Altera User Flash Memory (ALTUFM) IP Core User Guide

2014.08.18

UG-040105

Subscribe Send Feedback

The Altera User Flash Memory IP cores provide interface logic for a subset of parallel interface, serial peripheral interface (SPI), inter-integrated circuit (I2C,) and the built-in dedicated user flash memory (UFM) serial interface. This document describes the following integrated User Flash Memory IP cores:

- Altera User Flash Memory for I2C Interface Protocol (ALTUFM_I2C)
- Altera User Flash Memory for Parallel Interface Protocol (ALTUFM_PARALLEL)
- Altera User Flash Memory for SPI Interface Protocol (ALTUFM_SPI)
- Altera Serial Interface (ALTUFM_NONE)

Note: This IP core is not supported for Arria 10 designs.

Related Information

- **Introduction to Altera IP Cores**
- **Altera IP Release Notes**

Features

The ALTUFM IP core provides the following features:

- Up to 8K bits for non-volatile storage
- Two sectors for partitioned sector erase
- Interface protocols: parallel, SPI, I2C, and none (use dedicated UFM)
- Memory initialization using Memory Initialization File or HEX File
- Built-in oscillator that provides oscillator frequency for the user flash memory
- Program, erase, and busy signals
- Easy instantiation from the IP Catalog GUI

Device Support

The ALTUFM IP core supports the MAX® II and Max V devices.

101 Innovation Drive, San Jose, CA 95134

Resource Utilization and Performance

The ALTUFM IP core is only available for MAX II and MAX V devices. Resource usage is reported with different interface options. Configuration mode settings described in the following tables are available in the parameter editor.

Table 1: ALTUFM_SPI Resource Usage

Access Mode *(1)*	Optimization *(2)*	Configuration Mode *(3)*	Logic Element	UFM Blocks
Read/write	Balanced	Base mode	147	1
Read only	Balanced	Base mode	72	1
Read/write	Balanced	Extended mode	134	1
Read only	Balanced	Extended mode	40	1

Notes to Table 1–1:

1. Choose the access mode option to configure the UFM to the required mode of operation. Choose a design implementation that balances high performance with minimal logic usage. The balanced optimization logic option is set in Analysis and Synthesis settings (Assignments menu).
2. Choose the configure mode to set the mode of access with SPI interface to the UFM block. Set this option to Base Mode or Extended Mode depending on the widths of the address and data buses.

Table 2: ALTUFM_PARALLEL Resource Usage

Access Mode *(1)*	Optimization *(2)*	Address Width	Data Width	Logic Element	UFM Blocks
Read/write	Balanced	9	16	88	1
Read only	Balanced	9	16	69	1

Notes to Table 1–2:

1. Choose the access mode option to configure the UFM to the required mode of operation.
2. Choose a design implementation that balances high performance with minimal logic usage. The balanced optimization logic option is set in Analysis and Synthesis settings (Assignments menu).

Table 3: ALTUFM_NONE Resource Usage

Logic Elements	Optimization *(1)*	UFM Blocks
3	Balanced	1

Note to Table 1–3:

1. Choose a design implementation that balances high performance with minimal logic usage. The balanced optimization logic option is set in Analysis and Synthesis settings (Assignments menu).

. Choose the Access Mode option to configure the UFM to the required mode of operation.

Note: Memory size is available only for the I2C interface where the size of the memory to be protected is specified. This option is valid only when the access mode is set to **Read and Write**. Set this option in the parameter editor.

(Optimization effort for the I2C interface is set to balanced for all listed combinations).

Table 4: ALTUFM_I2C Resource Usage

Access Mode	Memory Size	Logic Element	UFM Blocks
Read only	1K	117	1
Read only	2K	117	1

Table 5: ALTUFM_I2C Resource Usage (Read/write access with Memory address erase)

Access Mode	Memory Size	Write Protection	UFM Blocks		
No Write Protection	Full Memory Write Protected	Half Memory Write Protected			
Read/write	1K	154 LE	154 LE	156 LE	1
Read/write	2K	155 LE	155 LE	157 LE	1
Read/write	4K	119 LE	119 LE	120 LE	1

Table 6: ALTUFM_I2C Resource Usage (Read/write access with Device address erase)

Access Mode	Memory Size	Write Protection	UFM Blocks		
No Write Protection	Full Memory Write Protected	Half Memory Write Protected			
Read/write	1K	141 LE	141 LE	142 LE	1
Read/write	2K	164 LE	164 LE	166 LE	1

Table 7: ALTUFM_I2C Resource Usage (Read/write access with A2 erase)

Access Mode	Memory Size	Write Protection	UFM Blocks		
No Write Protection	Full Memory Write Protected	Half Memory Write Protected			
Read/write	1K	151 LE	151 LE	153 LE	1
Read/write	2K	152 LE	152 LE	153 LE	1

Table 8: ALTUFM_I2C Resource Usage (Read/write access with No erase)

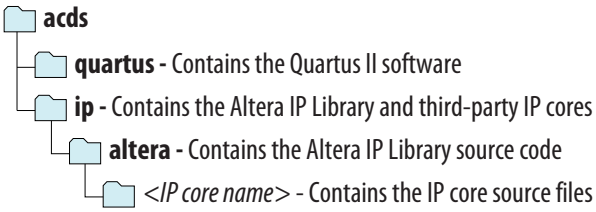
Access Mode	Memory Size	Write Protection	UFM Blocks		
No Write Protection	Full Memory Write Protected	Half Memory Write Protected			
Read/write	1K	144 LE	144 LE	146 LE	1
Read/write	2K	145 LE	145 LE	147 LE	1

This user guide assumes that you are familiar with IP cores and how to create them. If you are unfamiliar with Altera IP cores, refer to the *Introduction to Altera IP cores*.

Installing and Licensing IP Cores

The Altera IP Library provides many useful IP core functions for production use without purchasing an additional license. You can evaluate any Altera IP core in simulation and compilation in the Quartus II software using the OpenCore evaluation feature. Some Altera IP cores, such as MegaCore® functions, require that you purchase a separate license for production use. You can use the OpenCore Plus feature to evaluate IP that requires purchase of an additional license until you are satisfied with the functionality and performance. After you purchase a license, visit the Self Service Licensing Center to obtain a license number for any Altera product.

Figure 1: IP Core Installation Path



Note: The default IP installation directory on Windows is **<*drive*>:\altera**<*version number*>; on Linux it is <*home directory*>**/altera/** <*version number*>.

Related Information

- Altera Licensing Site
- Altera Software Installation and Licensing Manual

Customizing and Generating IP Cores

You can customize IP cores to support a wide variety of applications. The Quartus II IP Catalog and parameter editor allow you to quickly select and configure IP core ports, features, and output files.

IP Catalog and Parameter Editor (replaces MegaWizard Plug-In Manager)

The Quartus II IP Catalog (**Tools** > **IP Catalog**) and parameter editor help you easily customize and integrate IP cores into your project. You can use the IP Catalog and parameter editor to select, customize, and generate files representing your custom IP variation.

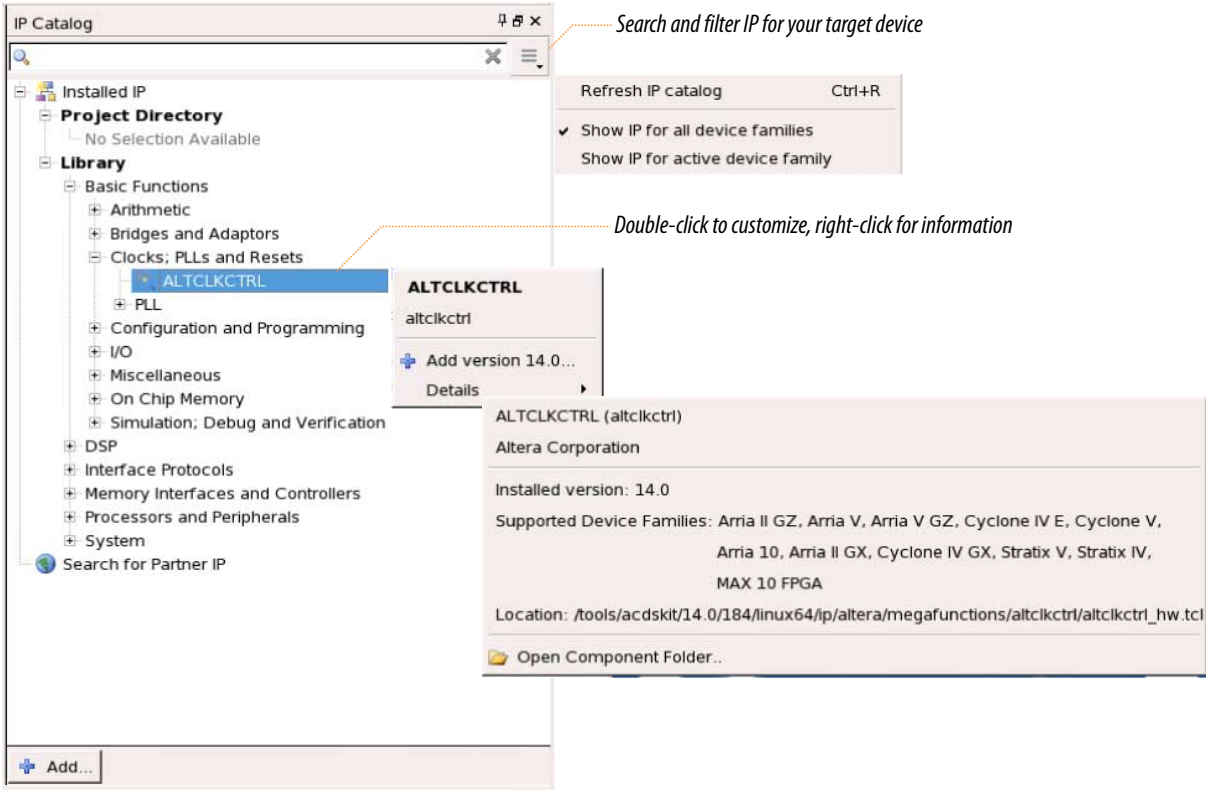
Note: The IP Catalog (**Tools** > **IP Catalog**) and parameter editor replace the MegaWizard™ Plug-In Manager for IP selection and parameterization, beginning in Quartus II software version 14.0. Use the IP Catalog and parameter editor to locate and paramaterize Altera IP cores.

