1. Generic Serial Flash Interface Intel® FPGA IP Core User Guide......................................................... 3
   1.1. Device Family Support.......................................................................................................................... 4
   1.2. Signals.................................................................................................................................................. 5
   1.3. Parameters.......................................................................................................................................... 6
   1.4. Register Map....................................................................................................................................... 7
   1.5. Using Generic Serial Flash Interface Intel FPGA IP............................................................................ 10
       1.5.1. Control Status Register Operations............................................................................................... 10
       1.5.2. Memory Operations..................................................................................................................... 11
       1.5.3. Byte Enabling............................................................................................................................... 12
   1.6. Generic Serial Flash Interface Intel FPGA IP Core Reference Design............................................... 12
       1.6.1. Hardware and Software Requirements........................................................................................ 13
       1.6.2. Functional Description................................................................................................................ 13
       1.6.3. Creating Nios II Hardware System............................................................................................... 15
       1.6.4. Integrating Modules into Intel Quartus Prime Project................................................................. 17
       1.6.5. Programming the .sof File.......................................................................................................... 17
       1.6.6. Building Application Software System using Nios II Software Build Tools.................................. 18
   1.7. Flash Access Using the Generic Serial Flash Interface Intel FPGA IP Core..................................... 20
       1.7.1. Flash Operations that Require Operation Code.............................................................................. 20
       1.7.2. Flash Operations to Read Flash Registers.................................................................................... 21
       1.7.3. Flash Operations to Write Flash Registers.................................................................................. 22
       1.7.4. Flash Operations that Require An Address.................................................................................. 23
       1.7.5. Read Memory from the Flash...................................................................................................... 24
       1.7.6. Program Flash.............................................................................................................................. 26
   1.8. Generic Serial Flash Interface Intel FPGA IP User Guide Archives.................................................. 27
1. Generic Serial Flash Interface Intel® FPGA IP Core User Guide

The Generic Serial Flash Interface Intel® FPGA IP core provides access to Serial Peripheral Interface (SPI) flash devices. The Generic Serial Flash Interface IP is a more efficient alternative compared to the ASMI Parallel and ASMI Parallel II Intel FPGA IP cores. The Generic Serial Flash Interface Intel FPGA IP core supports Intel configuration devices as well as flash from different vendors. Intel recommends you to use the Generic Serial Flash Interface Intel FPGA IP core for new designs.

You can use the Generic Serial Flash Interface Intel FPGA IP to write the following data to the flash device:

- Configuration memory—configuration data for Active Serial (AS) configuration scheme
- General purpose memory—application-specific data

The Generic Serial Flash Interface IP supports the following features:

- Single, dual or quad I/O mode
- Direct flash access via the Avalon® memory-mapped slave interface which allows the controller to directly execute codes from the flash
- Up to 3 multiple flash device support (Intel Arria® 10 and Intel Cyclone® 10 GX devices only)
- Generic control register for accessing flash control status registers
- Programmable clock generator with run-time baud rate change for device clock
- Programmable chip select delay
- Read data capturing logic when running with high frequency
- FPGA active serial memory interface (ASMI) block atom connection to the active serial (AS) pins or export to FPGA I/O pins

Related Information

- Generic Serial Flash Interface Intel FPGA IP Core Reference Design on page 12
- Generic Serial Flash Interface Intel FPGA IP Core Reference Design Files
- How do I enable Micron's MT25Q device support in replacement to End Of Life (EOL) EPCQ(>=256Mb) and EPCQ-L devices?
- Configuration Devices
  Provides more information on the third-party flash support.
- Using the Generic Serial Flash Interface (ODEVGSFI) Training Course

(1) The supported flash devices for configuration memory are, EPCQ, EPCQ-A, EPCQ-L, and Micron® MT25Q (256Mb to 2Gb) devices.
1.1. Device Family Support

The Generic Serial Flash Interface IP is supported in the following devices:

- Intel Agilex™
- Intel Stratix® 10
- Intel Arria 10
- Intel Cyclone 10 GX
- Intel Cyclone 10 LP
- Intel MAX® 10 (For general purpose memory only)
- Stratix V
- Arria V
- Cyclone V
- Stratix IV
- Cyclone IV
- Arria II

Note: For Intel Agilex, Intel Stratix 10, and Intel MAX 10 devices, export the flash pin by enabling **Enable SPI pins interface** parameter of this IP core.

Related Information

Configuration Devices

Provides more information about the third-party flash support.
1.2. Signals

Figure 1. Signal Block Diagram

Table 1. Ports Description

<table>
<thead>
<tr>
<th>Signal</th>
<th>Width</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>avl_csr_addr</td>
<td>6</td>
<td>Input</td>
<td>Avalon memory-mapped address bus. The address bus is in word addressing.</td>
</tr>
<tr>
<td>avl_csr_read</td>
<td>1</td>
<td>Input</td>
<td>Avalon memory-mapped read control to the CSR.</td>
</tr>
<tr>
<td>avl_csr_rddata</td>
<td>32</td>
<td>Output</td>
<td>Avalon memory-mapped read data bus from the CSR.</td>
</tr>
<tr>
<td>avl_csr_write</td>
<td>1</td>
<td>Input</td>
<td>Avalon memory-mapped write control to the CSR.</td>
</tr>
<tr>
<td>avl_csr_wrdata</td>
<td>32</td>
<td>Input</td>
<td>Avalon memory-mapped write data bus to CSR.</td>
</tr>
<tr>
<td>avl_csr_waitrequest</td>
<td>1</td>
<td>Output</td>
<td>Avalon memory-mapped waitrequest control from the CSR.</td>
</tr>
<tr>
<td>avl_csr_rddata_valid</td>
<td>1</td>
<td>Output</td>
<td>Avalon memory-mapped read data valid that indicates the CSR read data is available.</td>
</tr>
<tr>
<td>avl_mem_write</td>
<td>1</td>
<td>Input</td>
<td>Avalon memory-mapped write control to the memory</td>
</tr>
<tr>
<td>avl_mem_burstcount</td>
<td>7</td>
<td>Input</td>
<td>Avalon memory-mapped burst count for the memory. The value range from 1 to 64 (Max page size).</td>
</tr>
<tr>
<td>avl_mem_waitrequest</td>
<td>1</td>
<td>Output</td>
<td>Avalon memory-mapped waitrequest control from the memory.</td>
</tr>
<tr>
<td>avl_mem_read</td>
<td>1</td>
<td>Input</td>
<td>Avalon memory-mapped read control to the memory</td>
</tr>
<tr>
<td>flash_data</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>flash_dclk</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>flash_ncs</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

