Nios® V Embedded Processor Design Handbook

Updated for Intel® Quartus® Prime Design Suite: **23.4**

Online Version
Send Feedback

726952
2023.12.04

Contents

1. About the Nios® V Embedded Processor

1.1. Intel® FPGA and Embedded Processors Overview

Intel FPGA devices can implement logic that functions as a complete microprocessor while providing many options.

An important difference between discrete microprocessors and Intel FPGA is that Intel FPGA fabric contains no logic when it powers up. Before you run software on a Nios® V processor based system, you must configure the Intel FPGA device with a hardware design that contains a Nios V processor. The Nios V processor is a soft intellectual property (IP) processor based on the RISC-V specification. You can place the Nios V processor anywhere on the Intel FPGA, depending on the requirements of the design.

To enable your Intel® FPGA IP-based embedded system to behave as a discrete microprocessor-based system, your system should include the following:

- A JTAG interface to support Intel FPGA configuration, hardware and software debugging
- A power-up Intel FPGA configuration mechanism

If your system has these capabilities, you can begin refining your design from a pretested hardware design loaded in the Intel FPGA. Using an Intel FPGA also allows you to modify your design quickly to address problems or to add new functionality. You can test these new hardware designs easily by reconfiguring the Intel FPGA using your system's JTAG interface.

The JTAG interface supports hardware and software development. You can perform the following tasks using the JTAG interface:

- Configure the Intel FPGA
- Download and debug software
- Communicate with the Intel FPGA through a UART-like interface (JTAG UART terminal)
- Debug hardware (with the Signal Tap embedded logic analyzer)
- Program flash memory

After you configure the Intel FPGA with a Nios V processor-based design, the software development flow is similar to the flow for discrete microcontroller designs.

Related Information

- Nios V Processor Reference Manual
  Provides information about the Nios V processor performance benchmarks, processor architecture, the programming model, and the core implementation.
- Embedded Peripherals IP User Guide

- Nios V Processor Software Developer Handbook
  Describes the Nios V processor software development environment, the tools that are available, and the process to build software to run on Nios V processor.
- Ashling* RiscFree* Integrated Development Environment (IDE) for Intel FPGAs User Guide
  Describes the RiscFree* integrated development environment (IDE) for Intel FPGAs Arm*-based HPS and Nios V core processor.
- Nios V Processor Intel FPGA IP Release Notes

1.2. Intel Quartus® Prime Software Support

Nios V processor build flow is different for Intel Quartus® Prime Pro Edition software and Intel Quartus Prime Standard Edition software. Refer to *AN 980: Nios V Processor Intel Quartus Prime Software Support* for more information about the differences.

Related Information

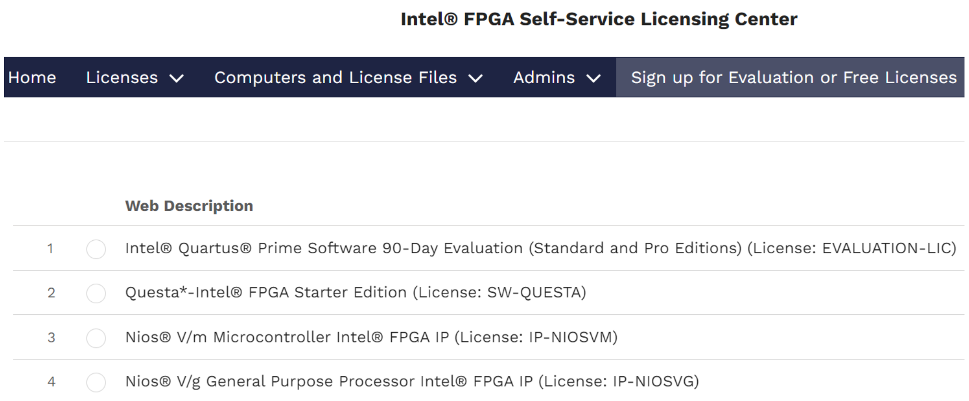
AN 980: Nios V Processor Intel Quartus Prime Software Support

1.3. Nios V Processor Licensing

Each Nios V processor core has its license key. You can acquire the Nios V Processor Intel FPGA IP licenses at zero cost.

