Using the UFM in MAX II Devices

December 2007, version 1.0 Application Note 489

Introduction

This application note discusses storing non-volatile information. Most CPLDs use serial EEPROMs to achieve non-volatile information storage, but MAX® II CPLDs are the only CPLDs that offer User Flash Memory (UFM), which allows a user to store non-volatile information of up to 8 Kbits. In addition to being programmable, the UFM also supports serial and parallel interfaces and other proprietary protocols. This application note explains how you can efficiently store and retrieve information in the UFM and uses the I²C protocol for interfacing and accessing the UFM of a MAX II CPLD.

User Flash Memory

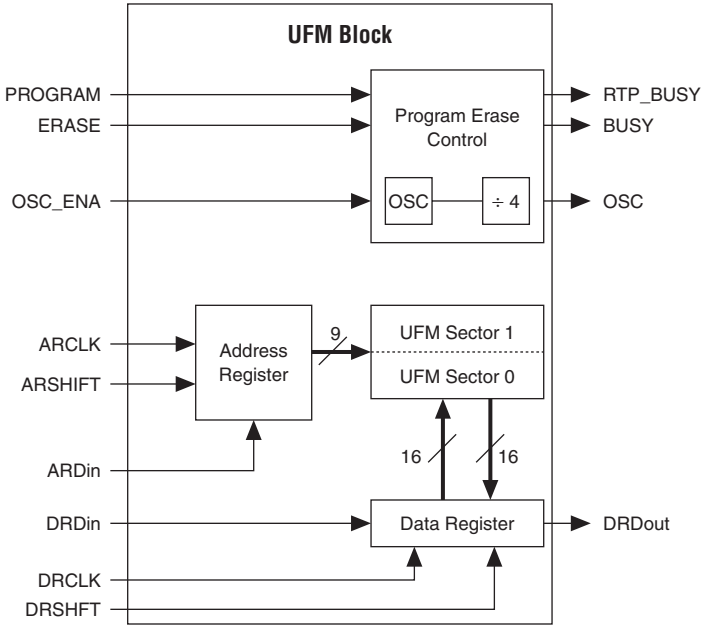
The following list contains example applications and features of the UFM:

- You can use the UFM to store vital non-volatile information such as ASSP or processor configuration bits, electronic ID information for a board during manufacturing, or to store information that is to be displayed on an LCD as soon as the processor is powered on.
- Features include parallel and serial interfaces such as the Serial Communication Interface (SCI), Serial Peripheral Interface (SPI), Inter-Integrated Circuit (I²C), Microwire, and other proprietary protocols. MAX II devices offer more interface flexibility than an off-the-shelf EEPROM device.
- The UFM has an internal oscillator that can be used to satisfy the clocking needs of any design; thus, eliminating the extra space and cost required for an external clocking circuit.

You can use MAX II CPLDs to incorporate logic and memory devices on a design board, thereby reducing chip-to-chip delays, minimizing board space, and reducing total system cost.

The UFM is divided into two sectors of 4 Kbits each. The address register indicates the address of the UFM memory location where data has to be written to or retrieved from. The data register holds the data that will be written or retrieved from the UFM. The Program Erase Control block is used to either program or erase the UFM and also to enable the internal oscillator. Figure 1 shows a block diagram of the UFM.

Figure 1. Block Diagram of the User Flash Memory (UFM)



Using the UFM in MAX II Devices

The UFM is divided into 2 sectors: Sector 0 and Sector 1. Each sector is 4096 bits and has an address range from 000h to 0FFh and 100h to 1FFh, respectively. The address locations are accessed by 9 bits, and each address location is capable of storing 8-bit data.

You can load and modify data into the UFM by Read/Stream Read, Program, and Erase operations. The Read/Stream Read operation is used to read the contents of the location pointed to by the address register. Consecutive memory locations can be read by incrementing the address register (Stream Read). The Program operation is used to load data into the UFM, while the Erase operation is performed when the content of the UFM has to be modified. However, erasure of a single address location is not possible. This operation either erases the entire UFM (A2A1A0 = 111) or the sector of the UFM indicated by the MSB of the address transmitted.

The UFM also contains an internal oscillator that can be enabled. This signal can be routed through the logic array block and can also be fed back to ARCLK and DRCLK.

