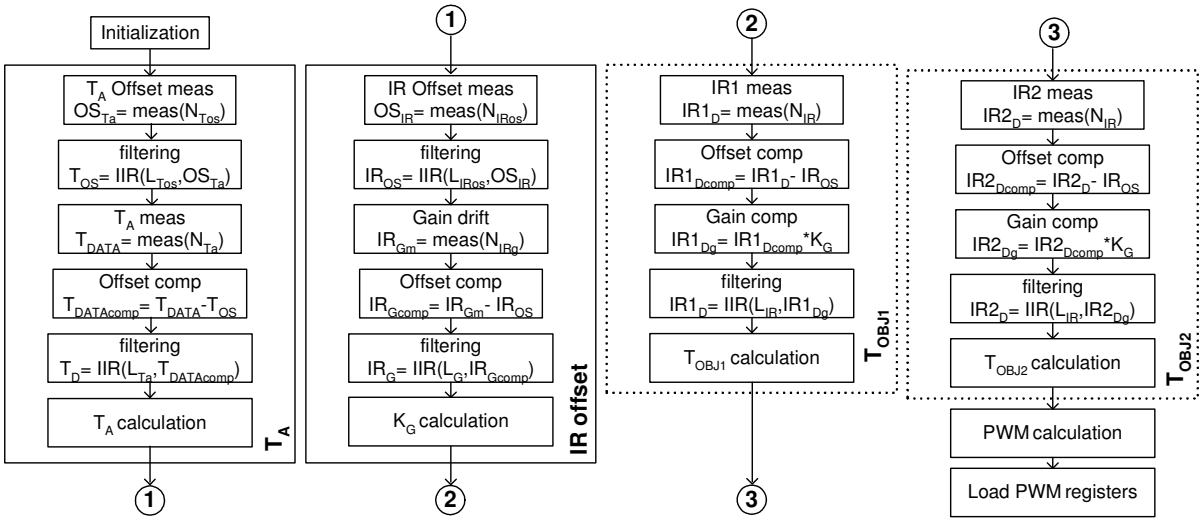


Fig.11

8 Electrical considerations of SMBus applications with MLX90614

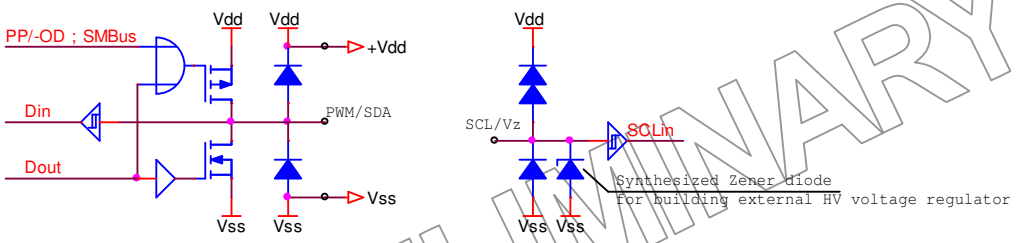


Fig.12: Input/Output pin schematics of MLX90614

For reliability reasons, the MLX90614 incorporates ESD clamp diodes to Vdd and from Vss. Therefore a powered-down MLX90614 will load the SMBus. This differs from the SMBus specification. In power-managed systems it is therefore needed to keep the MLX90614 powered when the SMBus is needed. This is no issue with sleep mode.

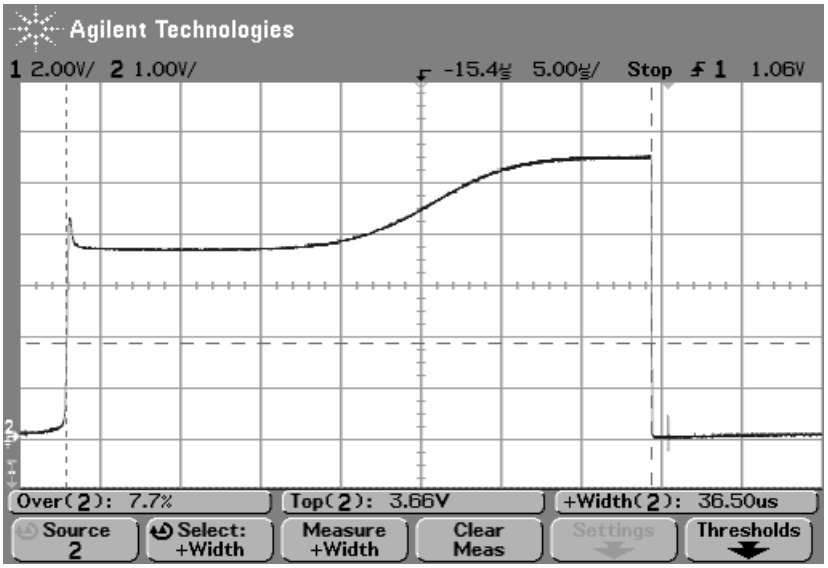


Fig.13: SCL line undershoot with MLX90614

A synthesized Zener diode is integrated on the SCL/Vz pin of the device. It allows simple implementation of higher voltage regulators. There are several things that influence the SMBus applications with that feature:

Transient response of SCL pin of MLX90614 adds undershoot to the SMBus SCL line. Rising edge on the bus results in partial opening of the zener diode, as shown on Fig.13. This undershoot is typically well beyond the threshold level for high-to-low transition on SCL line.

