USB Interface Adapter Evaluation Module

User's Guide

Literature Number: SLLU093
August 2006

Texas Instruments

Contents

List of Figures

List of Tables

Read This First

About This Manual

This user's guide describes the functions and operation of the USB Interface Adapter evaluation module, from different aspects of hardware design, firmware programming, communication protocols, GUI and PC libraries, etc.

How to Use This Manual

This document contains the following chapters:

- Chapter 1 – Introduction
- Chapter 2 – Summary of Hardware Design
- Chapter 3 – Firmware Programming and Communications Protocol
- Chapter 4 – Firmware Versioning Scheme
- Chapter 5 – GUI and PC Libraries
- Chapter 6 – Schematic, Bill of Materials, and PCB Layout

Information About Cautions and Warnings

CAUTION

This EVM contains components that can potentially be damaged by electrostatic discharge. Always transport and store the EVM in its supplied ESD bag when not in use. Handle using an antistatic wristband. Operate on an antistatic work surface. For more information on proper handling, refer to SSYA008.

WARNING

By default, the I^2C data and clock lines are pulled up internally to 3.3 V. If internal pullup resistors are not used and external ones are used instead, please make sure to pull up to 3.3 V only. Operation from 5 V is not specified and may permanently damage this USB Interface Adapter EVM.

The information in a caution or a warning is provided for your protection. Please read each caution and warning carefully.

FCC Warning

This equipment is intended for use in a laboratory test environment only. It generates, uses, and can radiate radio frequency energy and has not been tested for compliance with the limits of computing devices pursuant to subpart J of part 15 of FCC rules, which are designed to provide reasonable protection against radio frequency interference. Operation of this equipment in other environments may cause interference with radio communications, in which case, the user, at his/her own expense, will be required to take whatever measures may be required to correct this interference.

Introduction

This EVM serves as an interface adapter or a bridge between a host PC (IBM™ compatible) and one or multiple slave devices via a standard type-A to mini-B USB cable. The communication between the USB interface adapter and the host PC is via USB, while the communication between the USB interface adapter and the slave device(s) is via an inter-integrated circuit (I^2C) and/or general-purpose inputs/outputs GPIOs. The bridge converts communication transactions between the USB and I^2C/GPIO.

Note: For the USB interface adapter to communicate with multiple I^2C slaves, build a special 10-pin ribbon cable with multiple connector(s), so that all the I^2C slave devices can be daisy-chained together on the same ribbon cable.

1.1 List of Hardware Items for Operation

In order to operate this USB interface adapter, the following items are required (see Figure 1-1):

- USB interface adapter in an enclosure
- Standard type-A to type-mini-B (5-pin) USB cable
- 10-pin ribbon cable with connectors on both ends

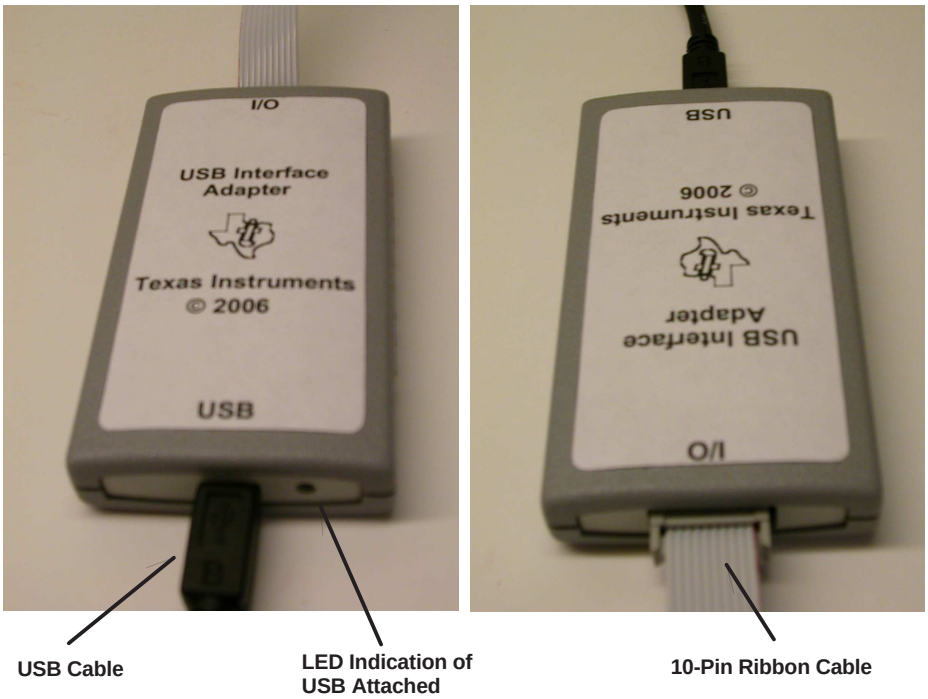

Figure 1-1. Connection of USB Interface Adapter, USB Cable, and Ribbon Cable

1.2 List of Software Items for Operation

In addition to the previously listed hardware items, the following software also is needed:

- I^2C/GPIO/PMBUS/SMBUS transport layer DLL driver (PMBus Transport (USB HID).dll) created by TI
- Demo GUI software (USB SAA GUI.exe) for I^2C/GPIO/PMBUS/SMBUS application

Both can be downloaded from the TI website in the file "USB interface adapter GUI.zip", which also contains other accessory files.

The USB interface adapter is recognized by a PC as a generic human interface device (HID), which is supported by the built-in USB/HID drivers of the Windows operating system. Therefore, it is plug and play and no proprietary USB driver is required.

1.3 Step-by-Step Instructions for Operation and Troubleshooting

After you have downloaded and installed the transport DLL and the GUI demo software, follow these steps to operate the USB interface adapter:

1. Plug in the USB cable to both the PC and the USB interface adapter and wait for the green LED to illuminate.
   Troubleshooting: If the green LED does not illuminate after 30 s, check to ensure the USB cable is securely connected. If the connection is secure, try a different USB port. If a different USB port does not solve the problem, try to reboot the computer. If rebooting the computer does not fix the problem, try a different USB cable. If trying different USB cable does not solve the problem, contact TI technical support for help.
2. Plug in the 10-pin ribbon cable to both the USB interface adapter and an I^2C slave board. Make sure that the notch on the ribbon-cable connector matches the keyhole of the socket in the enclosure.
3. Power up the I^2C slave board. Note that the default I^2C speed is set at 100 kHz and the default pullups for I^2C data and clock lines are set at 2.2 kΩ.
4. Run the DEMO GUI software and follow the instructions for the GUI.
   Troubleshooting: If the I^2C slave board cannot talk with the USB interface adapter, first check to ensure the 10-pin ribbon cable has been securely connected. If the ribbon cable has been securely connected, check the I^2C communication speed and the pullups for I^2C data and clock lines suitable for the application. If not, modify the I^2C communication speed and the pullups accordingly. If this still does not solve the problem, contact the manufacturer of the I^2C slave board for help.

Summary of Hardware Design

This chapter describes the major features of hardware design for the USB interface adapter. See Chapter 6 for the schematic, bill of materials (BOM), and PCB layout information.

2.1 Introduction

The hardware is based on a USB peripheral chip from TI (TUSB3210). The TUSB3210 has an 8052 core with enhanced performance. The PCB is a simple two-layer, top-side populated board (see the schematic in Chapter 6). The major features of the hardware design are detailed the in the following sections.

2.2 Ribbon-Cable Connector J2 and Pinout

This connector is used for communications and controls between the USB interface adapter and one more multiple I^2C slave device(s).

Table 2-1. Terminal Functions

TERMINAL		I/O	DESCRIPTION
NAME	NO.		
PMBCTRL5/GPIO7	1	I/O	Used as either the 5th PMBUS CONTROL line (output) or as GPIO pin 7 (input/output, with internal pullup enabled)
PMBCTRL4/GPIO6	2	I/O	Used as either the 4th PMBUS CONTROL line (output) or as GPIO pin 6 (input/output, with internal pullup enabled)
PMBCTRL3/GPIO5	3	I/O	Used as either the 3rd PMBUS CONTROL line (output) or as GPIO pin 5 (input/output, with internal pullup enabled)
PMBCTRL2/GPIO4	4	I/O	Used as either the 2nd PMBUS CONTROL line (output) or as GPIO pin 4 (input/output, with internal pullup enabled)
+3.3V	5	PWR	This pin can provide a 3.3-V output power supply at up to 100 mA. Any slave device(s) can use this power supply as long as the total current consumption is less than 100 mA.
Ground	6	GND	Common ground for the entire evaluation board
PMBCTRL1/GPIO3	7	I/O	Used as either the 1st PMBUS CONTROL line (output) or as GPIO pin 3 (input/output, with internal pullup enabled)
PMBALERT/SMBALERT/GPIO2	8	I/O	Used as either the ALERT line (input) for PMBUS or SMBUS communications, or as GPIO pin 2 (input/output, with internal pullup disabled, and with an external programmable pullup)
PMBC/SMBC/SCL/GPIO1	9	I/O	Used as either the CLOCK line (output) for PMBUS or SMBUS or I^2C communications, or as GPIO pin 1 (input/output, with internal pullup disabled, and with an external programmable pullup)
PMBD/SMBD/SDA/GPIO0	10	I/O	Used as either the DATA line (input/output) for PMBUS or SMBUS or I^2C communications, or as GPIO pin 0 (input/output, with internal pullup disabled, and with an external programmable pullup)

2.3 Configuration of Pullup Resistors for Pins 8, 9, and 10

The pullups for pins 8, 9, and 10 (see the schematic in Chapter 6) are configurable through communications with the embedded firmware in the TUSB3210 (see the command Set Pullup Resistors in Chapter 3). The default setting is for all these three pins to have pullups of 2.2 kΩ. Table 2-2 lists the possible options for each pin.

Table 2-2. Configurable Options of Pullup Resistors for Pins 8, 9, and 10

PULLUP RESISTOR OPTION (BYTE VALUE)	OPTION 1 (0x00)	OPTION 2 (0x01) (DEFAULT)	OPTION 3 (0x02)	OPTION 4 (0x03)
Pin 8 of J2: ALERT line	No pullup (open)	2.2 kΩ	N/A	N/A
Pin 9 of J2: CLOCK line	No pullup (open)	2.2 kΩ	1 kΩ	688 Ω
Pin 10 of J2: DATA line	No pullup (open)	2.2 kΩ	1 kΩ	688 Ω

2.4 EEPROM

The embedded firmware is stored in a 64-Kbits EEPROM (see U1 in the schematic in Chapter 6). The firmware is field programmable (see Chapter 5 for more information). Jumper shunt J1 is normally closed and is used for firmware development only. On power up, the TUSB3210 downloads the firmware image from the EEPROM to the internal static RAM and starts firmware execution (first with USB enumeration) from there.

2.5 LED Indication of USB Attached

A green LED is mounted next to the USB connector on the evaluation board. Every time the USB interface adapter is attached to a PC via a USB cable, and if the host PC detects it and goes through USB enumeration successfully, the embedded firmware illuminates the green LED (D1) to indicate that; otherwise, the LED remains off. After unplugging the USB cable, the LED turns off immediately.

Firmware Programming and Communications Protocol

The following sections describe the embedded firmware, as well as the communications protocol supported in the USB interface adapter and how to use each command in the protocol.

3.1 Firmware Features Summary

The embedded firmware in the TUSB3210 supports the following major features:

- Supports USB1.1 full speed at 12 Mbit/s. The USB interface adapter is recognized by the host as a generic human interface device (HID), so no proprietary USB or HID driver is needed. Therefore, the USB interface adapter can be treated as a USB plug-and-play device.
- Supports firmware programming in the external EEPROM so that firmware can be updated in the field. See the TI website (www.ti.com) for more detailed information on firmware programming.
- Configures different pullup resistors for ALERT, DATA, and CLOCK lines
- Supports flexible I^2C read or write transactions. The USB interface adapter is used as the I^2C master.
- Selects different clock speeds (either 100 kHz or 400 kHz) for I^2C, PMBUS, or SMBUS communications
- Provides firmware version information (see Chapter 4 for details on firmware versioning scheme)

3.2 Firmware Version Command and Response

COMMAND DESCRIPTION	USB PACKET BYTE 0 COMMAND CODE	USB PACKET BYTE 1	USB PACKET BYTE 2	USB PACKET BYTE 3	USB PACKET BYTE 4	USB PACKET BYTE 5	USB PACKET BYTES 6–63	COMMENTS
Firmware Version (Get Version)	0x00							See Chapter 4

COMMAND DESCRIPTION	USB PACKET BYTE 0 COMMAND CODE	USB PACKET BYTE 1	USB PACKET BYTE 2	USB PACKET BYTE 3	USB PACKET BYTE 4	USB PACKET BYTE 5	USB PACKET BYTES 6–63	COMMENTS
Response from Firmware Version	0x00							See Chapter 4

See Chapter 4 for a detailed explanation of how the Family Code, Major Version, and Minor Version bytes are assigned/interpreted.

