The Serial Flash Mailbox Client Intel FPGA IP provides access to quad serial flash devices (SPI).

For a complete list of supported flash memory devices refer to the Device Configuration - Support Center web page.

The Serial Flash Mailbox Client Intel FPGA IP core supports:
- Direct flash access (write and read) through the Avalon® Memory-Mapped (Avalon MM) interface
- Control register access for other operations through the control and status register (CSR) interface
- Up to 4 kilobytes (KB) data data transfers for each quad SPI read and write command
- Opcodes for the following quad SPI operations:
  - Open
  - Close
  - Set chip select
  - Read data from flash
  - Write data to flash
  - Erase sector
  - Read device register
  - Write device register
  - Send device opcode

Refer to the respective flash device datasheet for a complete list of supported operations for a particular device.

**Related Information**

**Introduction to Intel FPGA IP Cores**
Provides general information about all Intel FPGA cores, including parameterizing, generating, upgrading, and simulating IP cores.

**Serial Flash Mailbox Client IP Modules**

The following block diagram shows the modules that comprise the Serial Flash Mailbox Client IP.
You set up quad SPI commands by writing to the CSR. The Serial Flash Mailbox Client IP sends commands to the secure device manager (SDM) in the Intel® Stratix® 10 device. The SDM controls the quad SPI device.

For write commands you prestore the write data in the write data FIFO. For read commands, you read data from the read data FIFO. The write and read data FIFOs store up to 1024 words. The write and read data FIFOs are Avalon MM slaves. The Serial Flash Mailbox Client IP asserts the irq signal if the command results in an error response.
Device Family Support

The following lists the device support level definitions for Intel FPGA IPs:

- **Advance support** — The IP is available for simulation and compilation for this device family. Timing models include initial engineering estimates of delays based on early post-layout information. The timing models are subject to change as silicon testing improves the correlation between the actual silicon and the timing models. You can use this IP for system architecture and resource utilization studies, simulation, pinout, system latency assessments, basic timing assessments (pipeline budgeting), and I/O transfer strategy (data-path width, burst depth, I/O standards tradeoffs).

- **Preliminary support** — The IP is verified with preliminary timing models for this device family. The IP meets all functional requirements, but might still be undergoing timing analysis for the device family. It can be used in production designs with caution.

- **Final support** — The IP is verified with final timing models for this device family. The IP meets all functional and timing requirements for the device family and can be used in production designs.

<table>
<thead>
<tr>
<th>Device Family</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel Stratix 10</td>
<td>Final</td>
</tr>
</tbody>
</table>

*Note:* Intel does not provide simulation modes for the Mailbox Avalon ST Client Intel FPGA IP.

**Related Information**

- Serial Flash Client Intel FPGA IP Release Notes

**Signals**

**Figure 2. Top-Level Signals**

```
Serial Flash Mailbox
Client Intel FPGA IP

 csr_address[6:0]  csr_readdata[31:0]  csr_waitrequest
 csr_read          csr_readdatavalid  csr_readdatavalid
 csr_write         wr_mem_waitrequest
 csr_writedata[31:0]

de_rdl Mem_address            rd_mem_readdatavalid
 rd_mem_address              rd_mem_readdatavalid
 rd_mem_read
 wr_mem_address              wr_mem_write
 wr_mem_writedata[31:0]
 wr_mem_write

clk
reset
irq
```
### Signal Descriptions