This design uses I^2C serial interface to store and retrieve data from the UFM. The following summarizes the I^2C protocol:

- Uses two bidirectional bus lines: the SDA line is used for addresses and data transfer and the SCL line is used for the I²C clock. Both lines remain high (pulled up) when free.
- Communication begins with a start condition indicated by a high-to-low transition on the SDA line when SCL is high.
- The address of the slave is then sent on the SDA. Data transfer begins once the address is acknowledged by the slave. Data to be transmitted has to be held stable on the SDA line while the clock is high.
- Communication is terminated with a stop condition indicated by a low-to-high transition on the SDA line when the SCL is high.

Table 1 describes the signals encountered in the UFM block.

Table 1. Brief Description of the Signals Encountered in the UFM Block *Note (1)*

Signal	Description
DRDin	Shifts data into the data register on each DRCLK pulse.
DRCLK	Controls the shifting of data from DRDin to DRDout and the parallel loading of data from the UFM to the Data Register.
DRSHFT	High: Shifts in LSB from DRDin, shifts out MSB to DRDout. Low: Latches data from the UFM into the data register.
ARDin	Serial input to store the memory location address.
ARCLK	Controls the shifting of memory location address and the incrementing of the address present in the address register.
ARSHFT	High: Shifts address serially from ARDin to address register. Low: Increments the address in the address register.
PROGRAM	On the rising edge of this signal, data from the data register is written into the memory location pointed to by the address register.
ERASE	On the rising edge of this signal, the memory sector indicated by the MSB of the address register is erased.
OSC_ENA	Signal used to enable the internal oscillator of the UFM.
DRDout	Output of the data register. MSB is obtained first.
BUSY	Indicates that the memory is busy in a program or erase instruction.
RTP_BUSY	Needed if real-time ISP feature is enabled.

Note to Table 1:

(1) For more information, refer to the "UFM Interface Signals" table in the *Using the User Flash Memory in MAX II Devices* chapter of the *Altera MAX II Device Handbook*.

The Flash Memory megafunction is used to instantiate the UFM. It allows the user to choose their interface, the size of the memory required, options to write protect the data in the UFM, the option to enable the internal

oscillator and route it to a port on the CPLD and also set the first four bits of the UFM slave address (bits A6-A3). It also provides the option to wire the remaining three slave address bits (A2 to A0) on the board.

Instantiate the UFM Megafunction in the Quartus II Software

Perform the following steps to instantiate the UFM megafunction:

1. Open the project to instantiate the internal oscillator with Quartus II software.

2. On the Tools menu, click **MegaWizard Plug-In Manager**. The **MegaWizard Plug-In Manager [page 1]** dialog box appears.

3. Select **Create a new custom megafunction variation** and click **Next**. The **MegaWizard Plug-In Manager [page 2a]** dialog box appears. Table 2 shows you the options and settings in the **MegaWizard Plug-In Manager [page 2a]** dialog box

Table 2. MegaWizard Plug-In Manager [page 2a] Options

Options	Settings
Which device family will you be using?	Select **MAX II**.
Which megafunction would you like to customize?	Click the "+" icon to expand **Memory Compiler** and select **Flash Memory**.
Which type of output file do you want to create?	Select from **AHDL**, **VHDL**, or **Verilog HDL**.
What name do you want for the output file?	Type the name of your file.

4. Click **Next**. The **MegaWizard Plug-In Manager ALTUFM [page 3 of 5]** dialog box appears. Table 3 shows the options and settings in the ALTUFM wizard page 3.

Table 3. MegaWizard Plug-In Manager [page 3 of 5] Options

Options	Settings	Additional Options
What is the interface protocol?	None	Use arclkena port (clock enable for arclk)
		Use drclkena port (clock enable for drclk)
	Parallel	What is the width of the address bus?
		What is the width of the data bus?
		Use the 'osc' (oscillator) output port
	Serial Peripheral Interface (SPI)	Base mode (uses 8 bit address and data)
		Extended mode (uses 16 bit address and data)
		Use the 'osc' (oscillator) output port
	I²C	What is the MSB of the device address (in binary)?
		What is the size of the memory?
		Use the 'osc' (oscillator) output port
What is the access mode for the user flash memory?	Read/Write or Read Only	—

5. Click **Next**.

- If you select **I2C**, the ALTUFM wizard page 4 appears. Table 4 shows the options and settings in the ALTUFM wizard page 4.

