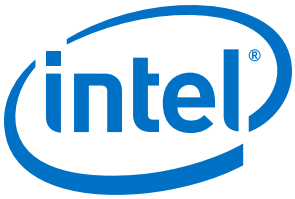


Generic Serial Flash Interface Intel FPGA IP Core User Guide

Updated for Intel® Quartus® Prime Design Suite: **18.1**


Subscribe


Send Feedback

UG-20161 | 2018.11.09
Latest document on the web: **PDF** | **HTML**

Contents

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 IP to write the following data to the flash device:

- Configuration memory(1)—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 (Avalon-MM) 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 11
- 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 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

Related Information

Configuration Devices
Provides more information about the third-party flash support.

1.2. Signals

Figure 1. Signal Block Diagram

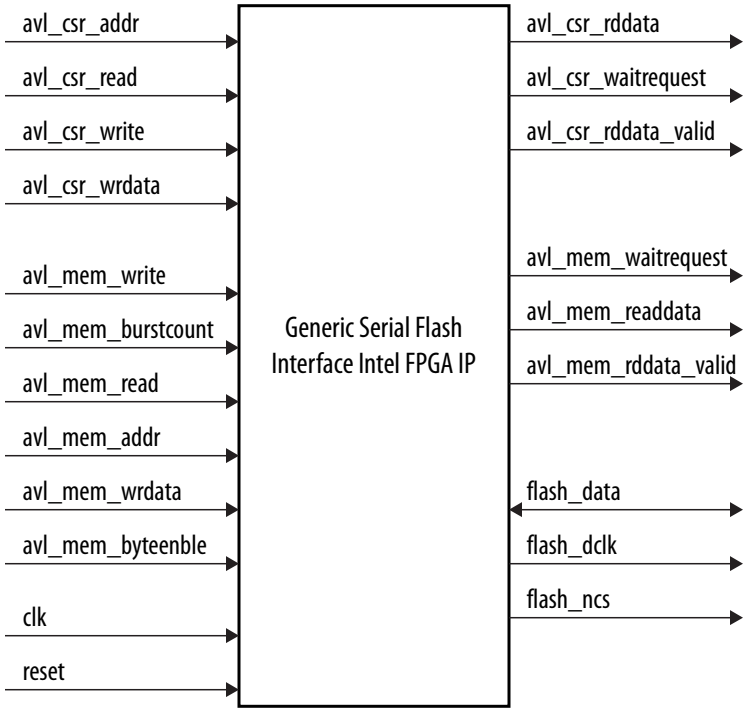


Table 1. Ports Description

Signal	Width	Direction	Description
Avalon®-MM slave interface for CSR (avl_csr)			
avl_csr_addr	6	Input	Avalon-MM address bus. The address bus is in word addressing.
avl_csr_read	1	Input	Avalon-MM read control to the CSR.
avl_csr_rddata	32	Output	Avalon-MM read data bus from the CSR.
avl_csr_write	1	Input	Avalon-MM write control to the CSR.
avl_csr_wrdata	32	Input	Avalon-MM write data bus to CSR.
avl_csr_waitrequest	1	Output	Avalon-MM waitrequest control from the CSR
avl_csr_rddata_valid	1	Output	Avalon-MM read data valid that indicates the CSR read data is available.
Avalon-MM slave interface for memory access (avl_ mem)			
avl_mem_write	1	Input	Avalon-MM write control to the memory
avl_mem_burstcount	7	Input	Avalon-MM burst count for the memory. The value range from 1 to 64 (Max page size).
avl_mem_waitrequest	1	Output	Avalon-MM waitrequest control from the memory.
avl_mem_read	1	Input	Avalon-MM read control to the memory
avl_mem_addr	N	Input	Avalon-MM 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-MM write data bus to the memory
avl_mem_readddata	32	Output	Avalon-MM read data bus from the memory.
avl_mem_rddata_valid	1	Output	Avalon-MM read data valid that indicates the memory read data is available.
avl_mem_byteenble	4	Input	Avalon-MM 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.

(2) Available when you enable the **Enable SPI pins interface** parameter.