The IP Catalog lists IP cores available for your design. Double-click any IP core to launch the parameter editor and generate files representing your IP variation. The parameter editor prompts you to specify an IP variation name, optional ports, and output file generation options. The parameter editor generates a top-level Qsys system file (**.qsys**) or Quartus II IP file (**.qip**) representing the IP core in your project. You can also parameterize an IP variation without an open project.

Use the following features to help you quickly locate and select an IP core:

- Filter IP Catalog to **Show IP for active device family** or **Show IP for all device families**.
- Search to locate any full or partial IP core name in IP Catalog. Click **Search for Partner IP**, to access partner IP information on the Altera website.
- Right-click an IP core name in IP Catalog to display details about supported devices, open the IP core's installation folder, andor view links to documentation.

Figure 2: Quartus II IP Catalog



Note: The IP Catalog is also available in Qsys (**View** > **IP Catalog**). The Qsys IP Catalog includes exclusive system interconnect, video and image processing, and other system-level IP that are not available in

the Quartus II IP Catalog. For more information about using the Qsys IP Catalog, refer to *Creating a System with Qsys* in the *Quartus II Handbook*.

Related Information

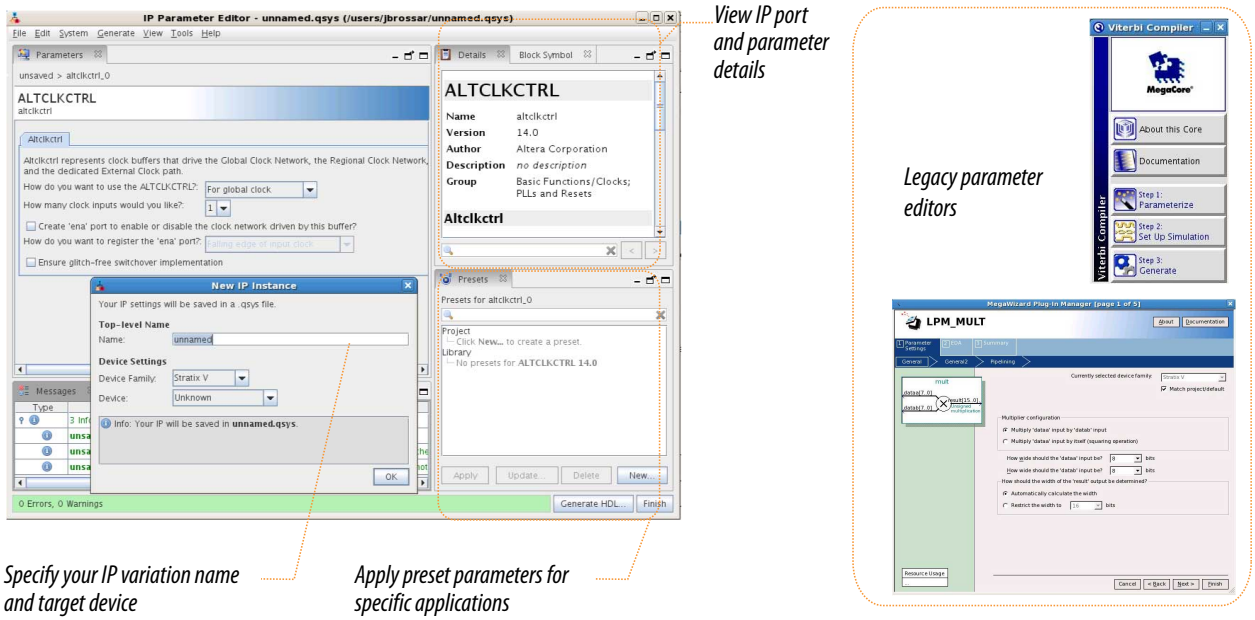
- **Creating a System with Qsys**

Using the Parameter Editor

The parameter editor helps you to configure IP core ports, parameters, and output file generation options.

- Use preset settings in the parameter editor (where provided) to instantly apply preset parameter values for specific applications.
- View port and parameter descriptions, and links to documentation.
- Generate testbench systems or example designs (where provided).

Figure 3: IP Parameter Editors

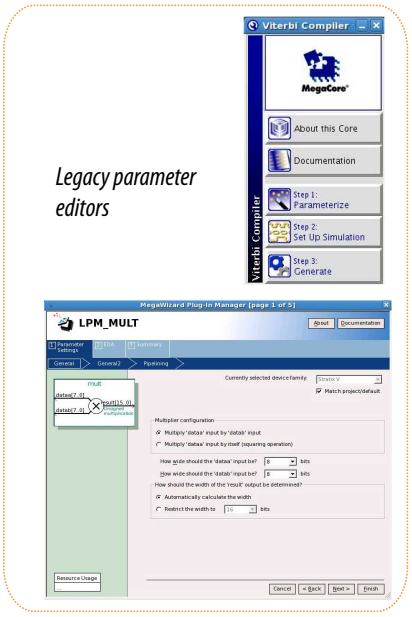


Specifying IP Core Parameters and Options (Legacy Parameter Editors)

The Quartus II software version 14.0 and previous uses a legacy version of the parameter editor for IP core configuration and generation. Use the following steps to configure and generate an IP variation using a legacy parameter editor.

Note: The legacy parameter editor generates a different output file structure than the latest parameter editor. Refer to *Specifying IP Core Parameters and Options* for configuration of IP cores in the Quartus II software version 14.0a10 and later.

Figure 4: Legacy Parameter Editors



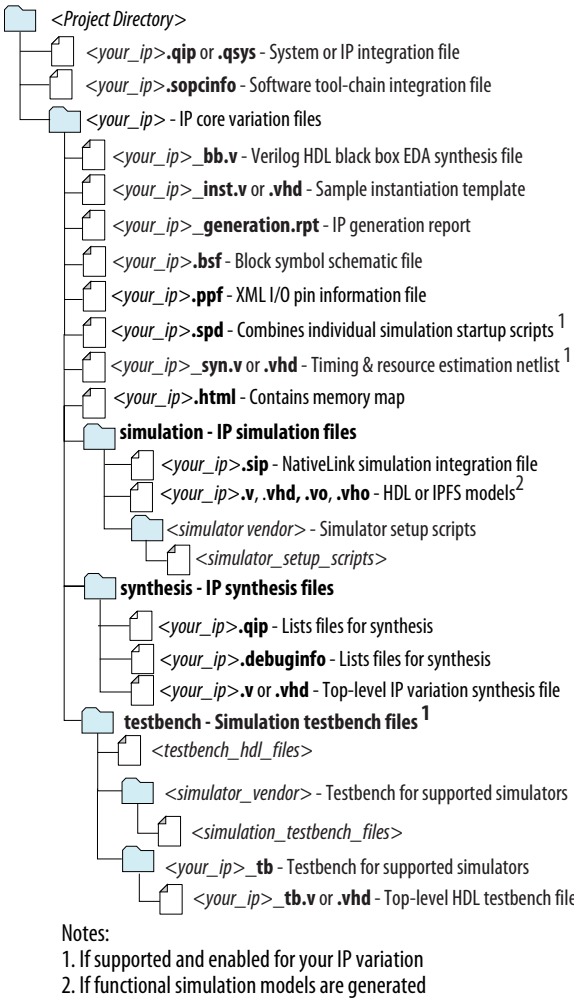
1. In the IP Catalog (**Tools** > **IP Catalog**), locate and double-click the name of the IP core to customize. The parameter editor appears.
2. Specify a top-level name and output HDL file type for your IP variation. This name identifies the IP core variation files in your project. Click **OK**.
3. Specify the parameters and options for your IP variation in the parameter editor. Refer to your IP core user guide for information about specific IP core parameters.
4. Click **Finish** or **Generate** (depending on the parameter editor version). The parameter editor generates the files for your IP variation according to your specifications. Click **Exit** if prompted when generation is complete. The parameter editor adds the top-level **.qip** file to the current project automatically.

