

Introduction

This document describes a System Management Bus (SMBus) controller in an Altera MAX II CPLD. The example shows the versatility of MAX II CPLDs.

SMBus

SMBus, a derivative of I²C, is a two-wire interface through which various system components can communicate with each other and with the rest of the system. At any time, only one device can master the bus to conduct transactions with one slave or with multiple slaves.

You can use an SMBus as a control bus for system and power management related tasks. You can remove individual control lines because an SMBus can be used to communicate with multiple devices. This not only reduces pin count, but also ensures future expansion.

The designed controller follows SMBus specifications, version 2.0 (see www.smbus.org/specs).

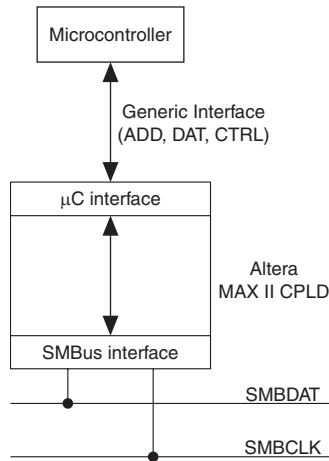
Using a MAX II CPLD as an SMBus Controller

The MAX II CPLD is a low cost, low power device. SMBus specification defines two classes of electrical characteristics: low power and high power. Implementing an SMBus controller in a MAX II CPLD with the SMBus operating at low power is a desirable solution in any low power consumption application.

The MAX II CPLD acts as a bridge between a host (for example, microcontroller or microprocessor) and the SMBus (see [Figure 1](#)). The controller incorporates a host interface and an SMBus interface with the control signals coming from the former to the latter. The designed controller is capable of functioning as a master or a slave.

The SMBus controller sits between a generic microcontroller bus (with address, data, and control signals) and the SMBus. It presents itself as a peripheral to the microcontroller and as an SMBus device (Master or Slave) on the SMBus.

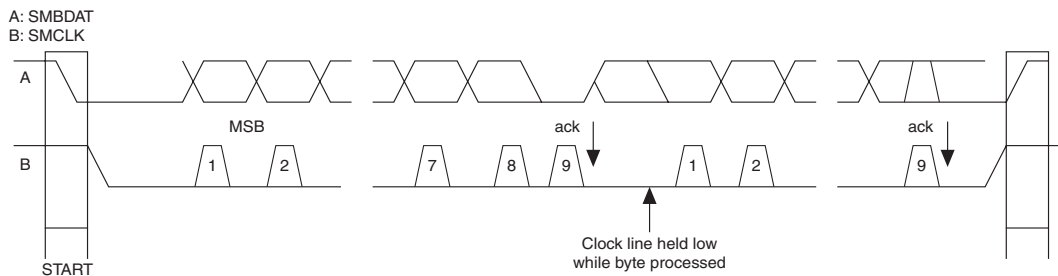
Figure 1. SMBus Block Diagram



Data Transfer on SMBus

Communication on the bus between a Master and a Slave is composed of four phases: START, Slave address, Data transfer, and STOP (see [Figure 2](#)). After the START phase, the Slave address is sent. Only the Slave whose address matches the address transmitted by the Master responds by sending back an acknowledge bit. When Slave addressing is achieved, the data transfer can proceed byte-by-byte. The Master can terminate the communication by generating a STOP signal to free the bus.

Figure 2. SMBus Data Transfer



The designed generic interface between the host and the SMBus controller includes an address bus, data bus, and the required control signals. The bus interface logic performs functions such as switching between Master and Slave mode, START/STOP signal generation, Packet Error Code (PEC) generation, R/W mode, and error notification.

The following features are incorporated in this design example:

- Generic and simple microcontroller interface
- Master and Slave mode of operation
- Arbitration lost interrupt with automatic mode switching from Master to Slave
- PEC generation and verification in master mode
- 98.215 KHz operation
- Clock low extension in both Master and Slave mode

Host Interface

The SMBus controller uses an asynchronous interface consisting of the signals shown in [Table 1](#).

Signal	Connection	Description
ADDRESS BUS [8]	Input	μC address bus used to select the desired register.
DATA BUS [8]	Bidir	μC data bus.
IRQ	Output	Interrupt request. This is an active high signal.
BUSY	Output	Indicates whether the bus is idle or busy. This is an active high signal.
CS	Input	Chip Select. This is an active low signal.
RD	Input	Places the data of the selected register on the data bus. This is an active high signal.
WR	Input	Writes the data present on the data bus to the selected register. This is an active high signal.
RESET	Input	Resets the controller. This is an active high signal.

All the signals are active high except CS, which is active low. When it is high, all the other lines become tri-stated except IRQ (Interrupt Request). [Table 2](#) show the various registers and their corresponding addresses.