<table>
<thead>
<tr>
<th>Signal</th>
<th>Width</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Avalon MM Control and Status Register Signals</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>csr_address</td>
<td>7</td>
<td>Input</td>
<td>Avalon MM address bus. The address bus uses word (32-bit) addressing.</td>
</tr>
<tr>
<td>csr_read</td>
<td>1</td>
<td>Input</td>
<td>Avalon MM read control for the CSR.</td>
</tr>
<tr>
<td>csr_readdata</td>
<td>32</td>
<td>Output</td>
<td>Avalon MM read data bus from the CSR.</td>
</tr>
<tr>
<td>csr_write</td>
<td>1</td>
<td>Input</td>
<td>Avalon MM write control to the CSR.</td>
</tr>
<tr>
<td>csr_writedata</td>
<td>32</td>
<td>Input</td>
<td>Avalon MM write data bus to CSR.</td>
</tr>
<tr>
<td>csr_waitrequest</td>
<td>1</td>
<td>Output</td>
<td>Avalon MM waitrequest control from the CSR.</td>
</tr>
<tr>
<td>csr_readdata_valid</td>
<td>1</td>
<td>Output</td>
<td>Avalon MM read data valid that indicates the CSR read data is available.</td>
</tr>
<tr>
<td><strong>Avalon MM Write Data Signals</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>wr_mem_write</td>
<td>1</td>
<td>Input</td>
<td>Avalon MM write control to the FIFO.</td>
</tr>
<tr>
<td>wr_mem_address</td>
<td>1</td>
<td>Input</td>
<td>Avalon MM address.</td>
</tr>
<tr>
<td>wr_mem_writedata</td>
<td>32</td>
<td>Input</td>
<td>Avalon MM write data bus to the memory.</td>
</tr>
<tr>
<td>wr_mem_waitrequest</td>
<td>1</td>
<td>Output</td>
<td>Avalon MM waitrequest control from the memory.</td>
</tr>
<tr>
<td><strong>Avalon MM Read Data Signals</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rd_mem_read</td>
<td>1</td>
<td>Input</td>
<td>Avalon MM read control to the FIFO.</td>
</tr>
<tr>
<td>rd_mem_readdata</td>
<td>32</td>
<td>Output</td>
<td>Avalon MM read data bus from the memory.</td>
</tr>
<tr>
<td>rd_mem_readdata_valid</td>
<td>1</td>
<td>Output</td>
<td>Avalon MM read data valid that indicates the memory read data is available.</td>
</tr>
<tr>
<td>rd_mem_address</td>
<td>1</td>
<td>Input</td>
<td>Avalon MM address.</td>
</tr>
<tr>
<td><strong>Clock and Reset</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>clk</td>
<td>1</td>
<td>Input</td>
<td>Input clock to clock the IP. The maximum frequency supported 250 MHz.</td>
</tr>
<tr>
<td>reset</td>
<td>1</td>
<td>Input</td>
<td>Synchronous reset to reset the Serial Flash Mailbox Client Intel FPGA IP.</td>
</tr>
<tr>
<td>Note: Refer to the AN 891: Using the Reset Release Intel FPGA IP for more information about reset in Intel Agilex™ devices.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>irq</td>
<td>1</td>
<td>Output</td>
<td>The irq signal asserts if the command STATUS is not OK. Read the ISR register Rsp_status field to determine the error.</td>
</tr>
</tbody>
</table>

**Related Information**

- AN 891: Using the Reset Release Intel FPGA IP
- Avalon Interfaces Specifications
  For more information about Avalon MM interfaces, including detailed signal definitions and timing diagrams.
### Register Map and Definitions

Table 3. Register Map and Definitions

- Each address offset in the table represents one word of memory address space.
- All registers have a default value of 0x0 unless otherwise stated.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Name</th>
<th>Field Name</th>
<th>R/W</th>
<th>Width</th>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>STATUS</td>
<td>Rsp_status</td>
<td>R</td>
<td>11</td>
<td>10:0</td>
<td>The status of executed commands. Refer to Response Codes on page 9 for the definitions of response codes.</td>
</tr>
</tbody>
</table>
| 1      | ISR          | Rddata_valid   | R   | 1     | 1     | When 1, indicates that readdata is available in the read data FIFO. The READ_FIFO_LEVEL specifies how many words are available in the read data FIFO. 
|        |              | Cmd_err        | R   | 1     | 0     | When 1, indicates an that the current command failed. Read the Rsp_status field of the STATUS register to determine the error condition. Assert reset to clear this bit. |
| 2      | IER          | Rdat_valid_en  | R/W | 1     | 1     | The enable bit for read data. The default value is 1.                       |
|        |              | Cmd_err_en     | R/W | 1     | 0     | The enable for command error responses. The default value is 1.             |
| 3      | CHIP_SELECT  | Chip_select    | R/W | 4     | 3:0   | Write the value of the flash device you want to select.                    |
| 4      | OPEN         | Open           | W   | 1     | 0     | Request exclusive access to the flash device. Write 1 to request exclusive access. The Rsp_status field of the STATUS register returns OK the SDM accepts request. |
| 5      | CLOSE        | Close          | W   | 1     | 0     | Write 1 to close exclusive access to the flash device. The Rsp_status field of the STATUS register returns OK if the SDM accepts request. |
| 6      | WR_ENABLE    | Wr_enable      | W   | 1     | 0     | Write 1 to assert the write a enable.                                     |
| 7      | WR_STATUS    | Wr_status      | W   | 8     | 7:0   | Writes the STATUS register of the flash memory.                            |
| 8      | RD_STATUS    | Rd_status      | R   | 8     | 7:0   | Rd_status contains the information from read STATUS register operation.(1) |
| 9      | SECTOR_ERASE | Sector_addr    | W   | 32    | 31:0  | Erases 64 KB in the flash memory, regardless of the flash memory sector size. The sector_address must be in most significant byte to least significant byte order. |