   Note: To manually add an IP variation generated with legacy parameter editor to a project, click **Project** > **Add/Remove Files in Project** and add the IP variation **.qip** file.

Files Generated for Altera IP Cores (version 14.0 and previous)

The Quartus II software version 14.0 and previous generates the following output for your IP core.

Figure 5: IP Core Generated Files



Parameter Settings

Table 9: ALTUFM_PARALLEL Parameters

Configuration Setting	Description
Which action do you want to perform?	You can select from the following options: **Create a new custom IP core variation**, **Edit an existing custom IP core variation**, or **Copy an existing custom IP core variation**.
Select a IP core from the list below	Select **ALTUFM_PARALLEL** from the **Memory Compiler** category.
Which device family will you be using?	Specify the device family that you want to use.

Configuration Setting	Description
Which type of output file do you want to create?	You can choose AHDL(**.tdf**), VHDL(**.vhd**), or Verilog HDL (**.v**) as the output file type.
What name do you want for the output file?	Specify the name of the output file.
Return to this page for another create operation	Turn on this option if you want to return to this page to create multiple IP cores.
Currently selected device family	Specifies the device family you chose on page 2a.
Match project/default	Turn on this option to ensure that the device selected matches the device family that is chosen in the previous page.
Read and write	Turn on this option if you want to enable the read and write access mode.
Read only	Turn on this option if you want to enable the read only mode.
'addr' width	Select the width for the address bus. The maximum size of the address bus can be 9.
'datain' width	Select the width for the data bus. The maximum size of the data bus can be 16.
Use 'osc' output port	Turn on this option to route the oscillator frequency to an external oscillator port.
Use 'oscena' input port	Turn on this option to enable the oscillator enable port.
Memory content initialization	• Select Initialize blank memory if you do not want to specify any initialization file. Select Initialize from hex or mif file to specify the initialization file. Type the file name or browse for the required file. • In the Quartus II software, the memory content values from your .hex or .mif are hard-coded into your ALTUFM IP core variation file when you generate the IP variation. If you change the contents of your .hex or .mif after generating the IP variation, these updates will not be reflected in simulation. This may cause a mismatch between simulation and device behavior because compilation and program file generation in the Quartus II software use the current .hex or .mif contents instead of the hard-coded values.To avoid this mismatch, regnenerate the ALTUFM IP core whenever you update your .hex or .mif.
Oscillator frequency	Specify the oscillator frequency for the user flash memory. This parameter is used for simulation purposes only. The values are 5.56MHz and 3.33MHz. If omitted, the default is 5.56MHz.
Erase time	Specify the erase time in unit of ns. Simulation erase time for the UFM block can only be from **1600** ns to **999999** ns.
Program time	Specify the program time in unit ns. Simulation erase time for the UFM block can only be from **1600** ns to **100000** ns.

Configuration Setting	Description
Generate netlist	Turn on this option if you want to generate a netlist for your third-party EDA synthesis tool to estimate the timing and resource usage of the IP core. If you turn on this option, a netlist file (**_syn.v**) is generated. This file is a representation of the customized logic used in the Quartus II software and provides the connectivity of the architectural elements in the IP core but may not represent true functionality.
Summary Page	Specify the types of files to be generated. The Variation file (<*function name*> **.v**) contains wrapper code in the language you specified on page 2a and is automatically generated. Choose from the following types of files: • 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*>**_inst.v**) • Verilog HDL black box file (<*function name*>**_bb.v)** For more information about the wizard-generated files, refer to the *Introduction to Altera IP Cores*.

Table 10: ALTUFM_SPI Parameter Settings

Configuration Setting	Description
Which action do you want to perform?	You can select from the following options: **Create a new custom IP core variation**, **Edit an existing custom IP core variation**, or **Copy an existing custom IP core variation**.
Select a IP core from the list below	Select **ALTUFM_SPI** from the **Memory Compiler** category.
Which device family will you be using?	Specify the device family that you want to use.
Which type of output file do you want to create?	You can choose AHDL(**.tdf**), VHDL(**.vhd**), or Verilog HDL (**.v**) as the output file type.
What name do you want for the output file?	Specify the name of the output file.
Return to this page for another create operation	Turn on this option if you want to return to this page to create multiple IP cores.
Currently selected device family	Specifies the device family you chose on page 2a.
Match project/default	Turn on this option to ensure that the device selected matches the device family that is chosen in the previous page.
Read and write	Turn on this option if you want to enable the read and write access mode.
Read only	Turn on this option if you want to enable the read only mode.

Configuration Setting	Description
Configuration mode	Select Base mode to use 8-bit address and data. Select Extended mode to use16-bit address and data.
Use 'osc' output port	Turn on this option to route the oscillator frequency to an external oscillator port.
Use 'oscena' input port	Turn on this option to enable the oscillator enable port.
Memory content initialization	• Select Initialize blank memory if you do not want to specify any initialization file. Select Initialize from hex or mif file to specify the initialization file. Type the file name or browse for the required file. • In the Quartus II software, the memory content values from your .hex or .mif are hard-coded into your ALTUFM IP core variation file when you generate the IP variation. If you change the contents of your .hex or .mif after generating the IP variation, these updates will not be reflected in simulation. This may cause a mismatch between simulation and device behavior because compilation and program file generation in the Quartus II software use the current .hex or .mif contents instead of the hard-coded values.To avoid this mismatch, regnenerate the ALTUFM IP core whenever you update your .hex or .mif.
Oscillator frequency	Specify the oscillator frequency for the user flash memory. This parameter is used for simulation purposes only. The values are 5.56MHz and 3.33MHz. If omitted, the default is 5.56MHz.
Erase time	Specify the erase time.
Program time	Specify the program time.
Generate netlist	Turn on this option if you want to generate a netlist for your third-party EDA synthesis tool to estimate the timing and resource usage of the IP core. If you turn on this option, a netlist file (**_syn.v**) is generated. This file is a representation of the customized logic used in the Quartus II software and provides the connectivity of the architectural elements in the IP core but may not represent true functionality.
Summary Page	Specify the types of files to be generated. The Variation file (<*function name*> **.v**) contains wrapper code in the language you specified on page 2a and is automatically generated. Choose from the following types of files: • 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*>**_inst.v**) • Verilog HDL black box file (<*function name*>**_bb.v)** For more information about the wizard-generated files, refer to the *Introduction to Altera IP Cores*.

Table 11: ALTUFM_I2C Parameter Settings

Configuration Setting	Description
Which action do you want to perform?	You can select from the following options: **Create a new custom IP core variation**, **Edit an existing custom IP core variation**, or **Copy an existing custom IP core variation**.
Select a IP core from the list below	Select **ALTUFM_I2C** from the **Memory Compiler** category.
Which device family will you be using?	Specify the device family that you want to use.
Which type of output file do you want to create?	You can choose AHDL(**.tdf**), VHDL(**.vhd**), or Verilog HDL (**.v**) as the output file type.
What name do you want for the output file?	Specify the name of the output file.
Return to this page for another create operation	Turn on this option if you want to return to this page to create multiple IP cores.
Currently selected device family	Specifies the device family you chose on page 2a.
Match project/default	Turn on this option to ensure that the device selected matches the device family that is chosen in the previous page.
Read and write	Turn on this option if you want to enable the read and write access mode.
Read only	Turn on this option if you want to enable the read only mode.
Device address MSB (binary)	Select 4-bit address that specifies the 4 MSBs of the device address.
Memory size	Select the memory size. Values are 1K, 2K, 4K, and 8K. If omitted, the default is 4K.
Use 'global_reset' input port	Turn on this option to enable global reset input port.
Use 'osc' output port	Turn on this option to route the oscillator frequency to an external oscillator port.
Use 'oscena' input port	Turn on this option to enable the oscillator enable port.
Write Configuration	Specify the write configuration for the I2C protocol. Select Single byte write or Page write. If you select Page write, you can only use 16 bytes.
Use 'wp' write protect input port	Select this option to use the write protect input port, and define how the write protect is applied; write protect the full memory or only the upper half of the memory.
Erase method	Select the erase method. Options are full erase, sector erase, and no erase. If you select the **Sector Erase Triggered by Byte Address** option, enter the binary address for sectors 0 and 1. Note that the **Sector Erase Triggered by Byte Address** option is only available if **Single byte write** is selected as the write configuration.

