Application Report
SLAA249B–April 2005–Revised September 2018

Implementing SMBus Using MSP430™ Hardware I^{2}C

Harman Grewal *MSP430*

ABSTRACT

This application report describes the implementation of the system management bus (SMBus) using the MSP430™ hardware I^{2}C peripheral. SMBus is used as a communication link for smart batteries, power-related devices, and a wide variety of other system devices. This report includes the support for master and slave protocols in a SMBus communication system.

The source code described in this application report is available from http://www.ti.com/lit/zip/slaa249. For additional help enabling communication with SMBus devices, see the MSP430 SMBUS Library.

Contents

List of Figures

Trademarks

MSP430 is a trademark of Texas Instruments.
All other trademarks are the property of their respective owners.

1 Introduction

SMBus is a two-wire serial interface based on the principles of I^{2}C. The two lines are serial clock (SCL) and serial data (SDA), which are tied to V_{CC} using pullup resistors. The devices communicating on this bus can drive the lines low or release them to high impedance. This connection is a wired-AND configuration. Multiple I^{2}C or SMBus devices can be connected on the bus, but pins of the MSP430 MCU should not be pulled above V_{CC}. For example, if the V_{CC} of the MSP430 MCU is at 3 V, then all devices on the bus must be pulled up to only 3 V.

A device performing data transfers on the bus can be considered as a master or a slave. Each master and slave device can either be a transmitter (send data) or a receiver (receive data), and the communication on the bus is always initiated by the master by providing a valid start condition and the SCL signal.

Multiple master and slave devices may be connected on the bus, but only one device may master the bus during a data transfer. Since more than one master may simultaneously attempt to take control of the bus and start a transmission, the I^{2}C/SMBus protocol provides an arbitration mechanism that relies on the wired-AND connection of all devices to the bus. A master device that generates a logic high on the SDA bus loses arbitration to a master that generates a logic low on the data bus. The MSP430 MCU master transmitter that loses arbitration switches to slave receiver mode and sets the arbitration lost flag, ALIFG. [1] Each device on the bus has a unique 7-bit address, which allows a total of 128 devices to be connected on the bus. Some addresses are dedicated SMBus addresses that are reserved and must not be assigned to any SMBus device; for example, the SMBus Alert response address (0001 100b). [2]

2 SMBus Protocols

The different communication protocols can be found in the System Management Bus specification. [2] The communication always begins with a valid start condition from the master followed by a 7-bit slave address and the read/write bit that defines the master as a receiver /transmitter respectively, except in the quick command protocol. In quick command protocol, the read/write bit is used to turn a device on/off or enable/disable a low-power mode. The read/write bit is followed by an Acknowledge from the slave. This is followed by 8-bit transfers that may be data, command, or Packet Error Check (PEC). An acknowledge is sent by the receiver after each byte is received. To end the transfer, a valid stop condition is initiated by the master.

The SMBus standard introduced the Packet Error Checking (PEC) mechanism to improve communication reliability. The PEC is a CRC-8 error check byte, calculated on all message bytes except the ACK, NACK, START, and STOP bits. The PEC is added to the message by the transmitter. The PEC in this application report is calculated using a cyclic redundancy check (CRC-8) polynomial, $C(x) = x^8 + x^2 + x^1 + 1$ and is calculated bit by bit in the order of bits received. See the SMBus specification for details on the PEC.

Another optional signal defined in the SMBus standard is the SMBALERT signal. This pin is also pulled up to V_{CC} through a resistor. A slave device can signal the master through SMBALERT to request communication with the master. The master acknowledges such a slave device by sending the SMBus alert response address (0001 1001b) on the bus. The slave device acknowledges this Alert command by returning its 7-bit slave address on the bus and the ALERT signal becomes inactive. The eighth bit can be a 0 or 1. If multiple devices pull the SMBALERT signal low, the lowest address device wins arbitration, and its signal becomes inactive after the corresponding slave address byte is put on the bus.

The SMBus operating frequency range is 10 kHz to 100 kHz. Because a minimum speed needs to be maintained in this communication, a slave can hold SCL low for only a specified amount of time before the master times out and issues a stop condition. A slave can hold the clock low for 25 ms before timeout occurs. After this time, the slave must be able to receive a new start condition within 35 ms. Additional timing information can be found on the SMBus specification site (http://www.smbus.org). [2]

3 Software

The firmware used to test this application is attached in a zip folder (http://www.ti.com/lit/zip/slaa249) along with this application report. Code coverage for this application report is supported in C and assembly using IAR and Code Composer Essentials.