**continued...**
Signal | Width | Direction | Description
--- | --- | --- | ---
avl_mem_addr | N | Input | Avalon memory-mapped address bus. The address bus is in word addressing. The width of the address depends on the flash memory density. If you are using Intel Arria 10, and Intel Cyclone 10 GX or any supported devices with general purpose I/O with multiples flashes, write the CSR to select the chip select. The IP targets the selected flash when being accessed via this address.
avl_mem_wrdata | 32 | Input | Avalon memory-mapped write data bus to the memory
avl_mem_readdata | 32 | Output | Avalon memory-mapped read data bus from the memory.
avl_mem_rddata_valid | 1 | Output | Avalon memory-mapped read data valid that indicates the memory read data is available.
avl_mem_byteenable | 4 | Input | Avalon memory-mapped write data enable bus to memory. During bursting mode, byteenable bus will be logic high, 4'b1111.

Clock and Reset

clk | 1 | Input | Input clock to clock the IP core.
reset | 1 | Input | Asynchronous reset to reset the IP core.

Interrupt

Irq | 1 | Output | Interrupt signal that indicate if there is an illegal write or illegal erase.

Conduit Interface(2)

flash_data | 4 | Bidirectional | Input or output port to feed data from the flash device.
flash_dclk | 1 | Output | Provides clock signal to the flash device.
flash_ncs | 1/3 | Output | Provides the ncs signal to the flash device.

1.3. Parameters

Table 2. Parameter Settings

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Legal Values</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device Density</td>
<td>1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048</td>
<td>Density of the flash device used in Mb.</td>
</tr>
<tr>
<td>Disable dedicated Active Serial interface</td>
<td>—</td>
<td>Routes the signals to the top level of your design. Enable this when you want to include the Serial Flash Loader Intel FPGA IP in your design.</td>
</tr>
<tr>
<td>Enable SPI pins interface</td>
<td>—</td>
<td>Translates the signals to the SPI pin interface.</td>
</tr>
<tr>
<td>Number of Chip Select used</td>
<td>1, 2, 3</td>
<td>Selects the number of chip select connected to the flash.</td>
</tr>
<tr>
<td>Enable flash simulation model</td>
<td>—</td>
<td>Uses the flash inside the device for simulation model.</td>
</tr>
</tbody>
</table>

(2) Available when you enable the Enable SPI pins interface parameter.
### 1.4. Register Map

#### Table 3. Register Map
- Each address offset in the following table represents 1 word of memory address space.
- IP_CLK is the clock that drives the IP
- SCLK is the clock that drives the flash device

<table>
<thead>
<tr>
<th>Offset (Hex)</th>
<th>Register Name</th>
<th>R/W</th>
<th>Field Name</th>
<th>Bit</th>
<th>Default Value (Hex)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Control Register</td>
<td>R/W</td>
<td>Addressing mode</td>
<td>8</td>
<td>0x0</td>
<td>Addressing mode for read and write operation:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• 0x0: 3-bytes addressing</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• 0x1: 4-bytes addressing</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>For 4-byte addressing mode, you must enable 4-byte address by sending</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>command to the flash.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>This bit affects direct access to memory via the Avalon memory-mapped</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>interface for both write and read operation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R/W</td>
<td>Chip select</td>
<td>7:4</td>
<td>0x0</td>
<td>Selects the flash device</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• 0x0: To select first device</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• 0x1: To select second device</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• 0x2: To select third device</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Reserved</td>
<td>3:1</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>R/W</td>
<td>Enable</td>
<td>0</td>
<td>0x1</td>
<td>Set this bit to 0 to disable the output of the IP and put all output signal</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>high impedance state. This can be used to share bus with other devices.</td>
</tr>
<tr>
<td>1</td>
<td>SPI Clock Baud-rate Register</td>
<td>R/W</td>
<td>Baud rate divisor</td>
<td>4:0</td>
<td>0x10</td>
<td>The IP has an internal clock divider to generate the clock that connects to the flash device. The possible divisor value is from 2 to 32 with the increment of 2. So, the maximum clock that the flash runs is half of the clock of the IP. Ex if the IP is run with 100Mhz clock, then the clock of the flash is at 50Mhz. By default, the clock is set to the lowest clock (/32) to ensure that the IP works in most cases. Divisor values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• 0x1 : /2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• 0x2 : /4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• 0x3 : /6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• ...</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• 0xF : /30</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• 0x10 : /32</td>
</tr>
<tr>
<td>2</td>
<td>CS Delay Setting Register</td>
<td>R/W</td>
<td>CS de-assert</td>
<td>7:4</td>
<td>0x0</td>
<td>Sets the chip select de-assertion delay.</td>
</tr>
</tbody>
</table>