Configuration Setting	Description
Memory content initialization	• Select Initialize blank memory if you do not want to specify any initialization file. Select Initialize from hex or mif file to specify the initialization file. Type the file name or browse for the required file. • IIn the Quartus II software, the memory content values from your .hex or .mif are hard-coded into your ALTUFM IP core variation file when you generate the IP variation. If you change the contents of your .hex or .mif after generating the IP variation, these updates will not be reflected in simulation. This may cause a mismatch between simulation and device behavior because compilation and program file generation in the Quartus II software use the current .hex or .mif contents instead of the hard-coded values.To avoid this mismatch, regnenerate the ALTUFM IP core whenever you update your .hex or .mif.
Oscillator frequency	Specify the oscillator frequency for the user flash memory. This parameter is used for simulation purposes only. The values are 5.56MHz and 3.33MHz. If omitted, the default is 5.56MHz.
Erase time	Specify the erase time.
Program time	Specify the program time.
Generate netlist	Turn on this option if you want to generate a netlist for your third-party EDA synthesis tool to estimate the timing and resource usage of the IP core. If you turn on this option, a netlist file (**_syn.v**) is generated. This file is a representation of the customized logic used in the Quartus II software and provides the connectivity of the architectural elements in the IP core but may not represent true functionality.
Summary Page	Specify the types of files to be generated. The Variation file (*<function name>* **.v**) contains wrapper code in the language you specified on page 2a and is automatically generated. Choose from the following types of files: • 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>***_inst.v**) • Verilog HDL black box file (*<function name>***_bb.v)** For more information about the wizard-generated files, refer to the *Introduction to Altera IP Cores*.

Table 12: ALTUFM_NONE Parameter Settings

Configuration Setting	Description
Which action do you want to perform?	You can select from the following options: **Create a new custom IP core variation**, **Edit an existing custom IP core variation**, or **Copy an existing custom IP core variation**.
Select a IP core from the list below	Select **ALTUFM_NONE** from the **Memory Compiler** category.

Configuration Setting	Description
Which device family will you be using?	Specify the device family that you want to use.
Which type of output file do you want to create?	You can choose AHDL(**.tdf**), VHDL(**.vhd**), or Verilog HDL (**.v**) as the output file type.
What name do you want for the output file?	Specify the name of the output file.
Return to this page for another create operation	Turn on this option if you want to return to this page to create multiple IP cores.
Currently selected device family	Specifies the device family you chose on page 2a.
Match project/default	Turn on this option to ensure that the device selected matches the device family that is chosen in the previous page.
Use 'arclkena' output port	Turn on this option to enable the arclkena port.
Use 'drclkena' input port	Turn on this option to enable drclkena port.
Memory content initialization	• Select Initialize blank memory if you do not want to specify any initialization file. Select Initialize from hex or mif file to specify the initialization file. Type the file name or browse for the required file. • In the Quartus II software, the memory content values from your .hex or .mif are hard-coded into your ALTUFM IP core variation file when you generate the IP variation. If you change the contents of your .hex or .mif after generating the IP variation, these updates will not be reflected in simulation. This may cause a mismatch between simulation and device behavior because compilation and program file generation in the Quartus II software use the current .hex or .mif contents instead of the hard-coded values.To avoid this mismatch, regnenerate the ALTUFM IP core whenever you update your .hex or .mif.
Oscillator frequency	Specify the oscillator frequency for the user flash memory. This parameter is used for simulation purposes only. The values are 5.56MHz and 3.33MHz. If omitted, the default is 5.56MHz.
Erase time	Specify the erase time.
Program time	Specify the program time.
Generate netlist	Turn on this option if you want to generate a netlist for your third-party EDA synthesis tool to estimate the timing and resource usage of the IP core. If you turn on this option, a netlist file (**_syn.v**) is generated. This file is a representation of the customized logic used in the Quartus II software and provides the connectivity of the architectural elements in the IP core but may not represent true functionality.

Configuration Setting	Description
Summary Page	Specify the types of files to be generated. The Variation file (*<function name>* **.v**) contains wrapper code in the language you specified on page 2a and is automatically generated. Choose from the following types of files: • 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>*_**inst.v**) • Verilog HDL black box file (*<function name>*_**bb.v)** For more information about the wizard-generated files, refer to the *Introduction to Altera IP Cores*.

Command Line Interface Parameters

Expert users can choose to instantiate and parameterize the IP core through the command-line interface using the clear box generator command. This method requires you to have command-line scripting knowledge.

For more information about using the clear box generator, refer to the *Introduction to Altera IP Cores*.

Table 13: ALTUFM_PARALLELParameters

Parameter Name	Type	Required	Description
`ACCESS_MODE`	String	No	Specifies the access mode for the user flash memory. Values are READ_WRITE and READ_ONLY. ***(1)***
`LPM_FILE`	String	No	Name of the Memory Initialization File (.mif) or Hexadecimal (Intel-Format) Files (.hex) containing RAM initialization data (<file name>), or UNUSED. If you use a HEX file to initialize with the memory, you must use a word size of 16 bits. Data bits need to be located in the MSBs of the data word and pad the LSBs with 1.
`OSC_FREQUENCY`	Integer	No	Specifies the oscillator frequency for the user flash memory. Values are 180000 ps (5.56 MHz) or 300000 ps (3.33 MHz). If omitted, the default is 180000 ps. The OSC_FREQUENCY setting controls the oscillator frequency during simulations and does not affect the device's oscillator frequency, which can range from 3.33 MHz (300000 ps) to 5.56 MHz (180000 ps). ***(2)***
`WIDTH_ADDRESS`	Integer	Yes	Specifies the width for the addr bus. ***(3)***
`WIDTH_DATA`	Integer	Yes	Specifies the width for the data bus. ***(3)***
`WIDTH_UFM_ADDRESS`	Integer	Yes	Specifies the internal address width for the user flash memory.

Parameter Name	Type	Required	Description
LPM_TYPE	String	No	Identifies the library of parameterized modules (LPM) entity name in VHDL Design Files (.vhd).

Notes to Table 2–5:

1. This parameter is used with the parallel or SPI interface protocol only.
2. This parameter is used for simulation purposes only.
3. This parameter is used with the parallel interface protocol only.

Table 14: ALTUFM_SPI Parameters

Parameter Name	Type	Required	Description
ACCESS_MODE	String	No	Specifies the access mode for the user flash memory. Values are READ_WRITE and READ_ONLY. This parameter is used with the parallel or SPI interface protocol only.
CONFIG_MODE	String	Yes	Specifies the configuration mode for the SPI interface protocol. Values are BASE and EXTENDED. The default is EXTENDED. When the CONFIG_MODE parameter is set to EXTENDED, the flash memory uses 16-bit word size for the address and data word. When the CONFIG_MODE parameter is set to BASE, the flash memory uses 8-bit word size for the address and data word and uses only sector 0 of the UFM. When the CONFIG_MODE parameter is specified to BASE and you specify a MIF or HEX file, you must specify the initial contents of sector 0 and 1 of the UFM. Addresses 0 through 255 of the MIF or HEX file must contain user data while addresses 256 through 511 should be set to 1. If you use a HEX file, you must specify all 16-bits of data. The user data (8-bits) should be placed in the MSBs of the data word and the LSBs should be padded with 1.
LPM_FILE	String	No	Name of the Memory Initialization File (.mif) or Hexadecimal (Intel-Format) Files (.hex) containing RAM initialization data (<file name>), or UNUSED.

Parameter Name	Type	Required	Description
OSC_FREQUENCY	Integer	No	Specifies the oscillator frequency for the user flash memory. This parameter is used for simulation purposes only. Values are 180000 ps (5.56 MHz) or 300000 ps (3.33 MHz). If omitted, the default is 180000 ps. The OSC_FREQUENCY setting controls the oscillator frequency during simulations and does not affect the device's oscillator frequency, which can range from 3.33 MHz (300000 ps) to 5.56 MHz (180000 ps).
WIDTH_UFM_ADDRESS	Integer	Yes	Specifies the internal address width for the user flash memory.
LPM_TYPE	String	No	Identifies the library of parameterized modules (LPM) entity name in VHDL Design Files (.vhd).