The firmware was tested using the MSP430F169 device and a MSP-FET430P140 target board as shown in Figure 1.

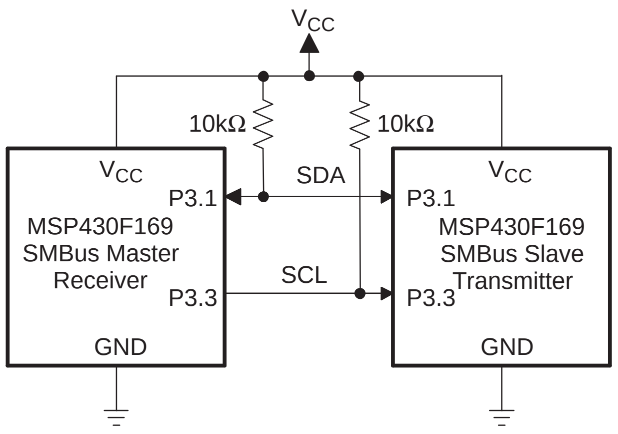

Figure 1. MSP430F169 Master-Slave SMBus Communication

4 Example Code

4.1 Example 1

fet140_SMB_mstr.c(.s43)

The master is the receiver and detects a timeout due to the slave holding the clock line low for a period greater than "timeout". TimerA0 is used to detect this timeout period, and the master issues a stop condition at the conclusion of the byte transfer currently in progress.

The hardware I^{2}C has the built-in I2CBB (Bus Busy) bit that is set after a start condition. I2CBB and I2CSCLLOW may be used to determine how long SCL has been held low after a start condition. The corresponding slave code source file is fet140_SMB_slav.c.

fet140_SMB_slav.c(.s43)

This is the slave code source for the master code in fet140_SMB_mstr.c. TimerA0 is used to generate the delay between consecutive data bytes. This simulates a delay in data handling causing SCL to be held low during that period, exercising the timeout features of the SMBus protocol. Data is transmitted inside the I2C_ISR.

4.2 Example 2

fet140_SMB_mstr_slvrst.c(.s43)

In this example the master device issues a start condition and waits in LPM0 for data reception.

fet140_SMB_slav_slvrst.c(.s43)

This is the slave code source used with fet140_SMB_mstr_slvrst.c (Master). The slave resets the communication port upon detecting a timeout. TimerA1 is used to detect the 25-ms timeout. TimerA2 is used to cause delay in data transfer to hold SCL low. TimerB0 is used as the capture input to detect SCL transitions to start/stop TimerA0 to detect timeout. SCL (Pin 31) is tied to TB0 (Pin 36) to detect transitions.

4.3 Example 3

fet140_SMB_mstr_PEC.c

This is the master code source that reads one data byte and the corresponding PEC byte from the slave. It performs a PEC using a CRC-8 algorithm on the received bytes and slave address.

fet140_SMB_slave_PEC.c

This is the slave code source used with fet140_SMB_mstr_PEC.c. The example writes one byte of data and one byte of PEC from the slave. The PEC byte is calculated using a CRC-8 algorithm on the transmitted byte and slave address.

4.4 Example 4

A test is also performed on the MSP430F169 device and SMBus-compatible TMP175 device. Pullup resistors (10 kΩ) are used on the SCL, SDA, and ALERT lines, as shown in Figure 2.

fet140_SMB_tmp175.c

This is the master code source used to communicate with the TMP175 digital temperature sensor. [3] The TMP175 is a SMBus-compatible slave device. This example sets up the TMP175 in 9-bit temperature mode with interrupt (TM = 1) to test the SMBALERT function. The master reads the temperature from the TMP175. Because the TMP175 is in the interrupt mode, the ALERT pin becomes active when the temperature equals or exceeds T(high) or equals or falls below T(low). The ALERT pin remains active until the device successfully responds to the SMBus alert response address. The alert pin is pulled up to V_{CC} through a resistor. This pin is tied to P2.0 (Pin 20) of the MSP430F169. P2.0 is setup to detect a falling edge transition and respond with the SMBus Alert response command.