The Nios V processor license key list is available in the Intel FPGA Self-Service Licensing Center. Click the **Sign up for Evaluation or Free License** tab, and select the corresponding options to make the request.

Figure 1. Intel FPGA Self-Service Licensing Center


Intel® FPGA Self-Service Licensing Center
Home
Licenses
Computers and License Files
Admins
Sign up for Evaluation or Free Licenses

		Web Description
1	○	Intel® Quartus® Prime Software 90-Day Evaluation (Standard and Pro Editions) (License: EVALUATION-LIC)
2	○	Questa*-Intel® FPGA Starter Edition (License: SW-QUESTA)
3	○	Nios® V/m Microcontroller Intel® FPGA IP (License: IP-NIOSVM)
4	○	Nios® V/g General Purpose Processor Intel® FPGA IP (License: IP-NIOSVG)

With the license keys, you can:

- Implement a Nios V processor within your system.
- Simulate the behavior of a Nios V processor system.
- Verify the functionality of the design, such as size and speed.
- Generate device programming files.
- Program a device and verify the design in hardware.

You do not need a license to develop software in the Ashling* RiscFree* IDE for Intel FPGAs.

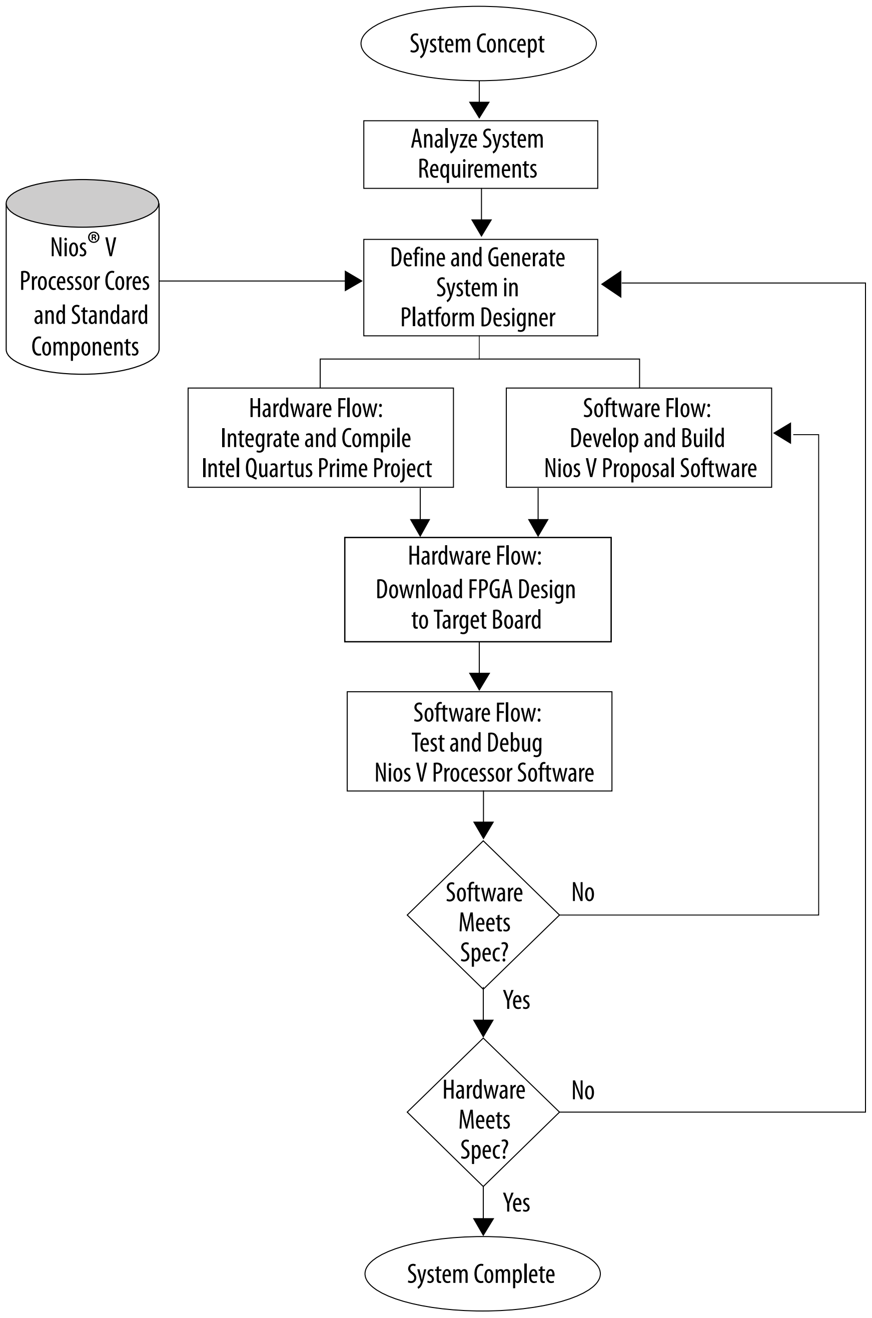
Related Information

- Intel FPGA Self-Service Licensing Center
  For more information about obtaining the Nios V Processor Intel FPGA IP license keys.
- Intel FPGA Software Installation and Licensing
  For more information about licensing the Intel FPGA software and setting up a fixed license and network license server.

1.4. Embedded System Design

The following figure illustrates a simplified Nios V processor based system design flow, including both hardware and software development.

Figure 2. Nios V Processor System Design Flow


System Concept
Analyze System Requirements
Nios® V Processor Cores and Standard Components
Define and Generate System in Platform Designer
Hardware Flow: Integrate and Compile Intel Quartus Prime Project
Software Flow: Develop and Build Nios V Proposal Software
Hardware Flow: Download FPGA Design to Target Board