Table 15: ALTUFM_I2CParameters

Parameter Name	Type	Required	Description
ACCESS_MODE	String	Yes	Specifies the access mode. Values are READ_WRITE and READ_ONLY. If omitted, the default is READ_WRITE.
INTENDED_DEVICE_FAMILY	String	No	Create the ALTMULT_ADD IP core with the **parameter editor** to calculate the value for this parameter. ***(1)***
ERASE_METHOD	String	No	Specifies the erase method. Values are MEM_ADD, A2_ERASE, DEV_ADD_111, and NO_ERASE. If omitted, the default is MEM_ADD.
FIXED_DEVICE_ADD	String	Yes	4-bit address that specifies the 4 MSBs of the device address.
LPM_FILE	String	No	Name of the Memory Initialization File (.mif) or Hexadecimal (Intel-Format) Files (.hex) containing RAM initialization data (<file name>), or UNUSED. If you use a HEX file to initialize with the memory, you must use a word size of 16 bits. Data bits need to be located in the MSBs of the data word and pad the LSBs with 1.
MEM_ADD_ERASE0	String	No	If the ERASE_METHOD parameter has a value of MEM_ADD, the MEM_ADD_ERASE0 parameter must specify the 8-bit memory address that erases sector 0 of the UFM block.
MEM_ADD_ERASE1	String	No	If the ERASE_METHOD parameter has a value of MEM_ADD, the MEM_ADD_ERASE1 parameter must specify the 8-bit memory address that erases sector 1 of the UFM block.

Parameter Name	Type	Required	Description
MEM_PROTECT	String	No	Specifies whether the write-protect port protects only the upper half of the UFM block or the entire UFM block from writes/erases. Values are FULL and UPPER_HALF. If omitted, the default is FULL.
MEMORY_SIZE	String	Yes	Specifies the memory size. Values are 1K, 2K, and 4K. If omitted, the default is 4K.
OSC_FREQUENCY	Integer	No	Specifies the oscillator frequency for the user flash memory. Values are 180000 ps (5.56 MHz) or 300000 ps (3.33 MHz). If omitted, the default is 180000 ps. The OSC_FREQUENCY setting controls the oscillator frequency during simulations and does not affect the device's oscillator frequency, which can range from 3.33 MHz (300000 ps) to 5.56 MHz (180000 ps). *(2)*
WRITE_MODE	String	No	Specifies the write mode. If omitted, the default is SINGLE_BYTE.
LPM_HINT	String	No	Allows you to assign Altera-specific parameters in VHDL Design Files (.vhd). The default is UNUSED.
LPM_TYPE	String	No	Identifies the library of parameterized modules (LPM) entity name in VHDL Design Files.

Notes to Table 2–7:

1. This parameter is used for modeling and behavioral simulation purposes
2. This parameter is used for simulation purposes only.

Table 16: ALTUFM_NONE Parameters

Parameter Name	Type	Required	Description
busy	Yes	Busy signal that indicates when memory is busy.	*(1)*
drdout	Yes	Data register output.	*(1)*
osc	No	Oscillator output.	If the osc port is specified, the oscena port is required. *(1)*
rtpbusy	Yes	Busy signal that indicates when in system configuration is using flash memory.	When the rtpbusy is high, it cannot be used. *(1)*

Note to Table 2–8:

1. This port is used without an interface protocol only.

C**Functional Description ###func_descript###

This chapter describes the functional description and the design examples of the ALTUFM IP core. This section also includes the ports descriptions of the ALTUFM IP core. You can use the ports to customize the ALTUFM IP core according to your application.

Upgrading IP Cores

IP core variants generated with a previous version of the Quartus II software may require upgrading before use in the current version of the Quartus II software. Click **Project** > **Upgrade IP Components** to identify and upgrade IP core variants.

The **Upgrade IP Components** dialog box provides instructions when IP upgrade is required, optional, or unsupported for specific IP cores in your design. You must upgrade IP cores that require it before you can compile the IP variation in the current version of the Quartus II software. Many Altera IP cores support automatic upgrade.

The upgrade process renames and preserves the existing variation file (**.v**, **.sv**, or **.vhd**) as <*my_variant*>_**BAK.v**, **.sv**, **.vhd** in the project directory.

Table 17: IP Core Upgrade Status

IP Core Status	Corrective Action
Required Upgrade IP Components	You must upgrade the IP variation before compiling in the current version of the Quartus II software.
Optional Upgrade IP Components	Upgrade is optional for this IP variation in the current version of the Quartus II software. You can upgrade this IP variation to take advantage of the latest development of this IP core. Alternatively you can retain previous IP core characteristics by declining to upgrade.
Upgrade Unsupported	Upgrade of the IP variation is not supported in the current version of the Quartus II software due to IP core end of life or incompatibility with the current version of the Quartus II software. You are prompted to replace the obsolete IP core with a current equivalent IP core from the IP Catalog.

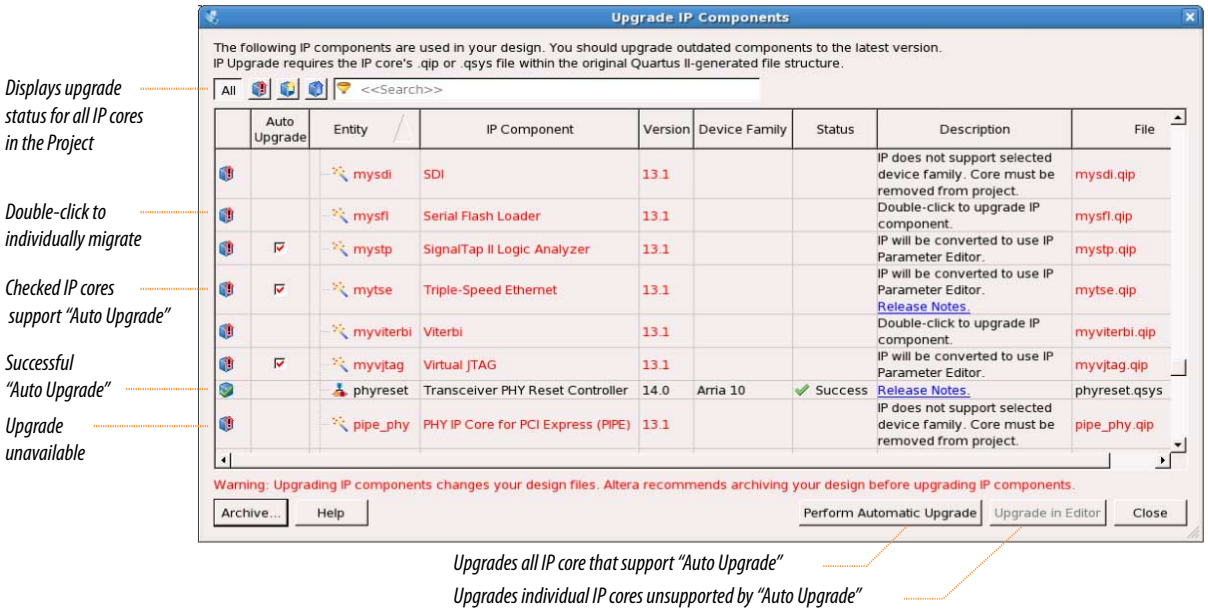
Before you begin

- Archive the Quartus II project containing outdated IP cores in the original version of the Quartus II software: Click **Project** > **Archive Project** to save the project in your previous version of the Quartus II software. This archive preserves your original design source and project files.
- Restore the archived project in the latest version of the Quartus II software: Click **Project** > **Restore Archived Project**. Click **OK** if prompted to change to a supported device or overwrite the project database. File paths in the archive must be relative to the project directory. File paths in the archive must reference the IP variation **.v** or **.vhd** file or **.qsys** file (not the **.qip** file).

1. In the latest version of the Quartus II software, open the Quartus II project containing an outdated IP core variation. The **Upgrade IP Components** dialog automatically displays the status of IP cores in your project, along with instructions for upgrading each core. Click **Project** > **Upgrade IP Components** to access this dialog box manually.

2. To simultaneously upgrade all IP cores that support automatic upgrade, click **Perform Automatic Upgrade**. The **Status** and **Version** columns update when upgrade is complete. Example designs provided with any Altera IP core regenerate automatically whenever you upgrade the IP core.

Figure 6: Upgrading IP Cores



Example 1: Upgrading IP Cores at the Command Line

You can upgrade IP cores that support auto upgrade at the command line. IP cores that do not support automatic upgrade do not support command line upgrade.

