Texas Instruments

Application Report

SLUA475–November 2016

SMBus Made Simple

PMP - Battery Charge Management

ABSTRACT

The System Management Bus (SMBus) is the most common form of communication for Texas Instruments advanced fuel gauges. Many customers want to design SMBus engines to communicate with TI advanced fuel gauges. Though this is possible, these designs sometime lead to confusion and frustration. Investigating SMBus errors or transaction failures can seem to be a difficult or daunting task. The purpose of this application report is to reduce the complexity and make learning SMBus easier. This report assumes some knowledge of I²C.

Contents

List of Figures

Trademarks

All trademarks are the property of their respective owners.

1 Getting to Know SMBus

Figure 1 shows some simple examples of generic SMBus transactions. These transactions are read/write words with and without packet error checking (PEC). Although a user's scope traces may not look exactly like these examples, it is easier to look at these theoretical examples and understand their content rather than considering actual scope traces. Examples of actual scope trace are given later in this document. Note that more detailed information can be gathered from these pictures than is discussed in this document. Simplified information is given in order to present only the basics of SMBus information. For most troubleshooting issues, the basics are all that users need to solve SMBus problems.

First, entire packets for read and write are examined. Only word communications are considered because they are common and relevant for most troubleshooting.

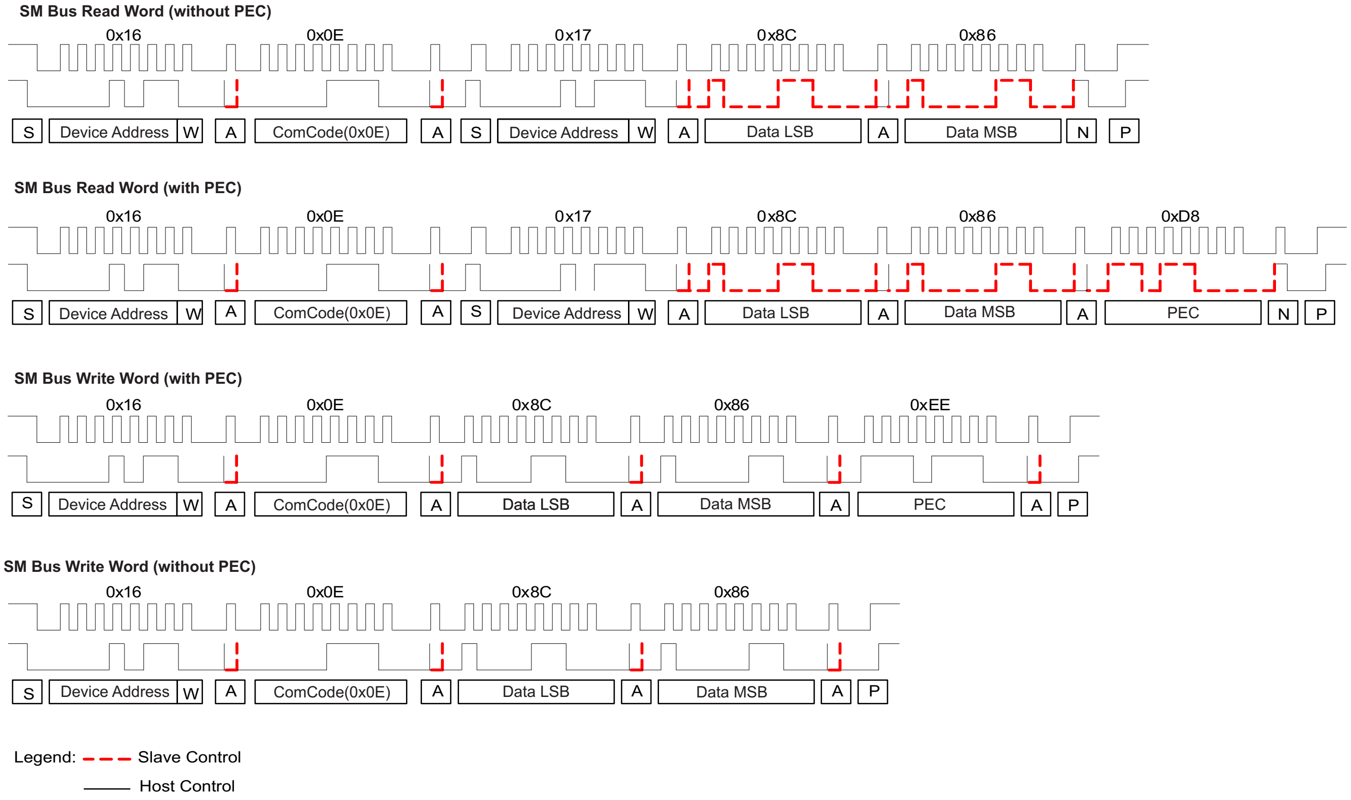


Figure 1. SMBus Transaction Examples

The following components make up the packet along with some of the relevant issues to consider.