---

(1) For Micron* devices, the flag status register provides the status of stacked devices. You can access the flag status register using the CONTROL command.
For example, to erase a sector of a Micron 2 gigabit (Gb) flash at address 0x04FF0000 using the opcode method, complete the following steps:
1. Get exclusive access to the serial flash device.
2. Enable the WR_ENABLE: 0x00000001
3. Set NUMB_BYTES to 4: 0x00000004
4. Write the CONTROL (opcode) for the sector erase: 0xDC000021
5. Write the flash address to WRITEDATA_0, the lower 4 bytes of write data: 0x0000FF04

For sequential erase commands in Micron devices, read bit 7 of the flag status register as detailed in the flash data sheet for the device. Use the CONTROL register read op code to perform this read.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Name</th>
<th>Field Name</th>
<th>R/W</th>
<th>Width</th>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>RD_DEVICE_ID</td>
<td>Device_id</td>
<td>R</td>
<td>32</td>
<td>31:0</td>
<td>Stores the device ID.</td>
</tr>
<tr>
<td>11 - 12</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>CONTROL</td>
<td>Opcode</td>
<td>R/W</td>
<td>8</td>
<td>31:24</td>
<td>Opcode of the flash device operation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Reserved[23:7]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Read_data_en</td>
<td>R/W</td>
<td>1</td>
<td>6</td>
<td>When 1, indicates the command has read data.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Write_data_en</td>
<td>R/W</td>
<td>1</td>
<td>5</td>
<td>When 1, indicates the command has write data.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Reserved[4:1]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>NUMB_BYTES</td>
<td>Number_bytes</td>
<td>R/W</td>
<td>4</td>
<td>3:0</td>
<td>The number of bytes to write or read from the device register (maximum 8 bytes).</td>
</tr>
<tr>
<td>15</td>
<td>WRITEDATA_0</td>
<td>Writedata_0</td>
<td>W</td>
<td>32</td>
<td>31:0</td>
<td>The lower 4 bytes of write data.</td>
</tr>
<tr>
<td>16</td>
<td>WRITEDATA_1</td>
<td>Writedata_1</td>
<td>W</td>
<td>32</td>
<td>31:0</td>
<td>The upper 4 bytes of write data.</td>
</tr>
<tr>
<td>17</td>
<td>READDATA_0</td>
<td>Readdata_0</td>
<td>R</td>
<td>32</td>
<td>31:0</td>
<td>The lower 4 bytes of read data.</td>
</tr>
<tr>
<td>18</td>
<td>READDATA_1</td>
<td>Readdata_1</td>
<td>R</td>
<td>32</td>
<td>31:0</td>
<td>The upper 4 bytes of read data.</td>
</tr>
<tr>
<td>19</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>WRITE_OP</td>
<td>Write_op</td>
<td>W</td>
<td>2</td>
<td>1:0</td>
<td>Write 2'b01 to perform write operation with address provided in offset 21 and write data in the FIFO. Write 2'b10 to flush data in write FIFO.</td>
</tr>
<tr>
<td>21</td>
<td>WRITE_ADDR</td>
<td>Write_addr</td>
<td>W</td>
<td>32</td>
<td>31:0</td>
<td>The device address for write operation.</td>
</tr>
<tr>
<td>22</td>
<td>WRITE_FIFO_LEVEL</td>
<td>Wr_fifo_level</td>
<td>R</td>
<td>32</td>
<td>31:0</td>
<td>Returns the fill level of the internal write data FIFO.</td>
</tr>
<tr>
<td>23</td>
<td>READ_OP</td>
<td>Read_op</td>
<td>W</td>
<td>2</td>
<td>1:0</td>
<td>Write 2'b01 to perform read operation with address provided in offset 24. Write 2'b10 to flush read data FIFO.</td>
</tr>
</tbody>
</table>

continued...
<table>
<thead>
<tr>
<th>Offset</th>
<th>Name</th>
<th>Field Name</th>
<th>R/W</th>
<th>Width</th>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>READ_ADDR</td>
<td>Read_addr</td>
<td>R/W</td>
<td>32</td>
<td>31:0</td>
<td>The device address for read operation.</td>
</tr>
<tr>
<td>25</td>
<td>READ_WORDS</td>
<td>Read_words</td>
<td>R/W</td>
<td>32</td>
<td>31:0</td>
<td>Number of words to read from device (maximum is 4 KB.)</td>
</tr>
<tr>
<td>26</td>
<td>READ_FIFO_LEVEL</td>
<td>Read_fifo_level</td>
<td>R</td>
<td>32</td>
<td>31:0</td>
<td>Specifies the fill level of the internal read data FIFO.</td>
</tr>
</tbody>
</table>

**Related Information**

**Response Codes** on page 9

**Response Codes**

The response code for each command along with read data, if applicable. Read the Rsp_status field of the STATUS register bit to determine if the command completed without errors.