Software Flow: Test and Debug Nios V Processor Software
Software Meets Spec?
No
Yes
Hardware Meets Spec?
No
Yes
System Complete

1.5. Nios V Processor Quick Start Guide

1.5.1. System Requirements

1.5.1.1. Hardware and Software Requirements

Intel uses the following hardware and software to build a Nios V processor system:

- Supported Intel FPGA devices. For more information about the supported devices, refer to the Table: *Nios V Processor Core and Device Support* in *AN 980: Nios V Processor Intel Quartus Prime Software Support*.
- Intel Quartus Prime software
  — Intel Quartus Prime Pro Edition software version 21.3 or later
  — Intel Quartus Prime Standard Edition software version 22.1 or later
- Ashling RiscFree IDE for Intel FPGAs

Note: Intel recommends you to install the same software version for all software.

You need to acquire the free license for the Nios V processor to compile the design in Intel Quartus Prime software.

Related Information

- Getting Started with Nios V Processor
  For more information about acquiring the license for the Nios V/m processor.
- AN 980: Nios V Processor Intel Quartus Prime Software Support

1.5.2. Nios V Processor Example Design

Note: For Intel Quartus Prime Standard Edition software, refer to *AN 980: Nios V Processor Intel Quartus Prime Software Support* for the steps to generate the example design.

To acquire the Nios V processor example design using Intel Quartus Prime Pro Edition software, follow these steps:

1. Go to Intel FPGA Design Store.
2. Search for *Arria10 - NIOSV based Helloworld example design on Arria10 devkit* package.
3. Click on the link at the title.
4. Accept the *Software License Agreement*.
5. Download the package according to the Intel Quartus Prime software version of your host machine.
6. Double-click to run the `top.par` file.
7. `top_project` folder is created by default after running the PAR file.
8. Open the `top_project` and refer to the `readme.txt` for how-to guide.