1.3. Parameters

Table 2. Parameter Settings

Parameter	Legal Values	Descriptions
Device Density	1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048	Density of the flash device used in Mb.
Disable dedicated Active Serial interface	—	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.
Enable SPI pins interface	—	Translates the signals to the SPI pin interface.
Number of Chip Select used	1 2 3	Selects the number of chip select connected to the flash.
Enable flash simulation model	—	Uses the flash inside the device for simulation model.

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

Offset (Hex)	Register Name	R/W	Field Name	Bit	Default Value (Hex)	Description
0	Control Register	Reserved		31:8	Reserved	
		R/W	Addressing mode	8	0x0	Addressing mode for read and write operation: • 0x0: 3-bytes addressing • 0x1: 4-bytes addressing For 4-byte addressing mode, you must enable 4-byte address by sending command to the flash. This bit affects direct access to memory via the Avalon-MM interface for both write and read operation.
		R/W	Chip select	7:4	0x0	Selects the flash device • 0x0: To select first device • 0x1: To select second device • 0x2: To select third device
		Reserved		3:1	Reserved	
		R/W	Enable	0	0x1	Set this bit to 0 to disable the output of the IP and put all output signal to high impedance state. This can be used to share bus with other devices.
1	SPI Clock Baud-rate Register	Reserved		31:5	Reserved	

continued...

Offset (Hex)	Register Name	R/W	Field Name	Bit	Default Value (Hex)	Description
		R/W	Baud rate divisor	4:0	0x10	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 run 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: • 0x1 : /2 • 0x2 : /4 • 0x3 : /6 • ... • 0xF : /30 • 0x10 : /32
2	CS Delay Setting Register	Reserved		31:8	Reserved	
		R/W	CS de-assert	7:4	0x0	Sets the chip select de-assertion delay. • 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
		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	Reserved		31:4	Reserved	
		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	Reserved		31:18	Reserved	
		R/W	Read data out transfer mode	17:16	0x0	Transfer mode for read data output.
		Reserved		15:14	Reserved	
		R/W	Read address transfer mode	13:12	0x0	Transfer mode for read address input Description as bit 1:0.

continued...

(3) Intel recommends that you set the chip select assertion delay to 5 if you are running the IP clock at 100 MHz.

Offset (Hex)	Register Name	R/W	Field Name	Bit	Default Value (Hex)	Description
		Reserved		11:10	Reserved	
		R/W	Write Data in transfer mode	9:8	0x0	Transfer mode for write data input Description as bit 1:0.
		Reserved		7:6	Reserved	
		R/W	Write address transfer mode	5:4	0x0	Transfer mode for write address input Description as bit 1:0.
		Reserved		3:2	Reserved	
		R/W	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] 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.
5	Read Instruction Register	Reserved		31:14	Reserved	
		R/W	Dummy cycles	12:8	0xA	Number of default dummy cycles used for read operation. Refer to the respective flash device datasheet.
		R/W	Read opcode	7:0	0x03	The opcode for read operation. Refer to the respective flash device datasheet to select the correct opcode according to the transfer mode setting.
6	Write Instruction Register	Reserved		31:16	Reserved	
		R/W	Polling opcode	15:8	0x05	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-MM interface. In applicable devices, you can set as the status register or flag status register.
		R/W	Write opcode	7:0	0x02	The opcode for write operation. Refer to the respective flash device datasheet to select the correct opcode according to the transfer mode setting.
7	Flash Command Setting Register(4)	Reserved		31:21	Reserved	
		R/W	Number of dummy cycles	20:16	0x0	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.

continued...

(4) Default setting is for read status command.

Offset (Hex)	Register Name	R/W	Field Name	Bit	Default Value (Hex)	Description
		R/W	Number of data bytes	15:12	0x08	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.
		R/W	Data type	11	0x01	Indicates the type of data (bit [15:12]). • 0: Number of byte declared in [15:12] is write data to flash device • 1: Number of byte declared in [15:12] is read data from flash device
		R/W	Number of address bytes	10:8	0x0	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.
		R/W	Opcode	7:0	0x05	The opcode of the operation.
8	Flash Command Control Register	Reserved		31:1	Reserved	
		W	Start	0	0x0	Write 1 to this bit to start the operation.
9	Flash Command Address Register	R/W	Stating address	31:0	31:0	Address of flash command.
A	Flash Command Write Data 0 Register	R/W	Lower 4 bytes write data	31:0	0x0	The first 4-byte of write data to flash device.
B	Flash Command Write Data 1 Register	R/W	Upper 4 bytes write data	31:0	0x0	The last 4-byte of write data to the flash device.
C	Flash Command Read Data 0 Register	R	Lower 4 bytes read data	31:0	0x0	The first 4-byte of read data from flash device.
D	Flash Command Read Data 1 Register	R	Upper 4 bytes read data	31:0	0x0	The last 4-byte of read data from the flash device.