3.3 I²C Write Command and Response

COMMAND DESCRIPTION	USB PACKET BYTE 0 COMMAND CODE	USB PACKET BYTE 1	USB PACKET BYTE 2	USB PACKET BYTE 3	USB PACKET BYTE 4	USB PACKET BYTE 5	USB PACKET BYTES 6–63	COMMENTS
I²C Write	0x14	A	B	C	C number of bytes (from 1 up to 60 bytes) to write			

COMMAND DESCRIPTION	USB PACKET BYTE 0 COMMAND CODE	USB PACKET BYTE 1	USB PACKET BYTE 2	USB PACKET BYTE 3	USB PACKET BYTE 4	USB PACKET BYTE 5	USB PACKET BYTES 6–63	COMMENTS
Response from I²C Write	0x94	Success: 0 Fail: 1						

The following describes the detailed bit banking of this command: **I²C Start → write A (= slave address shifted left by 1 bit + write bit of 0) → slave ACK → write B (which is usually either a command byte or a register address byte) → slave ACK → repeat (write 1 byte → slave ACK) for C number of times → I²C stop**. There is another more generic I²C Write transaction command in Section 3.10.

3.4 I²C Read Command and Response

COMMAND DESCRIPTION	USB PACKET BYTE 0 COMMAND CODE	USB PACKET BYTE 1	USB PACKET BYTE 2	USB PACKET BYTE 3	USB PACKET BYTE 4	USB PACKET BYTE 5	USB PACKET BYTES 6–63	COMMENTS
I²C Read	0x15	A	B	C	D (from 1 to 62 bytes)			

COMMAND DESCRIPTION	USB PACKET BYTE 0 COMMAND CODE	USB PACKET BYTE 1	USB PACKET BYTE 2	USB PACKET BYTE 3	USB PACKET BYTE 4	USB PACKET BYTE 5	USB PACKET BYTES 6–63	COMMENTS
Response from I²C Read	0x95	Success: 0 Fail: 1	D number of bytes read					

The following describes the detailed bit banking of this command: **I²C start → write A (= slave address shifted left by 1 bit + write bit of 0) → slave ACK → write B (which is usually either a command byte or a register address byte) → slave ACK → repeated start → write C (= slave address shifted left by 1 bit + read bit of 1) → slave ACK → read D number of bytes (with master ACKs in between) from a slave → master NACK → I²C stop**. There is another more generic I²C Read transaction command in Section 3.11.

3.5 Read/Write Port 0 Command and Response

COMMAND DESCRIPTION	USB PACKET BYTE 0 COMMAND CODE	USB PACKET BYTE 1	USB PACKET BYTE 2	USB PACKET BYTE 3	USB PACKET BYTE 4	USB PACKET BYTE 5	USB PACKET BYTES 6–63	COMMENTS
Read/Write Port 0	0x16	A (bit.x=1 for read/input, bit.x=0 for write /output)	B (byte to write)					B does not apply to port 0 read. Each GPIO can be configured as input or output.

COMMAND DESCRIPTION	USB PACKET BYTE 0 COMMAND CODE	USB PACKET BYTE 1	USB PACKET BYTE 2	USB PACKET BYTE 3	USB PACKET BYTE 4	USB PACKET BYTE 5	USB PACKET BYTES 6–63	COMMENTS
Response from Read/Write Port 0	0x96	Success: 0 Fail: 1	Byte read from port 0					

Each individual GPIO pin of the TUSB3210 port 0 can be configured as either an input or an output (see the schematic in Chapter 6).

GPIO pins 3 to 7 are configured to have internal pullups turned on so no special external pullups are needed unless stronger I/O drives are necessary.

GPIO pins 0, 1, and 2 are configured to be open-drain I/O ports. Use the software-configurable pullups to support I/O functions.

When pin 0 or pin 1 is used as a GPIO, the I^2C does not function properly anymore. Only after a power-on reset can the I^2C function be used again without using pins 0 and 1 as GPIOs. Therefore, it is recommended that no more than six GPIO pins are used, reserve pins 0 and 1 exclusively for I^2C applications.

3.6 EEPROM Program Command and Response

COMMAND DESCRIPTION	USB PACKET BYTE 0 COMMAND CODE	USB PACKET BYTE 1	USB PACKET BYTE 2	USB PACKET BYTE 3	USB PACKET BYTE 4	USB PACKET BYTE 5	USB PACKET BYTES 6–63	COMMENTS
EEPROM Program	0x18	A = Starting address MSB	B = Starting address LSB	C (# of bytes to program)	Data to be programmed into EEPROM (up to 32 bytes)			

COMMAND DESCRIPTION	USB PACKET BYTE 0 COMMAND CODE	USB PACKET BYTE 1	USB PACKET BYTE 2	USB PACKET BYTE 3	USB PACKET BYTE 4	USB PACKET BYTE 5	USB PACKET BYTES 6–63	COMMENTS
Response from EPROM Program	0x98	Success: 0 Fail: 1						

This command is used for programming a firmware image to the EEPROM (U1 on schematic) of the EVM. It is recommended that the whole firmware image be split into multiple blocks of 32 bytes and call this command repeatedly until the programming is completed.

3.7 EEPROM Read Command and Response

COMMAND DESCRIPTION	USB PACKET BYTE 0 COMMAND CODE	USB PACKET BYTE 1	USB PACKET BYTE 2	USB PACKET BYTE 3	USB PACKET BYTE 4	USB PACKET BYTE 5	USB PACKET BYTES 6–63	COMMENTS
EEPROM Read	0x19	A = Starting address MSB	B = Starting address LSB	C (# of bytes to read)				C ≤ 60

COMMAND DESCRIPTION	USB PACKET BYTE 0 COMMAND CODE	USB PACKET BYTE 1	USB PACKET BYTE 2	USB PACKET BYTE 3	USB PACKET BYTE 4	USB PACKET BYTE 5	USB PACKET BYTES 6–63	COMMENTS
Response from EEPROM Read	0x99	Success: 0 Fail: 1	C number of bytes read back from EEPROM (up to 32 bytes)					

This command works with the EEPROM Program command to verify if the firmware image has been programmed into the EEPROM correctly. For the best efficiency, try to read a block of 32 bytes at a time by calling this command.

3.8 Set Pullup Resistors Command and Response

COMMAND DESCRIPTION	USB PACKET BYTE 0 COMMAND CODE	USB PACKET BYTE 1	USB PACKET BYTE 2	USB PACKET BYTE 3	USB PACKET BYTE 4	USB PACKET BYTE 5	USB PACKET BYTES 6–63	COMMENTS
Set Pullup Resistors	0x1A	A (for SDA)	B (for SCL)	C (for ALERT)				0x00 for no pullups (switched off), 0x01 for 2.2 kΩ, 0x02 for 1 kΩ, 0x03 for 688 Ω. For ALERT, the option is either open (0x00) or 2.2 kΩ (0x01).

COMMAND DESCRIPTION	USB PACKET BYTE 0 COMMAND CODE	USB PACKET BYTE 1	USB PACKET BYTE 2	USB PACKET BYTE 3	USB PACKET BYTE 4	USB PACKET BYTE 5	USB PACKET BYTES 6–63	COMMENTS
Response from Set Pullup Resistors	0x9A	Success: 0 Fail: 1						Pullups have been configured

Byte A is for the DATA line for I^2C, PMBUS, or SMBUS communications; byte B is for the CLOCK line for I^2C, PMBUS, or SMBUS communications; byte C is for the ALERT line for PMBUS communications. If a slave device can provide one or multiple pullup resistors, switch off the pullups for DATA, SCL, or ALERT line accordingly.

3.9 Set I²C-PMBUS-SMBUS Speed Command and Response

COMMAND DESCRIPTION	USB PACKET BYTE 0 COMMAND CODE	USB PACKET BYTE 1	USB PACKET BYTE 2	USB PACKET BYTE 3	USB PACKET BYTE 4	USB PACKET BYTE 5	USB PACKET BYTES 6–63	COMMENTS
Set I²C-PMBUS-SMBUS Speed	0x1B	A (0 for 100 kHz, otherwise 400 kHz)						

COMMAND DESCRIPTION	USB PACKET BYTE 0 COMMAND CODE	USB PACKET BYTE 1	USB PACKET BYTE 2	USB PACKET BYTE 3	USB PACKET BYTE 4	USB PACKET BYTE 5	USB PACKET BYTES 6–63	COMMENTS
Response from Set I²C-PMBUS-SMBUS Speed	0x9B							Done with speed selection

For I²C, PMBUS, or SMBUS communications, the clock speed can be set to either 100 kHz or 400 kHz. As an I²C, PMBUS, or SMBUS master, the USB interface adapter supports clock stretching (up to 25 ms per transaction). The firmware sets the default clock speed to 100 kHz.

3.10 Generic I²C Write Command and Response

COMMAND DESCRIPTION	USB PACKET BYTE 0 COMMAND CODE	USB PACKET BYTE 1	USB PACKET BYTE 2	USB PACKET BYTE 3	USB PACKET BYTE 4	USB PACKET BYTE 5	USB PACKET BYTES 6–63	COMMENTS
Generic I²C Write	0x1C	A	A number of bytes (from 2 up to 62) to write					The first byte of data to write is usually (device address < 1 + write bit of 0).

COMMAND DESCRIPTION	USB PACKET BYTE 0 COMMAND CODE	USB PACKET BYTE 1	USB PACKET BYTE 2	USB PACKET BYTE 3	USB PACKET BYTE 4	USB PACKET BYTE 5	USB PACKET BYTES 6–63	COMMENTS
Response from Generic I²C Write	0x9C	Success: 0 Fail: 1						

This is a generic I²C Write transaction command that can be applied to majority of I²C applications. The detailed bit-banking is as follows: **I²C start → repeat (write 1 data byte, slave ACK) for A number of times → I²C stop**.

Basically, all the transaction bytes are placed one after another, such as address byte (7-bit slave address shifted left by 1 bit + write bit of 0), one or multiple command bytes, one or multiple data bytes into the A number of bytes buffer space, then the USB interface adapter carries out the bit banking. For maximum flexibility, this generic command for any I²C write transaction is recommended.

3.11 Generic I²C Read Command and Response

COMMAND DESCRIPTION	USB PACKET BYTE 0 COMMAND CODE	USB PACKET BYTE 1	USB PACKET BYTE 2	USB PACKET BYTE 3	USB PACKET BYTE 4	USB PACKET BYTE 5	USB PACKET BYTES 6–63		COMMENTS
Generic I²C Read	0x1D	A	A number of bytes (2 to 60) to write				Y = 1st byte to write + read bit 1 USB byte 62	X # of bytes (1 to 62) to read back USB byte 63	The 1st byte of data to write is usually (device address < 1 + write bit of 0).

COMMAND DESCRIPTION	USB PACKET BYTE 0 COMMAND CODE	USB PACKET BYTE 1	USB PACKET BYTE 2	USB PACKET BYTE 3	USB PACKET BYTE 4	USB PACKET BYTE 5	USB PACKET BYTES 6-63	COMMENTS
Response from Generic I²C Read	0x9D	Success: 0 Fail: 1	X number of bytes (1 to 62) read back from slave					

This is a generic I²C Read transaction command that can be applied to majority of I²C applications. The detailed bit-banking is as follows: **I²C start → repeat (write 1 data byte, slave ACK) for A number of times → repeated start → write Y → slave ACK → read X number of bytes (with master ACKs in between) from a slave → master NACK → I²C stop**.

Basically, all the transaction bytes are placed one after another, such as address byte (7-bit slave address shifted left by 1 bit + write bit of 0), one or multiple command bytes, then repeated start, write address byte Y (7-bit slave address shifted left by 1 bit + read bit of 1), and read back x number of bytes from the slave device being addressed. The USB interface adapter carries out the bit banking. For maximum flexibility, this generic command for any I²C read transactions is recommended.