Table 1. Example Design File Description

File	Description
`hw/`	Contains files necessary to run the hardware project.
`ready_to_test/`	Contains pre-built hardware and software binaries to run the design on the target hardware. For this package, the target hardware is Intel Arria® 10 SoC development kit.
`scripts/`	Consists of scripts to build the design.
`sw/`	Contains software application files.
`readme.txt`	Contains description and steps to apply the pre-bulit binaries or rebuild the binaries from scratch.

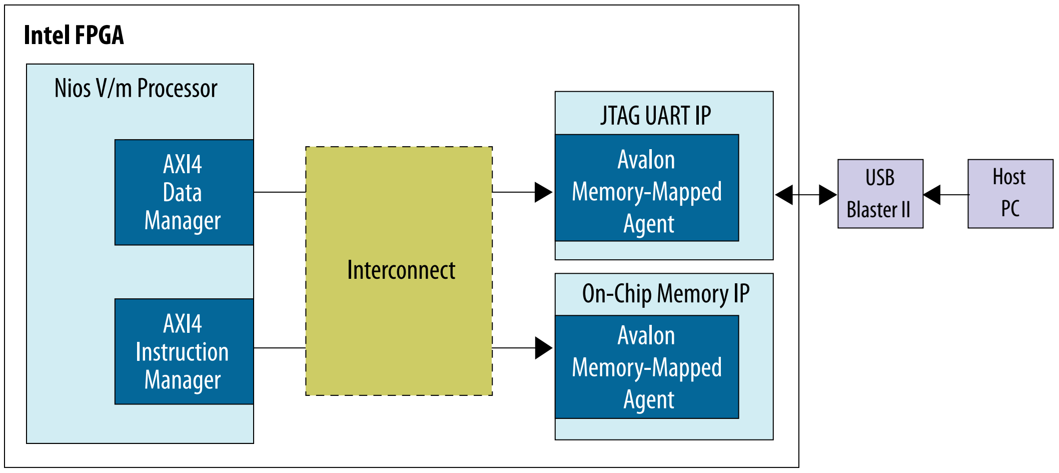
9. The Platform Designer system in the hw folder is ready for Software Design Flow.

The system is built using the IP blocks described in the following table and shown in the following figure.

Table 2. Component Description

Components	Description
Nios V/m Processor Intel FPGA IP	Runs application by executing instructions.
JTAG UART Intel FPGA IP	Enables serial character communication between Nios V/m processor and host computer.
On-Chip Memory Intel FPGA IP	Stores data and instructions.

Figure 3. Nios V/m Example Design Block Diagram



Note: For other Intel FPGA boards, you can modify the design to target your board by configuring the following settings:

- Target device setting from **Assignments ➤ Device...**
- Clock pin setting in the **Assignment Editor**

Related Information

- AN 980: Nios V Processor Intel Quartus Prime Software Support
- Intel FPGA Design Store

1.5.3. Software Design Flow

After generating the Nios V processor hardware system in Platform Designer, you need to create a Nios V processor software system. A basic software system comprises of a Board Support Package (BSP) and Application Project File (APP).

You can apply the pre-built BSP and APP projects in the sw folder. The BSP and APP projects are compatible with the Platform Designer system in the hw folder. The pre-built projects support the following operating systems:

- Intel HAL (`*_hal`)
- Micrium MicroC/OS-II (`*_ucosii`)

For custom BSP or APP projects, refer to Generating the Board Support Package and Generating the Application Project File.

Related Information

Nios V Processor Software System Design on page 32
For more information about RiscFree* IDE for Intel FPGA and Eclipse Embedded CDT.