**Table 4. Error Codes**

<table>
<thead>
<tr>
<th>Value (Hex)</th>
<th>Error Code Response</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>OK</td>
<td>Indicates that the command completed successfully. A command may erroneously return the OK status if a command, is partially successful. For example, if a read operation fails after returning some data.</td>
</tr>
<tr>
<td>1</td>
<td>INVALID_COMMAND</td>
<td>Indicates that the command is incorrectly formatted.</td>
</tr>
<tr>
<td>2</td>
<td>UNKNOWN_BR</td>
<td>Indicates that the command code is not understood.</td>
</tr>
<tr>
<td>3</td>
<td>UNKNOWN</td>
<td>Indicates that the currently loaded firmware cannot decode the command code.</td>
</tr>
<tr>
<td>4</td>
<td>INVALID_COMMAND_PARAMETERS</td>
<td>The length or indirect setting in header is not valid. Or the command data is invalid.</td>
</tr>
<tr>
<td>5</td>
<td>COMMAND_INVALID_ON_SOURCE</td>
<td>Command is from a source for which it is not enabled.</td>
</tr>
<tr>
<td>6</td>
<td>CLIENT_ID_NO_MATCH</td>
<td>Indicates that the Client ID requesting quad SPI or SD MMC access does not have exclusive access.</td>
</tr>
<tr>
<td>7</td>
<td>INVALID_ADDRESS</td>
<td>The address is invalid. This error indicates one of the following conditions:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• An unaligned address</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• An address range problem</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• A read permission problem</td>
</tr>
<tr>
<td>8</td>
<td>TIMEOUT</td>
<td>The command timed out.</td>
</tr>
<tr>
<td>9</td>
<td>HW_NOT_READY</td>
<td>The hardware is not ready. Can indicate either an initialization or configuration problem.</td>
</tr>
<tr>
<td>100</td>
<td>NOT_CONFIGURED</td>
<td>Indicates that the device is not configured.</td>
</tr>
<tr>
<td>1FF</td>
<td>ALT_SDM_MBOX_RESP_DEVICE_BUSY</td>
<td>Indicates that the device is busy.</td>
</tr>
<tr>
<td>2FF</td>
<td>ALT_SDM_MBOX_RESP_NO_VALID_RESP_AVAILABLE</td>
<td>Indicates that there is no valid response available.</td>
</tr>
<tr>
<td>3FF</td>
<td>ALT_SDM_MBOX_RESP_ERROR</td>
<td>General Error.</td>
</tr>
</tbody>
</table>
Using the Serial Flash Mailbox Client Intel FPGA IP

The following topics lists the steps you must follow for CSR write and read operations. All interfaces are Avalon MM compliant. Refer to the Avalon Interface Specification for more information Avalon interfaces.

Control and Status register (CSR) Operation

Follow these steps to perform a read or write to a specific address offset using the Serial Flash Mailbox Client Intel FPGA IP CSR.

1. Assert the csr_write or csr_read signals while the csr_waitrequest signal is low. If the csr_waitrequest signal is high, the csr_write or csr_read signals must be kept high until the csr_waitrequest signal goes low.

2. Depending on the operation, perform the following steps:
   - For read operations, set the address value on the csr_address bus.
   - For write operations, set the address value on the csr_address bus and the value data on the csr_writedata bus.

3. For read operations, you can retrieve the data after the csr_readdatavalid signal is high.

Write Operation

Complete the following steps to perform a write operation: The maximum write size is 4 KB (1000 words).

1. Request exclusive access to the flash memory using the QSPI_OPEN command.
2. Select the flash device using the QSPI_SET_CS command.
3. Erase the flash device using the QSPI_ERASE command.
4. Flush the write data FIFO by writing 2b′10 using the WRITE_OP command.
5. Pre-store the data you want to write to the flash device in the write data FIFO via write data interface. Write the write data FIFO:

   Note: The interface backpressures when the write data FIFO is full.

   a. Assert the wr_mem_write signal while the wr_mem_waitrequest signal is low. If the wr_mem_waitrequest signal is high, the wr_mem_write signal must remain high until the wr_mem_waitrequest signal goes low.

   b. Write the address value to the wr_mem_address bus. Write the data value to wr_mem_writedata bus.
Note: Refer to the base address assigned to wr_mem bus for Serial Flash Mailbox Client Intel FPGA IP in the Intel Quartus® Prime Platform Designer for list of address values that you can write.

c. Repeat step a and b to continuously pre-store the data into the write data FIFO. The FIFO has 1024 words. Consequently, the maximum write is 1024 words.

d. De-assert the wr_mem_write signal after writing all of the data into the write data FIFO.

e. Optional: You can read the fill level of the internal write data FIFO using the WRITE_FIFO_LEVEL command determine if the write data FIFO is full.

6. Read the Rsp_status field of the STATUS register to check the status of the write.

7. Start the write operation by transferring the data from the write data FIFO into the flash device by writing 2'b01 to the WRITE_OP command.