TimerA0 is set to have the master periodically send out the start condition to request a new temperature reading from the TMP175.

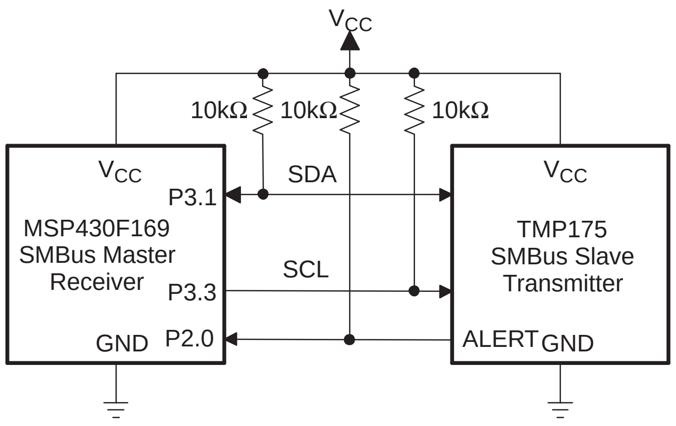

Figure 2. MSP430F169 Master-TMP175 Slave SMBus Communication

5 SMBus Using the USCI I²C Peripheral

This application report has been expanded for SMBus implementation using the newer USCI I²C module on the latest 2xx, 4xx, and 5xx families of devices. This module provides automatic clock activation for use with low-power modes when the module needs the clock for an I²C transaction. See the appropriate device data sheets and family user's guides for more details.

The tests on example 1 through 3 were performed using a MSP430F2619 and TI target board (MSP-TS430PM64). See the *MSP430 Hardware Tools User's Guide* for a schematic of the board.

The following codes (available from http://www.ti.com/lit/zip/slaa249) are used to test this implementation and the newer features and code enhancements are highlighted in each example mentioned below (as applicable).

5.1 Example 1a: Timeout Implementation for Master/Slave on a MSP430F2xx Device

msp430x26x_SMB_MST_timeout.c

The master transmits one byte and receives two bytes from the slave. The master detects a timeout due to the slave holding the clock line low for a period greater than "timeout", which is defined in the SMBUS specification. TimerA0 (clocked by SMCLK/DCO at 1.1 MHz) is used to detect this timeout period (as defined by CCR0 in the code), and the master issues a stop condition when the timeout is detected. An LED is toggled upon a timeout condition.

The corresponding slave code source file is msp430x26x_SMB_SLV_timeout.c.

This code simulates the READ WORD protocol of the SMBus specification.

msp430x26x_SMB_SLV_timeout.c

This is the slave code source for the master code in msp430x26x_SMB_MST_timeout.c. The slave receives one byte and transmits two bytes to the master. After the first byte is transmitted, the transmit interrupt enable (UCBxTXIE) is disabled to prevent the slave from sending the second byte and causing the clock line (SCL) to be held low during that period exercising the timeout features of the SMBus protocol. This creates a timeout condition. The master and slave on detecting the timeout proceed to reset the USCI logic allowing the I2C bus to be released. The functionality following a timeout can be varied as required by the user. On reset of the USCI module, the slave stays in LPM4 waiting for an instruction from the master, making use of the automatic clock activation feature on this device. In order to transmit and receive bytes without simulating a timeout, the user can comment out the following line in the TX ISR in the example:

```
// IE2 &= ~UCB0TXIE;                     // Read the comment below
```

This is also documented in the code.

5.2 Example 1b: Timeout Implementation for Master/Slave on a MSP430F5xx Device

This example demonstrates SMBus READ WORD protocol using the USCI module. The USCI module on the MSP4305xx device family uses a shared interrupt vector scheme that combines the transmit, receive, and status interrupts into a single interrupt vector compared to two interrupt vectors on the 2xx USCI module. A MSP430F5438 device and TI target board (MSP-TS430PZ5x100) was used to test this. See the *MSP430 Hardware Tools User's Guide* for a schematic of the board.