1.5.3.1. Generating the Board Support Package

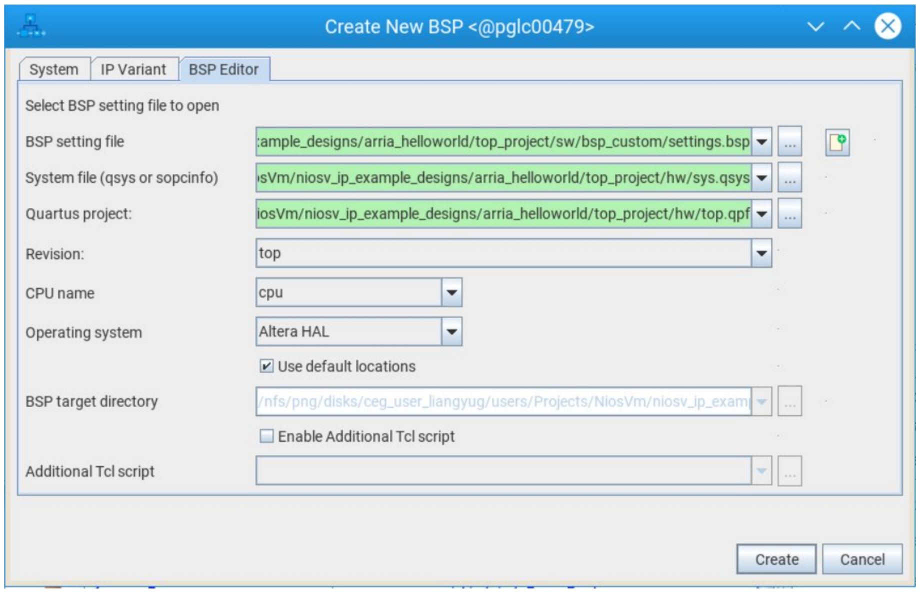
1.5.3.1.1. Generating the Board Support Package using the BSP Editor GUI

Platform Designer includes the BSP Editor board support package editing tool. A board support package (BSP) provides a software runtime environment for embedded systems, such as Nios V processor systems. The BSP Editor is a GUI tool that you can launch from Platform Designer to generate and configure BSP contents.

1. In the Intel Quartus Prime software, go to **Tools ➤ Platform Designer**.
2. In the Platform Designer window, go to **File ➤ New BSP**. The **Create New BSP** window appears.
3. For **BSP setting file**, create a BSP file (`settings.bsp`) in `<Project directory>/sw/bsp_custom/settings.bsp`.
4. For **System file (qsys or sopcinfo)**, select the Nios V processor Platform Designer system (`sys.qsys`).

   Note: For Intel Quartus Prime Standard Edition software, generate the BSP files using SOPCINFO file. Refer to *AN 980: Nios V Processor Intel Quartus Prime Software Support* for more information.
5. For **Quartus project**, select the example design Quartus Project File (`top.qpf`).
6. For **Revision**, select **top**.
7. For **CPU name**, select **cpu**.
8. Select the **Operating system** as **Altera HAL** or **Micrium MicroC/OS II**.
9. Click **Create** to create the BSP file.

Figure 4. Create New BSP window


Create New BSP <@pglc00479>

System | IP Variant | BSP Editor

Select BSP setting file to open

BSP setting file: :ample_designs/arria_helloworld/top_project/sw/bsp_custom/settings.bsp

System file (qsys or sopcinfo): ›sVm/niosv_ip_example_designs/arria_helloworld/top_project/hw/sys.qsys

Quartus project: iosVm/niosv_ip_example_designs/arria_helloworld/top_project/hw/top.qpf

Revision: top

CPU name: cpu

Operating system: Altera HAL

☑ Use default locations

BSP target directory: /nfs/png/disks/ceg_user_liangyug/users/Projects/NiosVm/niosv_ip_exam