3.12 Board Test Command and Response

COMMAND DESCRIPTION	USB PACKET BYTE 0 COMMAND CODE	USB PACKET BYTE 1	USB PACKET BYTE 2	USB PACKET BYTE 3	USB PACKET BYTE 4	USB PACKET BYTE 5	USB PACKET BYTES 6-63	COMMENTS
Board Test	0x7F							Start toggling eight LEDs connected to the eight GPIO pins simultaneously for 5 s.

COMMAND DESCRIPTION	USB PACKET BYTE 0 COMMAND CODE	USB PACKET BYTE 1	USB PACKET BYTE 2	USB PACKET BYTE 3	USB PACKET BYTE 4	USB PACKET BYTE 5	USB PACKET BYTES 6-63	COMMENTS
Response from Board Test	0xFF							Board test is done

This command is for internal testing of USB interface adapter evaluation boards during production only. Do not use this command for any other application.

3.13 Other Commands for Different Applications

For applications other than I²C/GPIO, such as PMBUS, SMBUS, HDQ, etc., see the corresponding literature for more details.

Firmware Versioning Scheme

This chapter briefly describes the firmware versioning scheme for the USB interface adapter. Understanding the versioning scheme is critical for using the device in the preset configuration. This topic is important for those who would like to make a custom implementation of firmware or to understand how the firmware version is validated by the host client software.

4.1 Introduction

The firmware can be no more than 8K bytes. This limits the amount of functionality that any single firmware image can have. All the functions needed for every application could not possibly be implemented in this space. In order to make the device extensible and to allow its use in many applications, it was designed in a way for a host application to reprogram the device with its own custom firmware whenever needed.

For example, some end devices may require protocol X to send and receive data, but the standard firmware does not support protocol X. In this case, the programmer of the GUI software that supports this end device writes custom firmware that conforms to the rules in Section 4.2. At run time, the GUI software finds the USB interface adapter and retrieves its firmware version number. If the version number is not compatible with the protocols needed, the GUI software then reprograms the device with its own version of firmware so that it can use it.

If all GUI software complies with this versioning system, then the same hardware can be reused for many applications without inhibiting other compliant software. **This also ensures that all implementations will be backward compatible with older GUI software.**

4.2 Rules for Implementing Compliant Custom Firmware

In order for custom firmware to be compatible with this versioning scheme, it must comply with the following rules:

1. The custom firmware must support pairs of commands listed in Table 4-1 (see Chapter 3 for the default implementation).

Table 4-1. List of Pairs of Commands Needed for Firmware Versioning

CODES (hex)	DESCRIPTION	INFORMATION
0x00 and 0x80	Firmware Version (Get Version)	Required to communicate its unique identity to the GUI software
0x18 and 0x98	EEPROM Pro	Required for the GUI software to reprogram the firmware on the embedded EEPROM
0x19 and 0x99	EEPROM Read	Required for the GUI software to validate the firmware image on the EEPROM

2. The custom firmware must have a unique, compliant version number that describes its functionality.

Note: Together, these two rules describe an interface that all GUI software can expect to be implemented. If GUI software cannot call the Firmware Version command, it cannot identify the firmware and should not attempt to use it as the outcome is unpredictable. If GUI software needs to reprogram the USB interface adapter, it expects command 0x18 to be an EEPROM programming. Conforming to this standard ensures that the firmware does not break the validation and reprogramming cycle.

4.3 Compliant Versioning

The versioning scheme uses three bytes to make up a unique identifier that the device can return (as the data part) in response to the Firmware Version command (see Chapter 3 for more details).

The first byte is the Family Code and it describes who is responsible for the firmware. These codes are given out in blocks to various groups with TI. For instance, the Digital Power Group owns codes 01–16 (or 0x01–0x10 in hex). Some Family Codes are reserved. Family codes 240–255 (or 0xF0–0xFF in hex) have been set aside for custom implementations by end users or anyone wanting to make a compliant version of custom firmware for their own private use.

This Family Code also denotes the permutations of protocols supported, such as Family Code 1 supports I^2C, PMBus, and SMBus, while Family Code 2 might support I^2C and Protocol X.

The second byte is the Major Version and it describes a revision to the permutations of protocols. If the protocols supported need new functionality or have changes to the way that they are called (signature changed) then this byte denotes the change.

The third byte is the Minor Version and it describes a revision to the permutations of protocols. If the firmware code has changed to fix a bug in the functions but no new public functions have been added and no changes have been made to the function signatures, this byte should be changed to denote the new version.

Table 4-2 summarizes the significance of the three-byte code.

Table 4-2. Three Bytes for Firmware Versioning

BYTE ORDER	MOST SIGNIFICANT BYTE	MIDDLE BYTE	LEAST SIGNIFICANT BYTE
Byte Meaning	Family Code	Major Version	Minor Version
Example Value	01 = Owned by Digital Power PMBus, SMBus, I^2C supported	01 = Major revision of the permutation of protocols	01 = Minor revision of the permutation of protocols

To implement this versioning scheme properly, a Windows™ application calls a Firmware Version command that, along with some others, is reserved and must be present in each implementation of the device. If the Family Code does not match the code that the application is looking for, it should not assume anything about what protocols are supported. The Family Code should immediately attempt to reprogram the firmware with its own version (binary image distributed with the application).

A Windows application that correlates Family Code and Major Version with the firmware implementation should then, theoretically, only have to check that the Minor Version is greater than or equal to its target Minor Version. A firmware programmer should be careful to never remove functionality from an implementation with the same Family Code/Major Version, or an older Windows application may attempt to make a malformed call to it when it attempts to communicate. In the case that such a drastic change is made, it is recommend that the firmware developer change the Family Code byte.

Table 4-3. Initial Family Codes

FAMILY CODE (DECIMAL)	GROUP/TEAM
00	Reserved
01–16	Digital Power + POE
17–32	HPA Design Tools team
33–48	Battery Management
49–239	Unassigned
240–255	Open for custom implementations (user defined)

GUI applications may use the flow chart in Figure 4-1 to determine whether or not to reprogram the firmware on the USB interface adapter.

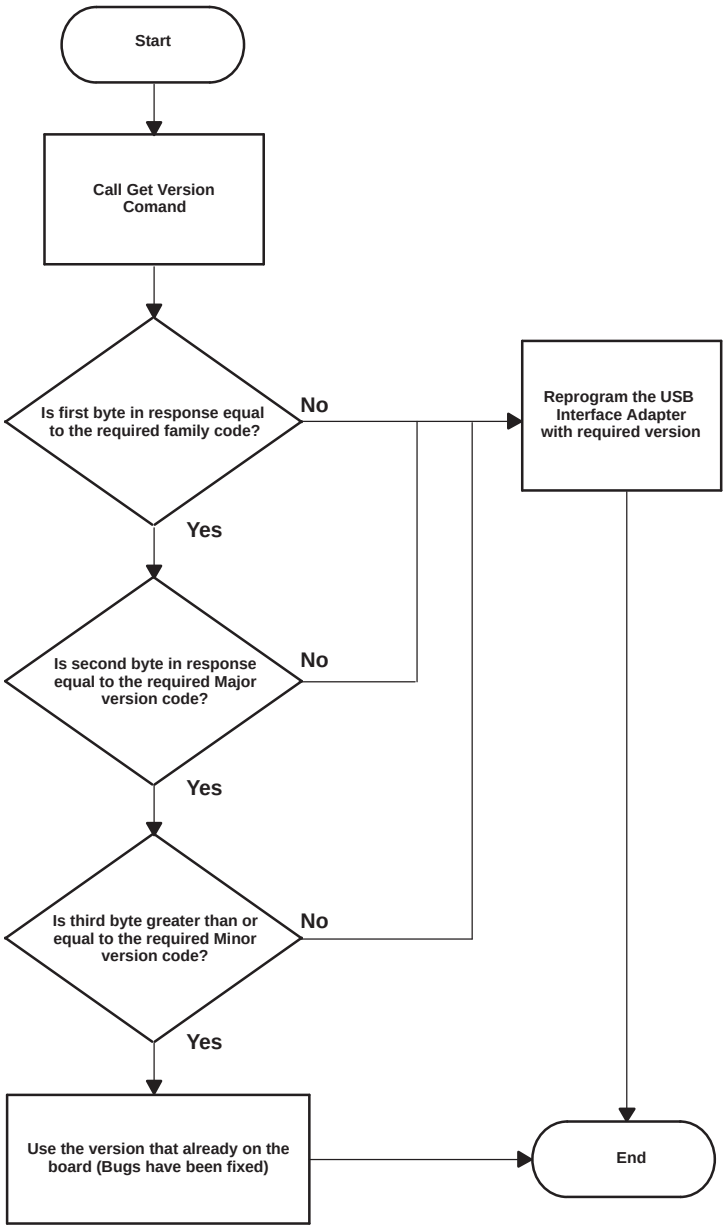

Figure 4-1. Flow Chart for Determining Firmware Programing

Figure 4-2 shows an example of how the firmware programming into EEPROM should function between applications.

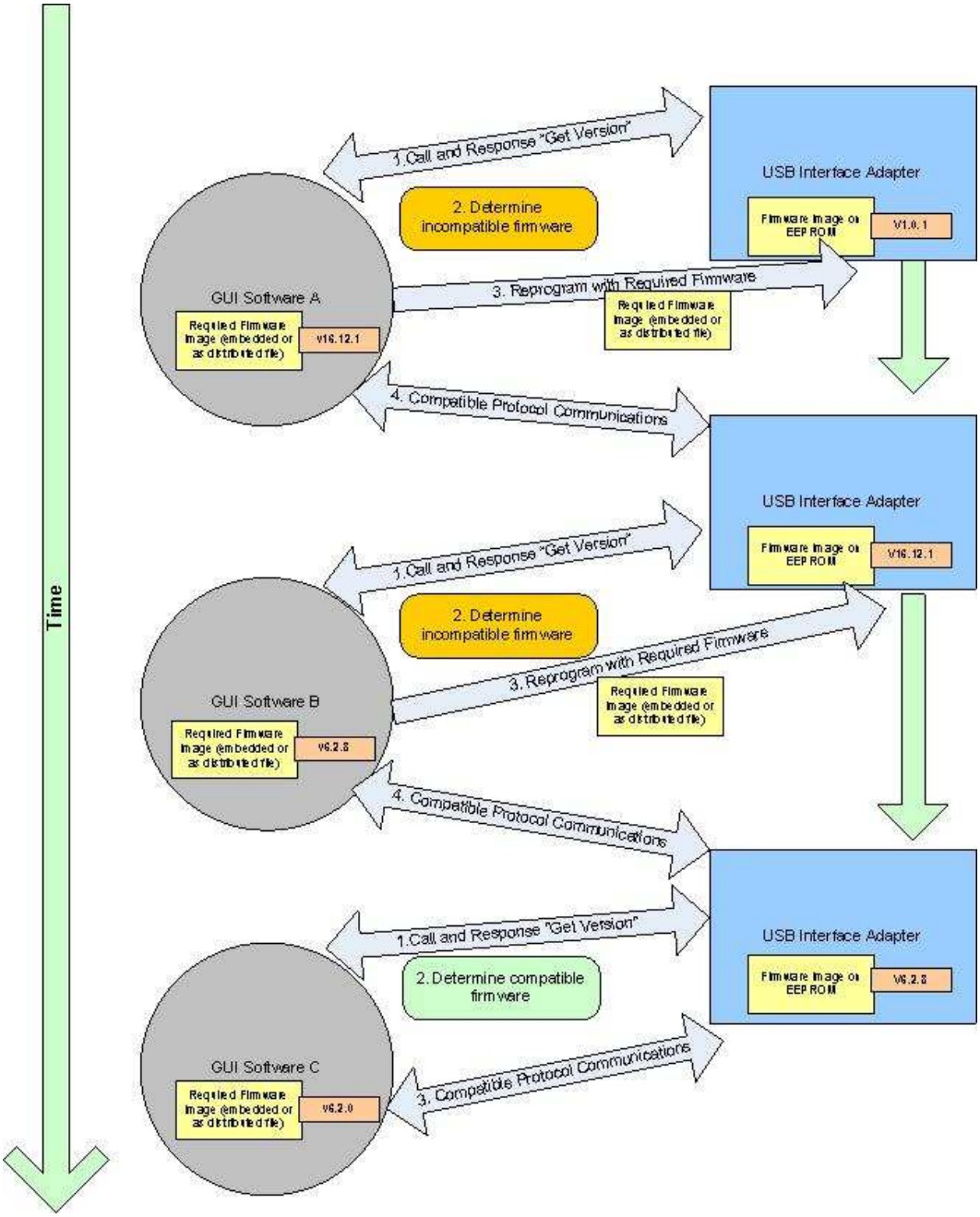

Figure 4-2. Interactions of Applications During Firmware Programming

GUI and PC Libraries

The adapter solution includes a set of PC libraries and a GUI that can be used to allow PC applications access to the adapter and target devices that the adapter supports.

