This chapter provides guidelines for user flash memory (UFM) applications by describing the features and functionality of the MAX® V UFM block and the Quartus® II ALTUFM megafunction.

Altera® MAX V devices feature a UFM block that can be used for storing non-volatile information up to 8 Kbits, similar to a serial EEPROM. The UFM provides an ideal storage solution that supports all protocols (serial peripheral interface (SPI), parallel, and other protocols) for interfacing through bridging logic designed into the MAX V logic array.

This chapter contains the following sections:

UFM Array Description

Each UFM array is organized as two separate sectors with 4,096 bits per sector. Each sector can be erased independently. Table 7–1 lists the dimensions of the UFM array.

Table 7–1. UFM Array Size

Device	Total Bits	Sectors	Address Bits	Data Width
5M40Z	8,192	2 (4,096 bits per sector)	9	16
5M80Z	8,192	2 (4,096 bits per sector)	9	16
5M160Z	8,192	2 (4,096 bits per sector)	9	16
5M240Z	8,192	2 (4,096 bits per sector)	9	16
5M570Z	8,192	2 (4,096 bits per sector)	9	16
5M1270Z	8,192	2 (4,096 bits per sector)	9	16
5M2210Z	8,192	2 (4,096 bits per sector)	9	16

Memory Organization Map

Table 7–2 lists the memory organization for the MAX V UFM block. There are 512 locations with 9 bits addressing a range of `000h` to `1FFh`. Each location stores 16-bit wide data. The MSB of the address register indicates the sector in operation.

Table 7–2. Memory Organization

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

Using and Accessing UFM Storage

You can use the UFM to store data of different memory sizes and data widths. Even though the UFM storage width is 16 bits, you can implement different data widths or a serial interface with the ALTUFM megafunction. Table 7–3 lists the different data widths available for the three types of interfaces supported in the Quartus II software, as well as no interface.

Table 7–3. Data Widths for Logic Array Interfaces

Logic Array Interfaces	Data Widths (Bits)	Interface Types
I²C	8	Serial
SPI	8 or 16	Serial
Parallel	Options of 3 to 16	Parallel
None	16	Serial

For more details about the logic array interface options in the ALTUFM megafunction, refer to "Software Support for UFM Block" on page 7–13.

☞ The UFM block is accessible through the logic array interface and the JTAG interface. However, the UFM logic array interface does not have access to the configuration flash memory (CFM) block.

UFM Functional Description

Figure 7–1 is the block diagram of the MAX V UFM block and the interface signals.
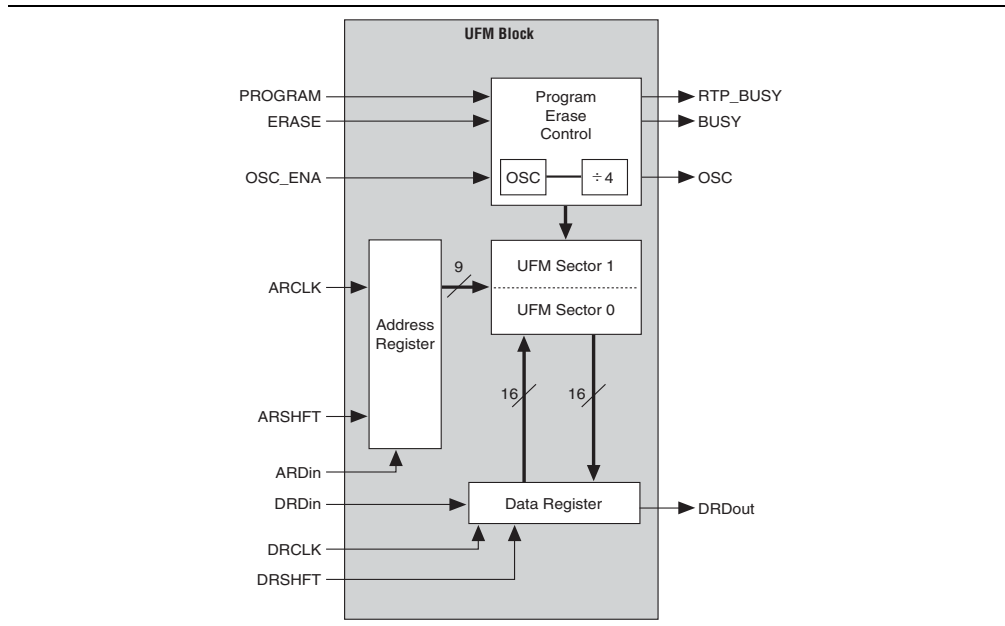
Figure 7–1. UFM Block and Interface Signals

Table 7–4 lists the MAX V UFM block input and output interface signals.

Table 7–4. UFM Interface Signals (Part 1 of 2)

Port Name	Port Type	Description
DRDin	Input	Serial input to the data register. It is used to enter a data word when writing to the UFM. The data register is 16 bits wide and data is shifted serially from the LSB to the MSB with each DRCLK. This port is required for writing, but unused if the UFM is in read-only mode.
DRCLK	Input	Clock input that controls the data register. It is required and takes control when data is shifted from DRDin to DRDout or loaded in parallel from the flash memory. The maximum frequency for DRCLK is 10 MHz.
DRSHFT	Input	Signal that determines whether to shift the data register or load it on a DRCLK edge. A high value shifts the data from DRDin into the LSB of the data register and from the MSB of the data register out to DRDout. A low value loads the value of the current address in the flash memory to the data register.
ARDin	Input	Serial input to the address register. It is used to enter the address of a memory location to read, program, or erase. The address register is 9 bits wide for the UFM size of 8,192 bits.
ARCLK	Input	Clock input that controls the address register. It is required when shifting the address data from ARDin into the address register or during the increment stage. The maximum frequency for ARCLK is 10 MHz.
ARSHFT	Input	Signal that determines whether to shift the address register or increment it on an ARCLK edge. A high value shifts the data from ARDin serially into the address register. A low value increments the current address by 1. The address register rolls over to 0 when the address space is at the maximum.

Table 7–4. UFM Interface Signals (Part 2 of 2)

Port Name	Port Type	Description
PROGRAM	Input	Signal that initiates a program sequence. On the rising edge, the data in the data register is written to the address pointed to by the address register. The BUSY signal asserts until the program sequence is completed.
ERASE	Input	Signal that initiates an erase sequence. On a rising edge, the memory sector indicated by the MSB of the address register is erased. The BUSY signal asserts until the erase sequence is completed.
OSC_ENA	Input	This signal turns on the internal oscillator in the UFM block. It is required when the OSC output is used, but optional otherwise. If OSC_ENA is driven high, the internal oscillator is enabled and the OSC output will toggle. If OSC_ENA is driven low, the internal oscillator is disabled and the OSC output drives constant high.
DRDout	Output	Serial output of the data register. Each time the DRCLK signal is applied, a new value is available. The DRDout data depends on the DRSHFT signal. When the DRSHFT signal is high, DRDout contains the new value that is shifted into the MSB of the data register. If DRSHFT is low, DRDout contains the MSB of the memory location read into the data register.
BUSY	Output	Signal that indicates when the memory is BUSY performing a PROGRAM or ERASE instruction. When it is high, the address and data register should not be clocked. The new PROGRAM or ERASE instruction is not executed until the BUSY signal is deasserted.
OSC	Output	Output of the internal oscillator. It can be used to generate a clock to control user logic with the UFM. It requires an OSC_ENA input to produce an output.
RTP_BUSY	Output	This output signal is optional and only needed if the real-time ISP feature is used. The signal is asserted high during real-time ISP and stays in the RUN_STATE for 500 ms before initiating real-time ISP to allow for the final read/erase/write operation. No read, write, erase, or address and data shift operations are allowed to be issued after the RTP_BUSY signal goes high. The data and address registers do not retain the contents of the last read or write operation for the UFM block during real-time ISP.

For more information about the interaction between the UFM block and the logic array of MAX V devices, refer to the *MAX V Device Architecture* chapter.
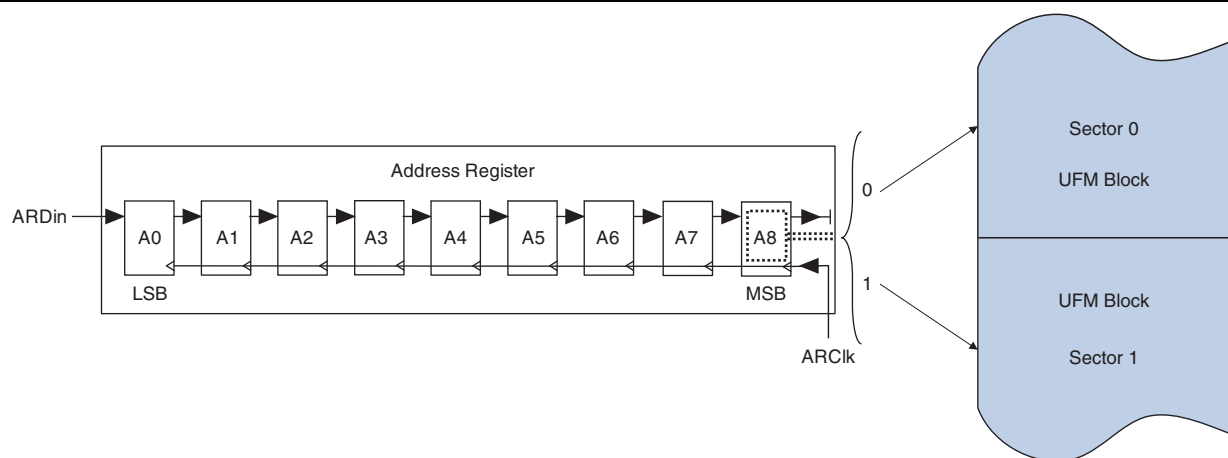
UFM Address Register

The MAX V UFM block is organized as a 512 × 16 memory. Because the UFM block is organized into two sectors, the MSB of the address indicates the sector that is used; 0 is for sector 0 (UFM0) while 1 is for sector 1 (UFM1). An ERASE instruction erases the content of the specific sector that is indicated by the MSB of the address register. Figure 7–2 shows the selection of the UFM sector using the MSB of the address register.

For more information about the erase mode, refer to "Erase" on page 7–11.

Figure 7–2. Selection of the UFM Sector Using the MSB of the Address Register

Three control signals exist for the address register: ARSHFT, ARCLK, and ARDin. ARSHFT is used as both a shift-enable control signal and an auto-increment signal. If the ARSHFT signal is high, a rising edge on ARCLK loads address data serially from the ARDin port and moves data serially through the register. A clock edge with the ARSHFT signal low increments the address register by 1. This implements an auto-increment of the address to allow data streaming. When a program, read, or erase sequence is executing, the address that is in the address register becomes the active UFM location.

UFM Data Register

The UFM data register is 16 bits wide with four control signals: DRSHFT, DRCLK, DRDin, and DRDout. DRSHFT distinguishes between clock edges that move data serially from DRDin to DRDout and clock edges that latch parallel data from the UFM sectors. If the DRSHFT signal is high, a clock edge moves data serially through the registers from DRDin to DRDout. If the DRSHFT signal is low, a clock edge captures data from the UFM sector pointed by the address register in parallel. The MSB is the first bit that is seen at DRDout. The data register DRSHFT signal is also used to enable the UFM for reading data. When the DRSHFT signal is low, the UFM latches data into the data register. Figure 7–3 shows the UFM data register.

Figure 7–3. UFM Data Register
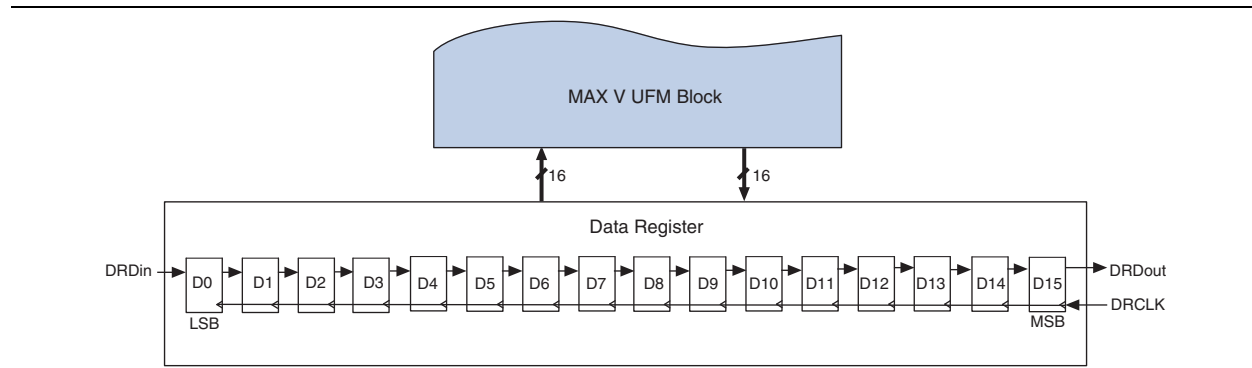
UFM Program/Erase Control Block

The UFM program/erase control block is used to generate all the control signals necessary to program and erase the UFM block independently. This block reduces the number of logic elements (LEs) necessary to implement a UFM controller in the logic array. It also guarantees correct timing of the control signals to the UFM. A rising edge on either PROGRAM or ERASE signal causes this control signal block to activate and begin sequencing through the program or erase cycle. At this point, for a program instruction, the data currently in the data register is written to the address pointed to by the address register.

Only sector erase is supported by the UFM. When an ERASE command is executed, this control block erases the sector whose address is stored in the address register. When the **PROGRAM** or **ERASE** command first activates the program/erase control block, the BUSY signal will be driven high to indicate an operation in progress in the UFM. After the program or erase algorithm is completed, the BUSY signal is forced low.