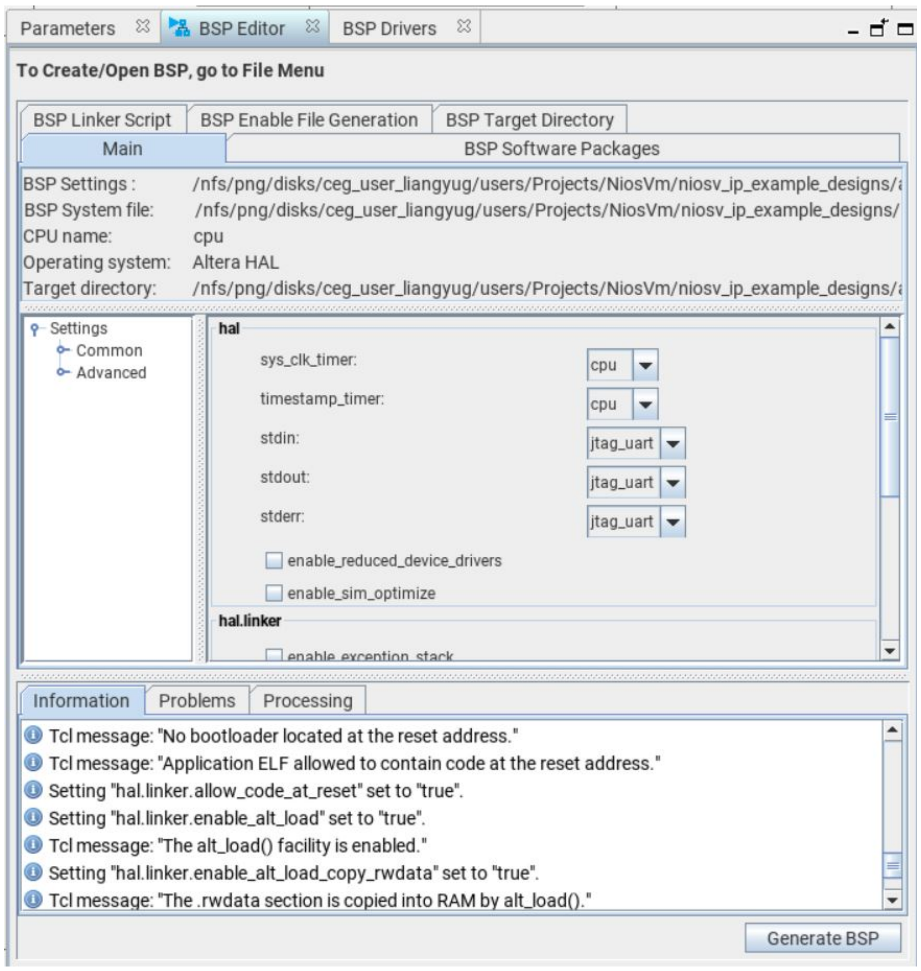
☐ Enable Additional Tcl script

Additional Tcl script:

Create | Cancel

10. Click **Generate BSP** to generate the BSP file.

Figure 5. BSP Editor



To Create/Open BSP, go to File Menu

BSP Settings : /nfs/png/disks/ceg_user_liangyug/users/Projects/NiosVm/niosv_ip_example_designs/
BSP System file: /nfs/png/disks/ceg_user_liangyug/users/Projects/NiosVm/niosv_ip_example_designs/
CPU name: cpu
Operating system: Altera HAL
Target directory: /nfs/png/disks/ceg_user_liangyug/users/Projects/NiosVm/niosv_ip_example_designs/

hal

sys_clk_timer: cpu
timestamp_timer: cpu
stdin: jtag_uart
stdout: jtag_uart
stderr: jtag_uart
☐ enable_reduced_device_drivers
☐ enable_sim_optimize

hal.linker

☐ enable_exception_stack

```
Tcl message: "No bootloader located at the reset address."
Tcl message: "Application ELF allowed to contain code at the reset address."
Setting "hal.linker.allow_code_at_reset" set to "true".
Setting "hal.linker.enable_alt_load" set to "true".
Tcl message: "The alt_load() facility is enabled."
Setting "hal.linker.enable_alt_load_copy_rwdata" set to "true".
Tcl message: "The .rwdata section is copied into RAM by alt_load()."
```

Note: In the **BSP Editor**, the default selection for **sys_clk_timer** and **timestamp_timer** are configured to **cpu** to use the Nios V processor's internal timer.