The GUI and PC libraries are designed using the Microsoft® .NET Framework, in order to leverage as many standard libraries as possible. As previously described in this document, the adapter presents itself as a USB HID to Windows. Because of this, no custom driver is required, and the application software on the PC can use the standard Microsoft Windows USB driver (usbhid.sys).

Like the adapter, the Windows libraries support all the different types of device interfaces: I^2C, SMBus, PMBus, and GPIO. In addition, the libraries support configuration and control of the adapter itself by allowing PC programs to update the EERPROM in the adapter, select switching in and out of resistors on data and clock lines, etc.

Figure 5-1 represents the software architecture for the PC that is used.

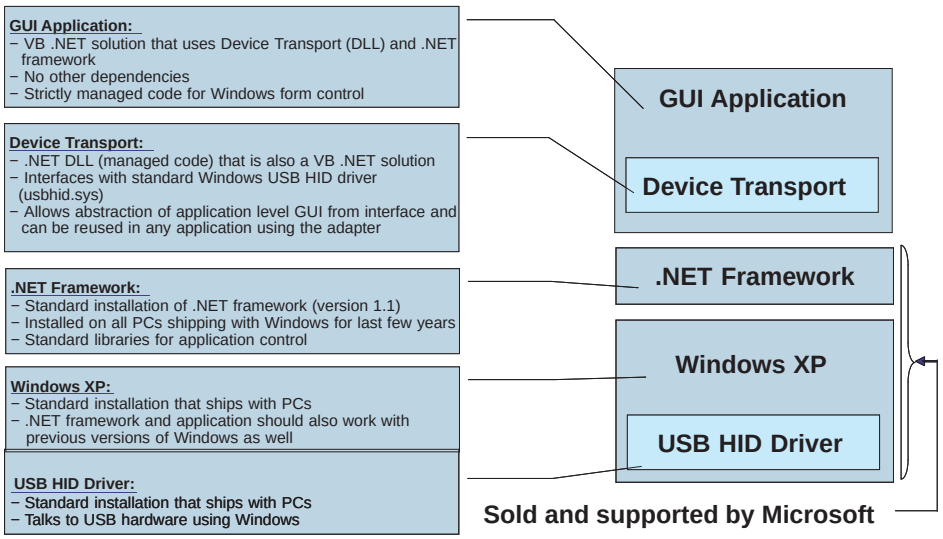

Figure 5-1. USB Adapter Software Architecture

The major pieces of the software architecture are as follows:

- Windows XP™
- .NET Framework
- Device transport
- GUI application

5.1 Windows XP

Windows XP provides the standard USB HID support to the application's libraries through a standard USB HID driver that comes with Microsoft Windows.

The .NET Framework is a set of libraries that Microsoft provides that allows Windows applications a rich set of services to use. Functionality such as form and window management, memory management, and database are all supported by the .NET Framework. Most recent applications written for Windows use the .NET Framework.

The .NET Framework is the only dependency that the device transport libraries and the GUI application have; there are no custom drivers required by this application. The .NET Framework is installed on all new PCs, so there should be nothing to do except unzip and run the GUI application that is delivered for the adapter Framework.

The current version of the device transport libraries and GUI Application use version 1.1 of the Microsoft .NET Framework. Depending on the version of development tools used to generate GUI libraries and applications, an update to the .NET Framework from Microsoft may be required.

The URL to update the Microsoft .NET Framework on your PC may be found at the following URL:

http://www.microsoft.com/downloads/details.aspx?FamilyID=0856EACB-4362-4B0D-8EDD-AAB15C5E04F5&displaylang=en

5.2 Device Transport

In order to enable new applications to be created with minimal effort by users of the adapter, a device transport library is provided.

This library is a .NET DLL that was generated using Microsoft Developer Studio® 2003 and is basically a set of .NET managed code. This library requires version 1.1 of the .NET Framework to be present on the machine that uses it.

This library is included in the sample GUI application that is provided with the adapter and is intended to be used by other applications, as well as to manage a target through the adapter.

To use this library, simply include it in the project being developed as a dependency.

This library provides the following functionality:

- Primitives that support all I/O types of the adapter
  - I^2C
    - I^2C read (variable length)
    - I^2C write (variable length)
  - SMBus
    - Receive byte
    - Read byte
    - Read word
    - Block read
    - Send byte
    - Write byte
    - Write word
    - Block write
    - Process call
    - Block write block read process call
    - PEC enable/disable
  - PMBus
    - Group command
    - Assert/deassert CONTROL line
    - Poll ALERT# signal
  - GPIO (read/write)
- General functionality
  - Reset adapter
  - Get device version
  - Select internal pullup resistors
  - Update/read EEPROM contents
  - Select 100-kHz/400-kHz bus speed

5.3 GUI Application

The sample application that is available with the adapter uses the device transport to communicate with the adapter, as well as the target devices behind the adapter. A screenshot is shown in Figure 5-2.

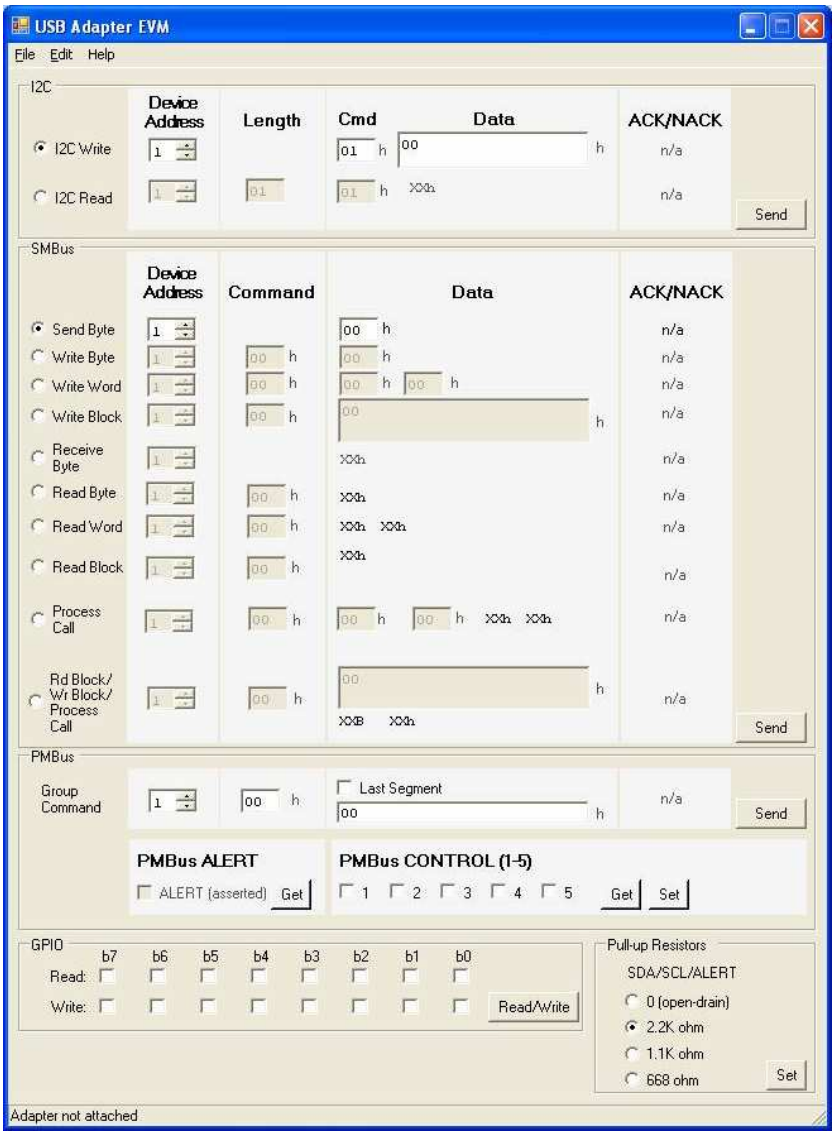

Figure 5-2. Sample GUI Application

From the main menu, update the firmware for the adapter, select the bus speed of the device (100 kHz or 400 kHz), and select whether PEC is enabled or disabled when communicating with the device.

The main portion of the form is split into five sections: I^2C, SMBus, PMBus, GPIO, and pullup resistors. From the I^2C, SMBus, and PMBus group boxes, the user can select each of the available transaction types that the adapter supports. The result of the transaction to the device is also communicated here as an acknowledge (ACK) or not acknowledge (NACK).

In the GPIO group box, each of the eight GPIO pins can be enabled and programmed as read or write. In addition, the state of the pin can be read or asserted to the desired value from this group box as well.

Finally, the GUI allows users to select the values of the pullup resistors used for the SDA/SCL lines. The adapter and the GUI allow users to select from 668 Ω, 1.1 kΩ, 2.2 kΩ, or open drain.

Schematic, Bill of Materials, PCB Layout

This chapter contains information on schematic design, with related information on bill of materials (BOM) and printed circuit board (PCB) layout for the USB interface adapter.