  Click **Next**. The ALTUFM wizard page 3 appears.

Table 4. MegaWizard Plug-In Manager [page 4 of 7] Options (Part 1 of 2)

Options	Settings	Additional Options
What is the write configuration for the I2C protocol?	Single byte write	—
	Page write	Select from **8 bytes**, **16 bytes**, **32 bytes**
Write protect	Use the 'wp' (write protect) input	Write protect applies to the full memory
		Write protect applies only to the upper half of the memory

Table 4. MegaWizard Plug-In Manager [page 4 of 7] Options (Part 2 of 2)

Options	Settings	Additional Options
What erase method should be used in I2C protocol?	Device Slave Address Full Erase (3 LSBs are 111)	—
	Sector Erase Triggered by Byte Address *(1)*	Sector 0: Trigger erase when writing to binary address (MSB is always '0')
		Sector 1: Trigger erase when writing to binary address (MSB is always '1')
	Sector Erase Triggered by 'a2' bit	—
	No Erase	—

Note to Table 4:
(1) This option is only available when you select **Single byte write** under **What is the write configuration for the I2C protocol?**

- If you select **None**, **Parallel**, or **Serial Peripheral Interface (SPI)**, the ALTUFM wizard page 4 appears. Table 5 shows the options and settings in the ALTUFM wizard page 3.

Table 5. MegaWizard Plug-In Manager [page 4 of 6] Options

Options	Settings	
Do you want to specify the initial content of the memory?	No, leave it blank	
	Yes, use this file for the memory content	Type or browse the File name:
What is the oscillator frequency for the User Flash Memory? (for simulation only)	Select **3.33 MHz** or **5.56 MHz**	
What is the erase time for the User Flash Memory? (for simulation only)	Type the value	
What is the program time for the User Flash Memory? (for simulation only)	Type the value	

6. Click **Next**. The ALTUFM wizard page 5 appears.

7. On the ALTUFM wizard page 5. If you select this option, the file for that netlist is also available. A gray checkmark indicates a file that is automatically generated, and a red checkmark indicates an optional file. Click **Next**.

8. The ALTUFM wizard page 6 displays a list of the types of files to be generated. The automatically generated Variation file contains wrapper code in the language you specified on page 2a. On page 7, you can specify additional types of files to be generated. Choose from the AHDL Include file (<*function name*>**.inc**), VHDL component declaration file, <*function name*>**.cmp**), Quartus II symbol file (<*function name*>**.bsf**), Instantiation template file (<*function name*>**.v**), and Verilog HDL black box file (<*function name*>**_bb.v**). Click **Finish**.

Implementation

You can implement this design example with an EPM240, or any other MAX II CPLD. The UFM in this design example is assigned to have an I²C interface. Access to the MAX II UFM is demonstrated with an I²C bus environment. Implementation involves using the design example source code attached with this application note and allocating the UFM's I²C interface lines to the MAX II's GPIOs. The UFM is then accessed to read or write with the help of an I²C simulator that is created using a PC parallel port and interfacing hardware to create an I²C compliant 2-wire bus. Details about setting up an I²C environment are described in the Dallas Semiconductor's Maxim application note AN3230, which can be found at:

www.maxim-ic.com/appnotes.cfm/an_pk/3230

Similar free software can be downloaded at:

http://files.dalsemi.com/system_extension/AppNotes/AN3315/ParDS2W.exe

Using this utility program with the parallel port and its interfacing hardware interact with the MAX II CPLD and provide the `SDA` and `SCL` connections (required on an I²C 2-wire system to access the UFM). When implemented, this design allows the I²C Master to access the UFM to read or write data. The I²C Master in this demonstration is the user interface on the PC running the parallel port I²C software. The UFM is the I²C slave.

Table 6 shows the implementation of this design example on the MDN-B2 demo board.

Table 6. EPM240G Pin Assignment

Signal	Pin
SCLK	Pin 39
SDA	Pin 40
a1	Pin 37
a2	Pin 38

Unused pins are assigned as **input tri-stated** in the **Device and Pin Options** dialog box in the Quartus II software. The Assignment Editor in the Quartus II software is used to enable **Auto Open-Drain Pins** on SCLK and SDA pins. These settings are followed by a compilation cycle.