- To upgrade a single IP core that supports auto-upgrade, type the following command:

```
quartus_sh –ip_upgrade –variation_files <my_ip_filepath/my_ip>.<hdl>
<qii_project>

Example:
quartus_sh -ip_upgrade -variation_files mega/pll25.v hps_testx
```

- To simultaneously upgrade multiple IP cores that support auto-upgrade, type the following command:

```
quartus_sh –ip_upgrade –variation_files "<my_ip_filepath/my_ip1>.<hdl>;
<my_ip_filepath/my_ip2>.<hdl>"  <qii_project>

Example:
quartus_sh -ip_upgrade -variation_files "mega/pll_tx2.v;mega/pll3.v" hps_testx
```

Note: IP cores older than Quartus II software version 12.0 do not support upgrade. Altera verifies that the current version of the Quartus II software compiles the previous version of each IP core. The *Altera IP Release Notes* reports any verification exceptions

for Altera IP cores. Altera does not verify compilation for IP cores older than the previous two releases.

Related Information

Altera IP Release Notes

Migrating IP Cores to a Different Device

IP migration allows you to target the latest device families with IP originally generated for a different device. Some Altera IP cores require individual migration to upgrade. The **Upgrade IP Components** dialog box prompts you to double-click IP cores that require individual migration.

1. To display IP cores requiring migration, click **Project** > **Upgrade IP Components**. The **Description** field prompts you to double-click IP cores that require individual migration.
2. Double-click the IP core name, and then click **OK** after reading the information panel.
   The parameter editor appears showing the original IP core parameters.
3. For the **Currently selected device family**, turn off **Match project/default**, and then select the new target device family.
4. Click **Finish**, and then click **Finish** again to migrate the IP variation using best-effort mapping to new parameters and settings. Click **OK** if you are prompted that the IP core is unsupported for the current device. A new parameter editor opens displaying best-effort mapped parameters.
5. Click **Generate HDL**, and then confirm the **Synthesis** and **Simulation** file options. Verilog is the parameter editor default HDL for synthesis files. If your original IP core was generated for VHDL, select **VHDL** to retain the original output HDL format.
6. To regenerate the new IP variation for the new target device, click **Generate**. When generation is complete, click **Close**.
7. Click **Finish** to complete migration of the IP core. Click **OK** if you are prompted to overwrite IP core files. The **Device Family** column displays the migrated device support. The migration process replaces <*my_ip*>**.qip** with the <*my_ip*>**.qsys** top-level IP file in your project.

   Note: If migration does not replace <*my_ip*>**.qip** with <*my_ip*>**.qsys**, click **Project > Add/Remove Files in Project** to replace the file in your project.

8. Review the latest parameters in the parameter editor or generated HDL for correctness. IP migration may change ports, parameters, or functionality of the IP core. During migration, the IP core's HDL generates into a library that is different from the original output location of the IP core. Update any assignments that reference outdated locations. If your upgraded IP core is represented by a symbol in a supporting Block Design File schematic, replace the symbol with the newly generated <*my_ip*>**.bsf** after migration.

   Note: The migration process may change the IP variation interface, parameters, and functionality. This may require you to change your design or to re-parameterize your variant after the **Upgrade IP Components** dialog box indicates that migration is complete. The **Description** field identifies IP cores that require design or parameter changes.

Related Information

Altera IP Release Notes

Functional Description

This chapter describes the functional description and the design examples of the ALTUFM IP core. This section also includes the ports descriptions of the ALTUFM IP core. You can use the ports to customize the ALTUFM IP core according to your application.

User Flash Memory

User flash memory (UFM) provides access to the serial flash memory blocks in MAX II and MAX V devices. The UFM can be used like a serial EEPROM for storing up to 8,192 bits of non-volatile information. The hardware interface is a simple 12-pin protocol, providing interfaces similar to the industry standards for serial EEPROMs. The UFM connects to the logic array through the MultiTrack™ interconnect, allowing any logic cell to interface with the UFM. The UFM features two sectors for partitioned sector erase, a built-in internal oscillator that drives internal device logic; program, erase, and busy signals, auto-increment addressing, and a serial interface to the internal device logic with a programmable interface. The ALTUFM IP core, available in Altera's Quartus® II software, provides interface logic for a subset of these interfaces (parallel and SPI).

Note: Any interfaces not provided by the IP core or design examples require you to create user logic to bridge the UFM block to your desired interface protocol.

Each UFM array is organized as two separate sectors, with 4,096 bits per sector. Each sector can be erased independently.

Table 18: MAX II UFM Array Size

Device	Total Bits	Sectors	Address Bits	Data Width
EPM240	8,192	2 (4096 bits per sector)	9	16
EPM570	8,192	2 (4096 bits per sector)	9	16
EPM1270	8,192	2 (4096 bits per sector)	9	16
EPM2210	8,192	2 (4096 bits per sector)	9	16

Memory Organization Map

The memory organization map includes 512 locations with 9 bits, addressing a range of 000h to 1FFh. Each location stores 16-bit wide data. The most significant bit (MSB) of the address register indicates the sector in operation.

Table 19: Memory Organization

Sector	Address Range	
1	100h	1FFh
0	000h	0FFh

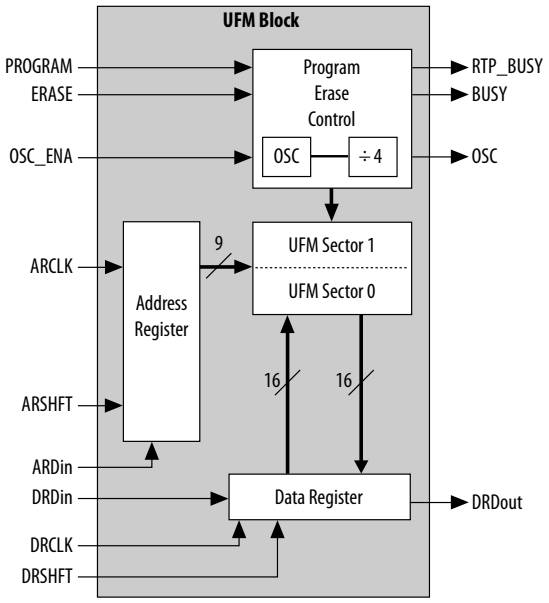
Using and Accessing UFM Storage

Use the UFM to store data of different memory sizes and data widths. The UFM storage width is 16 bits, however, you can implement different data widths or a serial interface using the ALTUFM IP core.

Table 20: Data Widths for Logic Array Interfaces

Logic Array Interface	Data Width (Bits)	Interface Type
SPI	8 or 16	Serial
Parallel	Options of 3 to 16	Parallel
I2C	8	Serial
None	16	Serial

Figure 7: MAX II UFM Block Diagram and Interface Signals Block Diagram



UFM Operating Modes

There are three UFM block modes:

- Read/stream read
- Program (write)

- Erase

The UFM block supports byte write, but does not support byte erase, requiring a sector-based erase sequence prior to any programming or writing. If the data content of a specific byte location needs to be overwritten in the UFM, the entire sector that byte resides in must be erased unless that byte location was already erased (all 1s).

If your design allows you to access the MAX II and MAX VUFM (write or erase), you must ensure that all the erase or write operations of the UFM are completed before starting any ISP session (including stand-alone verify, examine, setting security bit, and reading the contents of the UFM). Never start an ISP session when any erase or write operation of the UFM is in progress, as this may put the device in an unrecoverable state. This restriction does not apply to the read operation of the UFM.

The MAX II and MAX V UFM can be programmed, erased, and verified through the Joint Test Action Group (JTAG) port, or through connections to or from the logic array in accordance with IEEE Std. 1532-2002. There are 13 interface signals (Figure 3–1) to and from the UFM block and logic array, which allow the logic array to read or write to the UFM during device user mode. A reference design or user logic can be used to interface the UFM to many standard interface protocols such as Serial Communication Interface (SCI), Serial Peripheral Interface (SPI), Inter-Integrated Circuit (I2C), Microwire, or other proprietary protocols.

Serial Peripheral Interface

Serial peripheral interface (SPI) is a four-pin serial communication subsystem included on the Motorola 6805 and 68HC11 series microcontrollers. SPI allows the microcontroller unit to communicate with peripheral devices, and is capable of inter-processor communications in a multiple-master system.

The SPI bus consists of masters and slaves. The master device initiates and controls the data transfers, and provides the clock signal for synchronization. The slave device responds to the data transfer request from the master device. The master device in an SPI bus initiates a service request and the slave devices respond to the service request. The UFM is configured as the slave device for the SPI bus.

There are only four pins in SPI: SI, SO, SCK, and nCS. Data transmitted to the SI port of the slave device is sampled by the slave device at the positive SCK clock. Data transmits from the slave device through SO at the negative SCK clock edge. When nCS is asserted, it means the current device is being selected by the master device from the other end of the SPI bus for service. When nCS is not asserted, the SI and SCK ports should be blocked from receiving signals from the master device, and SO should be in high impedance state to avoid causing contention on the shared SPI bus. All instructions, addresses, and data are transferred with the MSB first, and start with high-to-low nCS transition.

