**Introduction**

The Altera® MAX® II CPLD family offers a non-volatile user flash memory (UFM) block. The UFM block provides the capability to store user-defined data which enables the integration of discrete, non-volatile storage into a single MAX II device—reducing system chip count and cost.

The System Management Bus (SMBus) is a two-wire interface protocol. Using the interface bus, various system component chips communicate with each other and the system. The SMBus provides a control bus for the whole system. A system may use the standard SMBus to pass messages to and from devices instead of using individual control lines and different interface protocols. This reduces pin count and reduces system complexity.

This design example allows you to access the MAX II UFM block through the SMBus interface protocol. The design example uses the MAX II logic array to implement an interface to a 2-Kbit addressable memory. The MAX II device acts as the slave device and receives instructions from a master device for data transfer.

This application note describes the SMBus protocols and the design example. The topics covered are:

- SMBus Interface Protocol
- SMBus–UFM Block Interface Design Example
- SMBus–UFM Block Interface Design Functional Description
- MAX II Device 5.0-V Compatibility
- Using the SMBus–UFM Block Interface Design Example with the Quartus II Software

**SMBus Interface Protocol**

The SMBus interface uses two communication wires for master and slave communication: the serial clock (SMBCLK) and the serial data (SMBDAT). Both lines are bidirectional. When the bus is idle, devices on the bus release both lines and pull-up resistors pull the lines high. The master device initiates data transfer between master and slave devices, and issues instructions to the slave device. Master devices in the SMBus interface also issue the SMBCLK signal to the slave devices for data transfer. **Figure 1** shows the connections between the devices and the bus lines.
The data length in the SMBus interface is an 8-bit long byte transfer with the most significant bit (MSB) transferred first. No restrictions exist on the number of bytes transferred. However, an acknowledge bit must follow every transfer byte. If the slave device is too busy to receive or transmit data, it can force the master device into the wait state by holding the clock line \textit{SMBCLK} low. The data reception and transmission resumes when the slave device releases the \textit{SMBCLK} clock line. \textit{Figure 2} shows a typical data transfer in the SMBus.

The following are the protocol definitions for the SMBus interface.

**Bus Lines Are Free**

Both \textit{SMBCLK} and \textit{SMBDAT} high indicate that the bus lines are free. A master device can initiate a new data transfer when the bus lines are free.

**Data Validity**

Changes to the \textit{SMBDAT} line while the \textit{SMBCLK} line is high, indicate a special condition, such as a Start, Stop, or Repeated Start condition. \textit{Figure 3} shows the data validity.
Start Condition (S)

A high-to-low transition on the SMBDAT line, while SMBCLK remains high, indicates a Start condition. A master device issues a Start condition to initiate a data transfer. When an SMBus-compatible device receives a Start condition, the device must reset its bus logic to anticipate the addressing from the master device for new communication.

Stop Condition (P)

A low-to-high transition on the SMBDAT line, while SMBCLK remains high, indicates a Stop condition. A master device issues a Stop condition to end a communication or data transfer and to free the bus line. A device cannot issue a Stop condition immediately after a Start or Repeat Start condition. Figure 4 shows the Start and Stop conditions.

Repeat Start (Sr)

The Repeat Start condition is identical to the Start condition. The master device uses Repeat Start between a Start and Stop condition to continue holding the bus busy while it initiates a new communication with other
devices. When an SMBus-compatible device receives a Repeat Start condition, the device must reset its bus logic to anticipate the addressing from the master device for new communication.

**Device Addressing**

When a master device issues the Start or Repeat Start condition, it starts with the first seven-bit transfer for device addressing. Device addressing tells an SMBus-compatible device that the master device has selected it for communication.

**Read or Write (Rd/Wr)**

The read or write is the eighth bit sent after the 7-bit device addressing. A value of 1 indicates a read operation, and a value of 0 indicates a write operation.

**Memory Addressing**

Memory addressing allows you to access a specific UFM block address. Specify an 8-bit address in order to write to or read from that address in the UFM block sector 0.