Refer to the following demo Notes (to demonstrate this design on the MDN-B2 demo board):

- Turn on the power to the demo board (using slide switch SW1).
- Download the design on to the MAX II CPLD through the JTAG header JP5 on the demo board and a conventional programming cable (ByteBlaster II or USB-Blaster).
- Keep SW4 on the demo board pressed before and during the start of the programming process. Once complete, turn off the power and remove the JTAG connector.
- Set up a parallel port driven I^2C environment on your PC:
  - Download a software utility such as the Maxim parallel port utility to communicate with the slave in I^2C defined protocol. Install the parallel port software. This example uses the program **ParDS2W.exe** at: http://files.dalsemi.com/system_extension/AppNotes/AN3315/ParDS2W.exe
  - You must install a parallel port driver to enable access to the parallel port in Windows XP or Windows 2000 for this parallel port utility. Direct-IO (www.direct-io.com) has a typical driver that you can use and which you can download at: www.direct-io.com/Direct-IO/directio.exe
  - After installation, you must configure the Direct-IO program. Open the Windows control panel and click on the Direct IO icon. Enter the Begin and End addresses of your parallel port (most often this is 378h through 37Fh, but confirm your PC's parallel port address by looking at settings in:

Control Panel/System/Hardware/Device Manager/Ports/
EC P Printer port (LPT)/Resources

- If your parallel port is configured to any type other than ECP, change it to ECP by changing the BIOS settings during start-up of your PC.
- Select the **Security Tab** of the Direct IO control panel and browse to the directory path of the **ParDS2W.exe** program. Click **Open**, and then click **Add**. You will see the path of this utility in the **Allowed Processes** field. Click **OK** to close the control panel window.
- Attach the parallel port I^2C dongle that is supplied along with the MDN-B2 demo board. Use an extension chord if necessary to extend the parallel port connection closer to your demo board.
- Attach the 4-pin socket on the pigtail of I^2C parallel port dongle to the I^2C header (JP3) of the demo board so that the red mark on the socket meets `pin#1` on the JP3 header.
- Set switches 1 and 2 of SW3 (8-way DIP switch on the MDN-B2 demo board) to their ON position.
- Open the ParDS2W program, select the appropriate parallel port address of your PC (as seen while configuring Direct IO), and set the **2-Wire Device Address** to `B0H`.
- Finally, you can test the I^2C setup by using the **Test Circuit** tab to see if you have a Test PASS message in the Status window. If you do, the I^2C environment is now set up.

- Using the **One byte Write/Read** section in the **2-Wire Utility** section of the parallel port utility program, perform a single byte **WRITE** operation to any specific memory address location by specifying an address and data (each a byte long, 2 hex digits)
- Similarly, perform a **READ** operation at the same address location and note the contents of that address location. It should be the value that was just written. Any other address location should have a content of FFh, unless it is written in.
- If you want to perform a **WRITE** into an address location more than once, it should be preceded by an erase operation.
- You can do a full content erase by selecting BEh as the slave address and doing a **WRITE** operation of FFh. This restores the content of the UFM to FF.
- You can set the 5th and 6th bit of the I^2C slave address (a2, a1) using switches 1 and 2 on SW3 of the demo board (8-way DIP switch). These are both set to 0, and the first 4 MSB of the I^2C slave address is set (1011 or Bh in this case) during instantiation of the Flash Memory megafunction.

Source Code

The design example for this application note is implemented in Verilog HDL and successful operation has been demonstrated using the MDN-B2 demo board. The source code, testbench, and complete Quartus II project are available at:

www.altera.com/literature/an/an489_design_example.zip

Conclusion

MAX II CPLDs are a great choice to implement a wide variety of logic solutions that require non-volatile memory support because of their unique on-chip user flash memory. Additionally, MAX II devices feature low power, easy and quick power-on, multi-volt capability, and a built-in internal oscillator, which makes them very versatile programmable logic devices.

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 Device Application Notes:
  AN 428: MAX II CPLD Design Guidelines
  AN 422: Power Management in Portable Systems Using MAX II CPLDs

Revision History

Table 7 shows the revision history for this application note.

Table 7. Revision History

Date and Version	Changes Made	Comments
December 2007, v1.0	Initial release.	—

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