8. Poll the Cmd_err field of the ISR register to check the status of the write transaction. The ISR writes a value of one to the Cmd_err field of the ISR register if a write is unsuccessful. You can also check the Rsp_status field of the STATUS register.

9. Repeat step 3 on page 10 to 8 on page 11 to continue to perform the subsequent write operation.

10. Release exclusive access to the flash device using the CLOSE command.

Figure 3. Write Operation Example Timing Diagram

You can use the Serial Flash Mailbox Client Intel FPGA IP to write the raw programming data (.rpd) file into the flash device. By default, the .rpd file is little-endian. To read flash data back correctly, you must transmit the data to flash memory in big-endian format. First reverse the bit order for each byte. Then write the data to flash from the least significant bit (LSB) to most significant bit (MSB) order. The following figure illustrates this process.
Figure 4. Connections for Little Endian Format When Bit Swap Is Off

<table>
<thead>
<tr>
<th>Byte3[7:0]</th>
<th>0x46</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0]</td>
<td>0</td>
</tr>
<tr>
<td>[1]</td>
<td>1</td>
</tr>
<tr>
<td>[2]</td>
<td>1</td>
</tr>
<tr>
<td>[3]</td>
<td>0</td>
</tr>
<tr>
<td>[4]</td>
<td>0</td>
</tr>
<tr>
<td>[5]</td>
<td>0</td>
</tr>
<tr>
<td>[6]</td>
<td>0</td>
</tr>
<tr>
<td>[7]</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Byte1[7:0]</th>
<th>0x12</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0]</td>
<td>1</td>
</tr>
<tr>
<td>[1]</td>
<td>0</td>
</tr>
<tr>
<td>[2]</td>
<td>1</td>
</tr>
<tr>
<td>[3]</td>
<td>0</td>
</tr>
<tr>
<td>[4]</td>
<td>0</td>
</tr>
<tr>
<td>[5]</td>
<td>0</td>
</tr>
<tr>
<td>[6]</td>
<td>1</td>
</tr>
<tr>
<td>[7]</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Byte2[7:0]</th>
<th>0x94</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0]</td>
<td>0</td>
</tr>
<tr>
<td>[1]</td>
<td>0</td>
</tr>
<tr>
<td>[2]</td>
<td>1</td>
</tr>
<tr>
<td>[3]</td>
<td>0</td>
</tr>
<tr>
<td>[4]</td>
<td>0</td>
</tr>
<tr>
<td>[5]</td>
<td>0</td>
</tr>
<tr>
<td>[6]</td>
<td>1</td>
</tr>
<tr>
<td>[7]</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>wr_mem_data[31:16]</th>
<th>0x46</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0]</td>
<td>0</td>
</tr>
<tr>
<td>[1]</td>
<td>1</td>
</tr>
<tr>
<td>[2]</td>
<td>2</td>
</tr>
<tr>
<td>[3]</td>
<td>3</td>
</tr>
<tr>
<td>[4]</td>
<td>4</td>
</tr>
<tr>
<td>[5]</td>
<td>5</td>
</tr>
<tr>
<td>[6]</td>
<td>6</td>
</tr>
<tr>
<td>[7]</td>
<td>7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>wr_mem_data[15:0]</th>
<th>0x12</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0]</td>
<td>0</td>
</tr>
<tr>
<td>[1]</td>
<td>1</td>
</tr>
<tr>
<td>[2]</td>
<td>2</td>
</tr>
<tr>
<td>[3]</td>
<td>3</td>
</tr>
<tr>
<td>[4]</td>
<td>4</td>
</tr>
<tr>
<td>[5]</td>
<td>5</td>
</tr>
<tr>
<td>[6]</td>
<td>6</td>
</tr>
<tr>
<td>[7]</td>
<td>7</td>
</tr>
</tbody>
</table>

| Note: |

**Related Information**
Generating Programming Files using Convert Programming Files in the Intel Stratix 10 Configuration User Guide

**Read Operation**

Complete the following steps to perform a read operation. The maximum read size is 4 KB (1000 words).

1. Request exclusive access to the flash memory using the **OPEN** command.
2. Select the flash device using the **CHIP_SELECT** command.
3. Specify the flash device address using the **READ_ADDR** command.
4. Specify the number using the READ_WORDS command.

5. Flush the read data FIFO before performing any read operations by writing $2b'10$ to the READ_OP command.

6. Start the read operation by transferring data from flash to the read data FIFO by writing $2'b01$ to the READ_OP command.

7. Poll the Rddata_valid of the ISR register to when the data stored in read data FIFO is ready to read. You can also read the fill level of the internal read data FIFO using the READ_FIFO_LEVEL command. You can also read the Cmd_err field of the ISR register to check the status of the read transaction. The ISR writes a value of one to the Cmd_err field of the ISR register if a read is unsuccessful. You can also check the Rsp_status field of the STATUS register.