Related Information

- Intel Quartus Prime Pro Edition User Guide: Platform Designer
  More information about Creating a Board Support Package with BSP Editor.
- Nios V Processor Software Developer Handbook: Board Support Package Editor
- AN 980: Nios V Processor Intel Quartus Prime Software Support
- Intel FPGA Design Store

1.5.3.1.2. Generating the Board Support Package using the Command-Line Interface

You can also generate the BSP file using the following step:

1. Launch the Nios V Command Shell
2. Based on your Intel Quartus Prime version, execute the following CLI command to generate the BSP file. Select the **type** as **hal**, **ucosii**, or **freertos**.

- In Intel Quartus Prime Pro Edition software:

  ```
  niosv-bsp -c --quartus-project=hw/top.qpf --qsys=hw/sys.qsys \
  --type=<hal, ucosii, or freertos> sw/bsp_custom/settings.bsp
  ```

- In Intel Quartus Prime Standard Edition software:

  ```
  niosv-bsp -c --quartus-project=hw/top.qpf --sopcinfo=hw/sys.sopcinfo \
  --type=<hal, ucosii, or freertos> sw/bsp_custom/settings.bsp
  ```

1.5.3.2. Generating the Application Project File

You can find the application source files at the following locations:

- Intel HAL application source file `hello.c` in the `sw/app` folder
- The Micrium MicroC/OS-II application source file `hello_ucosii.c` in the `sw/app_ucosii` folder

Follow these steps to generate the application project file:

1. Make a new directory as `<Project directory>/sw/app_custom`.
2. Copy the relevant application source file.
3. Launch the Nios V Command Shell.
4. Execute the command below to generate an application `CMakeLists.txt`.

   ```
   niosv-app --bsp-dir=sw/bsp_custom --app-dir=sw/app_custom \
   --srcs=sw/app_custom/hello.c --elf-name=hello.elf
   ```

1.5.3.3. Building the Application Project

You can build the application project using one of the following tools:

- RiscFree IDE for Intel FPGAs
- Eclipse Embedded CDT
- Command-line interface (CLI)

  ```
  cmake -G "Unix Makefiles" -DCMAKE_BUILD_TYPE=Debug -B \
  sw/app_custom/debug -S sw/app_custom
  make -C sw/app_custom/debug
  ```

Related Information

- RiscFree IDE for Intel FPGAs on page 35
- Eclipse CDT for Embedded C/C++ Developer on page 36
- Other Utilities Tools on page 37

1.5.4. Programming Nios V into the FPGA Device

To program Nios V processor based system into the FPGA and to run your application, use Intel Quartus Programmer tool.

1. To create the Nios V processor inside the FPGA device, download the `.sof` file onto the board with the following command.

Windows:

```
quartus_pgm -c 1 -m JTAG -o p;hw/output_files/top.sof@1
```

Linux:

```
quartus_pgm -c 1 -m JTAG -o p\;hw/output_files/top.sof@1
```

Note:
- `-c 1` is referring to cable number connected to the Host Computer.
- @1 is referring to device index on the JTAG Chain and may differ for your board.

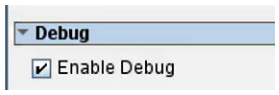
2. To run the Hello World application program, reset the Nios V processor system using the `toggle_issp.tcl` script.

```
quartus_stp -t scripts/toggle_issp.tcl
```

3. Download the `.elf` using the `niosv-download` command.

```
niosv-download <elf file>
```

Note: Set the **Enable Debug** option during configuration in Platform Designer to use `niosv-download` command.

