# Implementing SMBus Controller in Altera MAX Series

2014.09.22

AN-502



This design example shows the implementation of System Management Bus (SMBus) controller for Altera® MAX® II, MAX V, and MAX 10 devices.

SMBus, a derivative of $I^2C$, is a two-wire interface which various system components can communicate with each other. At any time, only one device can master the bus to conduct transactions with one or multiple slaves.

With SMBus you can:

- use an SMBus as a control bus for system and power management related tasks
- remove individual control lines because an SMBus can be used to communicate with multiple devices

With this setup, you can reduce pin count and ensures future expansion of your design.

**Related Information**

- **Design Example for MAX II**
  Provides the MAX II design file for this application note (AN 502).
- **Design Example for MAX 10**
  Provides the MAX 10 design file for this application note (AN 502).
- **Power Management in Portable Systems Using MAX II CPLDs**
- **MAX II CPLD Design Guidelines**

## Using a MAX II Device as an SMBus Controller

The detailed description of the implementation is based on the MAX II devices. This application can also be implemented in MAX V and MAX 10 devices.

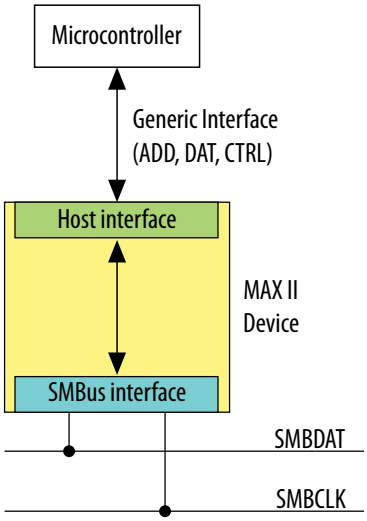
You can implement this design using an EPM1270. The design source code is compiled and programmed into the MAX II devices. Host interfacing ports and SMBus lines are mapped on convenient I/O pins.

The MAX II device 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 device with the SMBus operating at low power is a desirable solution in any low power consumption application.

The MAX II device acts as a bridge between a host (for example, microcontroller or microprocessor) and the SMBus. 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 between a Master and a Slave on the bus is composed of START, Slave address, Data transfer, and STOP phases.

The following are the communication phases flow:

1. After the START phase, the Slave address is sent.
2. Only the Slave whose address matches the address transmitted by the Master responds by sending back an acknowledge bit.
3. When Slave addressing is achieved, the data transfer will proceed byte-by-byte.
4. The Master can terminate the communication by generating a STOP signal to free the bus.

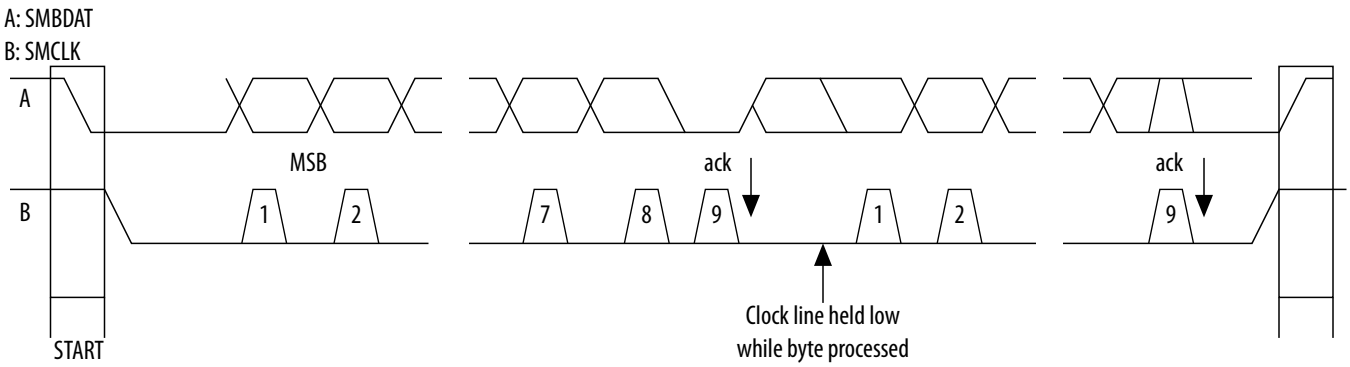
The bus interface logic performs the following functions:

- Switching between Master and Slave mode
- START/STOP signal generation
- Packet Error Code (PEC) generation
- R/W mode
- 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

**Figure 2: SMBus Data Transfer**




## Host Interface

The host interface of an SMBus controller uses interface signals and registers as generic interface to communicate with the host.

### Interface Signals

The SMBus controller uses an asynchronous interface consisting of the signals shown in the following table.

**Table 1: Asynchronous Interface Signals for SMBus Controller**

All the signals are active high except CS, which is active low. When it is high, all the other lines become tri-stated except Interrupt Request (IRQ).

| 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. |
| BUSY | Output | Indicates whether the bus is idle or busy. |
| CS | Input | Chip Select. |
| RD | Input | Places the data of the selected register on the data bus. |
| WR | Input | Writes the data present on the data bus to the selected register. |
| RESET | Input | Resets the controller. |

### Registers

The host interface includes address register, data register and status register. Each register has its own corresponding address.

**Table 2: Registers for Adress, Data and Status Registers**

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

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

## Address Register

The address register is an 8-bit register and stores the address of the slave module of the controller.

**Table 3: Address Register**

You can use the LSB to enable or disable the controller and the remaining 7 bits as the address of the controller. 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.

| 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: Data Register**

| Bit | Name | Description |
|---|---|---|
| 7..0 | Data Register | SMBus data |

## Status Register

The status register contains the status information of the ongoing process.

**Table 5: Status Register Bits**

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 can be read and/or written.

| 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 |

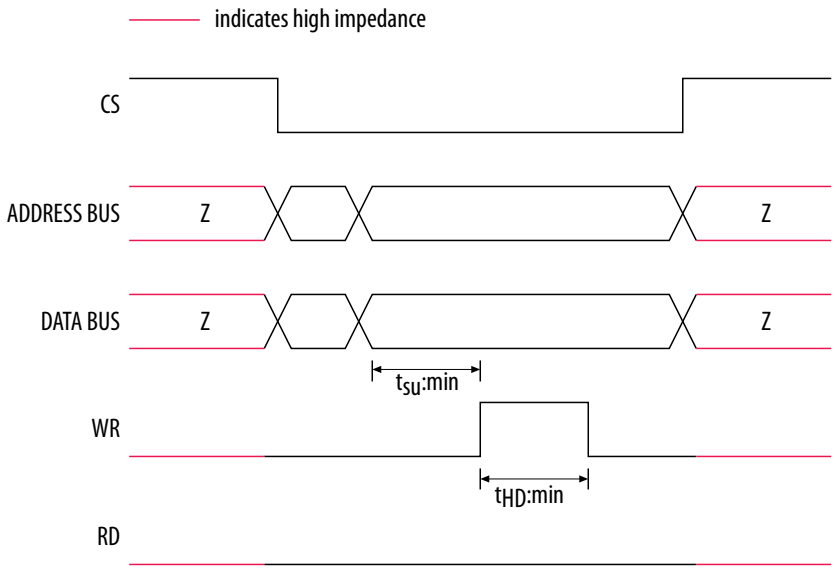
| Bit | Name | Description | |
|---|---|---|---|
| | | Set | Cleared |
| 6 | DTE | By the SMBus controller if data cannot be transferred only in slave mode | By the host before the next Master/ Slave operation |
| 5 | AL | Arbitration was lost | Normal operation |
| 4 | M/S | SMBus controller is functioning 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 onSMBDAT" |
| 2 | PEC | "Only in the master mode if the slave isPEC enabled" | By the microcontroller |
| 1 | STOP | Generates a stop condition | Normal operation |
| 0 | START | Generates a start condition | Normal operation |