- **Start bit:** Each packet of data must start with a start bit denoted with an *S*. The clock must wait at least 4 µs after the data line goes low before it goes low.
- **Device address 1:** The device address is sent by the host telling all slaves on the bus which slave acknowledges this particular communication packet.
  - SMBus can have multiple slaves, so all other slaves that do not have this address ignore the packet. Smart batteries have device address 0x16. Thus, this packet is acknowledged by any fuel gauge.
  - Only one device on the bus can have the same device address.
  - The last bit of the device address is the read/write bit. A 0 for this bit denotes a write, and a 1 denotes a read. The read/write bit in the first device address for a read is a 0 because a command code is being written to the slave first. A write packet has only one device address because the direction (read/write) does not change.
- **Acknowledge:** Denoted by an *A*. The slave must acknowledge that the device address was received.
- **Command code:** This is the command or slave data address that is written to in a write packet or read from in a read packet.

- **Repeated start (read):** Denoted by an *S*. A second start bit embedded within the packet is used to shift the bus to a read.
- **Device address 2 (read):** The second device address in a read packet is a legacy component. Because a read operation is two packets combined with a repeated start, it is not required because the slave responsible for this packet has already been established. The SMBus specification still requires this as part of the specification, so it is mandatory for communicating to all devices including TI fuel gauges. However, important information is in this byte such as the read/write bit, which is set to a 1. This setting tells the slave that this packet is a read, so it is prepared to clock out data.
- **Data LSB:** The first byte of data is the least-significant byte of the data word. The reason why SMBus sends the LSB first is because SMBus sends data in little-endian format. This means that data is sent in increasing numeric significance. Most modern computers store data in memory in this order.
- **Data MSB:** This is the second byte of data for the word sent and is the most-significant byte. Again, it is sent this way to conform to the little-endian format.
- **PEC:** The PEC byte is a checksum of the entire packet used to protect against data corruption.
- **Stop bit:** This is the end of the packet. It tells the slave device that the bus is done, so the slave can get ready for more communications. It is an important part of the packet. Users can experience trouble by leaving this stop bit off if they get all the data. Although TI fuel gauges will time out and reset eventually without this, it is important to keep all devices on the bus in a known state at the end of each packet sent. Even if the host has to stop the communication in the middle of the packet for some reason, the host always sends the stop bit to reset everything on the bus.

1.1 Closer Inspection

To consider SMBus communication in more detail, Figure 2 shows an SMBus read word and zooms into one byte of a data packet and the NACK/Stop bit. This diagram gives examples of most of the important bits of a total packet.

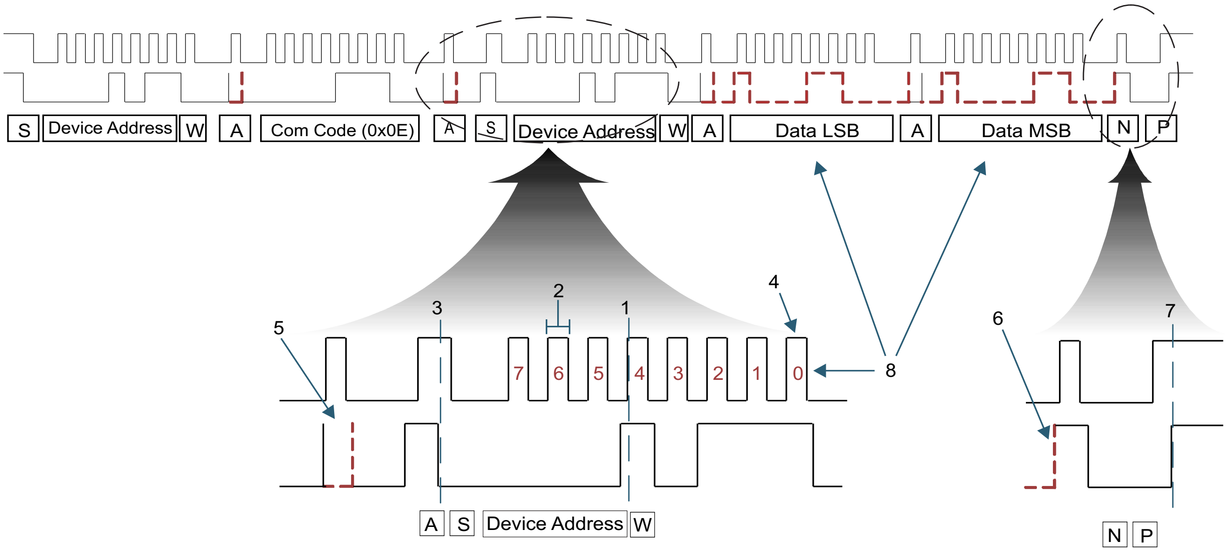


Figure 2. SMBus Read Word – Without PEC

Each byte is 8 bits long. Several things of interest can be derived by looking closely at this diagram:

1. **Data processing:** Each bit of data is processed by the SMBus engine on the rising clock edge. This is where the data is shifted into the engine. Note that the data must never change levels while the clock is high during an SMBus transaction except to create a start, restart, or stop bit.
2. **Clock timing:** The most common cause of difficulty with the SMBus is when host systems fail to follow the SMBus High clock timeout specification. If the clock is high at any time during a transaction for more than 50 µs, the SMBus engine interprets this as a bus idle condition and resets. This SMBus specification requirement can be more problematic than any other.
3. **Repeated start:** The repeated start bit is unique in that it shifts the focus of the current transaction

from a write to a read. Prior to the repeated start is a write to a command code with the read/write cleared in the device address, and after the repeated start, the bus shifts to a read of data with the read/write bit set.