Oscillator

`OSC_ENA`, one of the input signals in the UFM block, is used to enable the oscillator signal to output through the `OSC` output port. You can use this `OSC` output port to connect with the interface logic in the logic array. It can be routed through the logic array and fed back as an input clock for the address register (`ARCLK`) and the data register (`DRCLK`). The output frequency of the `OSC` port is one-fourth that of the oscillator frequency. As a result, the frequency range of the `OSC` port is 3.9 to 5.3 MHz. The maximum clock frequency accepted by `ARCLK` and `DRCLK` is 10 MHz and the duty cycle accepted by the `DRCLK` and `ARCLK` input ports is approximately 45% to 50%.

When the `OSC_ENA` input signal is asserted, the oscillator is enabled and the output is routed to the logic array through the `OSC` output. When the `OSC_ENA` signal is set low, the `OSC` output drives constant high. The routing delay from the `OSC` port of the UFM block to `OSC` output pin depends on placement. You can analyze this delay using the TimeQuest timing analyzer.

The undivided internal oscillator, which is not accessible, operates in a frequency range from 15.6 to 21.2 MHz. The internal oscillator is enabled during power-up, in-system programming, and real-time ISP. At all other times, the oscillator is not running unless the UFM is instantiated in the design and the `OSC_ENA` port is asserted. To see how specific operating modes of the ALTUFM megafunction handle `OSC_ENA` and the oscillator, refer to "Software Support for UFM Block" on page 7–13. For user-generated logic interfacing to the UFM, the oscillator must be enabled during program or erase operations, but not during read operations. The `OSC_ENA` signal can be tied low if you are not issuing any **PROGRAM** or **ERASE** commands.

☞ During real-time ISP operation, the internal oscillator automatically enables and outputs through the `OSC` output port (if this port is instantiated) even though the `OSC_ENA` signal is tied low. You can use the `RTP_BUSY` signal to detect the beginning and ending of the real-time ISP operation for gated control of this self-enabled `OSC` output condition.

☞ The internal oscillator is not enabled all the time. The internal oscillator for the program or erase operation is only activated when the flash memory block is being programmed or erased. During a read operation, the internal oscillator is activated whenever the flash memory block is reading data.

Instantiating the Oscillator without the UFM

You can use the MAX II/MAX V Oscillator megafunction selection in the MegaWizard™ Plug-In Manager to instantiate the UFM oscillator if you intend to use this signal without using the UFM memory block. Figure 7–4 shows the ALTUFM_OSC megafunction instantiation in the Quartus II software.

Figure 7–4. The Quartus II ALTUFM_OSC Megafunction
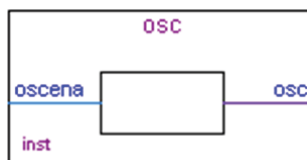
This megafunction is in the I/O folder on page 2a of the MegaWizard Plug-In Manager. On page 3 of the MAX II/MAX V Oscillator megafunction, you have an option to choose to simulate the OSC output port at its maximum or minimum frequency during the design simulation. The frequency chosen is only used as a timing parameter simulation and does not affect the real MAX V device OSC output frequency.

UFM Operating Modes

There are three different modes for the UFM block:

■ Read/Stream Read

■ Program (Write)

■ Erase

During program mode, address and data can be loaded concurrently. You can manipulate the UFM interface controls as necessary to implement the specific protocol provided the UFM timing specifications are met. Figure 7–5 through Figure 7–8 show the control waveforms for accessing UFM in three different modes. For program mode (Figure 7–7) and erase mode (Figure 7–8), the PROGRAM and ERASE signals can be asserted anytime after the address register and data register have been loaded. Do not assert the READ, PROGRAM, and ERASE signals or shift data and address into the UFM after entering the real-time ISP mode. You can use the RTP_BUSY signal to detect the beginning and end of real-time ISP operation and generate control logic to stop all UFM port operations. This user-generated control logic is only necessary for the ALTUFM_NONE megafunction, which provides no auto-generated logic. The other interfaces for the ALTUFM megafunction (ALTUFM_PARALLEL, ALTUFM_SPI, ALTUFM_I2C) contain control logic to automatically monitor the RTP_BUSY signal and will cease operations to the UFM when a real-time ISP operation is in progress.

☞ You can program the UFM or CFM block independently without overwriting the other block, which is not programmed. The Quartus II programmer provides the options to program the UFM and CFM blocks individually or together (the entire MAX V Device).

For guidelines about using ISP and real-time ISP while using the UFM block within your design, refer to *AN 100: In-System Programmability Guidelines*.

For a complete description of the device architecture, and for the specific values of the timing parameters listed in this chapter, refer to the *MAX V Device Architecture* chapter.

Read/Stream Read

The three control signals, `PROGRAM`, `ERASE`, and `BUSY` are not required during a read or stream read operation. To perform a read operation, the address register must be loaded with the reference address where the data is or is going to be located in the UFM. The address register can be stopped from incrementing or shifting addresses from `ARDin` by stopping the `ARCLK` clock pulse. `DRSHFT` must be asserted low at the next rising edge of `DRCLK` to load the data from the UFM to the data register. To shift the bits from the register, 16 clock pulses must be provided to read 16-bit wide data. You can use `DRCLK` to control the read time or disable the data register by discontinuing the `DRCLK` clock pulse. Figure 7–5 shows the UFM control waveforms during read mode.

The UFM block can also perform a stream read operation, using the address increment feature to read continuously from the UFM. Stream read mode is started by loading the base address into the address register. `DRSHFT` must then be asserted low at the first rising edge of `DRCLK` to load data into the data register from the address pointed to by the address register. `DRSHFT` will then assert high to shift out the 16-bit wide data with the MSB out first. Figure 7–6 shows the UFM control waveforms during stream read mode.
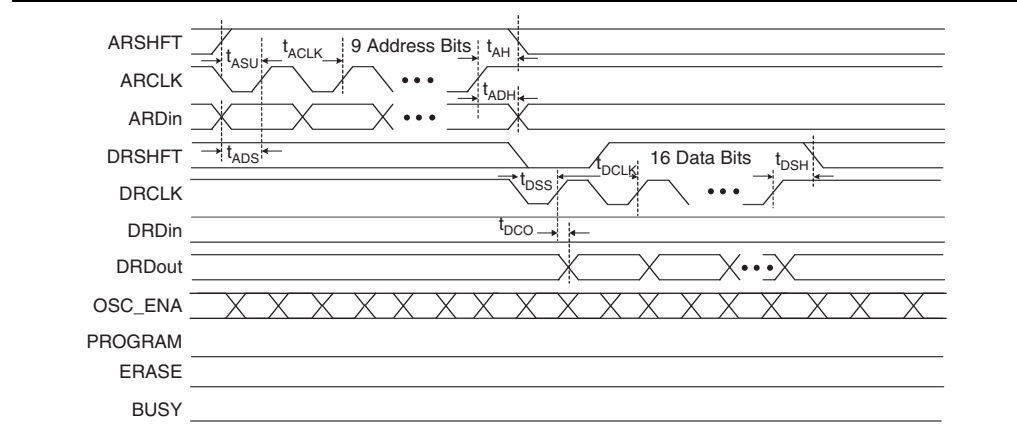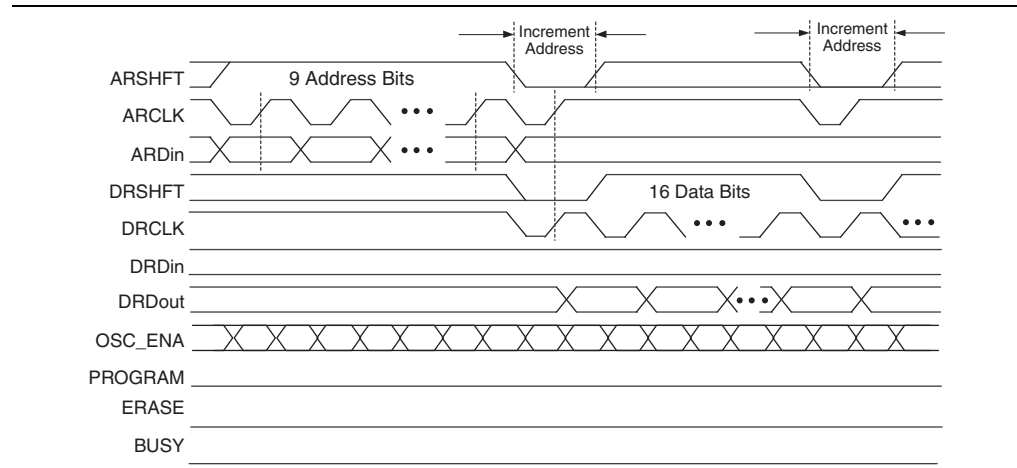
Figure 7–5. UFM Read Waveforms

Figure 7–6. UFM Stream Read Waveforms

Program

To program or write to the UFM, you must first perform a sequence to load the reference address into the address register. DRSHFT must then be asserted high to load the data serially into the data register starting with the MSB. Loading an address into the address register and loading data into the data register can be done concurrently. After the 16 bits of data have been successfully shifted into the data register, the PROGRAM signal must be asserted high to start writing to the UFM. On the rising edge, the data currently in the data register is written to the location currently in the address register. The BUSY signal is asserted until the program sequence is completed. The data and address register should not be modified until the BUSY signal is de-asserted, or the flash content will be corrupted. The PROGRAM signal is ignored if the BUSY signal is asserted. When the PROGRAM signal is applied at exactly the same time as the ERASE signal, the behavior is undefined and the flash content is corrupted. Figure 7–7 shows the UFM waveforms during program mode.

**Figure 7–7. UFM Program Waveforms**
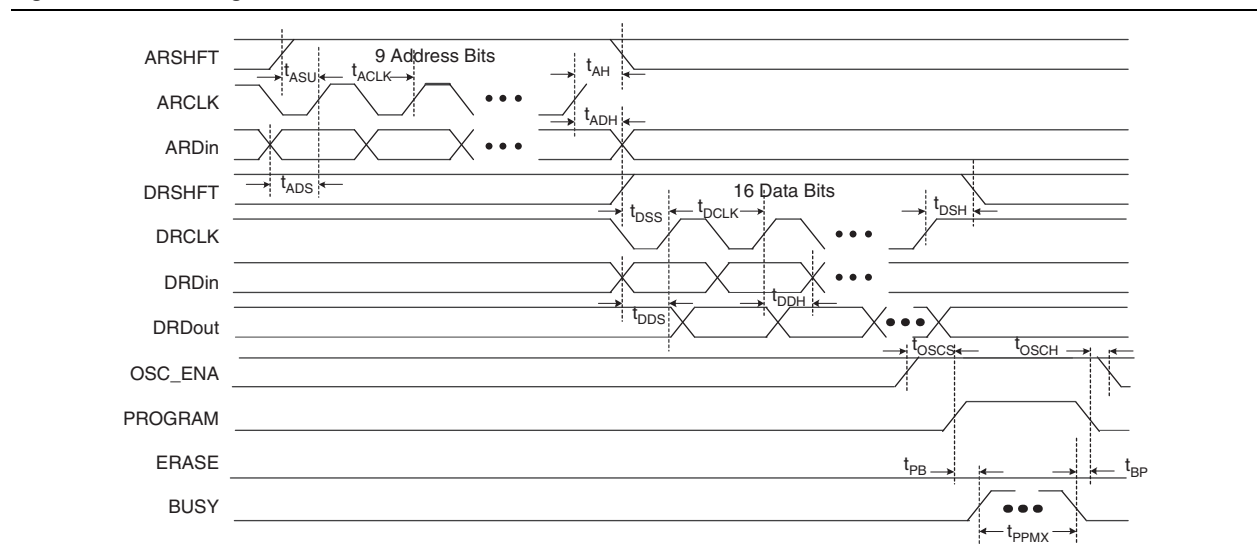
Erase

The ERASE signal initiates an erase sequence to erase one sector of the UFM. The data register is not needed to perform an erase sequence. To indicate the sector of the UFM to be erased, the MSB of the address register should be loaded with 0 to erase UFM sector 0, or 1 to erase UFM sector 1 (Figure 7–2 on page 7–5). On a rising edge of the ERASE signal, the memory sector indicated by the MSB of the address register will be erased. The BUSY signal is asserted until the erase sequence is completed. The address register should not be modified until the BUSY signal is de-asserted to prevent the flash content from being corrupted. This ERASE signal is ignored when the BUSY signal is asserted. Figure 7–8 illustrates the UFM waveforms during erase mode.

☞ When the UFM sector is erased, it has 16-bit locations all filled with FFFF. Each UFM storage bit can be programmed only once between erase sequences. You can write to any word up to two times providing the second programming attempt at that location only adds 0s. 1s are mask bits for your input word that cannot overwrite 0s in the flash array. New 1s in the location can only be achieved by an erase. Therefore, it is possible for you to perform byte writes because the UFM array is 16 bits for each location.