**Acknowledge (A)**

Immediately after receiving the entire 8 bits of data transferred by the transmitting device, the receiving device issues an Acknowledge bit. The receiving device pulls the SMBDAT line low when the SMBCLK clock pulses high to acknowledge the data transfer. Either the master or slave device can issue the Acknowledge bit, depending on the direction of the data transfer. To indicate Not-Acknowledge, the receiving device pulls the SMBDAT line high during the SMBCLK clock pulse.

If the master device does not receive the Acknowledge bit, or the slave device issued Not-Acknowledge after transferring a byte, the master device must generate a Stop condition to abort the transfer, or generate a Repeat Start condition to start a new transfer. If the slave device does not receive an Acknowledge bit after transferring a byte, the slave device terminates the data transfer and goes back to stand-by mode. However, at the last byte of transfer, if the master device does not generate an Acknowledge bit, it generates a Stop condition to indicate the end of a transfer.
For this design example, the MAX II device becomes a slave device for the SMBus interface. The master device generates clock signals on the SMBCLK line for communication on the SMBus interface. The SMBCLK signal is always an input for the SMBus interface design in the MAX II device. The master device transfers data to the MAX II slave device on the SMBDAT line at the positive clock edge of the SMBCLK clock. The slave device transfers data to the master device on the SMBDAT line at the negative clock edge of the SMBCLK clock.

Figure 5 shows the interface between the MAX II UFM block and the SMBus through the design example. Figure 6 shows the circuit diagram for the SMBus interface design.
The SMBus interface design allows access to all 256 addresses in sector 0 of the UFM block. However, you can access only the upper byte \([15..8]\) of each address. This emulates a 2-Kbit serial EEPROM memory. The design does not overwrite data stored in the lower byte \([7..0]\) of each address unless you perform the erase operation to erase a sector of the UFM block.

By default, the MAX II device uses the slave address of 1010110X, where “X” is slave address bit \([0]\). This bit is used to indicate a read or write operation. The first 7 bits 1010110 indicate the UFM block slave device. Other devices in the same bus should not use the same address. You can modify the Verilog HDL source code to change the slave address value.

**The Interface Operations**

This SMBus-UFM block interface design example allows four operations:

- Single-Byte Read Operation
- Page Read Operation
- Write Operation
- Erase Operation

**Single-Byte Read Operation**

The single-byte read operation allows you to read the data byte from a specific UFM block address. The master device issues a Repeat Start for the transfer and does not need to issue an Acknowledge bit after receiving
the data byte from the slave device. The master device ends the transmission by issuing a Stop condition after receiving the data. Figure 7 shows the interface protocol for the single-byte read operation.

Figure 7. Single-Byte Read Operation

<table>
<thead>
<tr>
<th>S</th>
<th>Slave Address (1010110)</th>
<th>Wr (0)</th>
<th>A</th>
<th>UFM Address [7..0]</th>
<th>A</th>
<th>Sr</th>
<th>Slave Address (1010101)</th>
<th>Rd (1)</th>
<th>A</th>
<th>Data Byte</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>From Master to Slave</td>
<td></td>
<td></td>
<td>From Slave to Master</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The following describes the single-byte read operation sequence.

1. The master device transmits the first byte, where the first 7 bits (1010 110) are for slave device addressing. The eighth bit is for read or write commands, and it has the value of 0 to indicate a write operation.

2. The slave device issues a 1-bit Acknowledge signal.

3. The master device transmits a second byte for memory addressing in the UFM block. These are bits [7..0] of the UFM block address.

4. The slave device issues a 1-bit Acknowledge signal.

5. The master device issues a Repeat Start condition.

6. The master device transmits the third byte, where the first 7 bits (1010 110) are for device addressing. The eighth bit is for read or write commands, and it has the value of 1 to indicate a read operation.

7. The slave device issues a 1-bit Acknowledge signal.

8. The slave device transmits one data byte.

9. The master device issues a Stop condition.

Page Read Operation

This operation allows you to read the upper-data byte from all addresses in sector 0 of the UFM block. The master device sends an Acknowledge bit after receiving each data byte from the slave device, except for the last byte. The master device ends the transmission by issuing a Stop signal after receiving the last data byte. Figure 8 shows the interface protocol for the page read operation.
The following describes the page read operation sequence.