4. **Read/write bit:** This bit is appended to the end of the device address. The device address is usually thought of as being 8 bits long, but it is actually 7 bits. So, the device address in an 8-bit format is a 0x16 in a write and a 0x17 after the repeated start in an SMBus read packet.
5. **Acknowledge:** All bytes are followed by an acknowledge (ACK) except for the last byte of a read packet when the host is responsible for NACK-ing the last byte. The slave expects a NACK of this byte even if it is a PEC byte (PEC is explained later in this document). Whoever receives the byte prior to the ACK is who is responsible for sending the ACK.
6. **No acknowledge:** A no acknowledge follows any byte that is not understood by the device receiving the previous data byte. The exception to this rule is the NACK required from the host after the last byte of data in a read packet (see number 5), which indicates to the slave that the host has received all bytes that it expected.
7. **Start and stop bit:** The stop bit is the final bit in the packet. Once this bit is sent by the host, the slave ignores anything on the bus until a start is detected and then only acknowledges its own device. By the SMBus specification, the fuel gauge must always acknowledge its device address.
8. **Bit order versus byte order:** This is important because the orders are opposite, which can be confusing. Each byte starts with the most-significant bit first and ends with the least-significant bit. However, the word of data is sent with the least-significant byte first and the most-significant byte last, which the SMBus specification requires. The bitwise order is normal; however, the bytewise order is in little-endian format as previously explained.

1.2 Final Considerations

The following are final considerations when examining all the points of a total packet.

1.2.1 Clock Stretching

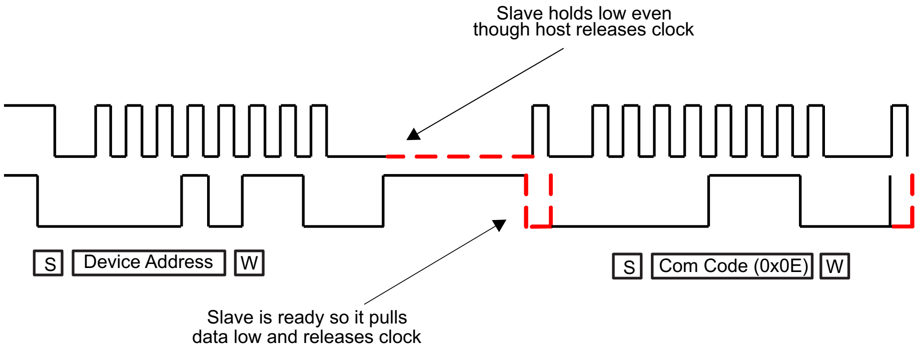

Slave holds low even though host releases clock
S
Device Address
W
S
Com Code (0x0E)
W
Slave is ready so it pulls data low and releases clock

Figure 3. SMBus Clock Stretch

Clock stretching is a simple way for the slave to indicate to the host that it is busy (see Figure 3). Any time the clock is low in the packet, the slave has the right to grab the clock and hold it low as long as required; however, the bit must not cause a packet timeout (25 ms).

The entire packet must not be longer than 25 ms, which includes the clock stretch. TI fuel gauges usually only clock stretch before or after the ACK or NACK bits.

1.2.2 Broadcasting (Master Mode Messages)

Sometimes, a slave can become the master of the bus. The SMBus specification allows for this possibility. All of TI's advanced fuel gauges have this ability. Depending on the fuel gauge (FG), this feature can be enabled or disabled in different ways. If enabled, the FG as master causes the bus to send alarms and charging voltage and charging current every 10 to 20 seconds to the host (0x10) and charger (0x12) device addresses

An SMBus master can only start a packet if the SMBus has been idle for more than 50 µs. Once this requirement has been met, the master immediately takes control of the bus by sending a start bit. All TI FGs function exactly like this, and because they are hardware-controlled SMBus engines, the FGs easily detect idle.

This type of behavior is difficult to create for an SMBus engine implemented in firmware. It is easy to detect 50 µs of idle time, but the port pins used to create the bus must be switched to outputs, and the start bit must be sent. During this time frame, another master can actually take control of the bus. If another master controls the bus, the firmware-controlled bus does not detect that the bus is no longer busy, which causes arbitration to be lost.

Because of this difficulty, most host systems do not recognize this part of the SMBus specification and, instead, act as the only master on the bus. Therefore, TI recommends that broadcasting messages be disabled for every application unless absolutely necessary.

1.2.3 PEC

PEC is a simple form of a checksum used for error checking. It is important to use PEC in all communications to ensure what was sent or received was actually intended. PEC is really just an extra byte of data added to the end of the communication packet that is derived from a simple CRC-8 checksum. All TI fuel gauges that support PEC also have the option to add PEC to the broadcast data if desired. However, most customers never use the broadcasts. Broadcasts must be completely disabled if not used.