continued...
<table>
<thead>
<tr>
<th>Offset (Hex)</th>
<th>Register Name</th>
<th>R/W</th>
<th>Field Name</th>
<th>Bit</th>
<th>Default Value (Hex)</th>
<th>Description</th>
</tr>
</thead>
</table>
|             |                        |     |                |     |                     | **0:** Chip select is de-asserted at the last falling edge of SCLK  
**n:** Chip select is de-asserted n number of clocks after the last falling edge of SCLK                                                                 |
|             | CS assert              | R/W | CS assert      | 3:0 | 0x0                 | Sets the chip select assertion delay.  
**0:** Chip select is asserted half flash clock period before the first rising edge of SCLK  
**n:** Chip select is asserted half flash clock period plus n number of IP_CLK. (3)                                                          |
| 3           | Read Capturing Register| R/W | Read delay     | 3:0 | 0x0                 | The clock to output timing of the flash plus the board trace, I/O pin timing can contribute to high value of delay to the data arriving at the IP logic. The delay capture provides a way for the IP to delay its reading logic to compensate for those delays. Delay the read data logic by a value of the IP_CLK cycles. |
| 4           | Operating Protocols Setting Register| R/W | Read data out transfer mode | 17:16 | 0x0 | Transfer mode for read data output. |
|             |                        |     |                |     |                     | **Reserved**                                                                                                                                   |
|             |                        |     |                |     |                     | **Reserved**                                                                                                                                   |
|             |                        |     |                |     |                     | **Reserved**                                                                                                                                   |
|             |                        |     | Read address transfer mode | 13:12 | 0x0 | Transfer mode for read address input Description as bit 1:0. |
|             |                        |     |                |     |                     | **Reserved**                                                                                                                                   |
|             |                        |     | Write Data in transfer mode | 9:8 | 0x0 | Transfer mode for write data input Description as bit 1:0. |
|             |                        |     |                |     |                     | **Reserved**                                                                                                                                   |
|             |                        |     | Write address transfer mode | 5:4 | 0x0 | Transfer mode for write address input Description as bit 1:0. |
|             |                        |     |                |     |                     | **Reserved**                                                                                                                                   |
|             |                        |     | Instruction transfer mode | 1:0 | 0x0 | Transfer mode for opcode:  
**0x0:** Standard SPI mode – command input is sent on DQ0  
**0x1:** Dual IO mode – command input is sent on DQ[1:0]  
**0x2:** Quad IO mode – command input is sent on DQ[3:0] |

(3) Intel recommends that you set the chip select assertion delay to 5 if you are running the IP clock at 100 MHz.
<table>
<thead>
<tr>
<th>Offset (Hex)</th>
<th>Register Name</th>
<th>R/W</th>
<th>Field Name</th>
<th>Bit</th>
<th>Default Value (Hex)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>This setting affects the flash command register. For example, if this field is set to 0x1, flash common operations (e.g. read id, read status, write status register) uses 0x1 as well.</strong></td>
</tr>
<tr>
<td>5</td>
<td>Read Instruction Register</td>
<td>R/W</td>
<td>Dummy cycles</td>
<td>12:8</td>
<td>0xA</td>
<td>Number of default dummy cycles used for read operation. Refer to the respective flash device datasheet.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Read opcode</td>
<td>7:0</td>
<td>0x03</td>
<td>The opcode for read operation. Refer to the respective flash device datasheet to select the correct opcode according to the transfer mode setting.</td>
</tr>
<tr>
<td>6</td>
<td>Write Instruction Register</td>
<td>R/W</td>
<td>Polling opcode</td>
<td>15:8</td>
<td>0x05</td>
<td>The opcode to check if the write operation has been completed. After write operation is completed, the IP releases the wait request of the Avalon memory-mapped interface. In applicable devices, you can set as the status register or flag status register.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Write opcode</td>
<td>7:0</td>
<td>0x02</td>
<td>The opcode for write operation. Refer to the respective flash device datasheet to select the correct opcode according to the transfer mode setting.</td>
</tr>
<tr>
<td>7</td>
<td>Flash Command Setting Register(4)</td>
<td>R/W</td>
<td>Number of dummy cycles</td>
<td>20:16</td>
<td>0x0</td>
<td>The number of dummy cycles. Set to 0 when the operation does not require any dummy cycles. Refer to the respective flash device datasheet for dummy clock requirements.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Number of data bytes</td>
<td>15:12</td>
<td>0x08</td>
<td>The number of write or read data. This works together with bit 11. If the value is Set to 0 if the operation has no write or read data, for example, write enable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Data type</td>
<td>11</td>
<td>0x01</td>
<td>Indicates the type of data (bit [15:12]).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• 0: Number of byte declared in [15:12] is write data to flash device</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• 1: Number of byte declared in [15:12] is read data from flash device</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Number of address bytes</td>
<td>10:8</td>
<td>0x0</td>
<td>Number of address bytes to send to the flash device. Either 3 or 4 bytes If this is set to zero then the operation does not carry any address byte.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Opcode</td>
<td>7:0</td>
<td>0x05</td>
<td>The opcode of the operation.</td>
</tr>
<tr>
<td>8</td>
<td>Flash Command Control Register</td>
<td>W</td>
<td>Start</td>
<td>0</td>
<td>0x0</td>
<td>Write 1 to this bit to start the operation.</td>
</tr>
</tbody>
</table>

(4) Default setting is for read status command.
<table>
<thead>
<tr>
<th>Offset (Hex)</th>
<th>Register Name</th>
<th>R/W</th>
<th>Field Name</th>
<th>Bit</th>
<th>Default Value (Hex)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>Flash Command Address Register</td>
<td>R/W</td>
<td>Stating address</td>
<td>31:0</td>
<td>31:0</td>
<td>Address of flash command.</td>
</tr>
<tr>
<td>A</td>
<td>Flash Command Write Data 0 Register</td>
<td>R/W</td>
<td>Lower 4 bytes write data</td>
<td>31:0</td>
<td>0x0</td>
<td>The first 4-byte of write data to flash device.</td>
</tr>
<tr>
<td>B</td>
<td>Flash Command Write Data 1 Register</td>
<td>R/W</td>
<td>Upper 4 bytes write data</td>
<td>31:0</td>
<td>0x0</td>
<td>The last 4-byte of write data to the flash device.</td>
</tr>
<tr>
<td>C</td>
<td>Flash Command Read Data 0 Register</td>
<td>R</td>
<td>Lower 4 bytes read data</td>
<td>31:0</td>
<td>0x0</td>
<td>The first 4-byte of read data from flash device.</td>
</tr>
<tr>
<td>D</td>
<td>Flash Command Read Data 1 Register</td>
<td>R</td>
<td>Upper 4 bytes read data</td>
<td>31:0</td>
<td>0x0</td>
<td>The last 4-byte of read data from the flash device.</td>
</tr>
</tbody>
</table>

### 1.5. Using Generic Serial Flash Interface Intel FPGA IP

The Generic Serial Flash Interface Intel FPGA IP core interfaces are Avalon memory-mapped compliant. For more details, refer to the Avalon specification.

#### 1.5.1. Control Status Register Operations

You can perform a read or write to a specific address offset using the Control Status Register (CSR).