Figure 7–8. UFM Erase Waveforms
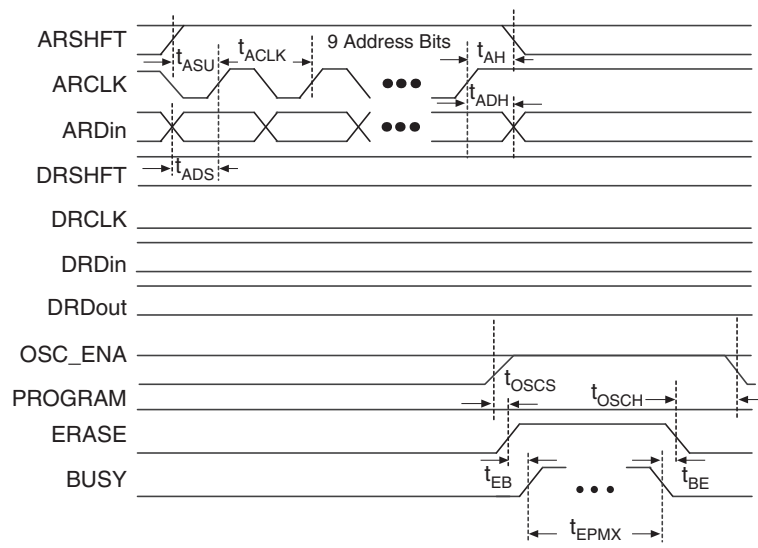
Programming and Reading the UFM with JTAG

In MAX V devices, you can write data to or read data from the UFM using the IEEE Std. 1149.1 JTAG interface. You can use a PC or UNIX workstation, the Quartus II Programmer, or the ByteBlasterMV™ or ByteBlaster™ II parallel port download cable to download Programmer Object File (**.pof**), Jam™ Standard Test and Programming Language (STAPL) Files (**.jam**), or Jam Byte-Code Files (**.jbc**) from the Quartus II software targeting the MAX V device UFM block.

☞ The **.pof**, **.jam**, and **.jbc** files can be generated using the Quartus II software.

Jam Files

Both **.jam** STAPL and **.jbc** files support programming for the UFM block.

Jam Players

Jam Players read the descriptive information in Jam files and translate them into data that programs the target device. Jam Players do not program a particular device architecture or vendor; they only read and understand the syntax defined by the Jam file specification. In-field changes are confined to the Jam file, not the Jam Player. As a result, you do not need to modify the Jam Player source code for each in-field upgrade.

There are two types of Jam Players to accommodate the two types of Jam files: an ASCII Jam STAPL Player and a Jam STAPL Byte-Code Player. Both ASCII Jam STAPL Player and Jam STAPL Byte-Code Player are coded in the C programming language for 16-bit and 32-bit processors.

For information about UFM operation during ISP, refer to *AN 100: In-System Programmability Guidelines*.

Software Support for UFM Block

The Altera Quartus II software includes sophisticated tools that fully utilize the advantages of the UFM block in MAX V devices, while maintaining simple, easy-to-use procedures that accelerate the design process. The following section describes how the ALTUFM megafunction supports a simple design methodology for instantiating standard interface protocols for the UFM block, such as:

■ I²C

■ SPI

■ Parallel

■ None (Altera Serial Interface)

This section includes the megafunction symbol, the input and output ports, and a description of the MegaWizard Plug-In Manager options. Refer to Quartus II Help for the ALTUFM megafunction Altera Hardware Description Language (AHDL) functional prototypes (applicable to Verilog HDL), VHDL component declarations, and parameter descriptions. You can access this megafunction from the Memory Compiler directory on page 2a of the MegaWizard Plug-In Manager.

The ALTUFM MegaWizard Plug-In Manager has separate pages that apply to the MAX V UFM block. During compilation, the Quartus II Compiler verifies the ALTUFM parameters selected against the available logic array interface options, and any specific assignments.

Inter-Integrated Circuit

Inter-Integrated Circuit (I²C) is a bidirectional two-wire interface protocol, requiring only two bus lines: a serial data/address line (SDA), and a serial clock line (SCL). Each device connected to the I²C bus is software addressable by a unique address. The I²C bus is a multi-master bus where more than one integrated circuit (IC) capable of initiating a data transfer can be connected to it, which allows masters to function as transmitters or receivers.

The ALTUFM_I2C megafunction features a serial, 8-bit bidirectional data transfer up to 100 Kbits per second. With the ALTUFM_I2C megafunction, the MAX V UFM and logic can be configured as a slave device for the I²C bus. The ALTUFM megafunction's I²C interface is designed to function similar to I²C serial EEPROMs.

The Quartus II software supports four different memory sizes:

■ (128 × 8) 1 Kbits

■ (256 × 8) 2 Kbits

■ (512 × 8) 4 Kbits

■ (1,024 × 8) 8 Kbits

I²C Protocol

The following defines the characteristics of the I²C bus protocol:

■ Only two bus lines are required: SDA and SCL. Both SDA and SCL are bidirectional lines that remain high when the bus is free.

■ Data transfer can be initiated only when the bus is free.

■ The data on the SDA line must be stable during the high period of the clock. The high or low state of the data line can only change when the clock signal on the SCL line is low.

■ Any transition on the SDA line while the SCL is high indicates a start or stop condition.

Table 7–5 lists the ALTUFM_I2C megafunction input and output interface signals.

Table 7–5. ALTUFM_I2C Interface Signals

Pin	Description	Function
SDA	Serial Data/Address Line	The bidirectional SDA port is used to transmit and receive serial data from the UFM. The output stage of the SDA port is configured as an open drain pin to perform the wired-AND function.
SCL	Serial Clock Line	The bidirectional SCL port is used to synchronize the serial data transfer to and from the UFM. The output stage of the SCL port is configured as an open drain pin to perform a wired-AND function.
WP	Write Protect	Optional active high signal that disables the erase and write function for read/write mode. The ALTUFM_I2C megafunction gives you an option to protect the entire UFM memory or only the upper half of memory.
A_2, A_1, A_0	Slave Address Input	These inputs set the UFM slave address. The A_6, A_5, A_4, A_3 slave address bits are programmable, set internally to 1010 by default.

START and STOP Condition

The master always generates start (S) and stop (P) conditions. After the start condition, the bus is considered busy. Only a stop (P) condition frees the bus. The bus stays busy if the repeated start (Sr) condition is executed instead of a stop condition. In this occurrence, the start (S) and repeated start (Sr) conditions are functionally identical.

A high-to-low transition on the SDA line while the SCL is high indicates a start condition. A low-to-high transition on the SDA line while the SCL is high indicates a stop condition. Figure 7–9 shows the start and stop conditions.

**Figure 7–9. Start and Stop Conditions**
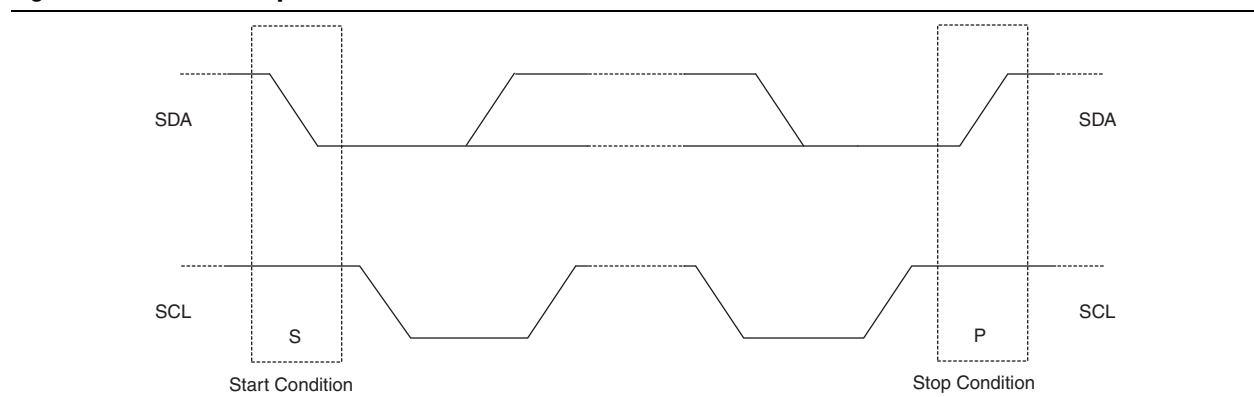
Acknowledge

Acknowledged data transfer is a requirement of I2C. The master must generate a clock pulse to signify the acknowledge bit. The transmitter releases the SDA line (high) during the acknowledge clock pulse.

The receiver (slave) must pull the SDA line low during the acknowledge clock pulse so that SDA remains a stable low during the clock high period, indicating positive acknowledgement from the receiver. If the receiver pulls the SDA line high during the acknowledge clock pulse, the receiver sends a not-acknowledge condition indicating that it is unable to process the last byte of data. If the receiver is busy (for example, executing an internally-timed erase or write operation), it will not acknowledge any new data transfer. Figure 7–10 shows the acknowledge condition on the I2C bus.

Figure 7–10. Acknowledge on the I2C Bus
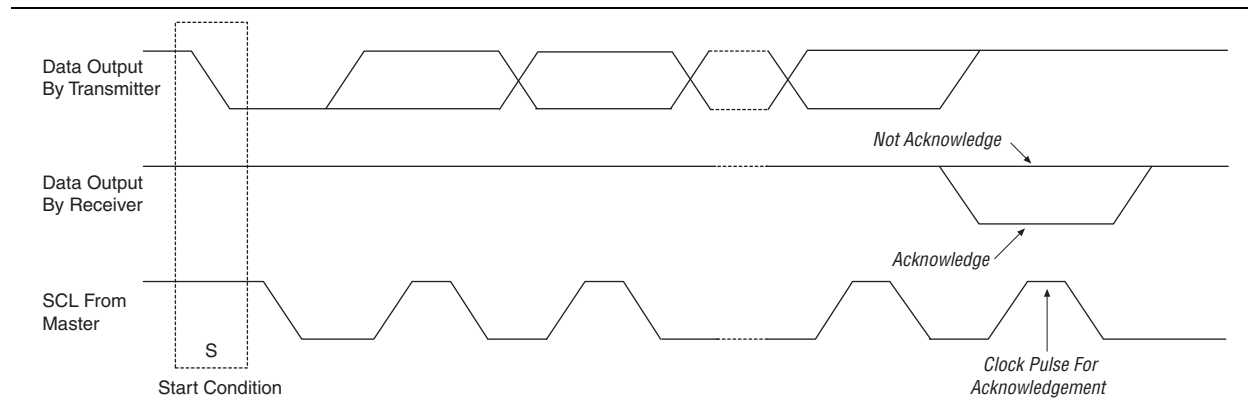
Device Addressing

After the start condition, the master sends the address of the particular slave device it is requesting. The four most significant bits (MSBs) of the 8-bit slave address are usually fixed while the next three significant bits (A_2, A_1, A_0) are device address bits that define which device the master is accessing. The last bit of the slave address specifies whether a read or write operation is to be performed. When this bit is set to 1, a read operation is selected. When this bit is set to 0, a write operation is selected.

The four MSBs of the slave address (A_6, A_5, A_4, A_3) are programmable and can be defined on page 3 of the ALTUFM MegaWizard Plug-In Manager. The default value for these four MSBs is 1010. The next three significant bits are defined using the three A_2, A_1, A_0 input ports of the ALTUFM_I2C megafunction. You can connect these ports to input pins in the design file and connect them to switches on the board. The other option is to connect them to V_{CC} and GND primitives in the design file, which conserves pins. Figure 7–11 shows the slave address bits.

Figure 7–11. Slave Address Bits
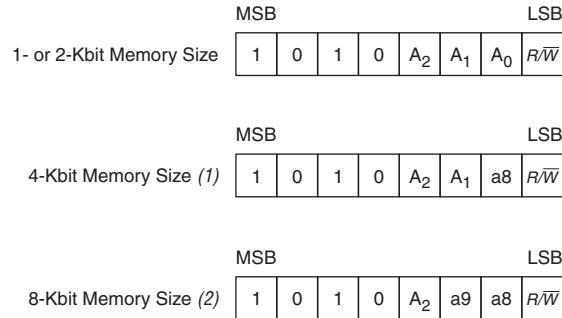
Notes to Figure 7–11:

(1) For the 4-Kbit memory size, the A_0 location in the slave address becomes the MSB (a8) of the memory byte address.

(2) For the 8-Kbit memory size, the A_0 location in the slave address becomes a8 of the memory byte address, while the A_1 location in the slave address becomes the MSB (a9) of the memory byte address.

After the master sends a start condition and the slave address byte, the ALTUFM_I2C logic monitors the bus and responds with an acknowledge (on the SDA line) when its address matches the transmitted slave address. The ALTUFM_I2C megafunction then performs a read or write operation to or from the UFM, depending on the state of the bit.

Byte Write Operation

The master initiates a transfer by generating a start condition, then sending the correct slave address (with the R/W bit set to 0) to the slave. If the slave address matches, the ALTUFM_I2C slave acknowledges on the ninth clock pulse. The master then transfers an 8-bit byte address to the UFM, which acknowledges the reception of the address. The master transfers the 8-bit data to be written to the UFM. After the ALTUFM_I2C logic acknowledges the reception of the 8-bit data, the master generates a stop condition. The internal write from the MAX V logic array to the UFM begins only after the master generates a stop condition. While the UFM internal write cycle is in progress, the ALTUFM_I2C logic ignores any attempt made by the master to initiate a new transfer. Figure 7–12 shows the byte write sequence.

Figure 7–12. Byte Write Sequence
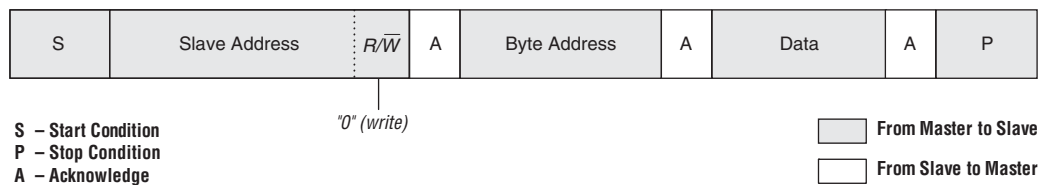
Page Write Operation

Page write operation has a similar sequence as the byte write operation, except that several bytes of data are transmitted in sequence before the master issues a stop condition. The internal write from the MAX V logic array to the UFM begins only after the master generates a stop condition. While the UFM internal write cycle is in progress, the ALTUFM_I2C logic ignores any attempt made by the master to initiate a new transfer. The ALTUFM_I2C megafunction allows you to choose the page size of 8 bytes, 16 bytes, or 32 bytes for the page write operation.