A common question is *Who sends (is responsible for) the PEC byte?* For this discussion, assume the master is the host system (notebook, PC, or another host), and the slave is the fuel gauge. In read operations, the slave (fuel gauge) is responsible for sending the PEC packet to the host. Then the host determines if the PEC is valid. In a write operation, the host is responsible for sending the PEC to the slave. Though some slave devices may not be fast enough, a TI fuel gauge always NACKs the PEC byte if an error is in the packet, which can sometimes be confusing. Therefore, a simple and easy way to remember this is that the SMBus device, which is responsible for sending the data of the last byte of the packet, is the one responsible for the PEC.

The SMBus specification has much more information about PEC relating to protection against devices that do not perform the PEC function reliably; however, they do not apply to TI fuel gauges. TI parts use a hardware PEC lookup; therefore, software does not interfere with the process. With TI fuel gauges, an ACK to a bad data packet's PEC byte will never be occur, instead, a NACK to a bad data packet occurs.

1.2.3.1 How to Calculate PEC

PEC calculations take some resources by the host system unless it has a hardware PEC engine. Although this document does not detail about how to do these calculations, many resources are available on the Internet, which include explanations and even code examples. Do an Internet search for PEC or CRC-8 computation. For an example of an online calculator that can check your code, refer to http://smbus.org/faq/crc8Applet.htm.

The following are two primary methods to calculate a PEC for a given data packet. Depending on your host CPU and memory, one of these methods should work for your application.

1. A lookup table method is time efficient. However, it requires a large data memory to implement.
2. Direct calculation of the PEC is simple to understand and takes little program memory; however, it is an iterative process that takes CPU time.

Consider the use of a logic analyzer or bus snooper for PEC calculations. Though a logic analyzer or a *bus snooper* may appear to be the simplest tools to monitor the SMBus traces, the following are a few of many reasons why an oscilloscope is preferable.

1. A logic analyzer only shows a vague or high level piece of what is actually on the bus (see Figure 4). A logic analyzer shows only transitions, not rise times, noise, or any other electrical aspects of the bus. It is necessary to see all the information when troubleshooting. A logic analyzer is a fine tool to see what is happening on a bus that has no issues. However, when working with critical communication failures, more detail is required than is available from a logic analyzer.

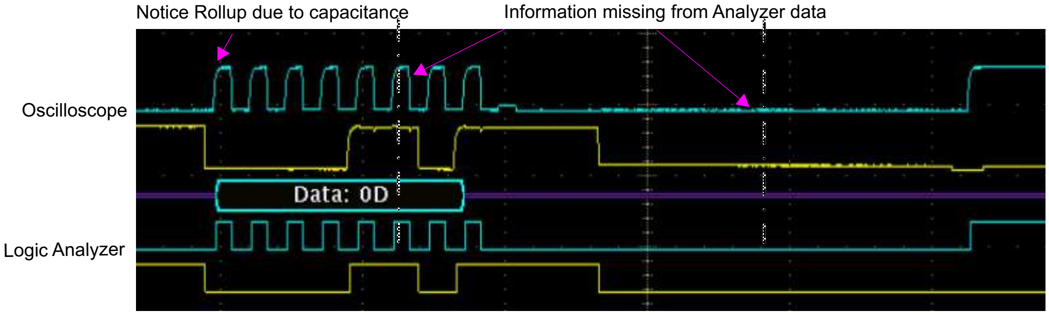

Notice Rollup due to capacitance
Information missing from Analyzer data
Oscilloscope
Data: 0D
Logic Analyzer

Figure 4. Oscilloscope Versus Logic Analyzer Comparison

2. A bus snooper is a tool that logs communications, and in that log, reports ASCII text representing what is happening on the bus. It gives information at an even higher level than the logic analyzer. The following are several lines out of a log file:

```
Msg 11 [S]#16 [A] #0E [A][S] #17 [A] #8C [A] #86 [A] #D8 [N][P]

Msg 12 [S]#16 [A] #0E [A][S] #17 [N]
```

It is apparent that this communication has a problem: the device NACKed its device address after the repeated start.

1.2.4 Examples

The following four examples illustrate why an oscilloscope is so useful. An oscilloscope is the tool that provides adequate information to calculate a PEC for a given data packet. Figure 5 through Figure 10 are examples of data that support this point.