8. Read the data stored in read data FIFO via read data interfaces.
   a. Assert the rd_mem_read signal while the rd_mem_waitrequest signal is low. If the rd_mem_waitrequest signal is high, the rd_mem_read signal must be kept high until the wr_mem_waitrequest signal goes low.
   b. Set the address value at the rd_mem_address bus.
      
      Note: Refer to the base address assigned to the rd_mem bus for the Serial Flash Mailbox Client Intel FPGA IP in the Intel Quartus Prime Platform Designer for the assigned addresses.
   c. Read the rd_mem_readdata bus if the rd_mem_readdatavalid signal is asserted.
   d. Repeat steps a to c to continuously read the data from the read data FIFO.
   e. De-assert the rd_mem_read signal once you have completed reading the data from the read data FIFO.
   f. Optional: You can read the fill level of the internal read data FIFO using the READ_FIFO_LEVEL command.

9. Repeat step 3 on page 12 to 8 on page 13 to continue to performing read operations.
   
   Note: You can check the STATUS register each time you send a command to ensure that the command completed successfully.
10. Release exclusive access to the flash device using the CLOSE command.

Design Example

The Serial Flash Mailbox Client Intel FPGA IP example design is available in the Intel Design Store. includes the following functions:

- Creates a flash image containing a configuration bitstream. This image includes the Serial Flash Mailbox Client Intel FPGA IP.
- Programs the flash memory using the configuration bitstream.
- Reads the status register of the flash memory device.
- Reads the flash memory device ID.
- Reads the flash memory device ID using the CONTROL command.
- Erases flash memory.
- Reads flash memory.
- Writes flash memory.

Related Information

Intel Stratix 10 Serial Flash Mailbox Client Intel FPGA IP Core Design Example
Prerequisites

You can create a very simple Intel Quartus Prime Pro Edition Platform Designer design example to exercise the Serial Flash Mailbox Client Intel FPGA IP. This design example must meet the following hardware and software requirements:

- You should be running the Intel Quartus Prime Pro Edition software version 18.0 or later.
- Your Platform Designer design example should include the components in the following figure:

**Figure 6. Required Communication and Host Components for the Serial Flash Mailbox Client Intel FPGA IP Design Example**

- Instantiate the JTAG to Avalon Master as host.
- Instantiate the Serial Flash Mailbox Client Intel FPGA IP.
- Connect the Serial Flash Mailbox Client Intel FPGA IP to JTAG to Avalon Master Bridge.
- Set base addresses for csr, rd_mem, and wr_mem.
- You should be using the Intel Stratix 10 SoC Development Kit.

Generating the Configuration Bitstream

Use the Programming File Generator to generate configuration bitstream.

1. On the Intel Quartus Prime File menu select **Programming File Generator**.
2. For **Device family** specify **Intel Stratix 10**.
3. For **Configuration mode** specify **Active Serial x4**
4. On the **Output Files** tab, select the **JTAG Indirect Configuration File (.jic)** output file type.
5. On the **Input Files** tab, click **Add Bitstream** and browse to your `.sof`.

6. On the **Configuration Device** tab select **Add Device** and select your device from the list of flash devices.
7. Click **Generate** to generate the configuration bitstream.

**Programming the Flash Memory with the Configuration Bitstream**

1. Open **Programmer** and click **Add File**. Select the generated JIC programming file (.jic) and click **Open**.
2. Check the **Program/Configure** check box for the attached .jic file.
3. To begin programming the flash memory with the configuration bitstream, click **Start**.
4. Configuration is complete when the progress bar reaches 100%. Power cycle the board. The Intel Stratix 10 device configures automatically using the Active Serial (AS) configuration scheme.
Reading the Flash Memory Device Status Register