6.1 Schematic

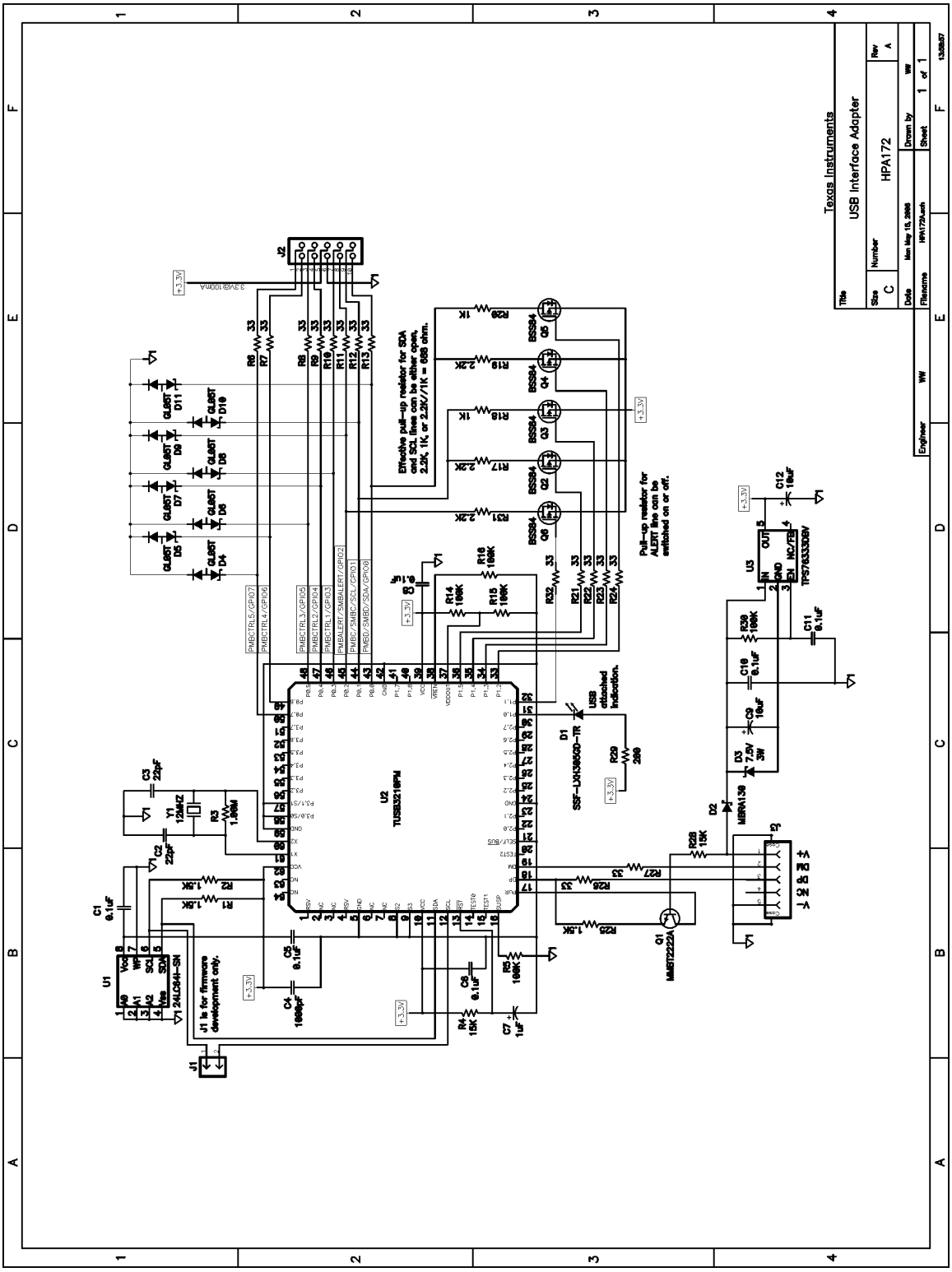

Figure 6-1. Schematic

6.2 Bill of Materials (BOM)

Table 6-1. Bill of Materials

QTY	REF DESIGNATOR	VALUE	DESCRIPTION	SIZE	PART NUMBER	MANUFACTURER
6	C1, C5, C6, C8, C10, C11	0.1 µF	Capacitor, Ceramic, 0.1 µF, 25 V, X7R, 10%	0603	C1608X7R1E104KT	TDK
2	C2, C3	22 pF	Capacitor, Ceramic, 22 pF, 50 V, C0G, 10%	0603	C1608C0G1H220KT	TDK
1	C4	1000 pF	Capacitor, Ceramic, 1000 pF, 100 V, C0G, 5%	0805	Std	Std
1	C7	1 µF	Capacitor, Tantalum, 1 µF, 16 V, 20%	3216	293D105X0016A2T	Vishay
2	C9, C12	10 µF	Capacitor, Tantalum, 10 µF, 10 V, 20%	3216	293D106X0010A2T	Vishay
1	D1	SSF-LXH305GD-TR	Diode, LED, 2.6 V, 25 mA	0.25 × 0.25	SSF-LXH305GD-TR	Lumex
1	D2	MBRA130	Diode, Schottky, 1 A, 30 V	SMA	MBRA130	IR
1	D3	7.5 V	Diode, Zener, 7.5 V, 3 W	SMB	1SMB5922BT3	On Semi
8	D4, D5, D6, D7, D8, D9, D10, D11	GL05T	Diode, TVS diode, Low Capacitance	SOT23	GL05T	General
1	J1	PTC36SAAN	Header, 2-pin, 100 mil spacing, (36-pin strip)	0.1 × 2	PTC36SAAN	Sullins
1	J2	86479-3	Connector, Male Right Angle 2×5 pin, 100 mil spacing, 4 Wall	0.607 × 0.484	86479-3	AMP
1	J3	UX60-MB-5ST	Connector, Recpt, USB-B, Mini, 5-pins, SMT	0.354 × 0.303	UX60-MB-5ST	Hirose
1	Q1	MMBT2222A	Transistor, NPN, High Performance, 500 mA	SOT-23	MMBT2222A	Fairchild
5	Q2, Q3, Q4, Q5, Q6	BSS84	MOSFET, Pch, –50 V, –0.13 A, 10 Ω	SOT-23	BSS84	Fairchild
3	R1, R2, R25	1.5 kΩ	Resistor, Chip, 1.5 kΩ, 1/16 W, 5%	0603	Std	Std
3	R17, R19, R31	2.2 kΩ	Resistor, Chip, 2.2 kΩ, 1/16 W, 5%	0603	Std	Std
2	R18, R20	1 kΩ	Resistor, Chip, 1 kΩ, 1/16 W, 5%	0603	Std	Std
1	R29	200 Ω	Resistor, Chip, 200 Ω, 1/16 W, 5%	0603	Std	Std
1	R3	1 MΩ	Resistor, Chip, 1 MΩ, 1/16 W, 1%	0603	Std	Std
2	R4, R28	15 kΩ	Resistor, Chip, 15 kΩ, 1/16 W, 5%	0603	Std	Std
5	R5, R14, R15, R16, R30	100 kΩ	Resistor, Chip, 100 kΩ, 1/16 W, 5%	0603	Std	Std
14	R6, R7, R8, R9, R10, R11, R12, R13, R21, R22, R23, R24, R26, R27, R32	33 Ω	Resistor, Chip, 33 Ω, 1/16 W, 5%	0603	Std	Std
1	U1	24LC64	IC, Serial EEPROM 64 kΩ, 1.8 V to 5.5 V, 400 kHz Max	SO8	24LC64	Microchip
1	U2	TUSB3210PM	IC, USB, General Purpose, Device Controller	0.48 × 0.48	TUSB3210PM	TI
1	U3	TPS76333DBV	IC, Micro-Power 150 mA LDO Regulator	SOT23-5	TPS76xxxDBV	TI
1	Y1	12 MHz		0.185 × 0.532	CY12BPSMD	Crystek
1		M3DDA-1018J-ND	Ribbon Cable, Socket-to-Socket, 10 pin	18	M3DDA-1018J-ND	3M
1			Shunt, 100 mil, Black	0.1	929950-00	3M
1			PCB, 3.12 in × 1.6 in × 0.062 in		HPA172E2	Any

Note: These assemblies are ESD sensitive, so ESD precautions shall be observed.

These assemblies must be clean and free from flux and all contaminants. Use of no clean flux is not acceptable.

These assemblies must comply with workmanship standards IPC-A-610 Class 2.

Ref designators marked with an asterisk ('**') cannot be substituted. All other components can be substituted with equivalent manufacturer's components.

6.3 Printed Circuit Board (PCB) Layout

Figure 6-2 through Figure 6-5 show the PCB layout.

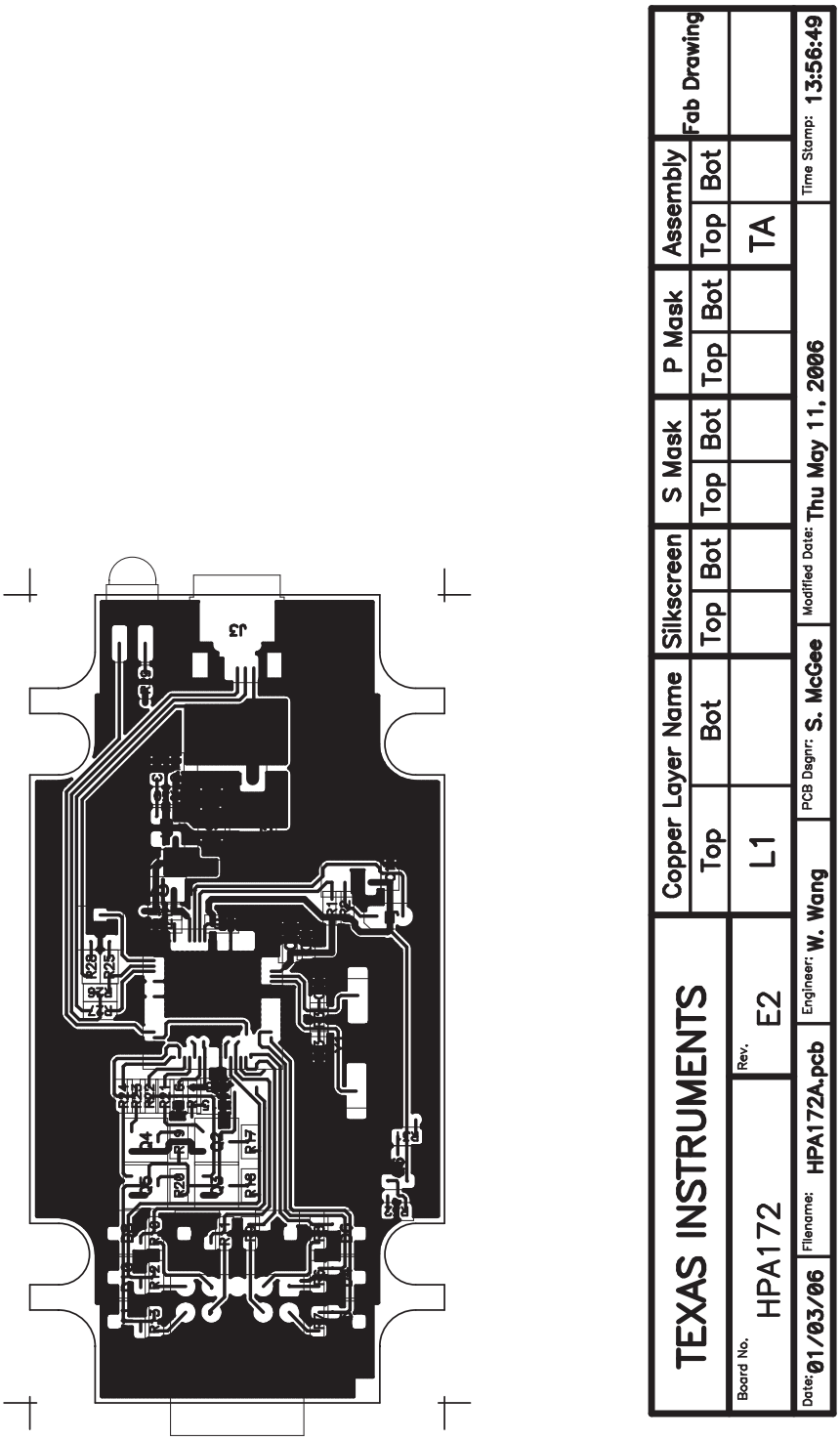

Figure 6-2. PCB Layout — Top L1 (Copper Layer) — Assembly TA

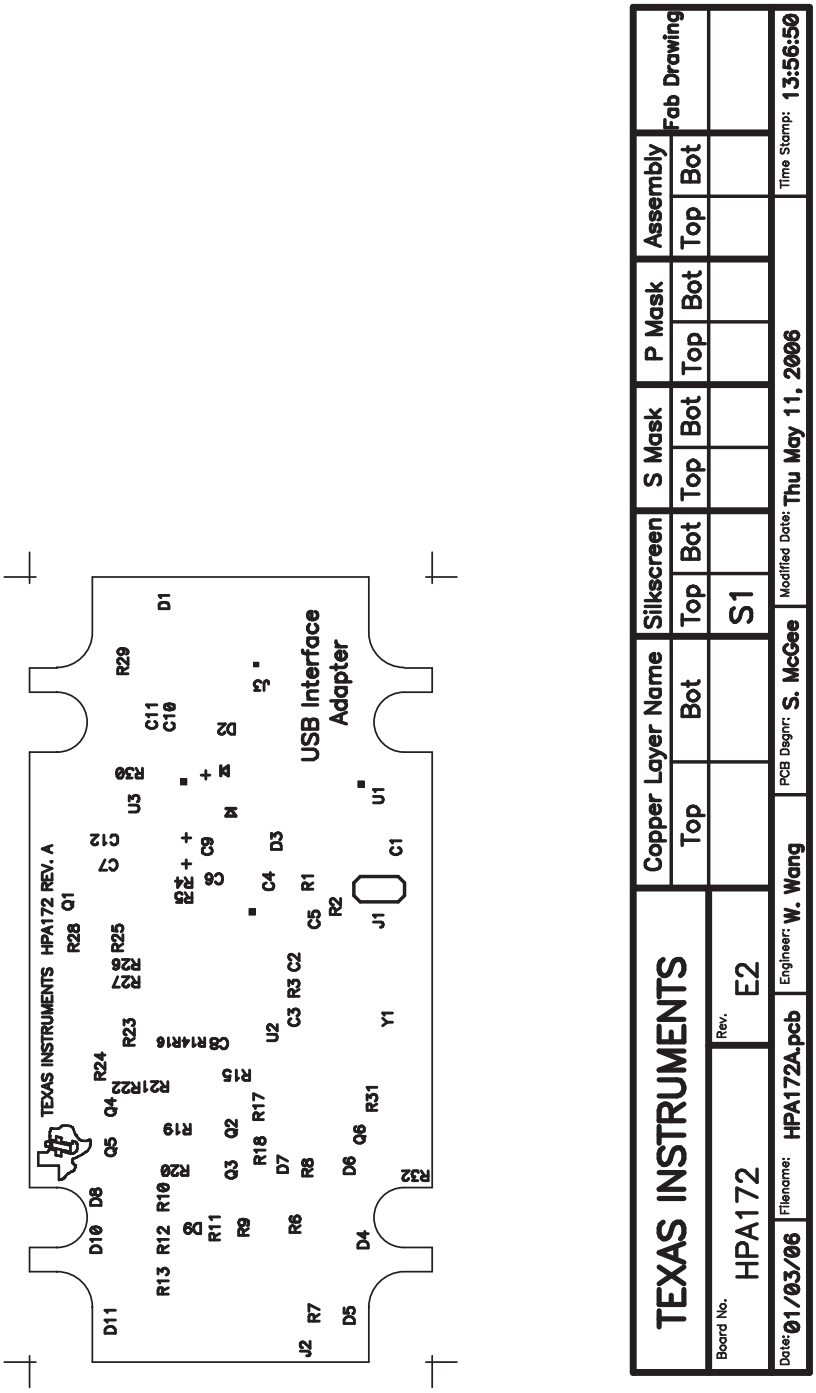

Figure 6-3. PCB Layout — Top S1 (Silkscreen Layer)