1.2.4.1 Example 1

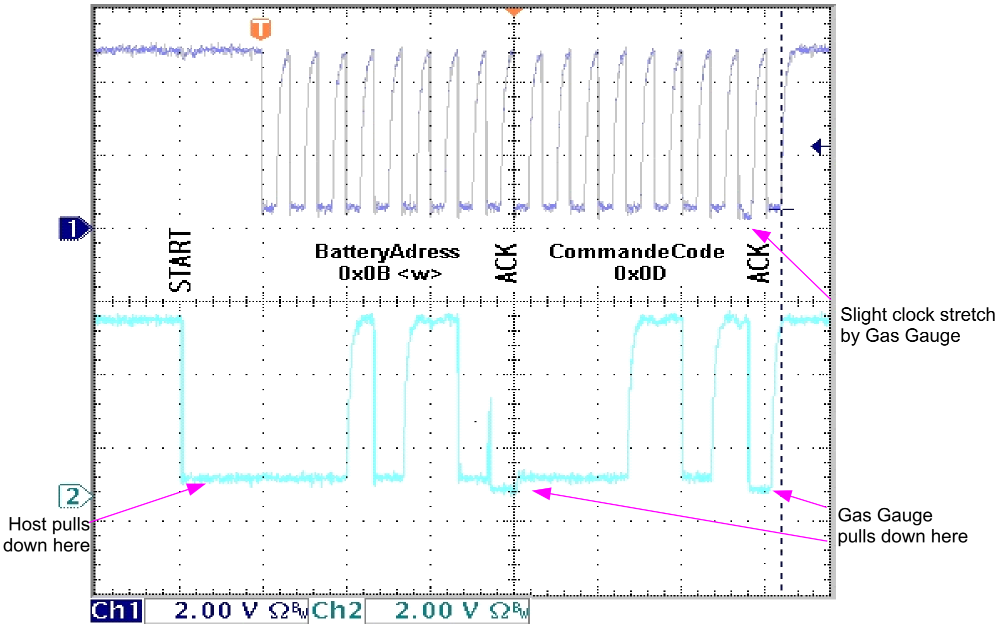


Figure 5. Example 1

Example 1 is shown in Figure 5. Notice that one can actually determine which device is pulling down on the bus at any given time. The *zero* threshold for the fuel gauge is at a lower voltage than the *zero* threshold for the host. The oscilloscope is an extremely useful tool when debugging to determine which device is in control of the bus. The reason this works is due to the current flowing through series protection resistors on the bus. The *zero* threshold is slightly different depending on which side of these series resistors the scope probe is connected with reference to ground.

Figure 6 is a modified picture showing different connection points for an oscilloscope probe. Each point has a different *zero* level voltage in reference to ground. The difference also varies depending on the resistors used in the circuit. This original picture was derived from the SMBus specification.

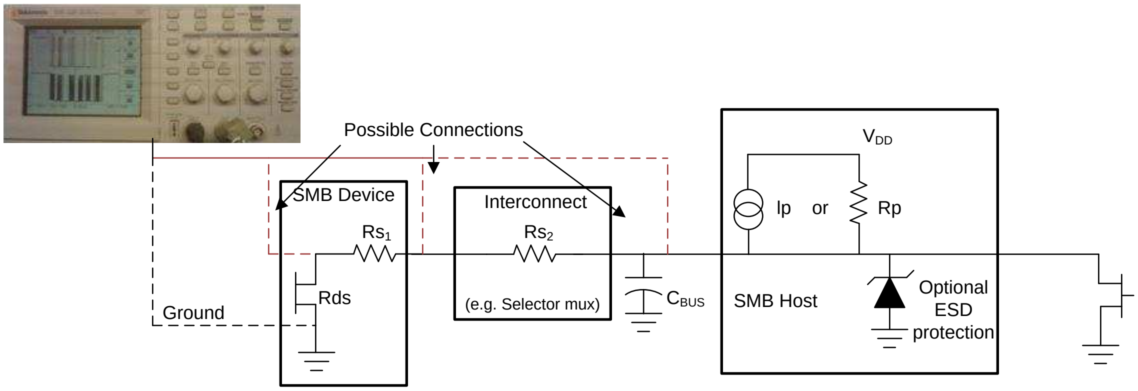
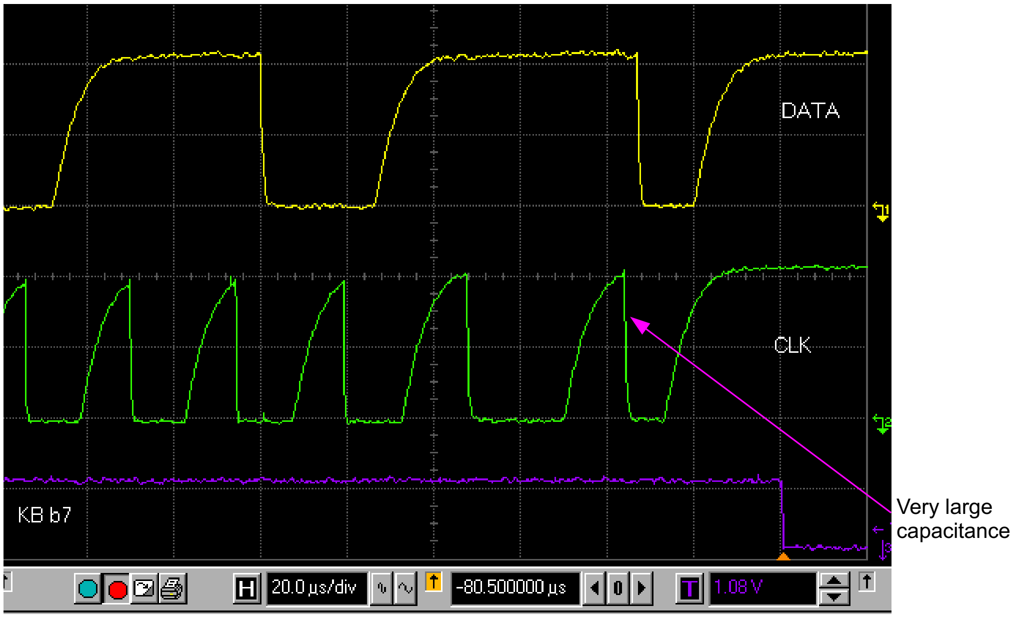