The nCS signal cannot be toggled simultaneously with the clock edge of SCK. During read/write mode, a low-to-high transition of nCS requires a minimum of 420 ns of hold time before it can be asserted low again. A high-to-low transition of nCS requires a minimum wait of 420 ns before the first SCK clock edge.

Parallel Interface

This interface allows for parallel communication between the UFM block and outside logic. Once the READ request, WRITE request, or ERASE request is asserted (active low assertion), the outside logic or device (such as a microcontroller) are free to continue their operation while the data in the UFM is retrieved, written, or erased. During this time, the nBUSY signal is driven “low” to indicate that it is not available to respond to any further request. After the operation is complete, the nBUSY signal is brought back to “high” to indicate that it is now available to service a new request. If it was the Read request, the DATA_VALID is driven “high” to indicate that the data at the DO port is the valid data from the last read address.

Asserting READ, WRITE, and ERASE at the same time is not allowed. Multiple requests are ignored and nothing is read from, written to, or erased in the UFM block. There is no support for sequential read and page write in the parallel interface.

Even though the altufm IP core allows you to select the address widths range from 3 bits to 9 bits, the UFM block always expects full 9 bits width for the address register. Therefore, the ALTUFM IP core always pads the remaining LSB of the address register with '0's if the register width selected is less than 9 bits. The address register will point to sector 0 if the address received at the address register starts with a '0'. On the other hand, the address register will point to sector 1 if the address received starts with a '1'.

Even though you can select an optional data register width of 3 to 16 bits using the altufm IP core, the UFM block always expects full 16 bits width for the data register. Reading from the data register will always proceed from MSB to LSB. The altufm IP core will always pad the remaining LSB of the data register with 1s if the user selects a data width of less than 16-bits.

During the read/write mode, a high-to-low transition of a mode signal (nREAD, nWRITE, or nERASE) requires a minimum of 420 ns of hold time before the instruction signal can be pulled high again. The address register and data input must be held for at least 420 ns once the mode signal is asserted low. The high-to-low transition of nBUSY requires a maximum wait of 210 ns once nRead, nWrite, or nErase is asserted low.

I2C Interface

The inter-integrated circuit (I2C) is a bidirectional two-wire interface protocol. Choose this interface to configure the UFM block and logic as a slave device for the I2C bus. The size of UFM memory, the access mode, the erase method, and the protection required for the UFM block all dictate the resources required on a particular device for this interface implementation.

For more information about using the ALTUFM IP core with the I2C interface, refer to *AN489: Using the UFM in MAX II Devices*.

None (Altera Serial Interface)

None means using the dedicated UFM serial interface. The built-in UFM interface uses 13 pins for the communication. You can produce your own interface design to communicate to/from the dedicated UFM interface and implement it in the logic array.

Common Applications

The MAX II and MAX V UFM block is the best choice for storing manufacturing data, helping to improve board space efficiency, and minimizing system cost by integrating board-level flash memory, EEPROM capabilities, and system logic into one device. You can customize the UFM communication system to comply with different manufacturers' standard interface protocols to access manufacturing product data.

The UFM block is used to replace on-board flash and EEPROM memory devices which are used to store ASSP or processor configuration bits, or electronic ID information for a board during manufacturing. Since you can program the UFM block to suit your needs, MAX II and MAX V devices offer more interface flexibility than an off-the-shelf EEPROM device.

Design Example: User Flash Memory with SPI Interface

This design example uses the ALTUFM IP core to implement user flash memory with the SPI Interface using the parameter editor in the Quartus II software.

In this example, you perform the following activities:

- Create user flash memory with an SPI interface using the ALTUFM IP core and the parameter editor
- Implement the design and assign the EPM2210F256C3 device to the project
- Compile and simulate the design

The design examples are available for download from the following locations:

- On the Documentation: Quartus II Development Software page, expand the **Using IP cores** section and then expand the **I/O** section.
- On the Documentation: User Guides section of the Altera website.

Generate the User Flash Memory

Perform the following steps to generate the user flash memory:

1. In the Quartus II software, open **alt_ufm_DesignExample.qar** and restore the archive file into your working directory.
2. In the IP Catalog (**Tools > IP Catalog**), locate and double-click the ALTUFM_SPI IP core. The parameter editor appears.
3. Specify the following parameters:

Table 21: Configuration Settings for ALTUFM Design Example

Configuration Setting	Value
What name do you want for the output file?	ufm_ex
Currently selected device family	**MAX II**
Match project/default	Turned on
Access mode	**Read and write**
Configuration mode	**Extended mode (16 bit address and data)**
Use 'osc' output port	Turned on
Use 'oscena' input port	Turned off
Memory content initialization	Initialize blank memory
Oscillator frequency	**3.33**
Erase time	—
Program time	—
Generate netlist	Turned off
Variation file	Turned on
Quartus II symbol file	Turned off

Configuration Setting	Value
Instantiation template file	Turned off
Verilog HDL black-box file	Turned on
AHDL Include file	Turned off
VHDL component declaration file	Turned off

The ALTUFM variation is now built.

Implement the User Flash Memory

This section describes how to assign the EPM2210F256C3 device to the project and compile the project.

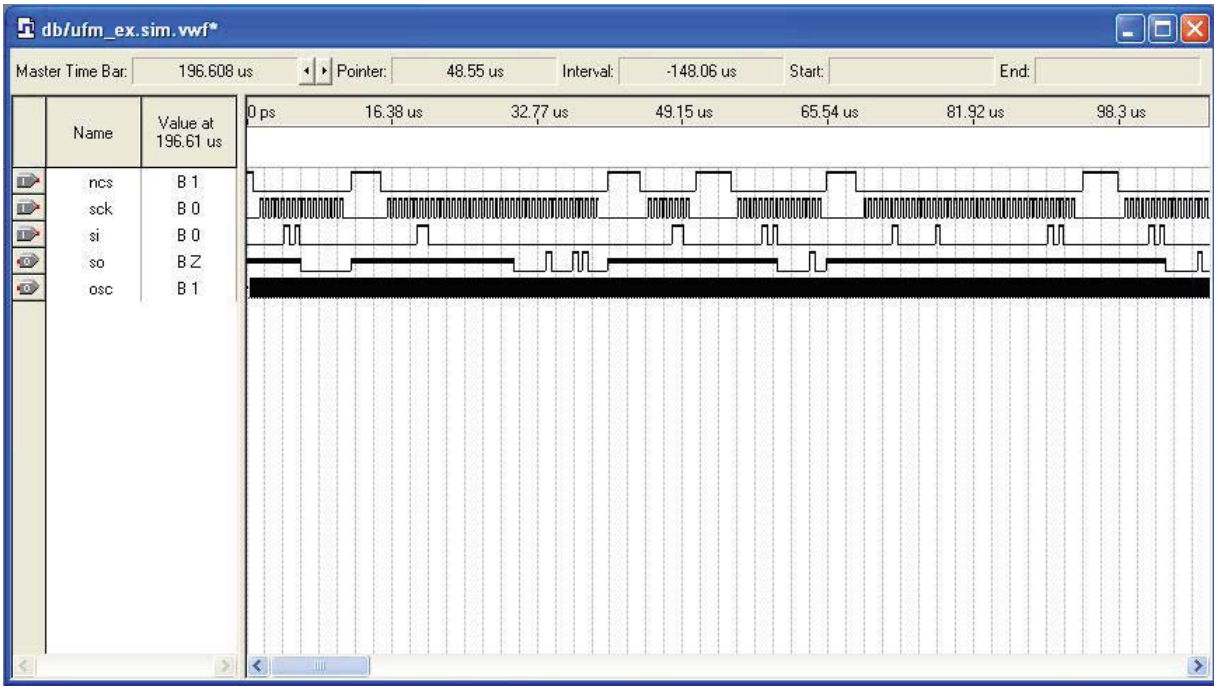
1. In the Quartus II software, on the Assignments menu, click Settings.
2. The Device Settings window displays.
3. In the Category list, select Device.
4. In the Family list, select MAX II.
5. In the Target device list, click Specific device selected in 'Available devices' list.
6. In the Available devices list, select EPM2210F256C3.
7. Leave the other options in the default state and click OK.
8. On the Processing menu, select Start Compilation to compile the design.
9. The Full compilation was successful box displays. Click OK.