## 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.

### Write Cycle

**Figure 3: Timing Diagram for Write Cycle**

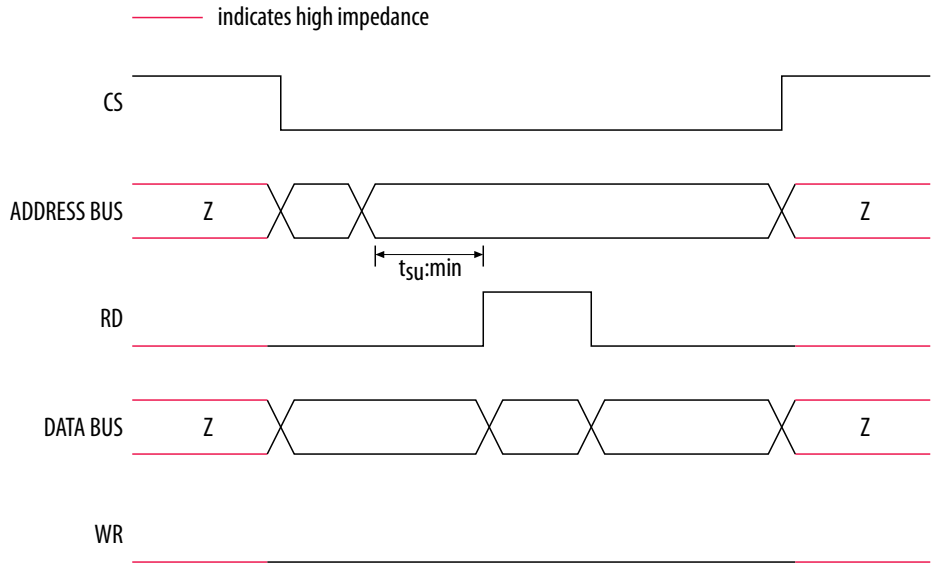



The following describes the write cycle:

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

### Read Cycle

**Figure 4: Timing Diagram for Read Cycle**




The following describes the read cycle:

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

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

## Using the Status Register

If the host wants to configure the controller in Master mode to perform a read operation with the CRC checking enabled:

1. The host sets 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.
2. The controller then serially outputs the data on the SMBDAT line.
3. After the controller receives an acknowledgement, the controller reads a byte of data and interrupts the host, which should now read the data from the data register.
4. If the host is not reading the data from the data register, the controller waits for approximately 32 ms and sets the IRQ and the busy_bus signals.
5. If the host wants the controller to generate a stop condition after reading a byte of data, the host sets the STOP bit in the status register. The STOP bit is generated after reading two bytes if the PEC bit is set. If the PEC bit is not set, the STOP bit is generated after reading only one byte of data.

## IRQ Signal

A raised IRQ signal means there is an interrupt request. IRQ goes high in Master Write, Master Read and Slave Mode if the following situation occurs.

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:

1. If the acknowledgement is not received from any of the slaves for the address byte transferred on the SMBDAT line.
2. If a byte of data is received from the slave, giving an indication to the host to read the byte.
3. 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

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 BUSY signal goes high in Master and Slave Mode if the following situation occurs.

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.

## Acknowledgments

Design example adapted for Altera MAX 10 FPGAs by:

Orchid Technologies Engineering and Consulting, Inc.

Maynard, Massachusetts 01754

TEL: 978-461-2000

WEB: **www.orchid-tech.com**

EMAIL: **info@orchid-tech.com**

## Document Revision History

| Date | Version | Changes |
|---|---|---|
| September 2014 | 2014.09.22 | • Added devices<br>• Updated template<br>• Restructure document |
| December 2007 | 1.0 | Initial release |