Here are the steps to read the flash memory device status register.

```python
# Set base address according to Platform Designer system
set CSR 0x00000000
set WR_MEM 0x00000240
set RD_MEM 0x00000248
# Set the variables to their respective offset
set AsmiCmdStatus [expr $CSR + [expr 0x00 << 2]]
set AsmiIsr [expr $CSR + [expr 0x01 << 2]]
set AsmiIer [expr $CSR + [expr 0x02 << 2]]
set AsmiChipSelect [expr $CSR + [expr 0x03 << 2]]
set AsmiOpen [expr $CSR + [expr 0x04 << 2]]
set AsmiClose [expr $CSR + [expr 0x05 << 2]]
set AsmiWrEnable [expr $CSR + [expr 0x06 << 2]]
set AsmiWrStatus [expr $CSR + [expr 0x07 << 2]]
set AsmiRdStatus [expr $CSR + [expr 0x08 << 2]]
set AsmiSectorErase [expr $CSR + [expr 0x09 << 2]]
set AsmiDeviceId [expr $CSR + [expr 0x0a << 2]]
set AsmiControl [expr $CSR + [expr 0x0d << 2]]
set AsmiNumbByte [expr $CSR + [expr 0x0e << 2]]
set AsmiWriteData0 [expr $CSR + [expr 0x10 << 2]]
set AsmiWriteData1 [expr $CSR + [expr 0x11 << 2]]
set AsmiReadData0 [expr $CSR + [expr 0x12 << 2]]
set AsmiReadData1 [expr $CSR + [expr 0x13 << 2]]
set AsmiWriteOp [expr $CSR + [expr 0x14 << 2]]
set AsmiWriteAddr [expr $CSR + [expr 0x15 << 2]]
set AsmiWriteFifoLevel [expr $CSR + [expr 0x16 << 2]]
set AsmiReadOp [expr $CSR + [expr 0x17 << 2]]
set AsmiReadAddr [expr $CSR + [expr 0x18 << 2]]
set AsmiReadNumWords [expr $CSR + [expr 0x19 << 2]]
set AsmiReadFifoLevel [expr $CSR + [expr 0x1a << 2]]

# Assign variable "m" to the string that is the 0th element in the list
# returned by get_service_paths master
set m [lindex [get_service_paths master] 0]
# Open the connection to the master module
open_service master $m
# Assign variable "m" to the string that is the 0th element in the list
# returned by get_service_paths master
set m [lindex [get_service_paths master] 0]
# Open the connection to the master module
open_service master $m

# Write Offset 4 to request access to the flash memory device.
master_write_32 $m $AsmiOpen 0x1

# Write Offset 3 to select the 1st flash memory device attached to the IP.
master_write_32 $m $AsmiChipSelect 0x0

# Read Offset 8 for status register of the device.
master_read_32 $m $AsmiRdStatus 1

# Write Offset 5 to close access to the flash memory device.
master_write_32 $m $AsmiClose 0x1
```

Reading the Flash Memory Device ID

Here are the steps to read the Device ID from a flash memory device:

```python
# Write Offset 4 to request access to the flash memory device.
master_write_32 $m $AsmiOpen 0x1
```
Reading the Flash Memory Device ID Using the Control Command

Here are the steps to read the flash memory device ID using the CONTROL command:

- Write Offset 4 to request access to the flash memory device.
  ```
  master_write_32 $m $AsmiOpen 0x1
  ```
- Write Offset 3 to select the 1st flash memory device attached to the IP.
  ```
  master_write_32 $m $AsmiChipSelect 0x0
  ```
- Writing the command argument to Offset 14 (specify the number bytes to 0x4)
  ```
  master_write_32 $m $AsmiNumbByte 0x4
  ```
- Writing the command argument to Offset 13 (the opcode to read device ID is 0xAF000041)
  ```
  master_write_32 $m $AsmiControl 0xAF000041
  ```
- Read Offset 17 to determine the lower 4 bytes of read data where the device ID obtained from control command is stored at.
  ```
  master_read_32 $m $AsmiReadData0 1
  ```
- Write Offset 5 to close access to the flash memory device.
  ```
  master_write_32 $m $AsmiClose 0x1
  ```

Erasing Flash Memory

Here are the steps to erase flash memory:

- Write Offset 4 to request access to the flash memory device.
  ```
  master_write_32 $m $AsmiOpen 0x1
  ```
- Write Offset 3 to select the 1st flash memory device attached to the IP.
  ```
  master_write_32 $m $AsmiChipSelect 0x0
  ```
- Write Offset 6 to perform write enable operation to the device.
  ```
  master_write_32 $m $AsmiWrEnable 0x1
  ```
- Writing the address argument to Offset 9 (Perform Sector Erase on address 0x03FF0000)
  ```
  master_write_32 $m $AsmiSectorErase 0x03FF0000
  ```
- Write Offset 5 to close access to the flash memory device.
  ```
  master_write_32 $m $AsmiClose 0x1
  ```

Reading Flash Memory

Here are the steps to read flash memory:

- Write Offset 4 to request access to the flash memory device.
  ```
  master_write_32 $m $AsmiOpen 0x1
  ```
- Write Offset 3 to select the 1st flash memory device attached to the IP.
  ```
  master_write_32 $m $AsmiChipSelect 0x0
  ```
Writing the address argument to Offset 24 (specify the device address for read operation to 0x03FF0000)
master_write_32 $m $AsmiReadAddr 0x03FF0000

Writing the command argument to Offset 25 (specify the number of words to read from device is 1)
master_write_32 $m $AsmiReadNumbWords 0x1