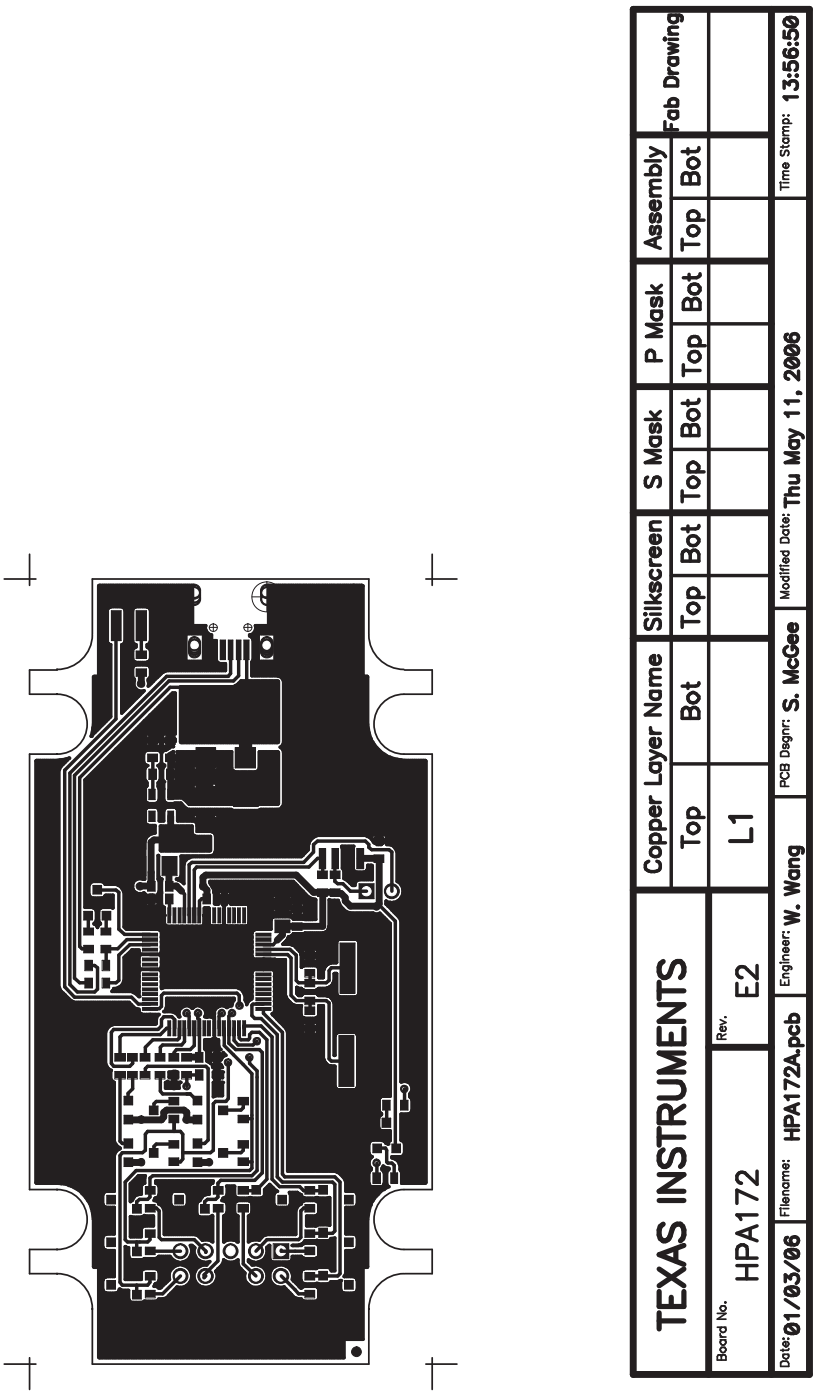

Figure 6-4. PCB Layout — Top L1 (Copper Layer)

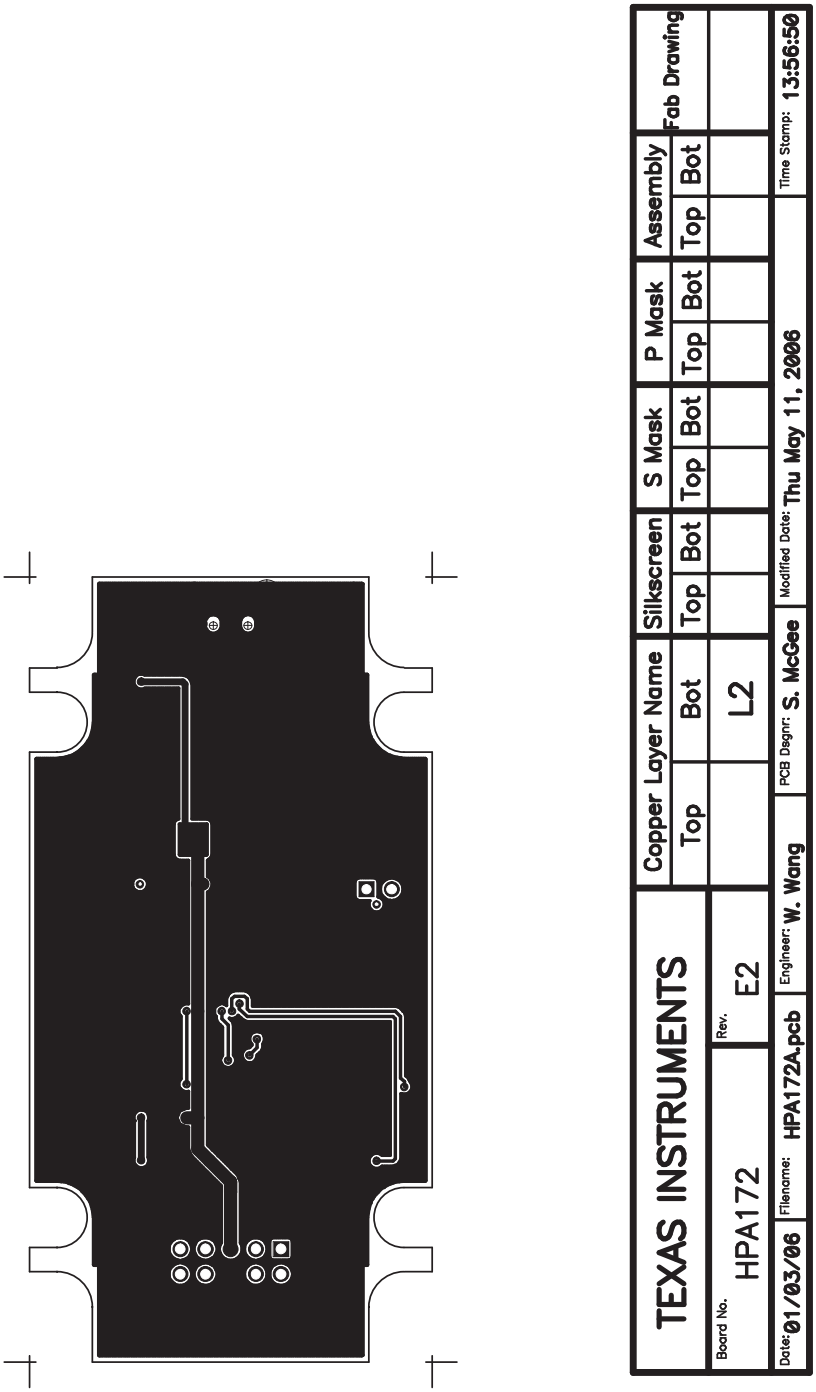

Figure 6-5. PCB Layout — Bottom L2 (Copper Layer)

Lists of Communication Protocol

This appendix contains a list of communication commands that can be sent from a host PC to the USB interface adapter, as well as a list of communication commands that can be sent from the USB interface adapter to the host PC.

A.1 Host PC to USB Interface Adapter Commands

COMMAND DESCRIPTION	USB PACKET BYTE 0 COMMAND CODE	USB PACKET BYTE 1	USB PACKET BYTE 2	USB PACKET BYTE 3	USB PACKET BYTE 4	USB PACKET BYTE 5	USB PACKET BYTES 6-63	COMMENTS
Firmware Version	0x00							See Chapter 4
I²C Write	0x14	A	B	C	C number of bytes (from 1 up to 60 bytes) to write			
I²C Read	0x15	A	B	C	D (from 1 to 62 bytes)			
Read/Write Port 0	0x16	A (bit.x=1 for read/input, bit.x=0 for write/output)	B (byte to write)					B does not apply to port 0 read. Each GPIO can be configured as input or output.
EEPROM Program	0x18	A = Starting address MSB	B = Starting address LSB	C (# of bytes to program)	Data to be programmed into EEPROM (up to 32 bytes)			
EEPROM Read	0x19	A = Starting address MSB	B = Starting address LSB	C (# of bytes to program)				C ≤ 60
Set Pullup Resistors	0x1A	A (for SDA)	B (for SCL)	C (for ALERT)				0x00 for no pullups (switch off), 0x01 for 2.2 kΩ, 0x02 for 1 kΩ, 0x03 for 688 Ω. For ALERT, the option is either open (0x00) or 2.2 kΩ (0x01).
Set I²C/PMBUS/SMBUS Speed	0x1B	A (0 for 100 kHz, otherwise 400 kHz)						
Generic I²C Write	0x1C	A	A number of bytes (from 2 up to 62) to write					The first byte of data to write is usually (device address << 1) + write bit 0.
Generic I²C Read	0x1D	A	A number of bytes (2 to 60) to write			Y = 1st byte to write + read bit 1 USB Byte 62	X # of bytes (1 to 62) to read back USB Byte 63	The first byte of data to write is usually (device address << 1) + write bit 0.

COMMAND DESCRIPTION	USB PACKET BYTE 0 COMMAND CODE	USB PACKET BYTE 1	USB PACKET BYTE 2	USB PACKET BYTE 3	USB PACKET BYTE 4	USB PACKET BYTE 5	USB PACKET BYTES 6-63	COMMENTS
Board Test	0x1F							Start toggling eight LEDs connected to the eight GPIO pins simultaneously for 5 s.

Note: See Chapter 3 for detailed explanation of each command.

A.2 USB Interface Adapter to Host PC Response Commands

COMMAND DESCRIPTION	USB PACKET BYTE 0 COMMAND CODE	USB PACKET BYTE 1	USB PACKET BYTE 2	USB PACKET BYTE 3	USB PACKET BYTE 4	USB PACKET BYTE 5	USB PACKET BYTES 6-63	COMMENTS
Response from Firmware Version	0x80	Family number (0 to 255)	Main version (0 to 255)	Minor version (0 to 255)				
Response from I^2C Write	0x94	Success: 0 Fail: 1						
Response from I^2C Read	0x95	Success: 0 Fail: 1	D number of bytes read					
Response from Read/Write Port 0	0x96	Success: 0 Fail: 1	Byte read from port 0					
Response from EEPROM Program	0x98	Success: 0 Fail: 1						
Response from EEPROM Read	0x99	Success: 0 Fail: 1	C number of bytes read back from EEPROM (up to 32 bytes)					
Response from Set Pullup Resistors	0x9A	Success: 0 Fail: 1						Pullups have been set.
Response from Set I^2C/PMBUS/SMBUS Speed	0x9B							Done with speed selection.
Response from Generic I^2C Write	0x9C	Success: 0 Fail: 1						
Response from Generic I^2C Read	0x9D	Success: 0 Fail: 1	X number of bytes (1 to 62) read back from slave					
Response from Board Test	0xFF							Board test is done.

Note: See Chapter 3 for detailed explanation of each command.

PMBUS/SMBUS Communications

This appendix describes the support for USB-to-PMBUS/SMBUS communications in the USB Interface Adapter and how to use it for your applications.