1. The master device transmits the first byte, where the first 7 bits (1010 110) are for slave device addressing. The eighth bit is for read or write commands, and it has the value of 0 to indicate a write operation.

2. The slave device issues a 1-bit Acknowledge signal.

3. The master device transmits a second byte for memory addressing in the UFM block. These are bits [7..0] of the UFM block address.

4. The slave device issues a 1-bit Acknowledge signal.

5. The master device issues a Repeat Start condition.

6. The master device transmits the third byte, where the first 7 bits (1010 110) are for device addressing. The eighth bit is for read or write commands, and it has the value of 1 to indicate read operation.

7. The slave device issues a 1-bit Acknowledge signal.

8. The slave device transmits the first data byte.

9. The master device issues a 1-bit Acknowledge signal.

10. The slave device transmits the next data byte.

11. Repeat Steps 9 and 10 until the last data byte.

12. The master device issues a Stop condition.
**Write Operation**

The write operation allows you to write one data byte to the upper byte of a specific address in the UFM block sector 0. Figure 9 shows the interface protocol for the write operation.

**Figure 9. Write Operation**

<table>
<thead>
<tr>
<th>S</th>
<th>Slave Address (1010110)</th>
<th>Wr (0)</th>
<th>A</th>
<th>UFM Address [7..0]</th>
<th>A</th>
<th>Data Byte</th>
<th>A</th>
<th>P</th>
</tr>
</thead>
</table>

The following describes the write operation sequence.

1. The master device transmits the first byte, where the first seven bits (1010 110) are for device addressing. The eighth bit is for read or write commands, and it has the value of 0 to indicate a write operation.

2. The slave device issues a 1-bit Acknowledge signal.

3. The master device transmits a second byte for memory addressing in the UFM block. These are bits [7..0] of the UFM block address.

4. The slave device issues a 1-bit Acknowledge signal.

5. The master device transmits a third byte as data.

6. The slave device issues a 1-bit Acknowledge signal.

7. The master device issues a Stop condition.

**Erase Operation**

The erase operation allows you to erase the UFM block sector 0. There are two ways to start the erase operation:

**Write 8 bits of 1 (0xFF) to Address 0x00 of the UFM Block**

Writing 8 bits of 1 to the first address of sector 0 starts the erase. Figure 10 shows the interface protocol for the erase operation.
The following describes the erase operation sequence.

1. The master device transmits the first byte, where the first seven bits (1010 110) are for device addressing. The eighth bit is for read or write commands, and it has the value of 0 to indicate a write operation.

2. The slave device issues a 1-bit Acknowledge signal.

3. The master device transmits a second byte for memory addressing in the UFM block. These are bits [7..0] of the UFM block address. For the erase operation, this should be 8 bits of 0.

4. The slave device issues a 1-bit Acknowledge signal.

5. The master device transmits a third byte as data. For the erase operation, this should be 8 bits of 1.

6. The slave device issues a 1-bit Acknowledge signal.

7. The master device issues a Stop condition.

**Issue Slave Address of 10101010 (0xAA)**

Issuing a slave address of 10101010 to the device can also start an erase. When you use this address for the erase purpose, other slave devices on the same bus cannot use this address. Also, you cannot have more than one SMBus slave device that uses the same erase trigger, unless you change the trigger address. Figure 11 shows the interface protocol for the erase operation.

**Figure 10. Erase Operation (Writing 8 bits of 1 to Address 0x00)**

<table>
<thead>
<tr>
<th>S</th>
<th>Slave Address (1010110)</th>
<th>Wr (0)</th>
<th>A</th>
<th>UFM Address (00000000)</th>
<th>A</th>
<th>Data Byte (11111111)</th>
<th>A</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>From Master to Slave</td>
<td></td>
<td></td>
<td>From Slave to Master</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 11. Erase Operation (Writing 8 bits of 1 to address 0x00)**

<table>
<thead>
<tr>
<th>S</th>
<th>Slave Address (1010101)</th>
<th>Wr (0)</th>
<th>A</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>From Master to Slave</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

From Slave to Master
The following describes the erase operation sequence.