To execute the read or read operation for the control status register, perform the following steps:

1. **Assert the avl_csr_write or avl_csr_read signal while the avl_csr_waitrequest signal is low** (if the waitrequest signal is high, the avl_csr_write or avl_csr_read signal must be kept high until the waitrequest signal goes low.)

2. **At the same time, set address value on avl_csr_address bus.** If it is a write operation, set value data on the avl_csr_writedata bus together with the address.

3. **If it is a read transaction, wait until avl_csr_readdatavalid signal is asserted high to retrieve the read data.**
   - For operations that require write value to flash, you must perform write enable operation first.
   - You must read the flag status register every time you issue a write or erase command.
   - In case of support multiples flash devices, you must write chip select register to select the correct flash device before performing any operation to the specific flash device.
1.5.2. Memory Operations

During flash memory access, the IP core performs the following steps to allow you to perform any direct read or write operation:

- Write enable for write operation
- Check flag status register to make sure the operation has been completed at the flash
- Release waitrequest signal when operation completed

Memory operations are similar to the Avalon memory-mapped operations. You must set the correct address on the address bus, write data if it is write transaction, drive burst count bus 1 if single transaction or desired burst count value and trigger the write or read signal.

Note: For multiple flash device setup, the address bus is extended to include the chip select value.

Figure 2. 8-Word Write Burst Waveform Example

Figure 3. 8-Word Reading Burst Waveform Example

Figure 4. 1-Byte Write bytenable = 4’b0001 Waveform Example

Note: There are two internal unconstrained clocks in the Generic Flash Serial Interface Intel FPGA IP core when you compile your design in the Intel Quartus® Prime Pro Edition software. Intel recommends that you constraint the path by using the following command:

```
create_generated_clock -name <name_of_generated_clock> -source [get_ports <input_clock_name>] -divide_by 2 [get_registers <path_of_the_unconstrained_path>]
```
1.5.3. Byte Enabling

1.5.3.1. Byte Enabling Supported Patterns

Table 4. Byte Enabling Supported Patterns

<table>
<thead>
<tr>
<th>Byte Enable Pattern</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>4'b0000</td>
<td>Supported (when burst is more than 1)</td>
</tr>
<tr>
<td>4'b0001</td>
<td>Supported</td>
</tr>
<tr>
<td>4'b0010</td>
<td>Supported</td>
</tr>
<tr>
<td>4'b0100</td>
<td>Supported</td>
</tr>
<tr>
<td>4'b1000</td>
<td>Supported</td>
</tr>
<tr>
<td>4'b0011</td>
<td>Supported</td>
</tr>
<tr>
<td>4'b0110</td>
<td>Supported</td>
</tr>
<tr>
<td>4'b1100</td>
<td>Supported</td>
</tr>
<tr>
<td>4'b0111</td>
<td>Supported</td>
</tr>
<tr>
<td>4'b1110</td>
<td>Supported</td>
</tr>
<tr>
<td>4'b1111</td>
<td>Supported</td>
</tr>
</tbody>
</table>

All write bursts greater than 1 is set to byte enable of 4'b1111, in which all byteenables are asserted through all the words of the burst. When a master wider than 32 bits is used to connect to the IP, the interconnect fabric of the Platform Designer produces multi-word bursts to adapt the wide master into the narrow 32-bit slave (the IP). Choose to use the byte enabling patterns in the Byte Enabling Supported Patterns table if the wide master intends to write only certain bytes in the entire transaction. You must ensure that the byte enabling pattern is contiguous for burst writes.

Note: For burst writes, the IP writes all bytes (4'b1111) into the flash even with your selected byte enable pattern. If you have not enabled the data, the IP writes 0xFF. The performance of the IP is still the same as writing 8 bytes for a 64-bit wide master, even if you have enabled only 1 byte using byte enable.

1.6. Generic Serial Flash Interface Intel FPGA IP Core Reference Design

The reference design implements the Generic Serial Flash Interface Intel FPGA IP to perform the following general-purpose memory operations:

- Read device ID
- Enable sector protect
- Perform sector erase
- Read and write data from and to flash devices

Related Information

- Generic Serial Flash Interface Intel FPGA IP Core User Guide on page 3
- Generic Serial Flash Interface Intel FPGA IP Core Reference Design Files
1.6.1. Hardware and Software Requirements

The following are the hardware and software requirements for the design example:

- Cyclone V E FPGA Development Kit
- Intel Quartus Prime software version 18.0 with Nios® II Software Build Tools for Eclipse
- Intel FPGA Download Cable II
- Tested flash devices:
  - Cypress* S70FL01G
  - Micron MT25Q01G
  - Micron MT25Q512
  - EPCQ256

1.6.2. Functional Description

1.6.2.1. Reference Design Components

Figure 5. Reference Design Block Diagram

Table 5. Reference Design Components Descriptions

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JTAG UART Intel FPGA IP</td>
<td>Enables communication between the Nios II processor and the host computer</td>
</tr>
<tr>
<td>Nios II Processor</td>
<td>Run application program by executing data and instruction</td>
</tr>
<tr>
<td>On-Chip Memory Intel FPGA IP</td>
<td>Store code and data</td>
</tr>
<tr>
<td>Generic Serial Flash Interface Intel FPGA IP</td>
<td>Controls vendor-independent flash device to perform flash interaction</td>
</tr>
</tbody>
</table>
1.6.2.2. Reference Design Application Program

Flow diagram sequence description:

1. The application program starts with identifying the flash device attached to the FPGA.

   *Note:* The flash devices serve as samples to demonstrate this reference design only.

2. The application program performs sector protection and erases the protected sector:
   a. To perform sector protect, the application program:
      i. Performs write enable command.
      ii. Performs write status register command to set block protect (BP) bit and Top/Bottom(TB) bit.
      iii. Polls write in progress (WIP) bit (bit 0 of status register) until it returns a 0 (ready).
      iv. Performs read status register command to check if sector protect operation succeeded or failed.
   b. To perform sector erase, the application program:
i. Performs write enable command.
ii. Performs sector erase command.
iii. Polls in progress (WIP) bit (bit 0 of status register) until it return a 0 (ready).
iv. Performs read status register to check whether erase operation succeeded or failed.