When an external regulator is build the SCL can no longer be used. This would cause the regulator to turn off and on with every clock cycle. Use of MLX90614 in >5V system is shown on Fig. 14. However, if it is needed to have SMBus communication with an MLX90614 already connected to a schematic like this, it will be necessary to override the power supply regulator as shown on Fig. 14. Then the SCL pin can be toggled and the SMBus communication will run.

SCL input leakage is increased. In worst case (over temperature and voltage on the SCL pin) this leakage may significantly exceed the sleep mode power drain of the MLX90614. Sleep

mode is available with the 3V version only, while the zener diode function can be used with the 5V version. However, the zener diode is present in all MLX90614 versions, so the leakage will be seen on the 3V version too. It is recommended to disable the pull-up on the SCL line in order to prevent the leakage from increasing the overall power-down power drain of the SMBus system.

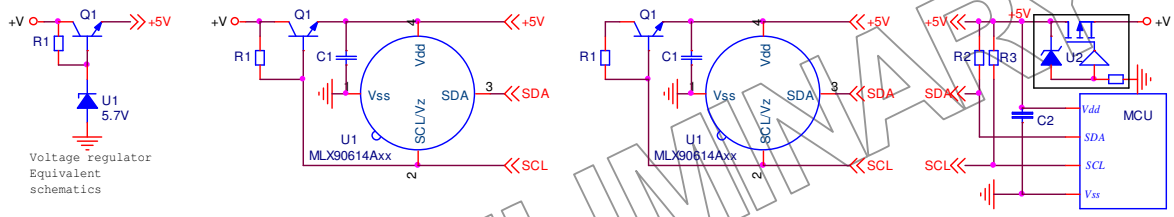


Fig.14: External voltage regulator with MLX90614

The input levels of the MLX90614 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 MLX90614 (refer to the data sheet) the specifications differs. When 5V (also applies for >5V applications) is used, the MLX90614 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 MLX90614 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 MLX90614Axx are Vin,L=0.5…1.5V and Vin,H=1.6…2.4V (over all temperatures and supply voltages), and for MLX90614Bxx – 0.5…1,5V and 1.2…2.8V (idem). However, this does not mean, that MLX90614Axx is likely to have Vin,L=1.5V and Vin,H=1.6V at the same time; also MLX90614Bxx 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 MLX90614, but is not likely to be really necessary. Detailed values (guaranteed by design, not test limits) are given below:

Table 8

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

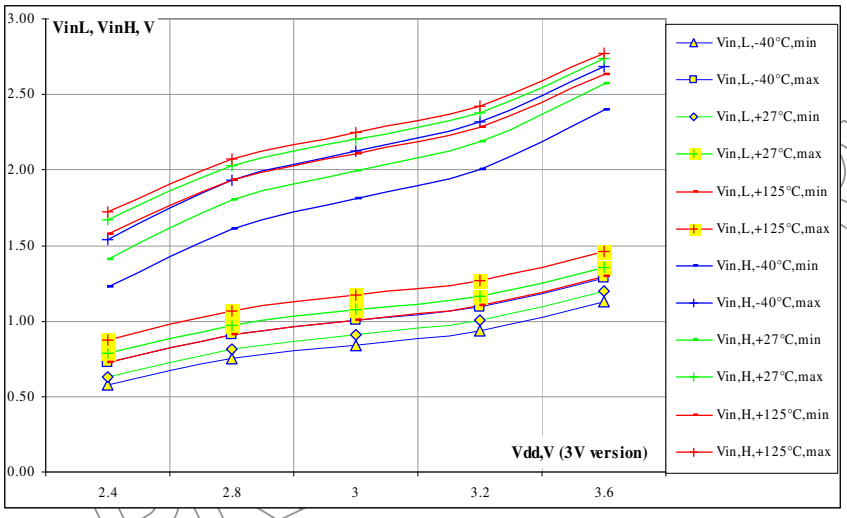


Fig.15: Input voltage levels versus power supply voltage and temperature

Both temperature and supply voltage increase both low and high thresholds.

9 Conclusion

The MLX90614 can easily be used in with the SMBus interface to build a network of sensors. Also for a single sensor the SMBus interface can be the preferred choice for communicating with the application controller.
In APPENDIX there are C functions which can be used to implement all SMBus transaction formats to communicate with MLX90614. Because software implementation of the SMBus is used, it is easy to adapt these functions for different microcontrollers.

♦ 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

//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 Managment 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
}
```

```
//***************************************************************************************************************
//                                          STOP CONDITION ON SMBus
//***************************************************************************************************************
//Name:         STOP_bit
//Function:     Generates STOP condition on SMBus
//Parameters:   No
//Return:       No
//Comments:     Refer to "System Managment 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()

//*****************************************************************************************************************
//                                              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()

//---------------------------------------------------------------------------------------------------------------
```