A write operation is only possible to an erased UFM block or word location. The UFM block differs from serial EEPROMs, requiring an erase operation before writing new data in the UFM block. A special erase sequence is required, as discussed in "Erase Operation".

Acknowledge Polling

The master can detect whether the internal write cycle is completed by polling for an acknowledgement from the slave. The master can re-send the start condition together with the slave address as soon as the byte write sequence is finished. The slave does not acknowledge if the internal write cycle is still in progress. The master can repeat the acknowledge polling and proceed with the next instruction after the slave acknowledges.

Write Protection

The ALTUFM_I2C megafunction includes an optional Write Protection (WP) port available on page 4 of the ALTUFM MegaWizard Plug-In Manager. In the MegaWizard Plug-In Manager, you can choose the WP port to protect either the full or upper half memory.

When WP is set to 1, the upper half or the entire memory array (depending on the write protection level selected) is protected, and the write and erase operations are not allowed. The ALTUFM_I2C megafunction acknowledges the slave address and memory address. After the master transfers the first data byte, the ALTUFM_I2C megafunction sends a not-acknowledge condition to the master to indicate that the instruction will not execute. When WP is set to 0, the write and erase operations are allowed.

Erase Operation

Commercial serial EEPROMs automatically erase each byte of memory before writing into that particular memory location during a write operation. However, the MAX V UFM block is flash based and only supports sector erase operations. Byte erase operations are not supported. When using read/write mode, a sector or full memory erase operation is required before writing new data into any location that previously contained data. The block cannot be erased when the ALTUFM_I2C megafunction is in read-only mode.

Data can be initialized into memory for read/write and read-only modes by including a memory initialization file (**.mif**) or hexadecimal file (**.hex**) in the ALTUFM MegaWizard Plug-In Manager. This data is automatically written into the UFM during device programming by the Quartus II software or third-party programming tool.

The ALTUFM_I2C megafunction supports four different erase operation methods shown on page 4 of the ALTUFM MegaWizard Plug-In Manager:

■ Full Erase (Device Slave Address Triggered)

■ Sector Erase (Byte Address Triggered)

■ Sector Erase (A_2 Triggered)

■ No Erase

These erase options only work as described if that particular option is selected in the MegaWizard Plug-In Manager before compiling the design files and programming the device. Only one option can be selected for the ALTUFM_I2C megafunction.

Each erase option is discussed in more detail in the following sections.

Full Erase (Device Slave Address Triggered)

The full erase option uses the A_2, A_1, A_0 bits of the slave address to distinguish between an erase or read/write operation. This slave operation decoding occurs when the master transfers the slave address to the slave after generating the start condition. If the A_2, A_1, and A_0 slave address bits transmitted to the UFM slave equals 111 and the four remaining MSBs match the rest of the slave addresses, then the Full Erase operation is selected. If the A_6, A_5, A_4, A_3 A_2, A_1, and A_0 slave address bits transmitted to the UFM match its unique slave address setting, the read/write operation is selected and functions as expected. As a result, this erase option utilizes two slave addresses on the bus reserving A_6, A_5, A_4, A_3, 1, 1, 1 as the erase trigger. Both sectors of the UFM block will be erased when the Full Erase operation is executed. This operation requires acknowledge polling. The internal UFM erase function only begins after the master generates a stop condition. Figure 7–13 shows the full erase sequence triggered by using the slave address.

If the memory is write-protected (WP = 1), the slave does not acknowledge the erase trigger slave address (A_6, A_5, A_4, A_3, 1, 1, 1) sent by the master. The master should then send a stop condition to terminate the transfer. The full erase operation will not be executed.

Figure 7–13. Full Erase Sequence Triggered Using the Slave Address
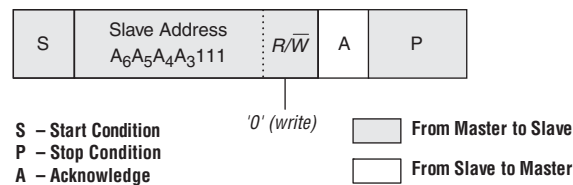
Sector Erase (Byte Address Triggered)

This sector erase operation is triggered by defining a 7- to 10-bit byte address for each sector depending on the memory size. The trigger address for each sector is entered on page 4 of the ALTUFM MegaWizard Plug-In Manager. When a write operation is executed targeting this special byte address location, the UFM sector that contains that byte address location is erased. This sector erase operation is automatically followed by a write of the intended write byte to that address. The default byte address location for UFM Sector 0 erase is address 0×00. The default byte address location for UFM Sector 1 erase is [(selected memory size)/2]. You can specify another byte location as the trigger-erase addresses for each sector.

This sector erase operation supports up to eight UFM blocks or serial EEPROMs on the I²C bus. This sector erase operation requires acknowledge polling.

Sector Erase (A$_2$ Triggered)

This sector erase operation uses the received A$_2$ slave address bit to distinguish between an erase or read/write operation. This slave operation decoding occurs when the master transmits the slave address after generating the start condition. If the A$_2$ bit received by the UFM slave is 1, the sector erase operation is selected. If the A$_2$ bit received is 0, the read/write operation is selected. While this reserves the A$_2$ bit as an erase or read/write operation bit, the A$_0$ and A$_1$ bits still act as slave address bits to address the UFM. With this erase option, there can be up to four UFM slaves cascaded on the bus for 1-Kbit and 2-Kbit memory sizes. Only two UFM slaves can be cascaded on the bus for 4-Kbit memory size, because A$_0$ of the slave address becomes the ninth bit (MSB) of the byte address. After the slave acknowledges the slave address and its erase or read/write operation bit, the master can transfer any byte address within the sector that must be erased. The internal UFM sector erase operation only begins after the master generates a stop condition. Figure 7–14 shows the sector erase sequence using the A$_2$ bit of the slave address.

Figure 7–14. Sector Erase Sequence Indicated Using the A$_2$ Bit of the Slave Address

S – Start Condition
P – Stop Condition
A – Acknowledge

'0' (write) (1)

From Master to Slave
From Slave to Master
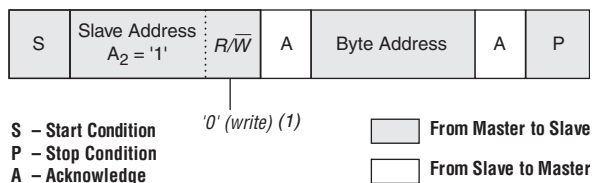
Note to Figure 7–14:

(1) A$_2$ = 0 indicates a read/write operation is executed in place of an erase. Here, the R/W bit determines whether it is a read or write operation.

If the ALTUFM_I2C megafunction is write-protected (WP=1), the slave does not acknowledge the byte address (that indicates the UFM sector to be erased) sent in by the master. The master should then send a stop condition to terminate the transfer and the sector erase operation will not be executed.

No Erase

The no erase operation never erases the UFM contents. This method is recommended when UFM does not require constant re-writing after its initial write of data. For example, if the UFM data is to be initialized with data during manufacturing using I^2C, you may not require writing to the UFM again. In that case, you should use the no erase option and save LE resources from being used to create erase logic.

Read Operation

The read operation is initiated in the same manner as the write operation except that the R/W bit must be set to 1. Three different read operations are supported:
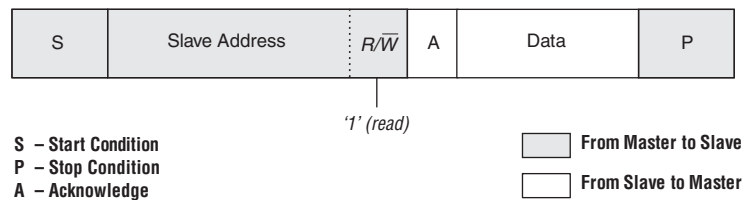
■ Current Address Read (Single Byte)

■ Random Address Read (Single byte)

■ Sequential Read (Multi-Byte)

After each UFM data has been read and transferred to the master, the UFM address register is incremented for all single and multi-byte read operations.

Current Address Read

This read operation targets the current byte location pointed to by the UFM address register. Figure 7–15 shows the current address read sequence.
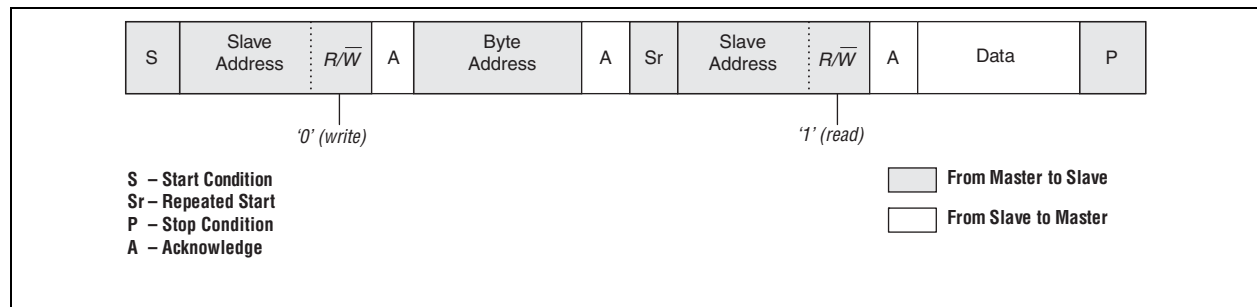
Figure 7–15. Current Address Read Sequence

S	Slave Address	$R/\overline{W}$	A	Data	P

'1' (read)

S – Start Condition
P – Stop Condition
A – Acknowledge

From Master to Slave
From Slave to Master

Random Address Read

Random address read operation allows the master to select any byte location for a read operation. The master first performs a "dummy" write operation by sending the start condition, slave address, and byte address of the location it wishes to read. After the ALTUFM_I2C megafunction acknowledges the slave and byte address, the master generates a repeated start condition, the slave address, and the R/W bit is set to 1. The ALTUFM_I2C megafunction then responds with acknowledge and sends the 8-bit data requested. The master then generates a stop condition. Figure 7–16 shows the random address read sequence.

Figure 7–16. Random Address Read Sequence

Sequential Read

Sequential read operation can be initiated by either the current address read operation or the random address read operation. Instead of sending a stop condition after the slave has transmitted one byte of data to the master, the master acknowledges that byte and sends additional clock pulses (on the SCL line) for the slave to transmit data bytes from consecutive byte addresses. The operation is terminated when the master generates a stop condition instead of responding with an acknowledge. Figure 7–17 shows the sequential read sequence.

Figure 7–17. Sequential Read Sequence
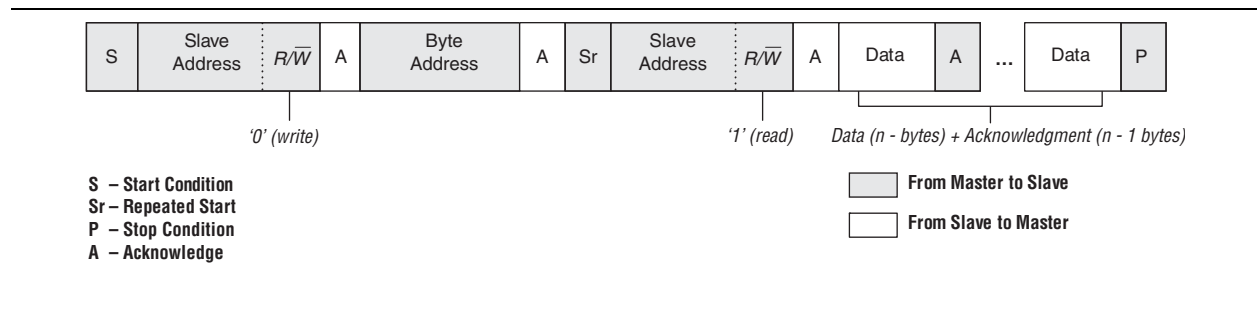
ALTUFM_I2C Interface Timing Specification

Figure 7–18 shows the timing waveform for the ALTUFM_I2C megafunction read/write mode.

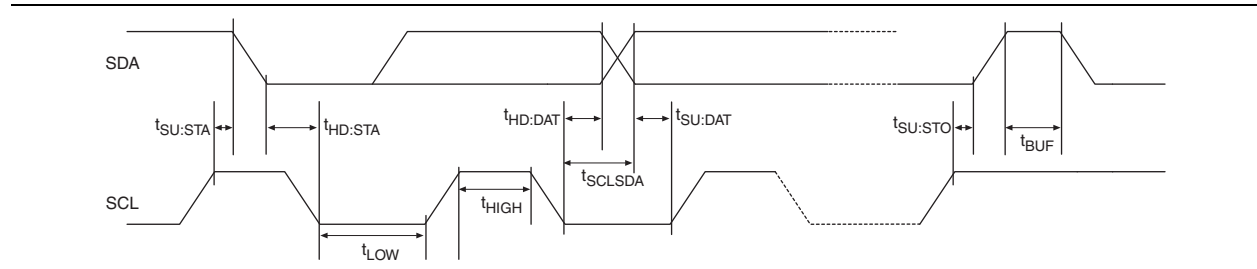**Figure 7–18. Timing Waveform for the ALTUFM_I2C Megafunction**

Table 7–6 through Table 7–8 list the timing specification needed for the ALTUFM_I2C megafunction read/write mode.