1.5. Using Generic Serial Flash Interface IP

The core interfaces are Avalon-MM 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 to 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-MM operations. You must set the correct at 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

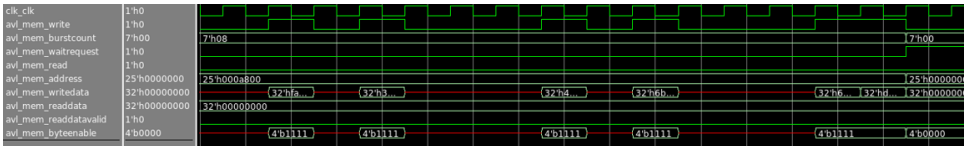


Figure 3. 8-Word Reading Burst Waveform Example

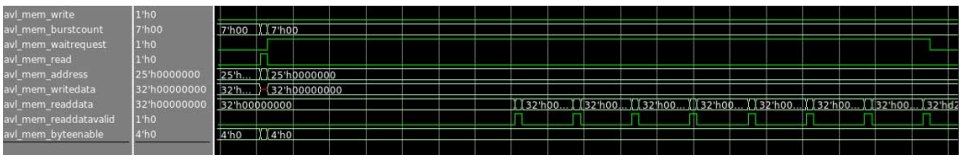
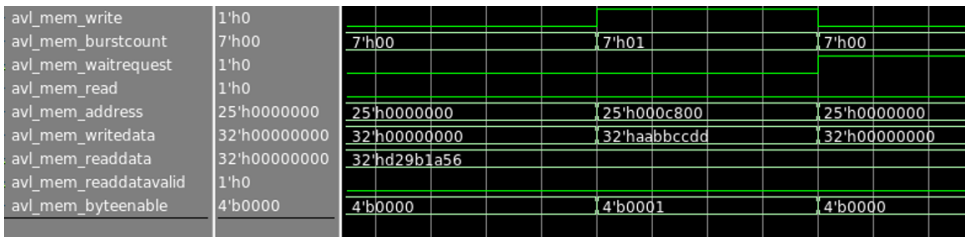


Figure 4. 1-Byte Write `byteenable = 4'b0001` Waveform Example



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 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