3. Erase error occurred because the sector is protected. The application program clears the error bit through:
   • Clear flag status register command (EPCQ-L or Micron).
   • Clear status register command (Cypress).

4. The application program disables the sector protect:
   a. Performs write enable command.
   b. Performs write status register command to clear BP bit and TB bit.
   c. Polls WIP bit (bit 0 of status register) until it returns a 0 (ready).
   d. Performs read status register command to check whether BP bit and TB bit has succeeded clear.

5. The application program performs flash device programming after the sector is not protected. The application program:
   a. Performs write memory into the address with empty memory.
   b. Polls WIP bit (bit 0 of status register) until it returns a 0 (ready)
   c. Performs read back memory of the address to confirm the address has programmed.

6. Repeat Step 2 and read back memory of the address. Memory is not erased because the sector is protected.

1.6.3. Creating Nios II Hardware System

1. In the Intel Quartus Prime software, go to File ➤ New Project Wizard.
2. Create a new Intel Quartus Prime Prime project named generic_flash_access in a new directory and select the Cyclone V E 5CEFA7F3117 device.
3. Select Tools ➤ Platform Designer, and save the file as generic_flash_access.qsys.
4. Double-click on the clock source clk_0 and change the Clock frequency to 100000000 Hz (100MHz).
5. Right click on clk_0 and rename it as sys_clk.
6. Add a Nios II processor:
   a. Go to Processor and Peripherals ➤ Embedded Processors ➤ Nios II Processor, and click Add.
   b. Click Finish to add the Nios II processor to the design and rename it as nios2.
      Note: Ignore any messages about parameters that have not been specified yet.
7. Add a Generic Serial Flash Interface IP:
a. Select Basic Functions ➤ Configuration and Programming ➤ Generic Serial Flash Interface Intel FPGA IP, and click Add. Rename this component as intel_generic_serial_flash_interface_top0.

b. Set the device density.
   
   Note: This reference design uses 1024MB flash device density.

c. Connect data_master of processor to avl_mem and avl_csr, and instruction_master of processor to only avl_csr of this component.

8. Add an On-chip Memory IP:
   a. Select Basic Functions ➤ On Chip Memory ➤ On-Chip Memory (RAM or ROM) Intel FPGA IP.
   b. Set the Total Memory Size to 40960 bytes (40 KBytes).
   c. Click Finish and rename as main_memory.
   d. Connect its slave to data_master and instruction_master of processor.

9. Add a JTAG UART IP:
   a. Go to Interface Protocols ➤ Serial ➤ JTAG UART Intel FPGA IP, and click Add.
   b. Click Finish and rename it as jtag_uart.
   c. Connect its avalon_jtag_slave port to the data_master port of the processor.
   d. In the IRQ column, connect the interrupt sender port from the Avalon_jtag_slave port to the interrupt receiver port of the processor and type 0.

10. Connect clock input of sys_clk to clock input of all other components.

11. Resolve all Nios II processor error messages before generating the Platform Designer system:
   a. Double click the Nios II processor nios2.
   b. Click Vectors, change both the Reset vector memory and Exception vector memory to main_memory.s1.
   c. Click System tab and click on the drop-down menu System and click Assign Base Address to auto assign base addresses for all the components.
   d. Under the same menu, click Create Global Reset Network to connect the reset signals to form a global reset network.
12. Generate the system:
   a. Click Generate HDL on the bottom of the window.
   b. When completed, the Platform Designer displays Generate: Completed successfully.

1.6.4. Integrating Modules into Intel Quartus Prime Project

1. In the Intel Quartus Prime software, select Assignment ➤ Settings.
2. In the Settings window, add generic_flash_access.qys file located in the synthesis folder and click Apply.
3. The generic_flash_access.qys file is shown under Files directory. Right click the file and choose Set as Top-Level Entity.
4. Go to Processing ➤ Start ➤ Start Analysis and Elaboration to allow the hardware system to determine input and output pins.
5. Start pin assignment by going to Assignments ➤ Pin Planner, and assign PIN_L14 as clk_clk and PIN_AA26 as reset_reset_n.
6. Go to Assignments ➤ Device ➤ Device and Pin Options ➤ Configuration, and change the Configuration scheme to Active Serial x1.
7. Processing ➤ Start ➤ Start Analysis and Synthesis to perform full hardware system compilation.

1.6.5. Programming the .sof File

1. In the Intel Quartus Prime Programmer, click on Hardware setup and choose the correct USB chain connecting your FPGA.
2. Click on Auto Detect and 5CEFA7F31 appears, and change the file to top.sof.
3. Enable Program/ Configure, and click Start.
1.6.6. Building Application Software System using Nios II Software Build Tools

1. In the Intel Quartus Prime, go to **Tools ➤ Nios II Software Build Tools for Eclipse**.

2. Browse to your workspace directory.

3. In the **Nios II Software Build Tools for Eclipse**, go to **File ➤ New ➤ Nios II Application and BSP from Template**.

4. In the **SOPC Information File name** field, select `generic_flash_access.sopcinfo` from your project directory and click **Open**.

5. For **Project Name**, set to `generic_flash_access`, choose **Hello World Small project template** and click **Finish**.

6. In the `generic_flash_access` project directory and replace the `hello_world_small.c` file with `main.c` and `operation.c` files attached in the reference design.