Table 7–6. I2C Interface Timing Specification

Symbol	Parameter	Min	Max	Unit
F_{SCL}	SCL clock frequency	—	100	kHz
$t_{SCL:SDA}$	SCL going low to SDA data out	—	15	ns
t_{BUF}	Bus free time between a stop and start condition	4.7	—	µs
$t_{HD:STA}$	(Repeated) start condition hold time	4	—	µs
$t_{SU:STA}$	(Repeated) start condition setup time	4.7	—	µs
t_{LOW}	SCL clock low period	4.7	—	µs
t_{HIGH}	SCL clock high period	4	—	µs
$t_{HD:DAT}$	SDA data in hold time	0	—	ns
$t_{SU:DAT}$	SDA data in setup time	20	—	ns
$t_{SU:STO}$	STOP condition setup time	4	—	ns

Table 7–7. UFM Write Cycle Time

Parameter	Min	Max	Unit
Write Cycle Time	—	110	µs

Table 7–8. UFM Erase Cycle Time

Parameter	Min	Max	Unit
Sector Erase Cycle Time	—	501	ms
Full Erase Cycle Time	—	1,002	ms

Instantiating the I²C Interface Using the Quartus II ALTUFM_I2C Megafunction

Figure 7–19 shows the ALTUFM_I2C megafunction symbol for a I²C interface instantiation in the Quartus II software.

Figure 7–19. ALTUFM_I2C Megafunction Symbol for the I²C Interface Instantiation in the Quartus II Software
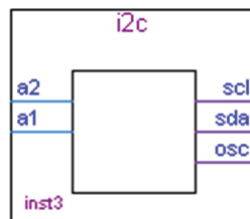
ALTUFM_I2C megafunction is under the **Memory Compiler** folder on page 2a of the MegaWizard Plug-In Manager. On page 3, you can choose whether to implement the **Read/Write** or **Read Only** mode for the UFM. You also have an option to choose the memory size for the ALTUFM_I2C megafunction as well as defining the four MSBs of the slave address (default 1010).

You can select the optional write protection and erase operation methods on page 4 of the ALTUFM MegaWizard Plug-In Manager.

☞ The UFM block's internal oscillator is always running when the ALTUFM_I2C megafunction is instantiated for both read-only and read/write interfaces.

Serial Peripheral Interface

Serial peripheral interface (SPI) is a four-pin serial communication subsystem included on the Motorola 6805 and 68HC11 series microcontrollers. It allows the microcontroller unit to communicate with peripheral devices, and is also 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 with the slave devices responding to the service request.

With the ALTUFM megafunction, the UFM and MAX V logic can be configured as a slave device for the SPI bus. The OSC_ENA is always asserted to enable the internal oscillator when the SPI megafunction is instantiated for both read only and read/write interfaces.

The Quartus II software supports both the Base mode (uses 8-bit address and data) and the Extended mode (uses 16-bit address and data). Base mode uses only UFM sector 0 (2,048 bits), while Extended mode uses both UFM sector 0 and sector 1 (8,192 bits). There are only four pins in SPI: SI, SO, SCK, and nCS. Table 7–9 describes the SPI pins and functions.

Table 7–9. SPI Interface Signals

Pin	Description	Function
SI	Serial Data Input	Receive data serially.
SO	Serial Data Output	Transmit data serially.
SCK	Serial Data Clock	The clock signal produced from the master device to synchronize the data transfer.
nCS	Chip Select	Active low signal that enables the slave device to receive or transfer data from the master device.

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 circuit diagram is shown in Figure 7–20.

Figure 7–20. Circuit Diagram for SPI Interface Read or Write Operations
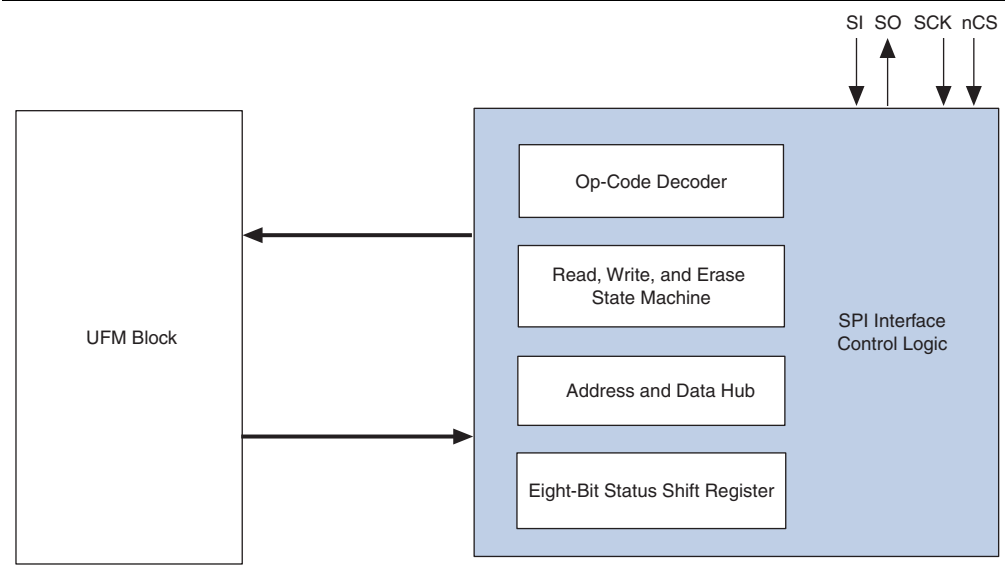
Opcodes

Table 7–10 lists the 8-bit instruction opcodes. After nCS is pulled low, the indicated opcode must be provided. Otherwise, the interface assumes that the master device has internal logic errors and ignores the rest of the incoming signals. When nCS is pulled back to high, the interface is back to normal. nCS should be pulled low again for a new service request.

Table 7–10. Instruction Set for SPI

Name	Opcode	Operation
WREN	00000110	Enable Write to UFM
WRDI	00000100	Disable Write to UFM
RDSR	00000101	Read Status Register
WRSR	00000001	Write Status Register
READ	00000011	Read data from UFM
WRITE	00000010	Write data to UFM
SECTOR-ERASE	00100000	Sector erase
UFM-ERASE	01100000	Erase the entire UFM block (both sectors)

The READ and WRITE opcodes are instructions for transmission, which means the data will be read from or written to the UFM.

WREN, WRDI, RDSR, and WRSR are instructions for the status register, where they do not have any direct interaction with UFM, but read or set the status register within the interface logic. The status register provides status on whether the UFM block is available for any READ or WRITE operation, whether the interface is WRITE enabled, and the state of the UFM WRITE protection. Table 7–11 lists the status register format. For the read only implementation of ALTUFM SPI (Base or Extended mode), the status register does not exist, saving LE resources.

Table 7–11. Status Register Format

Position	Status	Default at Power-Up	Description
Bit 7	X	0	—
Bit 6	X	0	—
Bit 5	X	0	—
Bit 4	X	0	—
Bit 3	BP1	0	Indicate the current level of block write protection *(1)*
Bit 2	BP0	0	Indicate the current level of block write protection *(1)*
Bit 1	WEN	0	1= SPI WRITE enabled state 0= SPI WRITE disabled state
Bit 0	nRDY	0	1 = Busy, UFM WRITE or ERASE cycle in progress 0 = No UFM WRITE or ERASE cycle in progress

Note to Table 7–11:

(1) For more information about status register bits BP1 and BP0, refer to Table 7–12 and Table 7–13 on page 7–34 .

The following sections describe the instructions for SPI.

READ

READ is the instruction for data transmission, where the data is read from the UFM block. When data transfer is taking place, the MSB is always the first bit to be transmitted or received. The data output stream is continuous through all addresses until it is terminated by a low-to-high transition at the nCS port. The READ operation is always performed through the following sequence in SPI, as shown in Figure 7–21:

1. nCS is pulled low to indicate the start of transmission.

2. An 8-bit READ opcode (00000011) is received from the master device. (If internal programming is in progress, READ is ignored and not accepted).

3. A 16-bit address is received from the master device. The LSB of the address is received last. Because the UFM block can take only nine bits of address maximum, the first seven address bits received are discarded.

4. Data is transmitted for as many words as needed by the slave device through SO for READ operation. When the end of the UFM storage array is reached, the address counter rolls over to the start of the UFM to continue the READ operation.

5. nCS is pulled back to high to indicate the end of transmission.

For SPI Base mode, the READ operation is always performed through the following sequence in SPI:

1. nCS is pulled low to indicate the start of transmission.

2. An 8-bit READ opcode (00000011) is received from the master device, followed by an 8-bit address. If internal programming is in progress, the READ operation is ignored and not accepted.

3. Data is transmitted for as many words as needed by the slave device through SO for READ operation. The internal address pointer automatically increments until the highest memory address is reached (address 255 only because the UFM sector 0 is used). The address counter will not roll over when address 255 is reached. The SO output is set to high-impedance (Z) when all eight data bits from address 255 have been shifted out through the SO port.

4. nCS is pulled back to high to indicate the end of transmission.

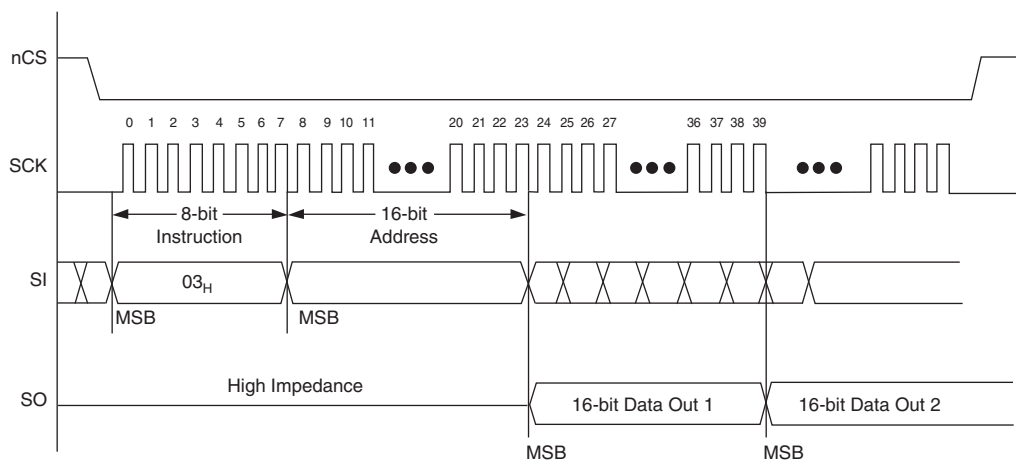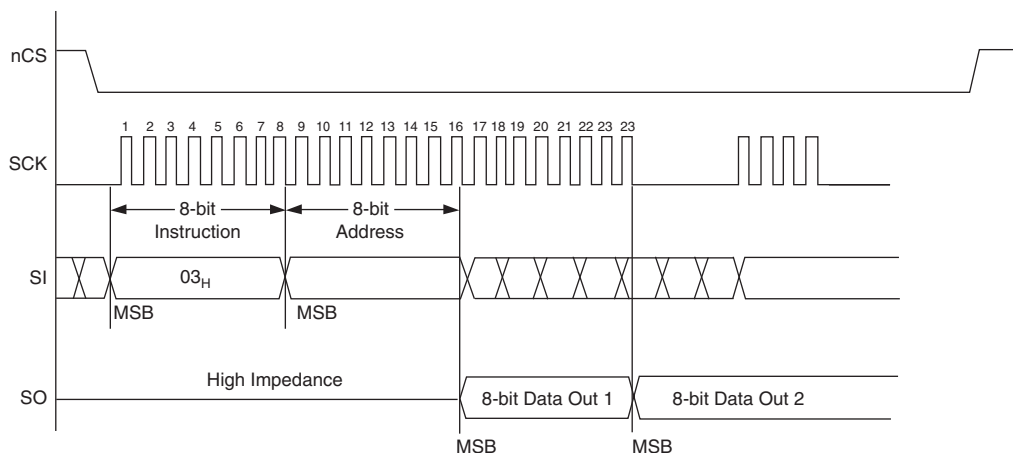**Figure 7–21. READ Operation Sequence for Extended Mode**

Figure 7–22 shows the READ operation sequence for Base mode.

Figure 7–22. READ Operation for Base Mode

WRITE

WRITE is the instruction for data transmission, where the data is written to the UFM block. The targeted location in the UFM block that will be written must be in the erased state (FFFFH) before initiating a WRITE operation. When data transfer is taking place, the MSB is always the first bit to be transmitted or received. nCS must be driven high before the instruction is executed internally. You may poll the nRDY bit in the software status register for the completion of the internal self-timed WRITE cycle. For SPI Extended mode, the WRITE operation is always done through the following sequence, as shown in Figure 7–23:

1. nCS is pulled low to indicate the start of transmission.

2. An 8-bit WRITE opcode (00000010) is received from the master device. If internal programming is in progress, the WRITE operation is ignored and not accepted.

3. A 16-bit address is received from the master device. The LSB of the address will be received last. Because the UFM block can take only nine bits of address maximum, the first seven address bits received are discarded.

4. A check is carried out on the status register (see Table 7–11) to determine if the WRITE operation has been enabled, and the address is outside of the protected region; otherwise, Step 5 is bypassed.

5. One word (16 bits) of data is transmitted to the slave device through SI.

6. nCS is pulled back to high to indicate the end of transmission.

For SPI Base mode, the WRITE operation is always performed through the following sequence in SPI:

1. nCS is pulled low to indicate the start of transmission.

2. An 8-bit WRITE opcode (00000010) is received. If the internal programming is in progress, the WRITE operation is ignored and not accepted.

3. An 8-bit address is received. A check is carried out on the status register (see Table 7–11) to determine if the WRITE operation has been enabled, and the address is outside of the protected region; otherwise, Step 4 is skipped.