Figure 6. Different Connection Points for an Oscilloscope Trace

1.2.4.2 Example 2



Very large capacitance

Figure 7. Example 2

The problem associated with Example 2 (shown in Figure 7) was a matter of receiving incorrect data. Although almost every bit received was correct, an incorrect bit was occasionally received. As pointed out in Figure 7, a large capacitance on the line sometimes caused a 1 to be read as a 0 because the data line was just on the edge of going up fast enough to be a 1 by the time the clock went to a 1. In Example 2 protection diodes were used on the bus that had a very high capacitance. By removing the high-capacitance protection diodes and using a more common protection circuitry as shown in Figure 8, the problem was solved.

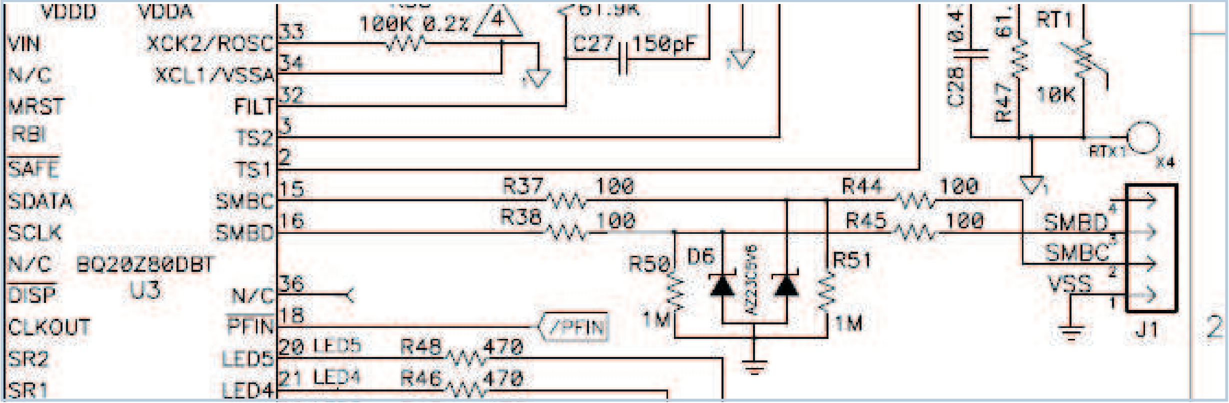
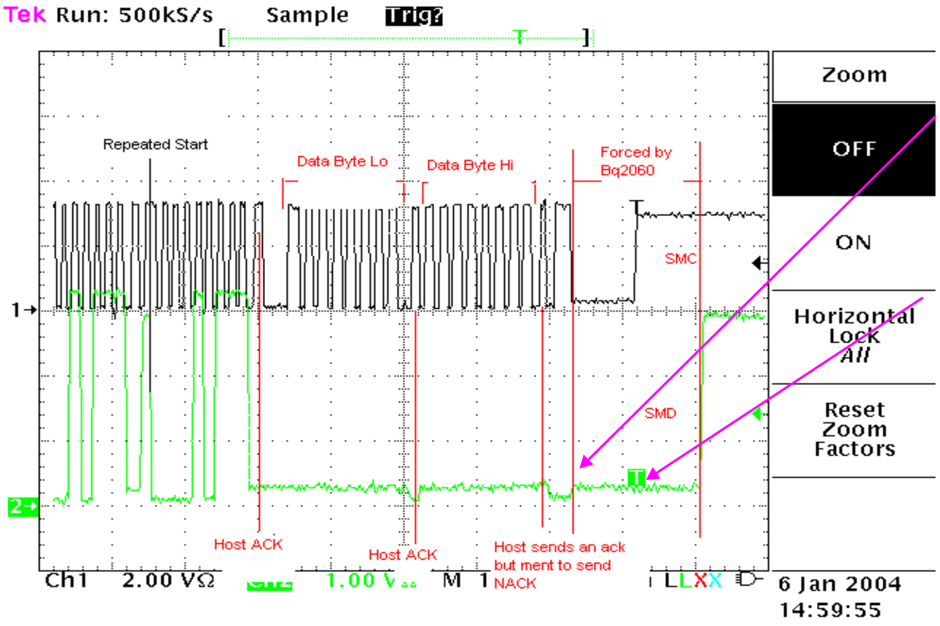