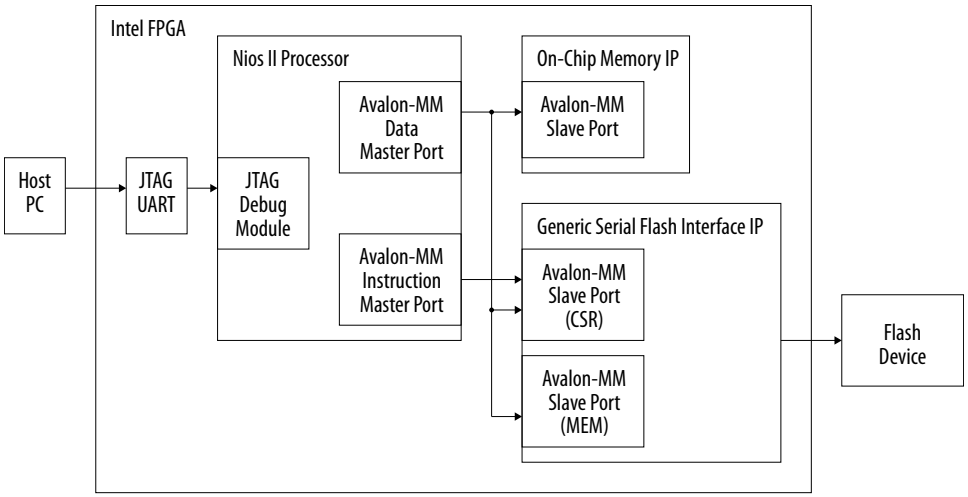
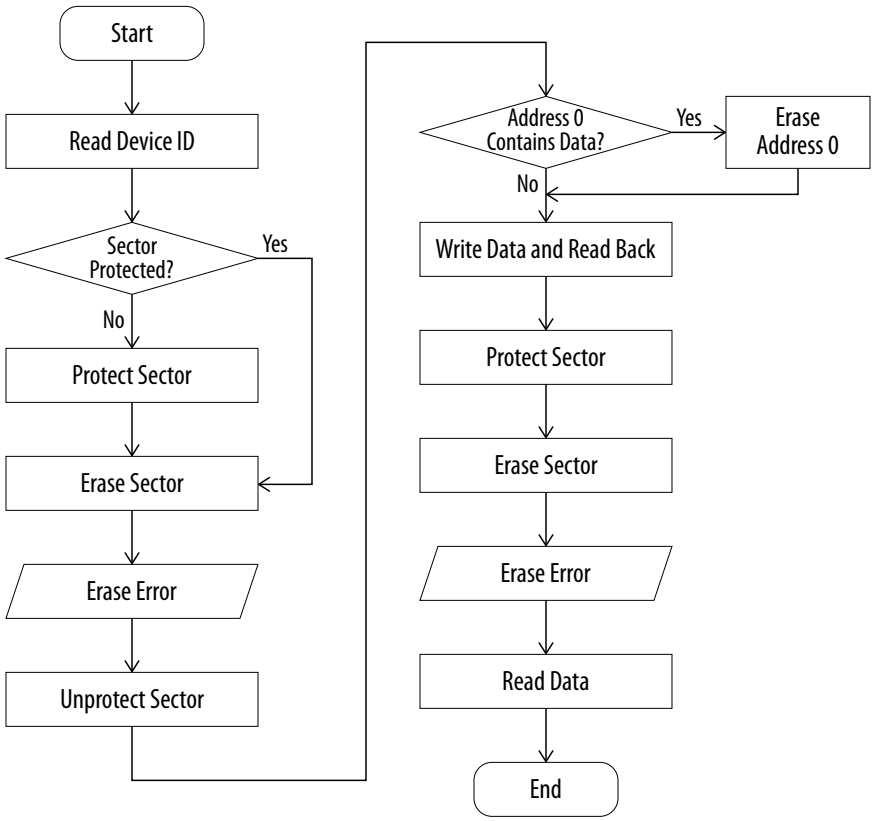


Table 4. Reference Design Components Descriptions

Component	Description
JTAG UART Intel FPGA IP	Enables communication between the Nios II processor and the host computer
Nios II Processor	Run application program by executing data and instruction
On-Chip Memory Intel FPGA IP	Store code and data
Generic Serial Flash Interface Intel FPGA IP	Controls vendor-independent flash device to perform flash interaction

1.6.2.2. Reference Design Application Program

Figure 6. Reference Design Application Program Flow Diagram



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:

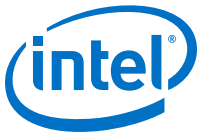
i. Performs write enable command.
ii. Performs sector erase command.
iii. Polls write in progress (WIP) bit (bit 0 of status register) until it return a 0 (ready).
iv. Performs read status register to check whether erase operation succeeded or failed.

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:

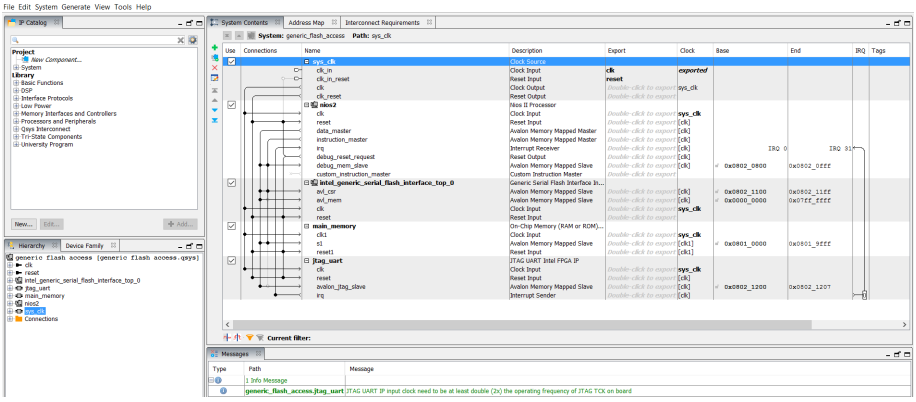
a. Double click the Nios II processor `nios2`.