B.1 Overview

The default firmware for USB Interface Adapter provides support for USB to PMBUS (revision 1.0) and SMBUS (revision 2.0) communications. Similar to its I^2C application, the USB Interface Adapter acts as a PMBUS/SMBUS master, which can talk with one or multiple PMBUS/SMBUS slave devices daisy-chained to a 10-pin ribbon cable.

B.2 Basic SMBUS Transaction Types Supported

Because PMBUS communication is derived from SMBUS communication, they share basic SMBUS transaction types in common. These basic SMBUS transaction types include:

- Send byte
- Receive byte
- Write byte
- Read byte
- Write word
- Read word
- Block write
- Block read
- Process call
- Block read – block write process call

In addition, PMBUS communication also supports Group command.

All these commands are implemented in the USB Interface Adapter firmware.

B.3 Special Signals Used for PMBUS/SMBUS Communications

Refer to Table 2-2 for special signal lines used for PMBUS/SMBUS communications.

Specifically, a slave device can assert an ALERT signal (by pulling it low) to report a fault or a warning condition to the USB Interface Adapter; the host PC can obtain the ALERT signal by polling the USB Interface Adapter. Note that the USB Interface Adapter does not support any Host Notify Protocol. This means that the USB Interface Adapter cannot become a PMBUS/SMBUS slave and, therefore, cannot support a slave to temporarily become a master and notify the host about the fault or the warning condition.

For PMBUS communications, the USB Interface Adapter provides five CONTROL signals, each of which can be used to enable/disable the powering up of one or multiple slave device(s).

Write Protect signal is not supported; neither is the RESET signal.

B.4 Packet Error Checking (PEC) Implementation

By default, the USB Interface Adapter firmware implements PEC for all the different commands. However, the implementation of PEC byte is optional. There is a special host PC to USB Interface Adapter command that can be used to turn PEC on or off (see Section B.23).

If PEC is on or enabled, the firmware in the USB Interface Adapter either appends a PEC byte to a command sent to one or multiple slave devices or checks the validity of a PEC byte received from a slave device.

B.5 Communication Clock Speed and Clock Stretching

The default clock speed is 100 kHz, which is compatible with PMBUS 1.0 and SMBUS 2.0. For compatibility with future PMBUS specs, the clock speed can be increased to 400 kHz (see Section 3.9).

For each PMBUS/SMBUS transaction, the accumulative time of clock stretching cannot be more 25 ms.

B.6 Configuration of Pullup Resistors for ALERT, DATA, and CLOCK Lines

Refer to Section 3.8 for more details on this topic.

B.7 Limitations

Because of the limited capabilities of the TUSB3210 USB peripheral chip, the USB Interface Adapter has the following limitations:

- Extended Command is not supported. Only the first 256 commands are supported.
- For any block write or block read command, the number of data bytes is limited to 32 bytes only, even though the PMBUS specs support up to 256 bytes of data per transaction.
- Address Resolution Protocol (ARP) is not supported, so no multiple masters are allowed.
- Host Notify Protocol is not supported (see Section B.3).

B.8 List of Host PC to USB Interface Adapter Commands

Table B-1. Host PC to USB Interface Adapter Commands

COMMAND DESCRIPTION	USB PACKET BYTE 0 COMMAND CODE	USB PACKET BYTE 1	USB PACKET BYTE 2	USB PACKET BYTE 3	USB PACKET BYTE 4	USB PACKET BYTE 5	USB PACKET BYTES 6–63	COMMENTS
Send Byte	0x01	A	B					
Receive Byte	0x02	A						
Write Byte	0x03	A	B	C				
Write Word	0x04	A	B	C	D			
Read Byte	0x05	A	B	C				
Read Word	0x06	A	B	C				
Process Call	0x07	A	B	C	D	E		
Block Write	0x08	A	B	C	C number of bytes (from 1 to 32 bytes) to write			
Block Read	0x09	A	B	C				
Block Read – Block Write Process Call	0x0A	A	B	C	D	C number of bytes (from 1 to 31 bytes) to write		
Group Command	0x0B	A	B	C	D (0xFF for last packet)	C number of bytes (from 1 to 32 bytes) to write		Write 0 to D if not for the last packet
Assert\Deassert CONTROL Lines	0x0C	Bit 0 for line 1, ..., bit 4 for line 5						Bit logic 1 for assert, bit logic 0 for deassert
Poll PMBUS Signal Lines	0x0F							
Turn On/Off PEC	0x11	0 for off, otherwise on						

B.9 List of USB Interface Adapter to Host PC Response Commands

Table B-2. USB Interface Adapter to Host PC Response Commands

COMMAND DESCRIPTION	USB PACKET BYTE 0 COMMAND CODE	USB PACKET BYTE 1	USB PACKET BYTE 2	USB PACKET BYTE 3	USB PACKET BYTE 4	USB PACKET BYTE 5	USB PACKET BYTES 6–63	COMMENTS
Response from Send Byte	0x81	Success: 0 Fail: 1						
Response from Receive Byte	0x82	Success: 0 Fail: 1	Byte read from slave					
Response from Write Byte	0x83	Success: 0 Fail: 1						
Response from Write Word	0x84	Success: 0 Fail: 1						
Response from Read Byte	0x85	Success: 0 Fail: 1	Byte read from slave					
Response from Read Word	0x86	Success: 0 Fail: 1	Low byte of word read from slave	High byte of word read from slave				
Response from Process Call	0x87	Success: 0 Fail: 1	Low byte of word read from slave	High byte of word read from slave				
Response from Block Write	0x88	Success: 0 Fail: 1						
Response from Block Read	0x89	Success: 0 Fail: 1	X	X number of bytes read from slave (up to 32 bytes)				
Response from Block Read – Block Write Process Call	0x8A	Success: 0 Fail: 1	X	X number of bytes read from slave (up to 31 bytes)				
Response from Group Command	0x8B	Success: 0 Fail: 1						
Response from Assert/Deassert CONTROL Lines	0x8C	Success: 0 Fail: 1						
Response from Poll PMBUS Signal Lines	0x8F	Bit 0 for CONTROL 1, …, bit 4 for CONTROL 5, bit 5 for ALERT, bits 6 and 7 not used						Logic high for 1, logic low for 0
Response from Turn On/Off PEC	0x91	Success: 0 Fail: 1						

See the following sections for detailed explanation of each command.

B.10 Send Byte Command and Response

COMMAND DESCRIPTION	USB PACKET BYTE 0 COMMAND CODE	USB PACKET BYTE 1	USB PACKET BYTE 2	USB PACKET BYTE 3	USB PACKET BYTE 4	USB PACKET BYTE 5	USB PACKET BYTES 6–63	COMMENTS
Send Byte	0x01	A	B					

COMMAND DESCRIPTION	USB PACKET BYTE 0 COMMAND CODE	USB PACKET BYTE 1	USB PACKET BYTE 2	USB PACKET BYTE 3	USB PACKET BYTE 4	USB PACKET BYTE 5	USB PACKET BYTES 6–63	COMMENTS
Response from Send Byte	0x81	Success: 0 Fail: 1						

The following describes the detailed bit banking of the Send Byte command (with PEC on):
PMBUS/SMBUS start → write A (= slave address shifted left by 1 bit + write bit of 0) → slave ACK → write B (a command byte defined by PMBUS or SMBUS) → slave ACK → write PEC byte → slave ACK → PMBUS/SMBUS stop

B.11 Receive Byte Command and Response

COMMAND DESCRIPTION	USB PACKET BYTE 0 COMMAND CODE	USB PACKET BYTE 1	USB PACKET BYTE 2	USB PACKET BYTE 3	USB PACKET BYTE 4	USB PACKET BYTE 5	USB PACKET BYTES 6–63	COMMENTS
Receive Byte	0x02	A						

COMMAND DESCRIPTION	USB PACKET BYTE 0 COMMAND CODE	USB PACKET BYTE 1	USB PACKET BYTE 2	USB PACKET BYTE 3	USB PACKET BYTE 4	USB PACKET BYTE 5	USB PACKET BYTES 6–63	COMMENTS
Response from Receive Byte	0x82	Success: 0 Fail: 1	Byte read from slave					

The following describes the detailed bit banking of the Receive Byte command (with PEC on):
PMBUS/SMBUS start → write A (= slave address shifted left by 1 bit + read bit of 1) → slave ACK → read in 1 data byte from slave → master ACK → read in PEC byte from slave → master NACK → PMBUS/SMBUS stop

B.12 Write Byte Command and Response

COMMAND DESCRIPTION	USB PACKET BYTE 0 COMMAND CODE	USB PACKET BYTE 1	USB PACKET BYTE 2	USB PACKET BYTE 3	USB PACKET BYTE 4	USB PACKET BYTE 5	USB PACKET BYTES 6–63	COMMENTS
Write Byte	0x03	A	B	C				

COMMAND DESCRIPTION	USB PACKET BYTE 0 COMMAND CODE	USB PACKET BYTE 1	USB PACKET BYTE 2	USB PACKET BYTE 3	USB PACKET BYTE 4	USB PACKET BYTE 5	USB PACKET BYTES 6–63	COMMENTS
Response from Write Byte	0x83	Success: 0 Fail: 1						

The following describes the detailed bit banking of the Write Byte command (with PEC on):
PMBUS/SMBUS start → write A (= slave address shifted left by 1 bit + write bit of 0) → slave ACK → write B (a command byte defined by PMBUS or SMBUS) → slave ACK → write C (a data byte) → slave ACK → write PEC byte → slave ACK → PMBUS/SMBUS stop

B.13 Write Word Command and Response

COMMAND DESCRIPTION	USB PACKET BYTE 0 COMMAND CODE	USB PACKET BYTE 1	USB PACKET BYTE 2	USB PACKET BYTE 3	USB PACKET BYTE 4	USB PACKET BYTE 5	USB PACKET BYTES 6–63	COMMENTS
Write Word	0x04	A	B	C	D			

COMMAND DESCRIPTION	USB PACKET BYTE 0 COMMAND CODE	USB PACKET BYTE 1	USB PACKET BYTE 2	USB PACKET BYTE 3	USB PACKET BYTE 4	USB PACKET BYTE 5	USB PACKET BYTES 6–63	COMMENTS
Response from Write Word	0x84	Success: 0 Fail: 1						

The following describes the detailed bit banking of the Write Word command (with PEC on):
PMBUS/SMBUS start → write A (= slave address shifted left by 1 bit + write bit of 0) → slave ACK → write B (a command byte defined by PMBUS or SMBUS) → slave ACK → write C (low byte of a data word) → slave ACK → write D (high byte of the data word) → slave ACK → write PEC byte → slave ACK → PMBUS/SMBUS stop

B.14 Read Byte Command and Response

COMMAND DESCRIPTION	USB PACKET BYTE 0 COMMAND CODE	USB PACKET BYTE 1	USB PACKET BYTE 2	USB PACKET BYTE 3	USB PACKET BYTE 4	USB PACKET BYTE 5	USB PACKET BYTES 6–63	COMMENTS
Read Byte	0x05	A	B	C				

COMMAND DESCRIPTION	USB PACKET BYTE 0 COMMAND CODE	USB PACKET BYTE 1	USB PACKET BYTE 2	USB PACKET BYTE 3	USB PACKET BYTE 4	USB PACKET BYTE 5	USB PACKET BYTES 6–63	COMMENTS
Response from Read Byte	0x85	Success: 0 Fail: 1	Byte read from slave					

The following describes the detailed bit banking of the Read Byte command (with PEC on):
PMBUS/SMBUS start → write A (= slave address shifted left by 1 bit + write bit of 0) → slave ACK → write B (a command byte defined by PMBUS or SMBUS) → slave ACK → repeated start → write C (= slave address shifted left by 1 bit + read bit of 1) → slave ACK → read in 1 data byte from slave → master ACK → read in PEC byte from slave → master NACK → PMBUS/SMBUS stop

B.15 Read Word Command and Response

COMMAND DESCRIPTION	USB PACKET BYTE 0 COMMAND CODE	USB PACKET BYTE 1	USB PACKET BYTE 2	USB PACKET BYTE 3	USB PACKET BYTE 4	USB PACKET BYTE 5	USB PACKET BYTES 6–63	COMMENTS
Read Word	0x06	A	B	C				