A1	A0	Selected Register
1	1	Address Register
0	0	Data Register
1	0	Status Register

A1 and A0 are the last two bits of the 8-bit wide address bus, with A0 being the LSB. The other six bits of the bus are all zeros (you can change this if necessary).

Address Register

The address register is an 8-bit register and stores the address of the slave module of the controller. The LSB is used to enable or disable the controller, because the address is only 7 bits. If the LSB is set, the controller is enabled. Clearing it forces the controller to not respond when it detects that its address is being sent on the bus ([Table 3](#)).

Bit	Name	Description
7..1	Slave Address	Address of the controller (in slave mode)
0	Enable/Disable	If set, enables the controller

Data Register

The data register contains the data to be written or read from the SMBDAT line. It is used to transfer the data from the SMBDAT line to the host and vice versa ([Table 4](#)).

Bit	Name	Description
7..0	Data Register	SMBus data

Status Register

The status register contains the status information of the on going process. [Table 5](#) shows the bit positions and descriptions.

Bit	Name	Description	
		Set	Cleared
7	AM	By the SMBus controller when Address matches in slave mode	By the SMBus controller after the operation to be carried in the slave mode is completed
6	DTE	By the SMBus controller if data cannot be transferred only in slave mode	By the host before the next Master/Slave operation
5	AL	Arbitration was lost	Normal operation
4	M/S	SMBus controller is functioning as a master	SMBus controller is functioning as a slave
3	R/W	Forces the SMBus controller to read data from SMBDAT	Forces the SMBus controller to write data on SMBDAT
2	PEC	Only in the master mode if the slave is PEC enabled	By the microcontroller
1	STOP	Generates a stop condition	Normal operation
0	START	Generates a start condition	Normal operation

The status register is cleared if a STOP is generated by the controller. Only the bits 4 to 0 in the status register are written by the host. The value on the remaining bits should not be changed by the host. All the registers listed in [Table 5](#) can be read and/or written.

Communicating with the SMBus Controller

Whenever the host wants to communicate with the controller, it should first read the status register to determine the present state of the controller and if necessary, write into it and then into other registers.

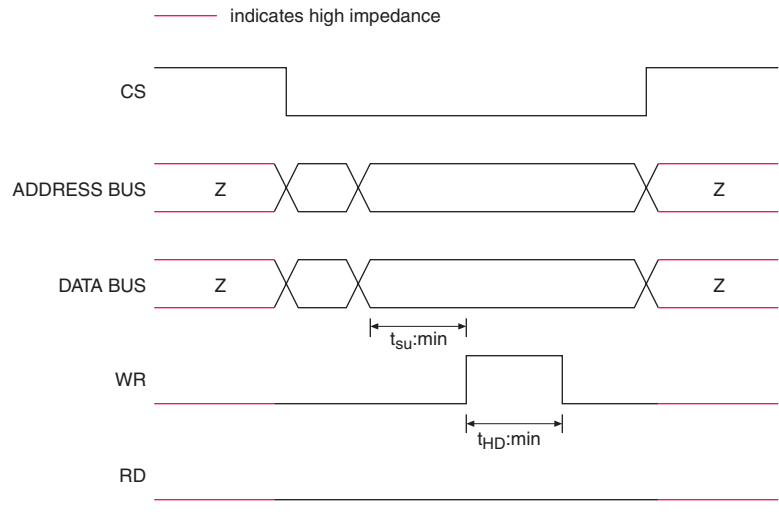
[Figure 3](#) (write cycle) and [Figure 4](#) (read cycle) show the sequence that the host must follow to communicate with the controller.

Write Cycle:

1. Make CS low.
2. Place address of the desired register on address bus.
3. Place data on data bus.

4. Assert \overline{WR} to write for a minimum of one internal clock period of the controller (5.5MHz).

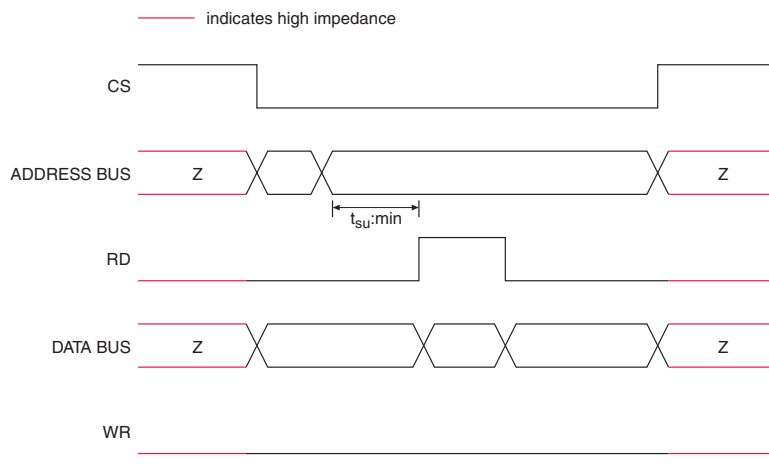
Figure 3. Write Cycle



Read Cycle:

1. Make CS low.
2. Place address of the desired register on address bus.
3. Assert RD to read data from data bus.

A read/write operation always reads/writes an entire register. Single bit operations are not possible.