1. The master device transmits the first byte, where the first seven bits (1010 101) are for device addressing for the erase purpose. The eighth bit is for read or write commands, and it has the value of 0 to indicate a write operation.

2. The slave device issues a 1-bit Acknowledge signal.

3. The master device issues a Stop condition.

**SMBus–UFM Block Interface Design Usage Guidelines**

There are two separate design examples that can be used for the SMBus interface. The first design example supports all three operations: read, write and erase. The resource usage for this design example is around 250 logic elements (LE) and fits into a EPM570, EPM1270, or EPM2210 device. The second design example supports the read operation only with an LE count of approximately 200. You can implement this design example in all MAX II devices.

You can easily modify and further optimize the design example to fit your project needs.

**Write or Erase During ISP**

Make sure that all the UFM block erase or write operations are completed before starting any in-system programming (ISP) session (including stand-alone verify, examine, setting security bit, and reading the UFM block contents). You should never start an ISP session when any UFM block erase or write operation is on going as this may put the device in an unrecoverable state; however, this does not apply to the UFM block read operation.

To ensure that no UFM block erase or write operation is in progress (just prior to programming a MAX II device) you should use the real-time ISP feature to program the device. With real-time ISP, the RTP_BUSY signal goes high and programming starts 400 ms after the RTP_BUSY signal is high. The design example monitors this signal and ensures that no erase or write operation is in progress when the RTP_BUSY signal is high.

**MAX II Device 5.0-V Compatibility**

MAX II devices feature MultiVolt™ I/O operation, allowing interfaces between devices with different I/O voltages. MAX II devices support 3.0-V, 2.5-V, 1.8-V and 1.5-V I/O voltages. The SMBus protocol allows the bus lines to operate up to 5.0 V. The MAX II device requires some modifications to operate up to 5.0 V.
The **SMBDAT** pin is a bidirectional I/O pin and the **SMBCLK** pin is an input pin. Assign the **SMBDAT** pin as an open-drain I/O pin. The banks of the I/O pins assigned for **SMBDAT** and **SMBCLK** need to be operating at 3.3-V $V_{CCIO}$.

An external weak pull-up resistor is required to pull the **SMBDAT** and **SMBCLK** bus lines up to 5.0 V. When determining the pull-up resistor value, consider two main factors:

- The resistor value should be large enough so that the $I_{OL}$ does not exceed the maximum $I_{OL}$ value of the device when the bus is low.
- The resistor value should be small enough so the voltage level does not exceed the maximum $V_{IL}$ when the bus is low.

Enable the internal PCI clamping diode to protect the MAX II I/O pins. The internal PCI clamping diode capability is available in bank 3 of the EPM1270 and EPM2210 devices. For other I/O banks in EPM1270 and EPM2210 devices and for all I/O banks in EPM240 and EPM570 devices, use an external diode.

For more information on the MAX II device 5.0-V compatibility, refer to the *Using MAX II Devices In Multi-Voltage System* chapter of the *MAX II Device Handbook*.

### Using the SMBus–UFM Block Interface Design Example with the Quartus II Software

This section covers the steps of using the design example with the Quartus® II software. The processes are:

- Initializing the UFM Block Contents
- Creating the UFM Block Megafuction
- Creating and Compiling The Project
- Verifying the Design

### Initializing the UFM Block Contents

This design example allows you to initialize the UFM block contents. You can program user data into the UFM block at the same time as you program the design into the device. This allows you initialize the UFM block with data and use the read only design. You can overwrite the data in the UFM block using the erase and write operations. To initialize the UFM block contents, create a memory initialization file (.mif) or a hexadecimal file (.hex) and specify the file contents. Create an **altufm_none** megafuction and initialize the UFM block with the .mif or .hex file contents.
Creating the MIF or HEX Files in Quartus II

In the Quartus II software, you can create either a .mif or .hex file to initialize the UFM block contents.

1. Select New (File menu).

2. Select Hexadecimal (Intel Format) File or Memory Initialization File under the Other Files tab in the New window. Click OK.