4. An 8-bit data is transmitted through SI.

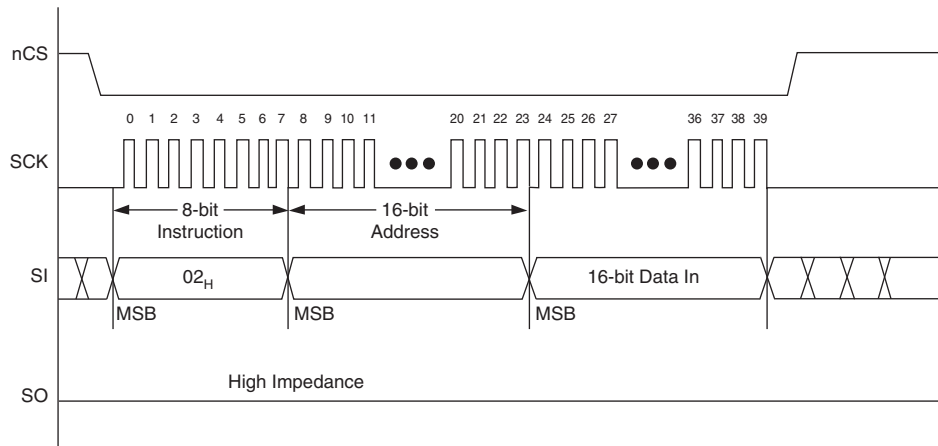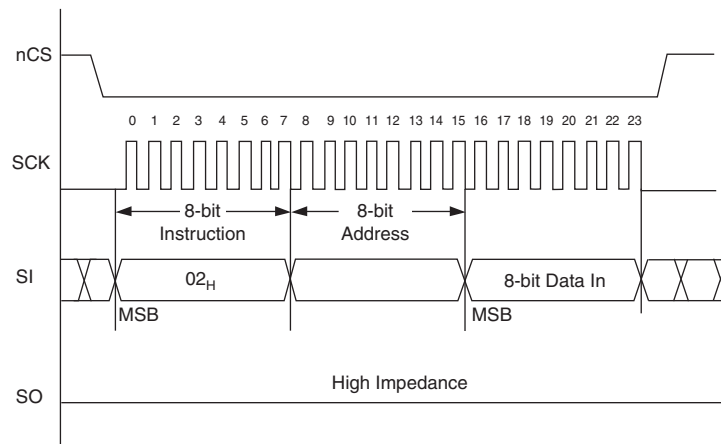5. nCS is pulled back to high to indicate the end of transmission.

Figure 7–23. WRITE Operation Sequence for Extended Mode

Figure 7–24 shows the WRITE operation sequence for Base mode.

Figure 7–24. WRITE Operation Sequence for Base Mode

SECTOR-ERASE

SECTOR-ERASE (SE) is the instruction of erasing one sector of the UFM block. Each sector contains 256 words. WEN bit and the sector must not be protected for SE operation to be successful. nCS must be driven high before the instruction is executed internally. You may poll the nRDY bit in the software status register for the completion of the internal self-timed SECTOR-ERASE cycle. For SPI Extended mode, the SE operation is performed in the following sequence, as shown in Figure 7–25:

1. nCS is pulled low.

2. Opcode 00100000 is transmitted into the interface.

3. The 16-bit address is sent. The eighth bit (the first seven bits will be discarded) of the address indicates which sector is erased; a 0 means sector 0 (UFM0) is erased, and a 1 means sector 1 (UFM1) is erased.

4. nCS is pulled back to high.

For SPI Base mode, the SE instruction erases UFM sector 0. Because there are no choices of UFM sectors to be erased, there is no address component to this instruction. The SE operation is always done through the following sequence in SPI Base mode:

1. nCS is pulled low.

2. Opcode 00100000 is transmitted into the interface.

3. nCS is pulled back to high.

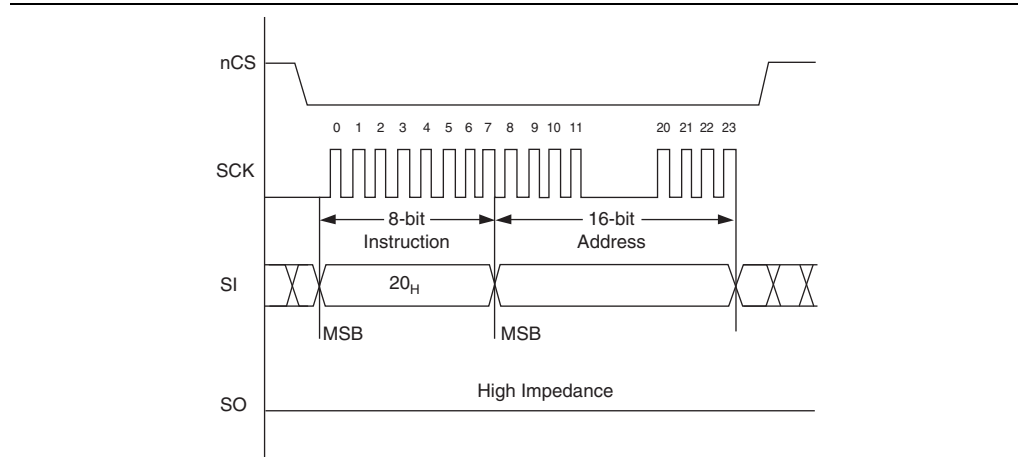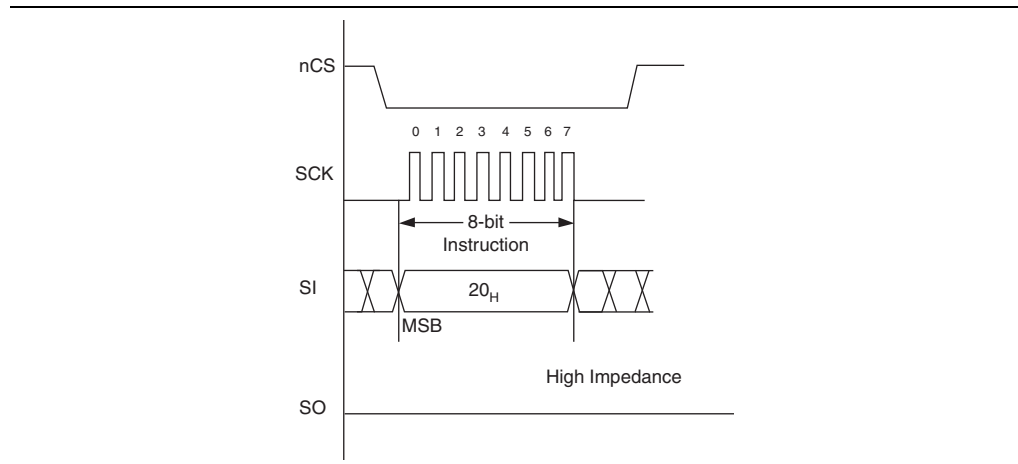**Figure 7–25. SECTOR-ERASE Operation Sequence for Extended Mode**

Figure 7–26 shows the SECTOR-ERASE operation sequence for Base mode.

**Figure 7–26. SECTOR_ERASE Operation Sequence for Base Mode**

UFM-ERASE

The UFM-ERASE (CE) instruction erases both UFM sector 0 and sector 1 for SPI Extended Mode. While for SPI Base mode, the CE instruction has the same functionality as the SECTOR-ERASE (SE) instruction, which erases UFM sector 0 only. WEN bit and the UFM sectors must not be protected for CE operation to be successful. nCS must be driven high before the instruction is executed internally. You may poll the nRDY bit in the software status register for the completion of the internal self-timed CE cycle. For both SPI Extended mode and Base mode, the CE operation is performed in the following sequence as shown in Figure 7–27:

1. nCS is pulled low.

2. Opcode 01100000 is transmitted into the interface.

3. nCS is pulled back to high.

Figure 7–27 shows the UFM-ERASE operation sequence.

**Figure 7–27. UFM-ERASE Operation Sequence**
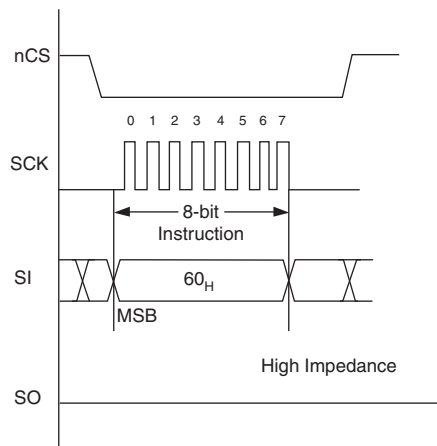
WREN (Write Enable)

The interface is powered-up in the write disable state. Therefore, WEN in the status register (refer to Table 7–11) is 0 at power-up. Before any write is allowed to take place, WREN must be issued to set WEN in the status register to 1. If the interface is in read-only mode, WREN does not have any effect on WEN, because the status register does not exist. After WEN is set to 1, it can be reset by the WRDI instruction; the WRITE and SECTOR-ERASE instructions will not reset the WEN bit. WREN is issued through the following sequence, as shown in Figure 7–28:

1. nCS is pulled low.

2. Opcode 00000110 is transmitted into the interface to set WEN to 1 in the status register.

3. After the transmission of the eighth bit of WREN, the interface is in wait state (waiting for nCS to be pulled back to high). Any transmission after this is ignored.

4. nCS is pulled back to high.

Figure 7–28. WREN Operation Sequence
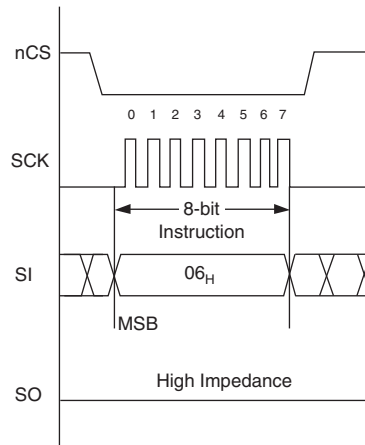
WRDI (Write Disable)

After the UFM is programmed, WRDI can be issued to set WEN back to 0, disabling WRITE and preventing inadvertent writing to the UFM. WRDI is issued through the following sequence, as shown in Figure 7–29:

1. nCS is pulled low.

2. Opcode 00000100 is transmitted to set WEN to 0 in the status register.

3. After the transmission of the eighth bit of WRDI, the interface is in wait state (waiting for nCS to be pulled back to high). Any transmission after this is ignored.
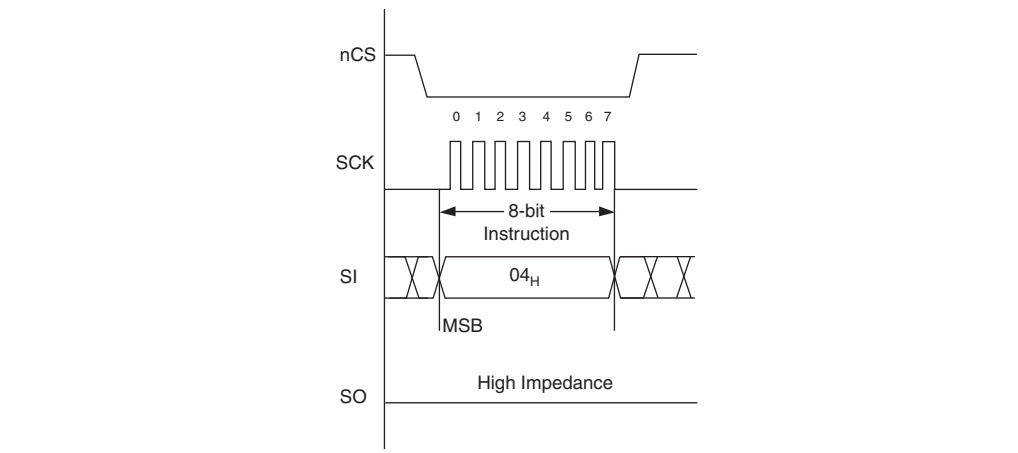
4. nCS is pulled back to high.

Figure 7–29. WRDI Operation Sequence

RDSR (Read Status Register)

The content of the status register can be read by issuing RDSR. After RDSR is received, the interface outputs the content of the status register through the SO port. Although the four most significant bits (Bit 7 to Bit 4) do not hold valuable information, all eight bits in the status register will output through the SO port. This allows future compatibility when Bit 7 to Bit 4 have new meaning in the status register. During the internal program cycle in the UFM, RDSR is the only valid opcode recognized by the interface (therefore, the status register can be read at any time), and nRDY is the only valid status bit. Other status bits are frozen and remain unchanged until the internal program cycle is ended. RDSR is issued through the following sequence, as shown in Figure 7–30:

1. nCS is pulled low.

2. Opcode 00000101 is transmitted into the interface.

3. SI ignores incoming signals; SO outputs the content of the status register, Bit 7 first and Bit 0 last.

4. If nCS is kept low, repeat step 3.

5. nCS is pulled back to high to terminate the transmission.