Figure 4. Read Cycle

The following is a typical example of using the status register:

If the host wants to configure the controller in Master mode to perform a read operation with CRC checking enabled, it will set the *START*, *PEC*, *R/W*, and *M/S* bits of the status register. The first byte of data that is transferred on the bus (address of the slave to be communicated) is written in the data register. The controller then serially outputs the data on the *SMBDAT* line. After it receives an acknowledgement, it reads a byte of data and interrupts the host, which should now read the data from the data register. If the data is not read, the controller waits for approximately 32 microseconds and sets the *IRQ* and the *busy_bus*. The host sets the *STOP* bit in the status register when it wants the controller to generate a stop condition after reading a byte of data. That is, if the *STOP* bit is set in the status register, *STOP* is generated after reading two bytes if *PEC* is set. If *PEC* is not set, *STOP* is generated after reading only one byte of data.

In Master mode, the controller always waits for approximately 32 microseconds for the host to respond after raising the *IRQ*.

IRQ goes high in the following situations:

In Master Write mode:

- If the data byte written in the data register is successfully transferred, it gives an indication to the host to write the next byte to the data register.
- If the acknowledgement is not received from the slave for the data transferred.

- If arbitration is lost.
- In the PEC set mode, if the IRQ is raised even after the STOP bit is set, it indicates that the PEC received from the slave did not match the PEC generated by the controller.

In Master Read mode:

- If the acknowledgement is not received from any of the slaves for the address byte transferred on the SMBDAT line.
- If a byte of data is received from the slave, giving an indication to the host to read the byte.
- If arbitration is lost.

In Slave mode (clock low stretching in Slave mode is not supported by this controller):

- If the address received from the Master matches with the data in the address register.
- If the reading/writing a byte of data on the SMBDAT line is completed—to give an indication to the host.

Busy signal indicates the following:

- If asserted, the busy signal indicates that data is being transferred on the SMBDAT line.
- If IRQ is asserted and the busy signal is low, it indicates that the current mode was carried out without any error.
- When no operation is being carried out if the BUSY signal is low, it indicates that there is no activity on the SMBus (that is, the SMBus is idle).
- If IRQ is asserted along with the busy signal, it indicates the failure of the current operation. The reasons for this may be one of the following:

In Master mode:

- An acknowledgement has not been received from the slave.
- Arbitration is lost.
- The host took longer than 32 microseconds (approximately) to respond after IRQ assertion.
- PEC received in the master read mode was different from the PEC calculated by the controller.

In Slave Mode:

- An acknowledgement has not been received in the slave write mode.
- The host could not respond within the low period of the clock on the SMBCLK line after the IRQ was asserted.
- STOP condition was detected on the SMBus.

Implementation

You can implement this design using an EPM1270. The design source code is compiled and programmed into the MAX II CPLDs. Host interfacing ports and SMBus lines are mapped on convenient I/O pins.

Source Code

This design example has been implemented in Verilog HDL and successful operation has been demonstrated using an accompanying testbench. The source code, testbench, and complete Quartus II project are available at:

www.altera.com/literature/an/an502_design_example.zip

Conclusion

As shown in this design example, MAX II CPLDs are an excellent choice to implement SMBus Controllers. Their low cost, low power, ability to tolerate any possible power-on sequence on their core, and I/O banks make them ideally suited for such applications.

Additional Resources

The following list contains additional resources:

- MAX II CPLD homepage:
www.altera.com/products/devices/cpld/max2/mx2-index.jsp
- MAX II Device Literature:
www.altera.com/literature/lit-max2.jsp
- MAX II Power-Down Designs:
www.altera.com/support/examples/max/exm-power-down.html
- MAX II Application Notes:
AN 428: MAX II CPLD Design Guidelines
AN 422: Power Management in Portable Systems Using MAX II CPLDs

Revision History

Table 6 shows the revision history for this application note.

<i>Table 6. Revision History</i>		
Date and Version	Changes Made	Summary
December 2007 v1.0	Initial release	—



101 Innovation Drive
San Jose, CA 95134
www.altera.com
Literature Services:
literature@altera.com

Copyright © 2007 Altera Corporation. All rights reserved. Altera, The Programmable Solutions Company, the stylized Altera logo, specific device designations, and all other words and logos that are identified as trademarks and/or service marks are, unless noted otherwise, the trademarks and service marks of Altera Corporation in the U.S. and other countries. All other product or service names are the property of their respective holders. Altera products are protected under numerous U.S. and foreign patents and pending applications, maskwork rights, and copyrights. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera Corporation. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.



I.S. EN ISO 9001