3. The number of words (address) must be 512, although you can only read from or write to the 256 addresses in sector 0 of the UFM block. Set the word size to 16. The design only allows you to access the upper byte, \([15..8]\), of the 256 addresses in the UFM block sector 0. Your initialization data must be specified for the first 256 addresses and the most significant byte for each 16-bit word. Click OK.
4. A .mif or .hex file displays. Enter values to store in the UFM block and save the file. It isn’t necessary to enter the value for addresses in sector 1 or the lower byte value for addresses in sector 0.
Creating the UFM Block Megafunction

Use the MegaWizard® Plug-In Manager to:

- Create an altufm_none megafunction
- Use the UFM block
- Initialize the UFM block contents

1. Select MegaWizard Plug-In Manager (Tools menu).
2. Select Create a new custom megafunction variation. Click Next.
3. Select MAX II from the Which device family will you be using? list.
4. Expand the Memory compiler folder and select Flash Memory.
5. From the Which type of output file do you want to create? option, choose a file type.
6. For the What name do you want for the output file? field, click Browse to locate your project directory. Type in the file name and click Save. Page 2 displays (Figure 15).
7. Click **Next**. Page 3 displays (Figure 16).
8. From the **What is the interface protocol?** option, choose **None**.

**Figure 16. Specifying the Interface Protocol**

![Interface Protocol Specification](image)

9. Click **Next**. Page 4 displays **(Figure 17)**.
Figure 17. Specifying the MIF File

10. From the **Do you want to specify the initial content of the memory?** option, choose **Yes, use this file for the memory content data**.

11. Click **Browse** to locate the `.mif` or `.hex` file that you created. Select the file and click **Open**.

12. Click **Next**. Page 5 displays (Figure 18).

13. Select the **Quartus symbol file** and click **Finish**.
Using the SMBus–UFM Block Interface Design Example with the Quartus II Software

Figure 18. Creating Additional Files

When the 'Finish' button is pressed, the MegaWizard Plug-In Manager will create the checked files in the following list. You may choose to include or exclude a file by checking or unchecking its corresponding checkbox, respectively. The state of checkboxes will be remembered for the next MegaWizard Plug-In Manager session.

The MegaWizard Plug-In Manager will create these files in the directory:
C:\altera\quartus50\n
<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uf_m_none.v</td>
<td>Variation file</td>
</tr>
<tr>
<td>uf_m_none.inc</td>
<td>VHDL Include file</td>
</tr>
<tr>
<td>uf_m_none.smp</td>
<td>VHDL Component declaration file</td>
</tr>
<tr>
<td>uf_m_none.bdf</td>
<td>Quater symbol file</td>
</tr>
<tr>
<td>uf_m_none_inst.v</td>
<td>Instantiation template file</td>
</tr>
<tr>
<td>uf_m_none_bb.v</td>
<td>Verilog Black Box declaration file</td>
</tr>
</tbody>
</table>

Figure 19. Creating Additional Files

Creating and Compiling The Project

Create a project using the Block Diagram/Schematic File as the top-level entity. Verify that the .bdf file of the design example and the UFM block megafunction are in the project directory. Make the necessary pin assignments and compile your project to obtain the Programmer Object File (.pof). Figure 19 shows the top-level block diagram file of the design.

Figure 19. Top Level File
Verifying the Design

The design example is hardware verified to ensure that it can work properly for the read or write, page-read, write, and erase operations.

You can verify and simulate the design using the Quartus II Simulator. The design example passes timing simulation to ensure that it can work properly for the read, write, and erase operations. Figure 20 shows the UFM block write and read or write operations. Data 1010 1100 is first written to address 1101 0011 (0xD3). Then, another data transfer is initiated to read-out the data from the same address.

Figure 20. Write and Read Operation

Conclusion

With an 8-Kbit UFM block, the MAX II device can replace small density serial EEPROMs. The SMBus to UFM block design example allows you to read, write, or erase the MAX II UFM block contents through the SMBus interface. With this design example, the MAX II device is the slave device and receives instructions from any master device in the bus for data transfer.