Functional Results—Simulate the User Flash Memory in the Quartus II Software

This section describes how to verify the design example you just created by simulating the design using the Quartus II Simulator. To set up the Quartus II Simulator, follow these steps:

1. On the Processing menu, click the **Generate Functional Simulation Netlist** option.
2. When the **Functional Simulation Netlist Generation was successful** message appears, click **OK**.
3. On the Assignments menu, click Settings and then select **Simulator Settings** from the Category list.
4. In the Category list, select Simulator.
5. In the Simulation mode list, select Functional.
6. Type ufm_ex_ip.vwf in the Simulation input box, or click Browse (...) to select the file in the project folder.
7. Turn on the End simulation at: option and type 50.0 and select ms from the list.
8. Turn on Automatically add pins to simulation output waveforms and Simulation coverage reporting options.
9. Turn off Check outputs option.
10. Turn off Overwrite simulation input file with simulation results.
11. Turn off Generate Signal Activity File option.
12. Click OK.
13. On the Processing menu, click **Start** Simulation to run a simulation.
14. The Simulation Report window appears.

Figure 8: Simulation Waveform



Functional Results—Simulate the User Flash Memory in ModelSim-Altera Software

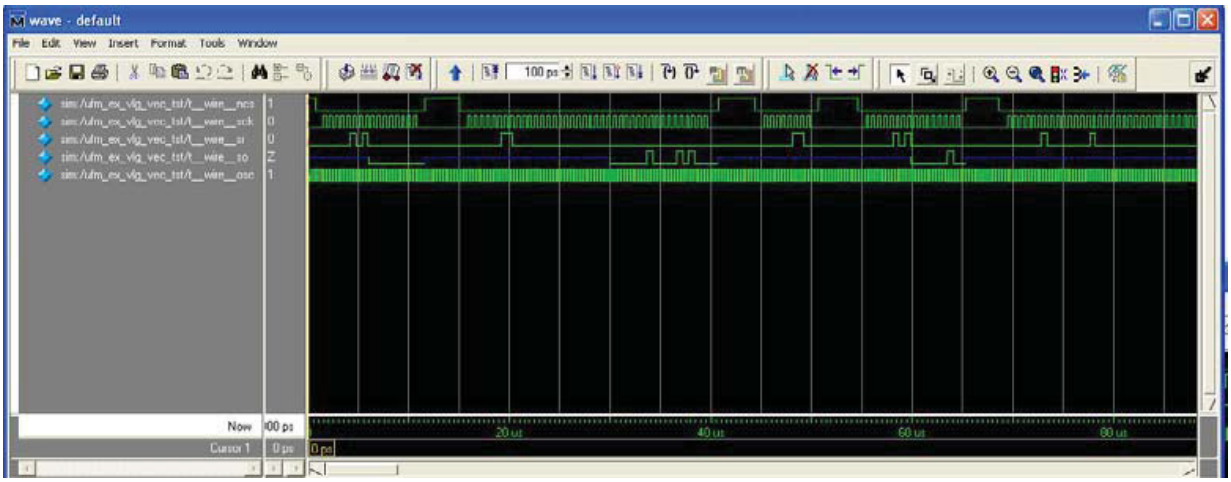
Simulate the design in the ModelSim-Altera software to compare the results of both simulators.

To set up the ModelSim-Altera software , follow these steps:

1. Unzip the **alt_ufm_msim.zip** file to any working directory on your PC.
2. Start the ModelSim-Altera software.
3. On the File menu, click **Change Directory**.
4. Select the folder in which you unzipped the files. Click **OK**.
5. On the Tools menu, click **Execute Macro**.
6. Select the **ufm_ex.do** file and click **Open**. This is a script file for ModelSim-Altera software that automated all necessary settings for the simulation.
7. Verify the results shown in the Waveform Viewer window.

You can rearrange signals, remove signals, add signals, and change the radix by modifying the script in **ufm_ex.do** accordingly.

Figure 9: ModelSim-Altera Software Simulation Waveforms



ALTUFM_PARALLEL Ports

Input Ports

Table 22: ALTUFM_PARALLEL Input Ports

Port Name	Required	Description	Comments
`addr[]`	Yes	Address bus.	Input port [WIDTH_ADDRESS-1..0] wide. The add port is used with the parallel interface protocol only. *(1)*
`di[]`	Yes	Data bus input.	Input port [WIDTH_DATA-1..0] wide. *(1)*
`nerase`	No	Erase input port.	When the ACCESS_MODE parameter is set to READ_ONLY, the nerase port cannot be used. *(1)*
`nread`	Yes	Read input port.	*(1)*
`nwrite`	No	Write input port.	*(1)*

Note to Table 3–5:

1. This port is used with the parallel interface protocol only.

Output Ports

Table 23: ALTUFM_PARALLEL Output Ports

Port Name	Required	Description	Comments
`data_valid`	Yes	Data output.	*(1)*
`nbusy`	Yes	Busy signal.	*(1)*
`do[]`	Yes	Data bus output.	Output port [WIDTH_DATA-1..0] wide. *(1)*

Note to Table 3–6:

1. This port is used with the parallel interface protocol only.

ALTUFM_SPI Ports

Input Ports

Table 24: ALTUFM_SPI Input Ports

Port Name	Required	Description	Comments
`ncs`	Yes	Device select input.	*(1)*
`sck`	Yes	Serial data clock.	*(1)*
`si`	Yes	Serial data input.	*(1)*

Note to Table 3–7:

1. This port is used with the SPI interface protocol only.

Output Ports

Table 25: ALTUFM_SPI Output Ports

Port Name	Required	Description	Comments
`so`	Yes	Serial bus output.	The si port is used with the SPI interface protocol only.

ALTUFM_I2C Ports

Input Ports

Table 26: ALTUFM_I2C Input Ports

Port Name	Required	Description	Comments
`a0`	Yes	Input port that specifies the LSB (bit 0) of the 7-bit device address.	*(1)*
`a1`	Yes	Input port that specifies the first bit of the 7-bit device address.	*(1)*
`a2`	Yes	Input port that specifies the second bit of the 7-bit device address.	*(1)*
`wp`	No	Write protect input port.	If wp port is set to 1, the memory is write protected and erase and write are disabled.

Note to Table 3–9:

1. This port can be used to vary the device address allocated to the ALTUFM_I2C IP core.

Output Ports

Table 27: ALTUFM_I2C Output Ports

Port Name	Required	Description	Comments
`osc`	No	Oscillator output port.	The osc port can be used to access the UFM internal oscillator. The oscillator can be used as a general-purpose clock for other logic circuitry.

Bidirectional Ports

Table 28: ALTUFM_I2C Bidirectional Ports

Port Name	Required	Description	Comments
`scl`	Yes	Bidirectional clock port.	Clock from master to slave.

Port Name	Required	Description	Comments
sda	Yes	Bidirectional clock port.	Data input from master and data output from slave.

ALTUFM_NONE Ports

Input Ports

Table 29: ALTUFM_NONE Input Ports

Port Name	Required	Description	Comments
arclk	Yes	Clock for the address register	*(1)*
ardin	Yes	Input for the address register.	*(1)*
arshft	Yes	Shift signal for the address register.	*(1)*
drclk	Yes	Clock for the data register.	*(1)*
drdin	Yes	Input for the data register.	*(1)*
drshft	Yes	Shift signal for the data register.	*(1)*
erase	Yes	Signal that controls the erase sequence.	*(1)*
oscena	No	Signal that enables the internal oscillator.	If the osc port is specified, the oscena port is required. *(1)*
program	No	Signal that initiates a program sequence.	*(1)*

Note to Table 3–12:

1. This port is used without an interface protocol only.

Output Ports

Table 30: ALTUFM_NONE Output Ports

Port Name	Required	Description	Comments
busy	Yes	Busy signal that indicates when memory is busy.	*(1)*

Port Name	Required	Description	Comments
drdout	Yes	Data register output.	*(1)*
osc	No	Oscillator output.	If the osc port is specified, the oscena port is required. *(1)*
rtpbusy	Yes	Busy signal that indicates when in system configuration is using flash memory.	When the rtpbusy is high, it cannot be used. *(1)*

Note to Table 3–13:

1. This port is used without an interface protocol only.

Document Revision History

The following table lists the revision history for this document.

Table 31: DateVersionChanges

Date	Version	Changes
August, 2014	2014.08.18	• Updated parameterization steps for legacy parameter editor. • Added note that this IP core does not support Arria 10 designs.
June 2014	2014.06.30	• Replaced MegaWizard Plug-In Manager information with IP Catalog. • Added standard information about upgrading IP cores. • Added standard installation and licensing information. • Removed outdated device support level information. IP core device support is now available in IP Catalog and parameter editor.
May 2014	3.2	Updated Page Write value in Table 2–3 on page 2–5
May 2012	3.1	Updated “Parameter Settings” on page 2–1.
March 2012	3.0	Added MAX V support.
August 2006	2.0	Updated for Quartus II 6.0 software.
July 2005	1.1	Updated for Quartus II 4.2 software.
May 2005	1.0	Initial release.