Figure 8. Common Protection Circuitry

1.2.4.3 Example 3



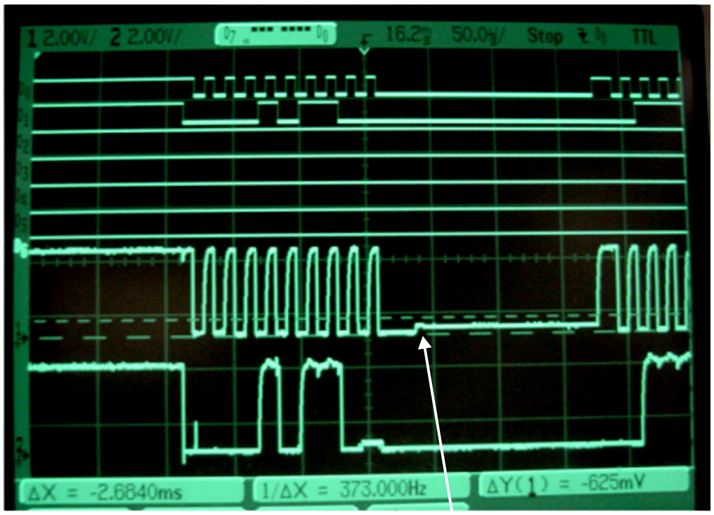
The “zero”voltage level c early shows the host is sending an ACK

Bq2060 holds SMD low waiting for clocks

Figure 9. Example 3

Figure 9 shows Example 3, an oscilloscope plot of a read word. This data poses the question of why the bq2060 was pulling the data line low as shown with the arrow. Available snooper data showed an ACK and then a STOP bit. The problem was the host did not intend to send a PEC byte, but because it sent an ACK (host pulled data line low at the clock), the bq2060 interpreted this as *send another byte of data*, which in this case, was the PEC byte. Therefore, the bq2060 held the data line low (trying to send a 0), waiting for clocks from the host. The host tried to send a stop bit because of the confusion. The solution to this issue was to make the host send a NACK after the high byte of data.

1.2.4.4 Example 4



Notice the very long clock stretch by the slave here

Figure 10. Example 4

The long clock stretch shown in Example 4 (Figure 10) is uncommon but still easily complies with the SMBus specification (25-ms packet length). Example 4 is a clear example of a clock stretch. As in this example, most clock stretches happen somewhere around the ACK and NACK clocks.

2 Most Common Problems

1. **Capacitance:** It is important that the data line is clean and well within an intended defined state when the clock pulse goes high; otherwise, results can be unpredictable. Capacitance or a weak pullup can cause troubleshooting difficulties because of inconsistent effectiveness. Rolloff caused by capacitance can cause the data line to be in an unknown state when the clock goes high, which causes confusion to the SMBus engine.
2. **Clock high time:** This is one of the more common overlooked problems. The clock must be high for less than 50 µs during the middle of a communication packet. This rule has no exceptions. If the clock is high longer than 50 µs, then the SMBus engine on the slave most likely times out.
3. **Communicating too fast:** This is common when using an I²C hardware engine to perform the SMBus communications. The SMBus engine is only specified at 100 kHz. Communicating faster than this causes *timing minimum* rules to be violated.
4. **Broadcast or other collisions:** In most TI advanced fuel gauge solutions, slave broadcasting (sometimes called master mode messaging) is disabled by default. This problem has been greatly reduced in recent years. Because most hosts or chargers do not accept broadcast messages, it is good practice to disable them if they are not being used.
5. **Wrong-sized pullup resistors:** Overall resistance must be approximately 10 kΩ for the SMBus clock line and 10 kΩ for the SMBus data line. Multiple masters can cause problems here because the resistors are in parallel, which reduces the overall resistance on the line. Whenever the SMBus line is pulled low, poor *low* voltage is noticeable. Remember to test the resistance using an oscilloscope on the receiving end of the line.
6. **Too-fast data to clock transitions:** This is often a problem on firmware-controlled host engines. For most high-speed processors, there must be enough *NOP* instructions between the data line going high and prior to the clock going high on each bit. This is especially important on the start, stop, and repeated start bits..
7. **Bad PEC computation:** To date, no TI fuel gauges that support PEC have been proven to produce an incorrect PEC computation. If you are getting an error because the PEC is incorrect, then most likely it is an error in your PEC computation code. Check for *roll off* or *carry* errors.
8. **Non-SMBus-compliant host:** Many failed communications occur because the host is not SMBus-compliant, and the user cannot change the host design. TI SMBus engines are mostly hardware engines and allow little manipulation. Therefore, it is important to ensure that the host is SMBus compliant prior to releasing the product.
9. **Using an existing I²C engine to run SMBus:** This usually works well, but the user must be attentive to the I²C speed. Many modern I²C hardware ports on microcontroller (MCU) units can run at 100 kHz or 400 kHz. Always use 100 kHz. Even then, the port may exceed 100 kHz at time. Ensure that the clock cannot be high for more than 50 µs.
10. **Host does not allow clock stretching:** This is especially a problem if the host is firmware driven. The host must check the clock after it sends it high to verify that it actually went high. If it does not do this, then the slave may be trying to slow the clock down and it ignores this and continues. This causes loss of arbitration, and both master and slave get confused.