Figure 7–30. RDSR Operation Sequence
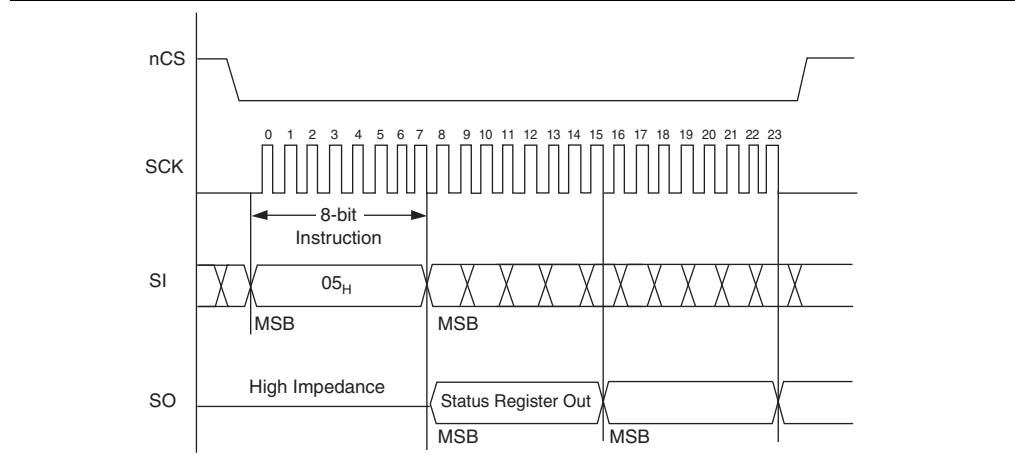
WRSR (Write Status Register)

The block protection bits (BP1 and BP0) are the status bits used to protect certain sections of the UFM from inadvertent write. The BP1 and BP0 status are updated by WRSR. During WRSR, only BP1 and BP0 in the status register can be written with valid information. The rest of the bits in the status register are ignored and not updated. When both BP1 and BP0 are 0, there is no protection for the UFM. When both BP1 and BP0 are 1, there is full protection for the UFM. BP0 and BP1 are set to 0 upon power-up. Table 7–12 lists the Block Write Protect Bits for Extended mode, while Table 7–13 lists the Block Write Protect Bits for Base mode. WRSR is issued through the following sequence, as shown in Figure 7–31:

1. nCS is pulled low.

2. Opcode 00000001 is transmitted into the interface.

3. An 8-bit status is transmitted into the interface to update BP1 and BP0 of the status register.

4. If nCS is pulled high too early (before all the eight bits in Step 2 or Step 3 are transmitted) or too late (the ninth bit or more is transmitted), WRSR is not executed.

5. nCS is pulled back to high to terminate the transmission.

Figure 7–31. WRSR Operation Sequence
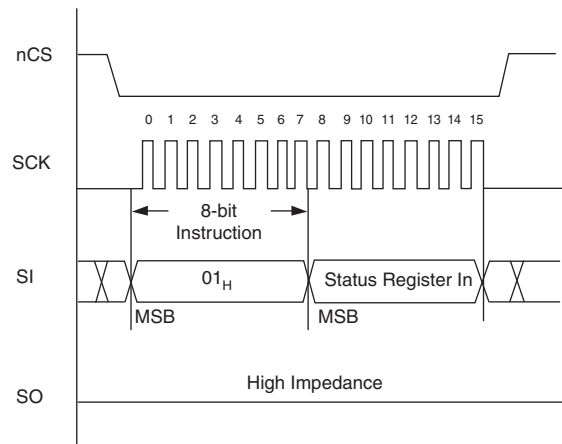
Table 7–12. Block Write Protect Bits for Extended Mode

Level	Status Register Bits		UFM Array Address Protected
	BP1	BP0	
0 (No protection)	0	0	None
3 (Full protection)	1	1	000 to 1FF

Table 7–13. Block Write Protect Bits for Base Mode

Level	Status Register Bits		UFM Array Address Protected
	BP1	BP0	
0 (No protection)	0	0	None
3 (Full protection)	1	1	000 to 0FF

ALTUFM SPI Timing Specification

Figure 7–32 shows the timing specification needed for the SPI Extended mode (read/write). These nCS timing specifications do not apply to the SPI Extended read-only mode nor to any of the SPI Base modes. However, for the SPI Extended mode (read only) and the SPI Base mode (both read only and read/write), the nCS signal and SCK are not allowed to toggle at the same time. Table 7–14 lists the timing parameters that only apply to the SPI Extended mode (read/write).
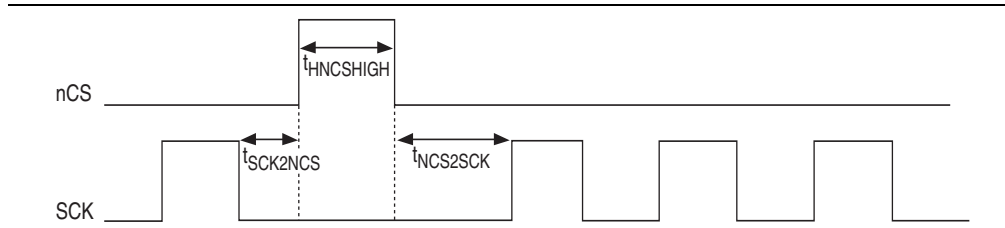
Figure 7–32. SPI Timing Waveform

Table 7–14. SPI Timing Parameters for Extended Mode

Symbol	Description	Minimum (ns)	Maximum (ns)
$t_{SCK2NCS}$	The time required for the SCK signal falling edge to nCS signal rising edge	50	—
$t_{HNCSHIGH}$	The time that the nCS signal must be held high	600	—
$t_{NCS2SCK}$	The time required for the nCS signal falling edge to SCK signal rising edge	750	—

Instantiating SPI Using Quartus II ALTUFM_SPI Megafunction

Figure 7–33 shows the ALTUFM_SPI megafunction symbol for SPI instantiation in the Quartus II software.

Figure 7–33. ALTUFM_SPI Megafunction Symbol for SPI Instantiation
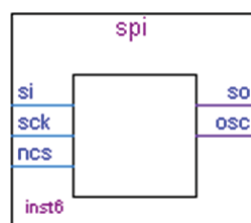
ALTUFM_SPI megafunction is under the **Memory Compiler** folder on page 2a of the MegaWizard Plug-In Manager. On page 3, you can choose whether to implement the **Read/Write** or **Read Only** mode as the access mode for the UFM. You can also select the configuration mode (Base or Extended) for SPI on this page. You can specify the initial content of the UFM block on page of the ALTUFM MegaWizard Plug-In Manager as discussed in "Creating Memory Content File" on page 7–39.

☞ The UFM block's internal oscillator is always running when the ALTUFM_SPI megafunction is instantiated for read/write interface. The UFM block's internal oscillator is disabled when the ALTUFM_SPI megafunction is instantiated for read only interface.

Parallel Interface

This interface allows for parallel communication between the UFM block and outside logic. After the READ request, WRITE request, or ERASE request is asserted (active low assertion), the outside logic or device (such as a microcontroller) can continue its 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. For both the read only and the read/write modes of the parallel interface, OSC_ENA is always asserted, enabling the internal oscillator. Table 7–15 lists the parallel interface pins and functions.

Table 7–15. Parallel Interface Signals

Pin	Description	Function
DI[15..0]	16-bit data Input	Receive 16-bit data in parallel. You can select an optional width of 3 to 16 bits using the ALTUFM megafunction.
DO[15..0]	16-bit data Output	Transmit 16-bit data in parallel. You can select an optional width of 3 to 16 bits using the ALTUFM megafunction.
ADDR[8..0]	Address Register	Operation sequence refers to the data that is pointed to by the address register. You can determine the address bus width using the ALTUFM megafunction.
nREAD	READ Instruction Signal	Initiates a read sequence.
nWRITE	WRITE Instruction Signal	Initiates a write sequence.
nERASE	ERASE Instruction Signal	Initiates a SECTOR-ERASE sequence indicated by the MSB of the ADDR[] port.
nBUSY	BUSY Signal	Driven low to notify that it is not available to respond to any further request.
DATA_VALID	Data Valid	Driven high to indicate that the data at the DO port is the valid data from the last read address for read request.

Even though the ALTUFM megafunction allows you to select the address widths range from 3 bits to 9 bits, the UFM block always expects a full 9 bits for the width of the address register. Therefore, the ALTUFM megafunction will always pad the remaining LSBs 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'. 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 megafunction, the UFM block always expects full 16 bits width for the data register. Reading from the data register always proceeds from MSB to LSB. The ALTUFM megafunction always pads the remaining LSBs of the data register with 1s if the user selects a data width of less than 16-bits.

ALTUFM Parallel Interface Timing Specification

Figure 7–34 shows the timing specifications for the parallel interface. Table 7–16 lists the parallel interface instruction signals. The nREAD, nWRITE, and nERASE signals are active low signals.

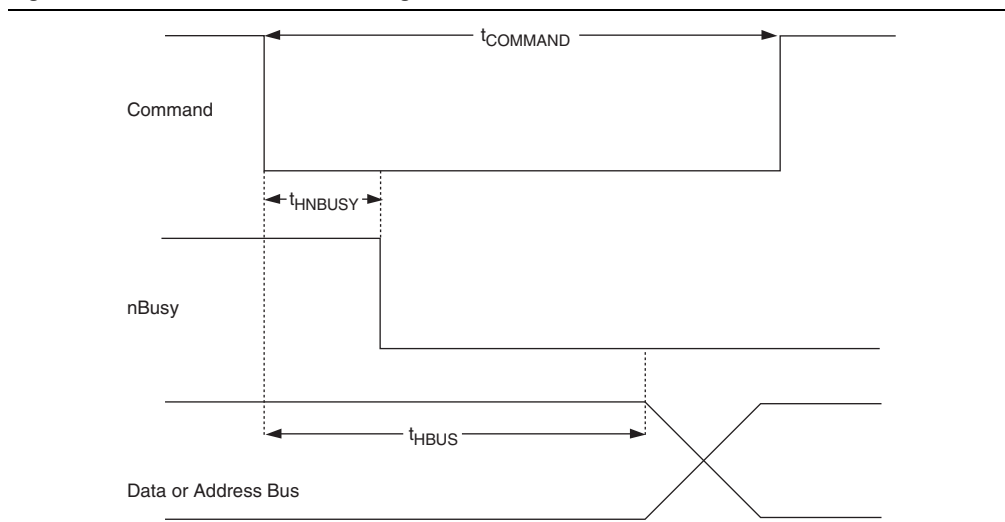**Figure 7–34. Parallel Interface Timing Waveform**

Table 7–16. Parallel Interface Timing Parameters

Symbol	Description	Minimum (ns)	Maximum (ns)
$t_{COMMAND}$	The time required for the command signal (nREAD/nWRITE/nERASE) to be asserted and held low to initiate a read/write/erase sequence	600	3,000
t_{HNBUSY}	Maximum delay between command signal's falling edge to the nBUSY signal's falling edge	—	300
t_{HBUS}	The time that the data and address buses must be present at the data input and address register ports after the command signal has been asserted low	600	—

Instantiating Parallel Interface Using Quartus II ALTUFM_PARALLEL Megafunction

Figure 7–35 shows the ALTUFM_PARALLEL megafunction symbol for a parallel interface instantiation in the Quartus II software.

Figure 7–35. ALTUFM_PARALLEL Megafunction Symbol for Parallel Interface Instantiation
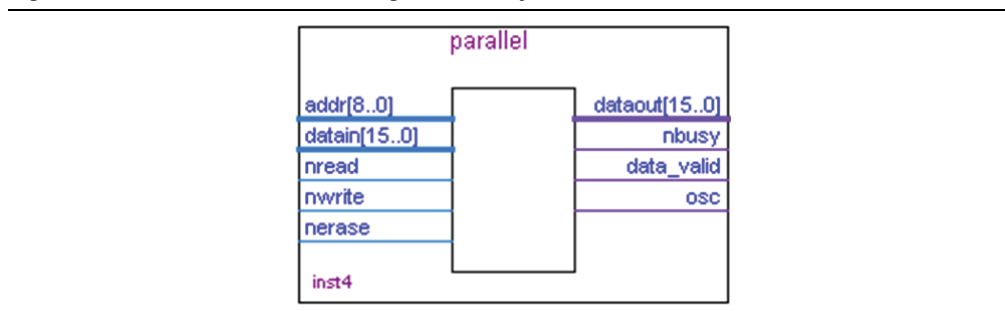
ALTUFM_PARALLEL megafunction is under the **Memory Compiler** folder on page 2a of the MegaWizard Plug-In Manager. On page 3, you can choose whether to implement the **Read/Write** or **Read Only** mode for the UFM. You also have an option to choose the width for address bus (up to 9 bits) and for the data bus (up to 16 bits). You can specify the initial content of the UFM block on page 4 of the ALTUFM MegaWizard Plug-In Manager as discussed in "Creating Memory Content File" on page 7–39.

☞ The UFM block's internal oscillator is always running when the ALTUFM_PARALLEL megafunction is instantiated for read/write interface. The UFM block's internal oscillator is disabled when the ALTUFM_PARALLEL megafunction is instantiated for a read only interface.

None (Altera Serial Interface)

Select **None** for the interface protocol to use the dedicated UFM serial interface. The built-in UFM interface uses 13 pins for the communication. The functional description of the 13 pins are described in Table 7–4 on page 7–3. You can produce your own interface design to communicate to/from the dedicated UFM interface and implement it in the logic array.

Instantiating None Using Quartus II ALTUFM_NONE Megafunction

Figure 7–36 shows the ALTUFM_NONE megafunction symbol for None instantiation in the Quartus II software.

Figure 7–36. ALTUFM_NONE Megafunction Symbol for None Instantiation

ALTUFM_NONE megafunction is under the **Memory Compiler** folder on page 2a of the MegaWizard Plug-In Manager. You can specify the initial content of the UFM block on page 3 of the ALTUFM MegaWizard Plug-In Manager as discussed in "Creating Memory Content File".

Creating Memory Content File

You can initialize the content of the UFM through a memory content file. The Quartus II software supports two types of initial memory content file format: Memory Initialization File (**.mif**) and Hexadecimal File (**.hex**). A new memory content file for the UFM block can be created by clicking **New** on the File menu. Select the **.mif** or **.hex** file in the **Other Files** tab.

After clicking **OK**, a dialog box appears. In this dialog box, the **Number of words** represents the numbers of address lines while the **Word size** represents the data width. To create a memory content file for the ALTUFM megafunction, enter 512 for the number of words and 16 for the word size.