Writing the command argument to Offset 23 (Specify 0x2 to flush out data inside read FIFO)
master_write_32 $m $AsmiReadOp 0x2

Writing the command argument to Offset 23 (Specify 0x1 to perform read operation)
master_write_32 $m $AsmiReadOp 0x1

Read Offset 26 to determine the fill level of the internal read data FIFO.
master_read_32 $m $AsmiReadFifoLevel 1

Read the data stored in read data FIFO via the base address of rd_mem in the IP.
master_read_32 $m $RD_MEM 1

Write Offset 5 to close access to the flash memory device.
master_write_32 $m $AsmiClose 0x1

---

Writing Flash Memory

Here are the steps to write to flash memory:

Write Offset 4 to request access to the flash memory device.
master_write_32 $m $AsmiOpen 0x1

Write Offset 3 to select the 1st flash memory device attached to the IP.
master_write_32 $m $AsmiChipSelect 0x0

Write Offset 6 to perform write enable operation to the device.
master_write_32 $m $AsmiWrEnable 0x1

Writing the command argument to Offset 20 (Specify 0x2 to flush out data inside write FIFO)
master_write_32 $m $AsmiWriteOp 0x2

Pre-store the data that you want to write into flash memory in write data FIFO via the base address of wr_mem in the IP (Specify 0x11223344 to write into write data FIFO)
master_write_32 $m $WR_MEM 0x11223344

Read Offset 22 to determine the fill level of the internal write data FIFO.
master_read_32 $m $AsmiWriteFifoLevel 1

Read the address argument to Offset 21 (specify the device address for write operation to 0x03FF0000)
master_write_32 $m $AsmiWriteAddr 0x03FF0000

Writing the command argument to Offset 20 (Specify 0x1 to perform write operation)
master_write_32 $m $AsmiWriteOp 0x1

Write Offset 5 to close access to the flash memory device.
master_write_32 $m $AsmiClose 0x1
Serial Flash Mailbox Client Intel FPGA IP Core User Guide Archives

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>Serial Flash Mailbox Client Intel FPGA IP</td>
</tr>
<tr>
<td>18.0</td>
<td>Serial Flash Mailbox Client Intel FPGA IP</td>
</tr>
</tbody>
</table>
## Document Revision History for the Serial Flash Mailbox Client Intel FPGA IP User Guide

<table>
<thead>
<tr>
<th>Document Version</th>
<th>Intel Quartus Prime Version</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>2019.12.11</td>
<td>19.3</td>
<td>Corrected definition of the SECTOR_ERASE command. This command performs a 64 KB erase, regardless of the actual sector size for a particular flash vendor.</td>
</tr>
<tr>
<td>2019.11.26</td>
<td>19.3</td>
<td>Removed Operations topic. The Serial Flash Mailbox Client Intel FPGA IP User Guide does not support these commands.</td>
</tr>
</tbody>
</table>
| 2019.09.30       | 19.3                        | Made the following changes:  
  - Changed the name of this IP from Stratix 10 Serial Flash Mailbox Client Intel FPGA IP to Serial Flash Mailbox Client Intel FPGA IP.  
  - Added support for Micron and Macronix flash devices.  
  - Added Operation Commands topic covering the quad SPI commands available to access flash memory devices.  
  - Added the missing irq signal description.  
  - Added Serial Flash Mailbox Client topic.  
  - Made the following changes to the Write Operation and Read Operation steps:  
    1. The operations include the QSPI_OPEN and QSPI_SET_CS commands before specifying the flash address.  
    2. The operations include the QSPI_CLOSE command after completing the write or read.  
  - Corrected bit ordering for the SECTOR_ERASE and RD_DEVICE_ID commands. The correct bit ordering is [31:0].  
  - Made the sector erase command mandatory for write operations.  
  - Removed references to electrically programmable configuration quad-serial low voltage (EPCQ-L) devices. As of 2018, EPCQ-L are obsolete.  
  - Added an example to illustrate bit swapping for the .rpd format which is little endian.  
  - Edited the entire user guide for clarity and style.  
  - Corrected minor errors and typos. |
| 2019.05.17       | 19.1                        | Updated Table: Signal Description to add a note regarding IP core instantiation guidelines to the reset signal.  
  - Added the Stratix 10 Serial Flash Mailbox Client Intel FPGA IP Archives topic. |
| 2019.04.01       | 19.1                        | Updated Table: Register Map and Definitions to update the field name for ISR from Cmd_err1 to Cmd_err. |
| 2018.12.24       | 18.0                        | Added a design example section.  
  - Updated Table: Register Map and Definitions.  
  - Updated Table: Stratix 10 Serial Flash Mailbox Client Intel FPGA IP Response Codes.  
  - Corrected minor typographical errors. |
| 2018.05.07       | 18.0                        | Initial release. |