3 Glossary

ACK: A bit in the SMBus communication packet used to signify an acknowledgment of the previous byte of data.

Broadcast: A term to indicate when the TI fuel gauge becomes a master on the bus and broadcasts information to the host as a master. This is also sometimes called master mode messaging.

FG: Acronym for fuel gauge

LSB: Acronym for least-significant byte; two bytes of data.

Master: The device on the SMBus that controls the current communication packet. The master controls the clock line on the bus for any given packet.

MSB: Acronym for most-significant byte; two bytes of data.

NACK: A bit in the SMBus communication packet used to signify a no-acknowledgment of the previous byte of data. This abbreviation usually signifies an SMBus error, or it comes at the end of the last byte of a read communication packet.

Packet: A complete SMBus transaction, either read or write, from the start bit to stop bit.

Repeated start: A second start bit in the communication packet used in a SMBus read to transition from writing the SMBus command code to reading data from that command code.

Slave: This is the opposite of the master. The slave does not control the clock line except for clock stretching used to slow the transaction down to allow the slave more time, if needed.

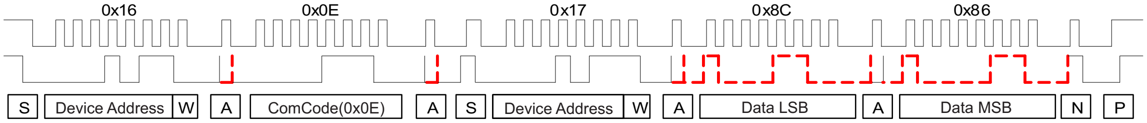
Word: For this document, an SMBus word signifies two bytes of data (0xFFFF), translated from 0–65535 unsigned or –32768 to 32767 signed

4 References

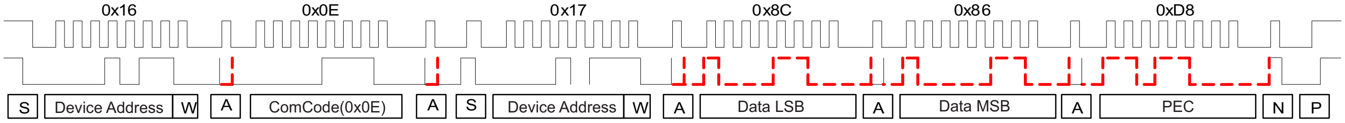
1. SMBus, *CRC-8 Calculator* (http://smbus.org/faq/crc8Applet.htm)
2. SBS Implementers Forum (SBS-IF), *Smart Data Battery Specification* (http://sbs-forum.org/specs/sbdat110.pdf)
3. SBS Implementers Forum (SBS-IF), *System Management Bus (SMBus) Specification* (http://www.smbus.org/specs/smbus20.pdf)

SMBus Reference Sheet

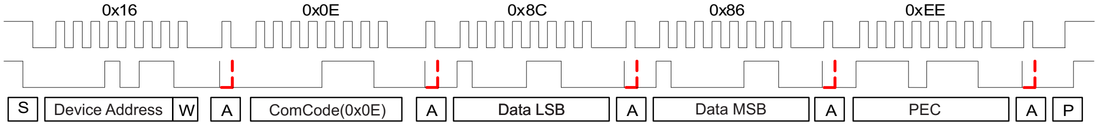
SM Bus Read Word (without PEC)



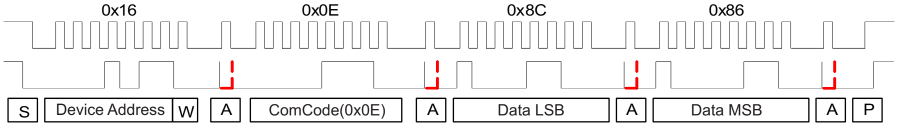
SM Bus Read Word (with PEC)



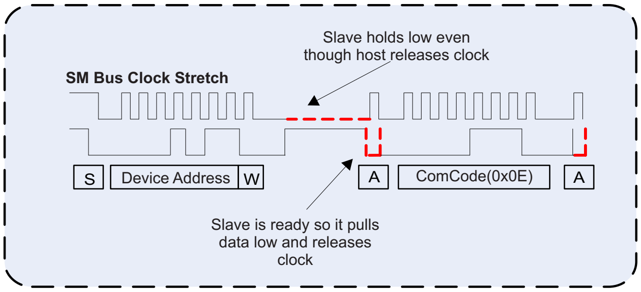
SM Bus Write Word (with PEC)



SM Bus Write Word (without PEC)



Legend: – – – Slave Control
—— Host Control


SM Bus Clock Stretch
Slave holds low even though host releases clock
S
Device Address
W
A
ComCode(0x0E)
A
Slave is ready so it pulls data low and releases clock

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES "AS IS" AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI's products are provided subject to TI's Terms of Sale or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI's provision of these resources does not expand or otherwise alter TI's applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265