7. Select the `main.c` file and go to **Project ➤ Build Project** to create the `generic_flash_access.elf` file.

8. Select the `generic_flash_access.elf` file and go to **Run ➤ Run As ➤ Nios II Hardware**.

9. The **Nios II Console** prints the following results.

### 1.6.6.1. Reference Design Results

**Cypress S70FL01G:**

<table>
<thead>
<tr>
<th>Flash Device: Cypress flash S70FL01G</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device ID: 4d210201</td>
</tr>
<tr>
<td>All sectors in this flash device is not protected</td>
</tr>
<tr>
<td>Now performing sector protection...</td>
</tr>
<tr>
<td>All sectors in this flash device is now successfully protected</td>
</tr>
<tr>
<td>Trying to erase sector 0...</td>
</tr>
<tr>
<td>ERASE ERROR as sector is protected!</td>
</tr>
<tr>
<td>Now perform sector unprotect...</td>
</tr>
<tr>
<td>Sector unprotect successfully! :)</td>
</tr>
<tr>
<td>Reading data at address 0...</td>
</tr>
<tr>
<td>Memory content at address 0: abcd1234</td>
</tr>
<tr>
<td>Trying to erase sector 0...</td>
</tr>
<tr>
<td>Sector erase successfully. Sector 0 is now empty.</td>
</tr>
<tr>
<td>Writing data to address 0...</td>
</tr>
<tr>
<td>Read back data from address 0...</td>
</tr>
<tr>
<td>Current memory in address 0: abcd1234</td>
</tr>
<tr>
<td>Read data match with data written. Write memory successful. Now performing sector protection...</td>
</tr>
<tr>
<td>All sectors in this flash device is now successfully protected</td>
</tr>
<tr>
<td>Trying to erase sector 0...</td>
</tr>
<tr>
<td>ERASE ERROR as sector is protected!</td>
</tr>
<tr>
<td>Current memory in address 0: abcd1234</td>
</tr>
<tr>
<td>Read data match with data written previously. Sector erase does not perform during sector is protected.</td>
</tr>
</tbody>
</table>

**Micron MT25Q01G:**

<table>
<thead>
<tr>
<th>Flash Device: Micron flash MT25Q01G</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device ID: 1021ba20</td>
</tr>
<tr>
<td>All sectors in this flash device is not protected</td>
</tr>
<tr>
<td>Now performing sector protection...</td>
</tr>
<tr>
<td>All sectors in this flash device is now successfully protected</td>
</tr>
<tr>
<td>Trying to erase sector 0...</td>
</tr>
</tbody>
</table>
Erase Error as erase is not allow during sector is protected!
Now perform sector unprotect... :) 
Reading data at address 0...
Memory content at address 0: abcd1234
Address 0 containing data, it is not empty.
Trying to erase sector 0...
Sector erase successfully. Sector 0 is now empty.
Memory not containing data...
Writing data to address 0...
Read back data from address 0...
Current memory in address 0: abcd1234
Read data match with data written. Write memory successful.
All sectors in this flash device is now successfully protected
Trying to erase sector 0...
ERASE ERROR as sector is protected!
Current memory in address 0: abcd1234
Read data match with data written previously. Sector erase does not perform during sector is protected.

Micron MT25Q512:

Flash Device: Micron flash MT25Q512
Device ID: 1020ba20
All sectors in this flash device is not protected
Now performing sector protection...
All sectors in this flash device is now successfully protected
Trying to erase sector 0...
Erase Error as erase is not allow during sector is protected!
Now perform sector unprotect... :) 
Reading data at address 0...
Memory content at address 0: abcd1234
Address 0 containing data, it is not empty.
Trying to erase sector 0...
Sector erase successfully. Sector 0 is now empty.
Memory not containing data...
Writing data to address 0...
Read back data from address 0...
Current memory in address 0: abcd1234
Read data match with data written. Write memory successful.
All sectors in this flash device is now successfully protected
Trying to erase sector 0...
ERASE ERROR as sector is protected!
Current memory in address 0: abcd1234
Read data match with data written previously. Sector erase does not perform during sector is protected.

EPCQ256:

Flash Device: EPCQ256
Device ID: 1019ba20
All sectors in this flash device is not protected
Now performing sector protection...
All sectors in this flash device is now successfully protected
Trying to erase sector 0...
Erase Error as erase is not allow during sector is protected!
Now perform sector unprotect... :) 
Reading data at address 0...
Memory content at address 0: abcd1234
Address 0 containing data, it is not empty.
Trying to erase sector 0...
Sector erase successfully. Sector 0 is now empty.
Memory not containing data...
Writing data to address 0...
Read back data from address 0...
Current memory in address 0: abcd1234
Read data match with data written. Write memory successful.
Now performing sector protection...
All sectors in this flash device is now successfully protected
Trying to erase sector 0...
ERASE ERROR as sector is protected!
Current memory in address 0: abcd1234
Read data match with data written previously. Sector erase does not perform
during sector is protected.

1.7. Flash Access Using the Generic Serial Flash Interface Intel FPGA IP Core

This section provides information on how to use the registers of this IP core to perform flash access. To begin, build the Platform Designer system with a few components (clock, jtag master, pll, and this IP core) as shown below. Then, use the flash operations in the next example.

Figure 8. Example of Creating Flash Access Using the Generic Serial Flash Interface Intel FPGA IP Core

Note: You must set the MSEL pins of the FPGA devices to the AS configuration mode. For Intel MAX 10 devices, you must enable the Enable SPI Pins Interface parameter of this IP core.

Flash operations are divided into several categories. Example of operations, registers to use, and sample .tcl scripts for each category are provided.

1.7.1. Flash Operations that Require Operation Code

The following flash operations require an operation code:

- Write enable
- Enter 4-byte addressing mode
- Exit 4-byte addressing mode
- Clear flag status register
- Clear status register

The following registers are used for operations that require an operation code:

- Flash command setting register
- Flash command control register
Example 1. Perform the Write Enable Operation for the Flash

```bash
proc write_enable { } {
    global mp flash_cmd_setting flash_cmd_ctrl flash_cmd_write_data_0
    master_write_32 $mp $flash_cmd_setting 0x00000006
    master_write_32 $mp $flash_cmd_ctrl 0x1
}
```

To perform the write enable operation for the flash, follow these steps:
1. Define the global variables.
2. Customize the write enable operation by writing to the flash command setting register.
   a. Set bit [7:0] of this register to 06 as 06h is the operation code of the write enable operation.
3. Write 1 to bit 0 of the flash command control register to start the write enable operation.