This code is tested with the SMBus compatible BQ27541 device and a MSP430 as a Slave.

msp430x5xx_SMB_MST_timeout.c

The master transmits one byte and receives two bytes from the slave. The master detects a timeout due to the slave holding the clock line low for a period greater than "timeout". TimerA0 is used to detect this timeout period, and the master issues a stop condition when the timeout is detected. An LED is toggled upon a timeout condition.

The corresponding slave code source file is msp430x5xx_SMB_SLV_timeout.c.

This code simulates the READ WORD protocol of the SMBus specification.

msp430x5xx_SMB_SLV_timeout.c

This is the slave code source for the master code in msp430x26x_SMB_MST_timeout.c. The slave receives one byte and transmits two bytes to the master. After the first byte is transmitted, the transmit interrupt enable (UCxxTXIE) is disabled to prevent the slave from sending the second byte and causing the clock line (SCL) to be held low during that period, exercising the timeout features of the SMBus protocol. This creates a timeout condition that is detected by the master, causing the master and slave to reset (or perform any other desired function). An LED is toggled upon a timeout condition. The slave stays in LPM4 waiting for an instruction from the master, making use of the automatic clock activation feature on this device. To transmit and receive bytes in a normal fashion (by avoiding the timeout condition), the user can comment out the following line in the TX ISR from this code:

```
// IE2 &= ~UCB0TXIE;                     // Read the comment below
```

This is also documented in the code.

5.3 Example 2: Implementation of CRC-8 PEC

msp430x26x_SMB_mstr_PEC.c

This is the master code source that reads one data byte and the corresponding PEC byte from the slave. It performs a PEC using a CRC-8 algorithm on the received bytes (Data byte and CRC from slave) and slave address. An improved and more efficient CRC-8 algorithm is used. [4]

msp430x26x_SMB_slave_PEC.c

This is the slave code source used with msp430x26x_SMB_mstr_PEC.c. The example writes one byte of data and one byte of PEC from the slave. The PEC byte is calculated using a CRC-8 algorithm on the transmitted byte and slave address. An improved and more efficient CRC-8 algorithm is used.

5.4 Example 3: Implementation With SMBUS Slave TMP175

This example is implemented on the MSP430F2619 device and SMBus-compatible TMP175 digital temperature sensor device. Pullup resistors (10 kΩ) are used on the SCL, SDA, and ALERT lines, as shown in Figure 2.

msp430x26x_SMB_tmp175.c

This is the master code source used to communicate with the TMP175 digital temperature sensor. [3] The TMP175 is a SMBus-compatible slave device. This example sets up the TMP175 in 9-bit temperature mode with interrupt (TM = 1) to test the SMBALERT function. The master reads the temperature from the TMP175. Since the TMP175 is in the interrupt mode, the ALERT pin becomes active when the temperature equals or exceeds T(high) or equals or falls below T(low). See the TMP175 data sheet for more details. [3] The ALERT pin remains active until the device successfully responds to the SMBus alert response address. The alert pin is pulled up to V_{CC} through a resistor. This pin is tied to P2.0 of the MSP430F2619. P2.0 is set to detect a falling edge transition, and the MSP430 master responds by sending the Alert Response Address (ARA) on the bus. The slave device that pulled the alert line low responds by sending its device address as the data byte. The master then receives the slave device address, determines which slave issued the alert, and proceeds accordingly. This portion has not been implemented in the example code but the I2C transaction framework is provided for the user. See reference [2] for more details on Alert Response Address.

TimerA0 (clocked by SMCLK/DCO at 1.1 MHz) is set to have the master periodically send out the start condition to request a new temperature reading from the TMP175.

6 Conclusion

The hardware I²C peripheral and its built-in features help the MSP430 MCU to easily adopt SMBus protocols. The ultra low-power operation of the MSP430 MCU is useful when interfacing with power management devices such as the smart battery systems used in notebook computers, cameras, cellular phones, or other portable electronic devices on a SMBus network.

7 References

1. MSP430x1xx Family User’s Guide
2. System Management Bus (SMBus) Specification, Version 2, Aug 2000
3. TMPx75 Temperature Sensor With I²C and SMBus Interface in Industry Standard LM75 Form Factor and Pinout
4. CRC Implementation with MSP430 MCUs
5. MSP430 SMBUS Library

Revision History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Changes from October 15, 2009 to September 25, 2018 **Page**

- Editorial changes throughout document 1
- Moved former Appendix A to Section 5 4