COMMAND DESCRIPTION	USB PACKET BYTE 0 COMMAND CODE	USB PACKET BYTE 1	USB PACKET BYTE 2	USB PACKET BYTE 3	USB PACKET BYTE 4	USB PACKET BYTE 5	USB PACKET BYTES 6–63	COMMENTS
Response from Read Word	0x86	Success: 0 Fail: 1	Low byte of word read from slave	High byte of word read from slave				

The following describes the detailed bit banking of the Read Word command (with PEC on):
PMBUS/SMBUS start → write A (= slave address shifted left by 1 bit + write bit of 0) → slave ACK → write B (a command byte defined by PMBUS or SMBUS) → slave ACK → repeated start → write C (= slave address shifted left by 1 bit + read bit of 1) → slave ACK → read in the low byte of a data word from slave → master ACK → read in the high byte of the data word from slave → master ACK → read in PEC byte from slave → master NACK → PMBUS/SMBUS stop

B.16 Process Call Command and Response

COMMAND DESCRIPTION	USB PACKET BYTE 0 COMMAND CODE	USB PACKET BYTE 1	USB PACKET BYTE 2	USB PACKET BYTE 3	USB PACKET BYTE 4	USB PACKET BYTE 5	USB PACKET BYTES 6–63	COMMENTS
Process Call	0x07	A	B	C	D	E		

COMMAND DESCRIPTION	USB PACKET BYTE 0 COMMAND CODE	USB PACKET BYTE 1	USB PACKET BYTE 2	USB PACKET BYTE 3	USB PACKET BYTE 4	USB PACKET BYTE 5	USB PACKET BYTES 6–63	COMMENTS
Response from Process Call	0x87	Success: 0 Fail: 1	Low byte of word read from slave	High byte of word read from slave				

The following describes the detailed bit banking of the Process Call command (with PEC on):
PMBUS/SMBUS start → write A (= slave address shifted left by 1 bit + write bit of 0) → slave ACK → write B (a command byte defined by PMBUS or SMBUS) → slave ACK → write C (low byte of a data word) → slave ACK → write D (high byte of the data word) → slave ACK → repeated start → write E (= slave address shifted left by 1 bit + read bit of 1) → slave ACK → read in the low byte of a data word from slave → master ACK → read in the high byte of the data word from slave → master ACK → read in PEC byte from slave → master NACK → PMBUS/SMBUS stop

B.17 Block Write Command and Response

COMMAND DESCRIPTION	USB PACKET BYTE 0 COMMAND CODE	USB PACKET BYTE 1	USB PACKET BYTE 2	USB PACKET BYTE 3	USB PACKET BYTE 4	USB PACKET BYTE 5	USB PACKET BYTES 6–63	COMMENTS
Block Write	0x08	A	B	C	C number of bytes (from 1 up to 32 bytes) to write			

COMMAND DESCRIPTION	USB PACKET BYTE 0 COMMAND CODE	USB PACKET BYTE 1	USB PACKET BYTE 2	USB PACKET BYTE 3	USB PACKET BYTE 4	USB PACKET BYTE 5	USB PACKET BYTES 6–63	COMMENTS
Response from Block Write	0x88	Success: 0 Fail: 1						

The following describes the detailed bit banking of the Block Write command (with PEC on):
PMBUS/SMBUS start → write A (= slave address shifted left by 1 bit + write bit of 0) → slave ACK → write B (a command byte defined by PMBUS or SMBUS) → slave ACK → write C (number of bytes to write to slave) → slave ACK → repeat (write next byte → slave ACK) for C number of times → write PEC byte → slave ACK → PMBUS/SMBUS stop

B.18 Block Read Command and Response

COMMAND DESCRIPTION	USB PACKET BYTE 0 COMMAND CODE	USB PACKET BYTE 1	USB PACKET BYTE 2	USB PACKET BYTE 3	USB PACKET BYTE 4	USB PACKET BYTE 5	USB PACKET BYTES 6–63	COMMENTS
Block Read	0x09	A	B	C				

COMMAND DESCRIPTION	USB PACKET BYTE 0 COMMAND CODE	USB PACKET BYTE 1	USB PACKET BYTE 2	USB PACKET BYTE 3	USB PACKET BYTE 4	USB PACKET BYTE 5	USB PACKET BYTES 6–63	COMMENTS
Response from Block Read	0x89	Success: 0 Fail: 1	X	X number of bytes read from slave (up to 32 bytes)				

The following describes the detailed bit banking of the Block Read command (with PEC on):
PMBUS/SMBUS start → write A (= slave address shifted left by 1 bit + write bit of 0) → slave ACK → write B (a command byte defined by PMBUS or SMBUS) → slave ACK → repeated start → write C (= slave address shifted left by 1 bit + read bit of 1) → slave ACK → read in X (number of data bytes to read from slave) → master ACK → repeat (read in next byte → master ACK) for X number of times → read in PEC byte from slave → master NACK → PMBUS/SMBUS stop

B.19 Block Read – Block Write Process Call Command and Response

COMMAND DESCRIPTION	USB PACKET BYTE 0 COMMAND CODE	USB PACKET BYTE 1	USB PACKET BYTE 2	USB PACKET BYTE 3	USB PACKET BYTE 4	USB PACKET BYTE 5	USB PACKET BYTES 6–63	COMMENTS
Block Read – Block Write Process Call	0x0A	A	B	C	D	C number of bytes (from 1 up to 31 bytes) to write		

COMMAND DESCRIPTION	USB PACKET BYTE 0 COMMAND CODE	USB PACKET BYTE 1	USB PACKET BYTE 2	USB PACKET BYTE 3	USB PACKET BYTE 4	USB PACKET BYTE 5	USB PACKET BYTES 6–63	COMMENTS
Response from Block Read – Block Write Process Call	0x8A	Success: 0 Fail: 1	X	X number of bytes read from slave (up to 31 bytes)				

The following describes the detailed bit banking of the Block Read – Block Write Process Call command (with PEC on):
PMBUS/SMBUS start → write A (= slave address shifted left by 1 bit + write bit of 0) → slave ACK → write B (a command byte defined by PMBUS or SMBUS) → slave ACK → write C (number of data bytes to write) → slave ACK → repeat (write next data byte → slave ACK) for C number of times → repeated start → write D (= slave address shifted left by 1 bit + read bit of 1) → slave ACK → read in X (number of data bytes to read from slave) → master ACK → repeat (read in next byte → master ACK) for X number of times → read in PEC byte from slave → master NACK → PMBUS/SMBUS stop

B.20 Group Command and Response

COMMAND DESCRIPTION	USB PACKET BYTE 0 COMMAND CODE	USB PACKET BYTE 1	USB PACKET BYTE 2	USB PACKET BYTE 3	USB PACKET BYTE 4	USB PACKET BYTE 5	USB PACKET BYTES 6–63	COMMENTS
Group Command	0x0B	A	B	C	D (0xFF for last packet)	C number of bytes (from 1 up to 32 bytes) to write		Write 0 to D if not for the last packet

COMMAND DESCRIPTION	USB PACKET BYTE 0 COMMAND CODE	USB PACKET BYTE 1	USB PACKET BYTE 2	USB PACKET BYTE 3	USB PACKET BYTE 4	USB PACKET BYTE 5	USB PACKET BYTES 6–63	COMMENTS
Response from Group Command	0x8B	Success: 0 Fail: 1						

The following describes the detailed bit banking of the Group command (with PEC on):
PMBUS/SMBUS start → write A (= slave address shifted left by 1 bit + write bit of 0) → slave ACK → write B (a command byte defined by PMBUS or SMBUS) → slave ACK → write C (number of bytes to write to slave) → slave ACK → repeat (write next byte → slave ACK) for C number of times → write PEC byte → slave ACK → if byte D = 0xFF, then (PMBUS/SMBUS stop); otherwise, wait to receive the next USB packet for more writing to slave(s)

B.21 Assert/Deassert CONTROL Lines Command and Response

COMMAND DESCRIPTION	USB PACKET BYTE 0 COMMAND CODE	USB PACKET BYTE 1	USB PACKET BYTE 2	USB PACKET BYTE 3	USB PACKET BYTE 4	USB PACKET BYTE 5	USB PACKET BYTES 6–63	COMMENTS
Assert/Deassert CONTROL Lines	0x0C	Bit 0 for line 1, …, bit 4 for line 5						Bit logic 1 for assert, bit logic 0 for deassert

COMMAND DESCRIPTION	USB PACKET BYTE 0 COMMAND CODE	USB PACKET BYTE 1	USB PACKET BYTE 2	USB PACKET BYTE 3	USB PACKET BYTE 4	USB PACKET BYTE 5	USB PACKET BYTES 6–63	COMMENTS
Response from Assert/Deassert CONTROL Lines	0x8C	Success: 0 Fail: 1						

Each CONTROL line can be used to enable/disable the powering up of one of multiple slaves.

B.22 Poll PMBUS Signal Lines Command and Response

COMMAND DESCRIPTION	USB PACKET BYTE 0 COMMAND CODE	USB PACKET BYTE 1	USB PACKET BYTE 2	USB PACKET BYTE 3	USB PACKET BYTE 4	USB PACKET BYTE 5	USB PACKET BYTES 6–63	COMMENTS
Poll PUMBUS Signal Lines	0x0F							

COMMAND DESCRIPTION	USB PACKET BYTE 0 COMMAND CODE	USB PACKET BYTE 1	USB PACKET BYTE 2	USB PACKET BYTE 3	USB PACKET BYTE 4	USB PACKET BYTE 5	USB PACKET BYTES 6–63	COMMENTS
Response from Poll PMBUS Signal Lines	0x8F	Bit 0 for CONTROL 1, …, bit 4 for CONTROL 5, bit 5 for ALERT, bits 6 and 7 not used						Logic high for 1, logic low for 0

Note that there is currently no support for RESET signal or WRITE_PROTECT signal.

B.23 Turn On/Off PEC Command and Response

COMMAND DESCRIPTION	USB PACKET BYTE 0 COMMAND CODE	USB PACKET BYTE 1	USB PACKET BYTE 2	USB PACKET BYTE 3	USB PACKET BYTE 4	USB PACKET BYTE 5	USB PACKET BYTES 6–63	COMMENTS
Turn On/Off PEC	0x11	0 for off, otherwise on						

COMMAND DESCRIPTION	USB PACKET BYTE 0 COMMAND CODE	USB PACKET BYTE 1	USB PACKET BYTE 2	USB PACKET BYTE 3	USB PACKET BYTE 4	USB PACKET BYTE 5	USB PACKET BYTES 6–63	COMMENTS
Response from Turn On/Off PEC	0x91	Success: 0 Fail: 1						

The default is PEC on. For data integrity and hence power-system reliability during PMBUS/SMBUS communications in switching power environments, TI strongly recommends the adoption of PEC byte for every PMBUS/SMBUS communication transaction.

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
Low Power Wireless	www.ti.com/lpw	Telephony	www.ti.com/telephony
		Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

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

Общество с ограниченной ответственностью «МосЧип» ИНН 7719860671 / КПП 771901001
Адрес: 105318, г.Москва, ул.Щербаковская д.3, офис 1107

Данный компонент на территории Российской Федерации

Вы можете приобрести в компании MosChip.

Для оперативного оформления запроса Вам необходимо перейти по данной ссылке:

http://moschip.ru/get-element

Вы можете разместить у нас заказ для любого Вашего проекта, будь то серийное производство или разработка единичного прибора.

В нашем ассортименте представлены ведущие мировые производители активных и пассивных электронных компонентов.

Нашей специализацией является поставка электронной компонентной базы двойного назначения, продукции таких производителей как XILINX, Intel (ex.ALTERA), Vicor, Microchip, Texas Instruments, Analog Devices, Mini-Circuits, Amphenol, Glenair.

Сотрудничество с глобальными дистрибьюторами электронных компонентов, предоставляет возможность заказывать и получать с международных складов практически любой перечень компонентов в оптимальные для Вас сроки.

На всех этапах разработки и производства наши партнеры могут получить квалифицированную поддержку опытных инженеров.

Система менеджмента качества компании отвечает требованиям в соответствии с ГОСТ Р ИСО 9001, ГОСТ РВ 0015-002 и ЭС РД 009

Офис по работе с юридическими лицами:

105318, г.Москва, ул.Щербаковская д.3, офис 1107, 1118, ДЦ «Щербаковский»

Телефон: +7 495 668-12-70 (многоканальный)

Факс: +7 495 668-12-70 (доб.304)

E-mail: info@moschip.ru

Skype отдела продаж:
moschip.ru
moschip.ru_4
moschip.ru_6
moschip.ru_9