1.7.2. Flash Operations to Read Flash Registers

The following flash operations are used to read flash registers:
- Read device ID
- Read status register
- Read flag status register
- Read configuration register
- Read bank register
- Read enhanced volatile configuration register

The following registers are used to read the status of a register:
- Flash command setting register
- Flash command control register
- Flash command read data 0 register

Example 2. Perform the Read Device ID Operation

```bash
proc read_device_id { } {
    global mp flash_cmd_setting flash_cmd_ctrl flash_cmd_read_data_0
    master_write_32 $mp $flash_cmd_setting 0x0000489F
    master_write_32 $mp $flash_cmd_ctrl 0x1
    set device_id [master_read_32 $mp $flash_cmd_read_data_0 1]
    puts $device_id
}
```
To perform the read device ID operation, follow these steps:

1. Define the global variables.
2. Customize the read device ID operation by writing to the flash command setting register.
   a. Set bit [7:0] of this register to 9F as 9Fh is the operation code of the read device ID operation.
   b. Set bit [10:8] to 0 as this operation does not carry any address byte.
   c. Set bit 11 to 1 as the number of byte declared in bit [15:12] is the read data from the flash device.
   d. Set bit [15:12] to 4 as you will be reading 4 bytes device ID data from the flash.
3. Write 1 to bit 0 of the flash command control register to start the read device ID operation.
4. Read the device ID from the flash command read data 0 register.

### 1.7.3. Flash Operations to Write Flash Registers

The following flash operations are used to write flash registers:

- Write enhanced volatile configuration register
- Write bank register
- Write status register
- Write configuration register

**Note:**
You must execute the write enable operation before you start these operations.

The following registers are used to write the status of a register:

- Flash command setting register
- Flash command control register
- Flash command write data 0 register

#### Example 3. Perform the Write Status Register Operation to Protect Sector of Flash

```bash
proc write_status_register { } {
  global mp flash_cmd_setting flash_cmd_write_data_0 flash_cmd_ctrl
  master_write_32 $mp $flash_cmd_setting 0x00001001
  master_write_32 $mp $flash_cmd_write_data_0 0x0000007c
  master_write_32 $mp $flash_cmd_ctrl 0x1
}
```

To perform the write status register operation, follow these steps:

1. Define the global variables.
2. Customize the write status register operation by writing to the flash command setting register.
a. Set bit [7:0] of this register to 01 as 01h is the operation code of the write status register operation.
b. Set bit [10:8] to 0 as this operation does not carry any address byte.
c. Set bit 11 to 0 as the number of byte declared in bit [15:12] is the write data to the flash device.
d. Set bit [15:12] to 1 as you will be writing 1 byte (8 bits) of data into the status register.

3. Write the data to set the sector protection into the flash command write data 0 register.
   a. Bit 6 and bit [4:2] of the status register are the block protect bits and bit 5 is the Top/Bottom bit. In this example, protection is required for all sectors from the bottom of the memory array. For more information, refer to the respective flash datasheet.

4. Write 1 to bit 0 of the flash command control register to start the write status register for the sector protect operation.

### 1.7.4. Flash Operations that Require An Address

The following flash operations require an address:
- Sector erase
- Bulk erase
- Die erase

**Note:** You must execute the write enable operation before you start these operations.

The following registers are used for operations that require an address:
- Flash command setting register
- Flash command control register
- Flash command address register

#### Example 4. Perform the Flash Sector Erase Operation

```tcl
proc erase_sector { } {
    global mp flash_cmd_setting flash_cmd_ctrl flash_cmd_addr_register
    master_write_32 $mp $flash_cmd_setting 0x000004D8
    master_write_32 $mp $flash_cmd_addr_register 0x00001000
    master_write_32 $mp $flash_cmd_ctrl 0x1
}
```

To perform the flash sector erase operation, follow these steps:
1. Define the global variables.
2. Customize the sector erase operation by writing to the flash command setting register.
a. Set bit [7:0] of this register to D8 as D8h is the operation code of the sector erase operation.

b. Set bit [10:8] to 4 as 4 bytes of address will be sent to the flash device.

c. Set bit 11 to 0 as the number of byte declared in bit [15:12] is the write data to the flash device.

3. Specify any address within the sector that you want to erase and write it to the flash command address register.
   a. In this example, we are performing the erase sector operation for address 00001000.

4. Write 1 to bit 0 of the flash command control register to start the sector erase operation.

This IP core supports flash in the extended, dual, and quad I/O protocols. Currently, the protocols supported by this IP core is a single-transfer rate (STR) only. This IP core supports both the 3-byte and 4-byte addressing modes. Different protocols and addressing modes to read memory and program operations are explained in the following sections.

1.7.5. Read Memory from the Flash

The following registers are used to perform the read memory:
- Operating protocols setting register
- Control register
- Read instruction register

Example 5. Perform the Read Memory (Extended Mode)

```bash
proc read { } {
    global mp operating_protocols_setting control_register read_instr
    master_write_32 $mp $operating_protocols_setting 0x00000000
    master_write_32 $mp $control_register 0x00000001
    master_write_32 $mp $read_instr 0x00000003
    master_read_32 $mp 0x01000000 0x1
}
```

To perform the read memory for the extended mode, follow these steps:

1. Define the global variables.

2. Write to the operating protocols setting register to set the transfer mode of the read memory operation. In this example, the transfer mode for read is (1-1-1).
   a. Set the instruction transfer mode [1:0] to 0, read address transfer mode [13:12] to 0, and read data out transfer mode [17:16] to 0.

3. Write to the control register to choose the byte addressing mode of the read memory operation.
a. This example is using the 3-byte addressing mode. Set bit 8 to 0.

4. Write to the read instruction register to customize the read memory operation.
   a. Set the read operation code [7:0] to 03 as 03h is the operation code for read.
   b. Set the dummy cycles [12:8] to 0 as the read operation does not contain any dummy cycles.

5. After setting the registers, you can perform read memory content in the address.
   a. In this example, 1 word of data is read from address 0x01000000.