The memory content is written into a **.hex** file. On the Tools menu, click **MegaWizard Plug-In Manager**. The memory content file (**data.hex**) is included on the respective ALTUFM MegaWizard Plug-In Manager. Click **Yes** to use this file for the memory content file. Click **Browse** to include the memory content file.

Memory Initialization for the ALTUFM_PARALLEL Megafunction

For the parallel interface, if a **.hex** file is used to initialize the memory content for the ALTUFM megafunction, you must fully specify all 16 bits in each memory address, regardless of the data width selected. If your data width is less than 16 bits wide, your data must be placed in the MSBs of the data word and the remaining LSBs must be padded with 1's.

For an example, if address_width = 3 and data_width = 8 are selected for the ALTUFM_PARALLEL megafunction, the **.hex** file should contain eight addresses of data (2^3 addresses), each word containing 16 bits. If the initial content at the location 000 is intended to be 10101010, you should specify 1010101011111111 for address 000 in the **.hex** file.

☞ This specification applies only to **.hex** files used with the parallel interface. **.mif**s do not require you to fully specify 16 bits for each data word. However, both **.mif** and **.hex** files require you to specify all addresses of data according to the address_width selected in the megafunction.

Memory Initialization for the ALTUFM_SPI Megafunction

The same 16-bit data padding mentioned for ALTUFM_PARALLEL is required for **.hex** files used with the SPI Base (8 bits) and Extended (16 bits) mode interface. In addition, for SPI Base and Extended modes, you must fully specify memory content for all 512 addresses (both sector 0 and sector 1) in the **.mif** and **.hex** files, even if sector 1 is not used. You can put valid data for SPI Base mode addresses 0 to 255 (sector 0), and initialize sector 1 to all ones.

Memory Initialization for the ALTUFM_I2C Megafunction

The MAX V UFM physical memory block contains a 16-bit wide and 512 deep (9-bit address) array. The ALTUFM_I2C megafunction uses the following smaller array sizes:

■ An 8-bit wide and 128 deep (7-bit address) mapping for 1 Kbit memory size

■ An 8-bit wide and 256 deep (8-bit address) mapping for 2 Kbits memory size

■ An 8-bit wide and 512 deep (9-bit address) mapping for 4 Kbits memory size

■ An 8-bit wide and 1,024 deep (10-bit address) mapping for 8 Kbits memory size

Altera recommends that you pad the **.mif** or **.hex** file for both address and data width to fill the physical memory map for the UFM block and ensure the **.mif** or **.hex** file represents a full 16-bit word size and a 9-bit address space.

Memory Map for 1-Kbit Memory Initialization

Figure 7–37 shows the memory map initialization for the ALTUFM_I2C megafunction of 1-Kbit memory size. The ALTUFM_I2C megafunction byte address location of 00h to 3Fh is mapped to the UFM block address location of 000h to 03Fh. The ALTUFM_I2C megafunction byte address location of 40h to 7Fh is mapped to the UFM block address location of 1C0h to 1FFh. Altera recommends that you pad the unused address locations of the UFM block with all 1s.
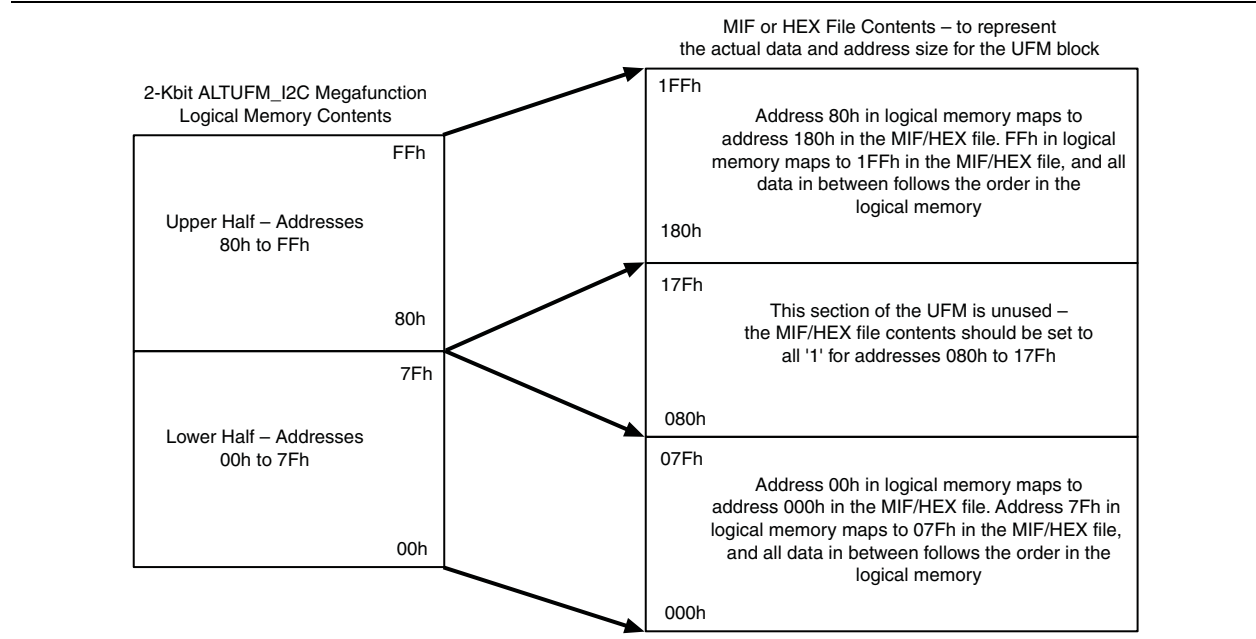
Figure 7–37. Memory Map for 1-Kbit Memory Initialization
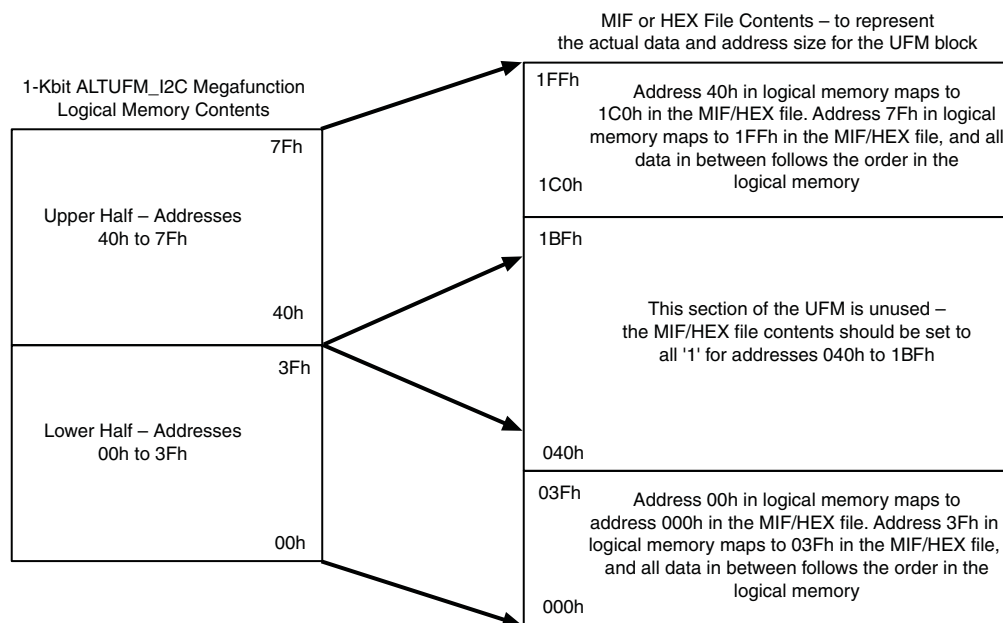
Memory Map for 2-Kbit Memory Initialization

Figure 7–38 shows the memory map initialization for the ALTUFM_I2C megafunction of 2 Kbits of memory. The ALTUFM_I2C megafunction byte address location of `00h` to `7Fh` is mapped to the UFM block address location of `000h` to `07Fh`. The ALTUFM_I2C megafunction byte address location of `80h` to `FFh` is mapped to the UFM block address location of `180h` to `1FFh`. Altera recommends that you pad the unused address location of the UFM block with all `1s`.
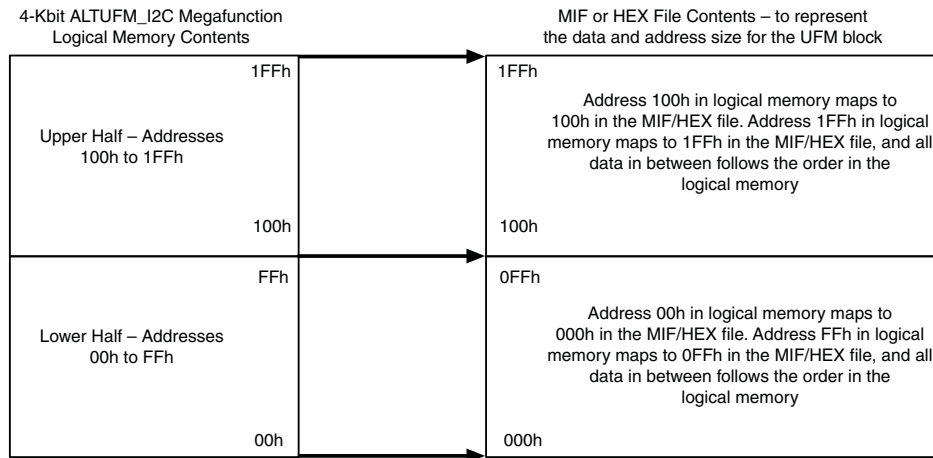
Figure 7–38. Memory Map for 2-Kbit Memory Initialization

Memory Map for 4-Kbit Memory Initialization

Figure 7–39 shows the memory map initialization for the ALTUFM_I2C megafunction of 4-Kbit memory. The ALTUFM_I2C megafunction byte address location of 00h to FFh is mapped to the UFM block address location of 000h to 0FFh. The ALTUFM_I2C megafunction byte address location of 100h to 1FFh is mapped to the UFM block address location of 100h to 1FFh.
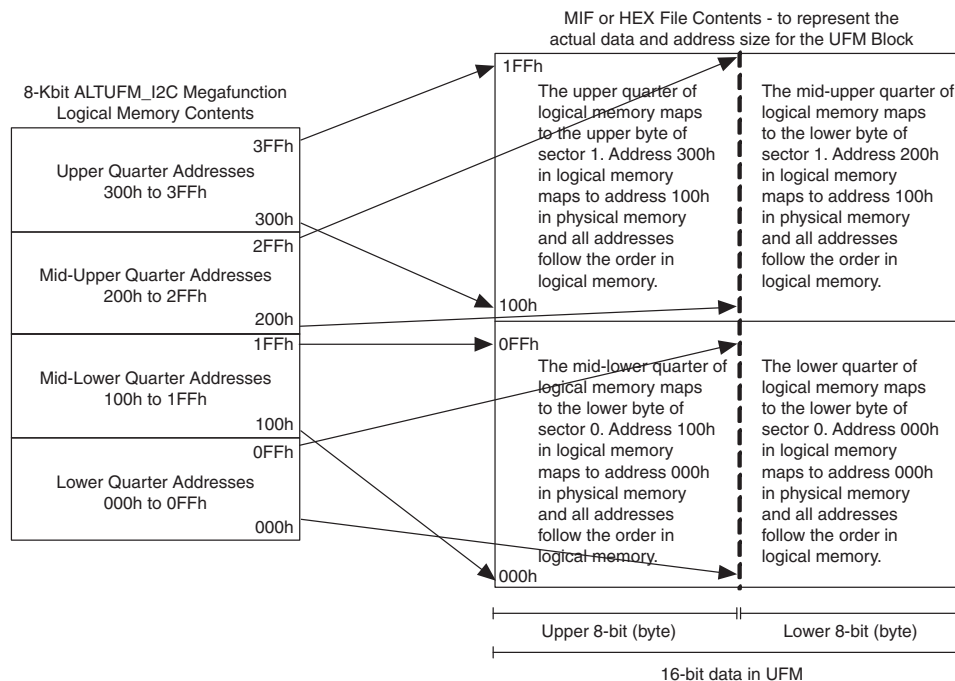
Figure 7–39. Memory Map for 4-Kbit Memory Initialization

Memory Map for 8-Kbit Memory Initialization

Figure 7–40 shows the memory map initialization for the ALTUFM_I2C megafunction of 8-Kbit memory. The ALTUFM_I2C megafunction of 8-Kbit memory fully utilizes all the memory locations in the UFM block.
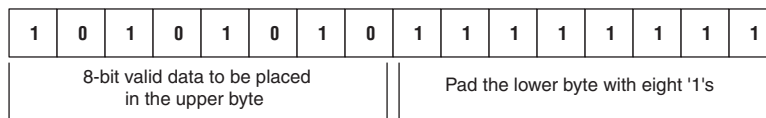
Figure 7–40. Memory Map for 8-Kbit Memory Initialization

Padding Data into Memory Map

The ALTUFM_I2C megafunction uses the upper 8 bits of the UFM 16-bit word; therefore, the 8 least significant bits should be padded with 1s, as shown in Figure 7–41.

Figure 7–41. Padding Data into Memory Map

1	0	1	0	1	0	1	0	1	1	1	1	1	1	1	1

8-bit valid data to be placed in the upper byte   Pad the lower byte with eight '1's

Simulation Parameters

In the ALTUFM megafunction, you have an option to simulate the OSC output port at the maximum or the minimum frequency during the design simulation. The frequency chosen is only used as the timing parameter for the Quartus II simulator and does not affect the real MAX V device OSC output frequency.

Document Revision History

Table 7–17 lists the revision history for this chapter.

Table 7–17. Document Revision History

Date	Version	Changes
January 2011	1.1	Updated "Oscillator" section.
December 2010	1.0	Initial release.