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.

Figure 7. Completed Platform Designer Connection



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:

```
Flash Device: Cypress flash S70FL01G
Device ID: 4d210201
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 sector is protected!
Now perform sector unprotect...
Sector unprotect successfully! :)
Reading data at address 0...
Memory content at address 0: abcd1234
Trying to erase sector 0...
Sector erase successfully. Sector 0 is now empty.
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.
```

Micron MT25Q01G:

```
Flash Device: Micron flash MT25Q01G
Device ID: 1021ba20
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...
Sector unprotect successfully! :)
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.
```

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...
Sector unprotect successfully! :)
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.
```

EPCQ256:

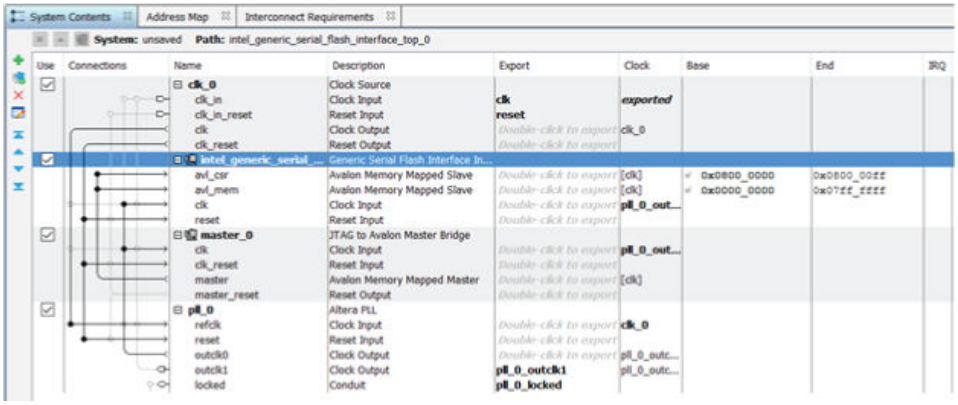
```
Flash Device: EPCQ256
Device ID: 1019ba20
All sectors in this flash device is not protected
Now performing sector protection...
All sectors in this flash device is now successfully protected
Trying to erase sector 0...
Erase Error as erase is not allow during sector is protected!
Now perform sector unprotect...
Sector unprotect successfully! :)
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 Intel FPGA IP core to perform flash access. To begin, build the Platform Designer system with a few components (`clock`, `jtag master`, `pll`, and this Intel FPGA 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 Intel FPGA 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

```
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

```
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

```
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

```
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 Intel FPGA IP core supports flash in the extended, dual, and quad I/O protocols. Currently, the protocols supported by this Intel FPGA IP core is a single-transfer rate (STR) only. This Intel FPGA 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)

```
proc read { } {

global mp operating_protocols_setting control_register read_instr

master_write_32 $mp $operating_protocols_setting 0x000000000

master_write_32 $mp $control_register 0x00000001

master_write_32 $mp $read_instr 0x00000003

master_read_32 $mp 0x0100000 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)

```
proc dual_output_fast_read { } {

global mp operating_protocols_setting control_register read_instr

master_write_32 $mp $operating_protocols_setting 0x00011001

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)

```
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 Intel FPGA IP core releases the wait request of the Avalon–MM 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)

```
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 0x00000101
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 Intel FPGA IP core releases the wait request of the Avalon–MM 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 Core User Guide Archives

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

IP Core Version	User Guide
18.0	Generic Serial Flash Interface Intel FPGA IP Core User Guide

1.9. Document Revision History for the Generic Serial Flash Interface Intel FPGA IP Core User Guide

Document Version	Intel Quartus Prime Version	Changes
2018.11.09	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	• 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	Initial release.