Example 6. Perform the Dual-Output Fast Read (Dual-SPI Mode)

```java
def dual_output_fast_read():
    global mp, operating_protocols_setting, control_register, read_instr
    master_write_32(mp, operating_protocols_setting, 0x0011001)
    master_write_32(mp, control_register, 0x00000101)
    master_write_32(mp, read_instr, 0x00000A3B)
    master_read_32(mp, 0x00000100, 0x1)
```

To perform the dual-output fast read mode, follow these steps:
1. Define the global variables.
2. Write to the operating protocols setting register to set the transfer mode of the read memory operation. In this example, the transfer mode for read is (2-2-2).
   a. Set the instruction transfer mode [1:0] to 1, read address transfer mode [13:12] to 1, and read data out transfer mode [17:16] to 1.
3. Write to the control register to choose the byte addressing mode of the read memory operation.
   a. This example is using the 4-byte addressing mode. Set bit 8 to 1.
4. Write to the read instruction register to customize the read memory operation.
   a. Set the read operation code [7:0] to 3B as 3Bh is the operation code for the dual-output fast read.
   b. Set the dummy cycles [12:8] to A as the dual-output fast read operation contains 10 dummy cycles.
5. After setting the registers, you can perform dual-output fast read memory content in the address.
   a. In this example, the memory content is read from address 0x00000100.
1.7.6. Program Flash

The following registers are used to perform program flash:

- Operating protocols setting
- Control register
- Write instruction

Example 7. Perform Page Program (Extended Mode)

```plaintext
proc page_program { } {
    global mp operating_protocols_setting control_register write_instr
    master_write_32 $mp $operating_protocols_setting 0x00000000
    master_write_32 $mp $control_register 0x00000001
    master_write_32 $mp $write_instr 0x00007002
    master_write_32 $mp 0x00001000 0x1234abcd
}
```

To perform the page program for the extended mode, follow these steps:

1. Define the global variables.
2. Write to the operating protocols setting register to set the transfer mode of the program operation. In this example, the transfer mode for read is (1-1-1).
   a. Set the instruction transfer mode \([1:0]\) to 1, write address transfer mode \([5:4]\) to 1, and write data in transfer mode \([9:8]\) to 1.
3. Write to the control register to choose the byte addressing mode of the write operation.
   a. This example is using the 3-byte addressing mode. Set bit 8 to 0.
4. Write to the write instruction register to customize the program operation.
   a. Set the write operation code \([7:0]\) to 02 as 02h is the operation code for page program.
   b. Set the polling operation code \([15:8]\) to 70 as 70h is the operation code for the read flag status register. After completing the write operation, the IP core releases the wait request of the Avalon memory-mapped interface. For the flash to have the read flag status register, you can use the read status register (05h).
5. After setting the registers, you can start to program the memory into the address.
   a. In this example, 1234abcdh is written to the memory address 0x00001000.
Example 8. Perform 4-byte Quad Input Fast Program (Quad SPI Mode)

```c
proc fourbyte_quad_input_fast_program { } {
    global mp operating_protocols_setting control_register write_instr
    master_write_32 $mp $operating_protocols_setting 0x00000222
    master_write_32 $mp $control_register 0x0000101
    master_write_32 $mp $write_instr 0x00007034
    master_write_32 $mp 0x00002000 0xabcd1234
}
```

To perform the 4-byte quad input fast program, follow these steps:

1. Define the global variables.
2. Write to the operating protocols setting register to set the transfer mode of the program operation. In this example, the transfer mode for 4-byte quad input fast program is (4-4-4).
   a. Set the instruction transfer mode \([1:0]\) to 2, write address transfer mode \([5:4]\) to 2, and write data in transfer mode \([9:8]\) to 2.
3. Write to the control register to choose the byte addressing mode of the write operation.
   a. This example is using the 4-byte addressing mode. Set bit 8 to 1.
4. Write to the write instruction register to customize the program operation.
   a. Set the write operation code \([7:0]\) to 34 as 34h is the operation code for the 4-byte quad input fast program.
   b. Set the polling operation code \([15:8]\) to 70 as 70h is the operation code for the read flag status register. After completing the write operation, the IP core releases the wait request of the Avalon memory-mapped interface. For the flash to have the read flag status register, you can use the read status register (05h).
5. After setting the registers, you can start to program the memory into the address.
   a. In this example, 1234abcdh is written to the memory address 0x00002000.

### 1.8. Generic Serial Flash Interface Intel FPGA IP User Guide

Archives

IP versions are the same as the Intel Quartus Prime Design Suite software versions up to v19.1. From Intel Quartus Prime Design Suite software version 19.2 or later, IP cores have a new IP versioning scheme.

If an IP core version is not listed, the user guide for the previous IP core version applies.

<table>
<thead>
<tr>
<th>IP Core Version</th>
<th>User Guide</th>
</tr>
</thead>
<tbody>
<tr>
<td>19.1</td>
<td>Generic Serial Flash Interface Intel FPGA IP Core User Guide</td>
</tr>
<tr>
<td>18.1</td>
<td>Generic Serial Flash Interface Intel FPGA IP Core User Guide</td>
</tr>
<tr>
<td>18.0</td>
<td>Generic Serial Flash Interface Intel FPGA IP Core User Guide</td>
</tr>
</tbody>
</table>
# 1.9. Document Revision History for the Generic Serial Flash Interface Intel FPGA IP User Guide

<table>
<thead>
<tr>
<th>Document Version</th>
<th>Intel Quartus Prime Version</th>
<th>IP Version</th>
<th>Changes</th>
</tr>
</thead>
</table>
• Added the Byte Enabling section.  
• Added a note to the Device Family Support topic.  
• Updated the Memory Operations topic.  
• Updated for latest branding standards. |
| 2019.11.27       | 19.3                        | 19.1       | Added a note to the Memory Operations topic. |
| 2019.09.30       | 19.3                        | 19.1       | • Added support for Intel Agilex devices.  
• Updated the Device Family Support topic.  
• Made minor editorial updates to the document. |
| 2018.11.09       | 18.1                        | 18.1       | • Added the Flash Access Using the Generic Serial Flash Interface Intel FPGA IP Core section.  
• Added the Generic Serial Flash Interface Intel FPGA IP Core User Guide Archives section.  
• Updated the Generic Serial Flash Interface Intel FPGA IP Core User Guide section to provide more information on the Generic Serial Flash Interface Intel FPGA IP core.  
• Updated the signal names of the Signal Block Diagram figure.  
• Updated the Conduit Interface signal names in the Ports Description table.  
• Updated the description of the write opcode field name of the write instruction register in the Register Map table. |
| 2018.05.16       | 18.0                        | 18.0       | • Updated the Generic Serial Flash Interface Intel FPGA IP Core Reference Design Files link.  
• Added Flash Command Address Register in the Register Map. |
| 2018.05.07       | 18.0                        | 18.0       | Initial release. |