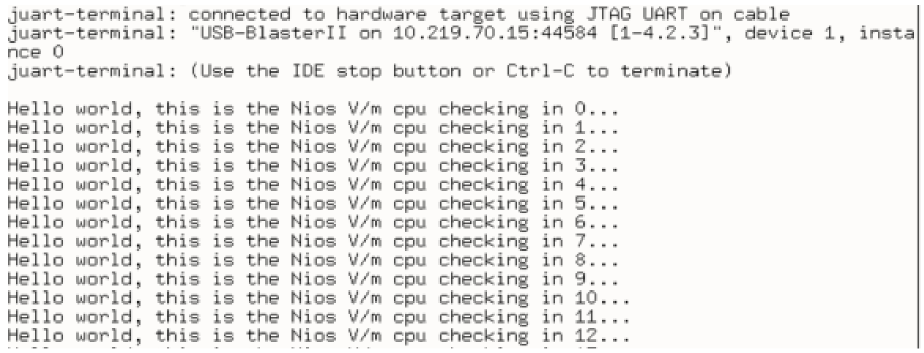


4. Use the JTAG UART terminal to print the stdout and stderr of the Nios V processor system.

```
juart-terminal
```

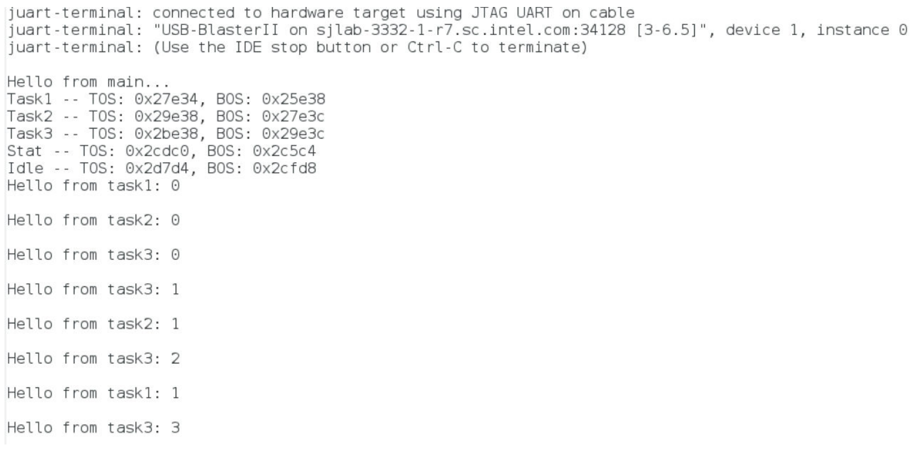
5. The Hello World application displays as shown in the following figures.

Figure 6. Output of the Hello World application using hello.c



```
juart-terminal: connected to hardware target using JTAG UART on cable
juart-terminal: "USB-BlasterII on 10.219.70.15:44584 [1-4.2.3]", device 1, insta
nce 0
juart-terminal: (Use the IDE stop button or Ctrl-C to terminate)

Hello world, this is the Nios V/m cpu checking in 0...
Hello world, this is the Nios V/m cpu checking in 1...
Hello world, this is the Nios V/m cpu checking in 2...
Hello world, this is the Nios V/m cpu checking in 3...
Hello world, this is the Nios V/m cpu checking in 4...
Hello world, this is the Nios V/m cpu checking in 5...
Hello world, this is the Nios V/m cpu checking in 6...
Hello world, this is the Nios V/m cpu checking in 7...
Hello world, this is the Nios V/m cpu checking in 8...
Hello world, this is the Nios V/m cpu checking in 9...
Hello world, this is the Nios V/m cpu checking in 10...
Hello world, this is the Nios V/m cpu checking in 11...
Hello world, this is the Nios V/m cpu checking in 12...
```

Figure 7. Output of the Hello World application using hello_ucosii.c



```
juart-terminal: connected to hardware target using JTAG UART on cable
juart-terminal: "USB-BlasterII on sjlab-3332-1-r7.sc.intel.com:34128 [3-6.5]", device 1, instance 0
juart-terminal: (Use the IDE stop button or Ctrl-C to terminate)

Hello from main...
Task1 -- TOS: 0x27e34, BOS: 0x25e38
Task2 -- TOS: 0x29e38, BOS: 0x27e3c
Task3 -- TOS: 0x2be38, BOS: 0x29e3c
Stat -- TOS: 0x2cdc0, BOS: 0x2c5c4
Idle -- TOS: 0x2d7d4, BOS: 0x2cfd8
Hello from task1: 0

Hello from task2: 0

Hello from task3: 0

Hello from task3: 1

Hello from task2: 1

Hello from task3: 2

Hello from task1: 1

Hello from task3: 3
```

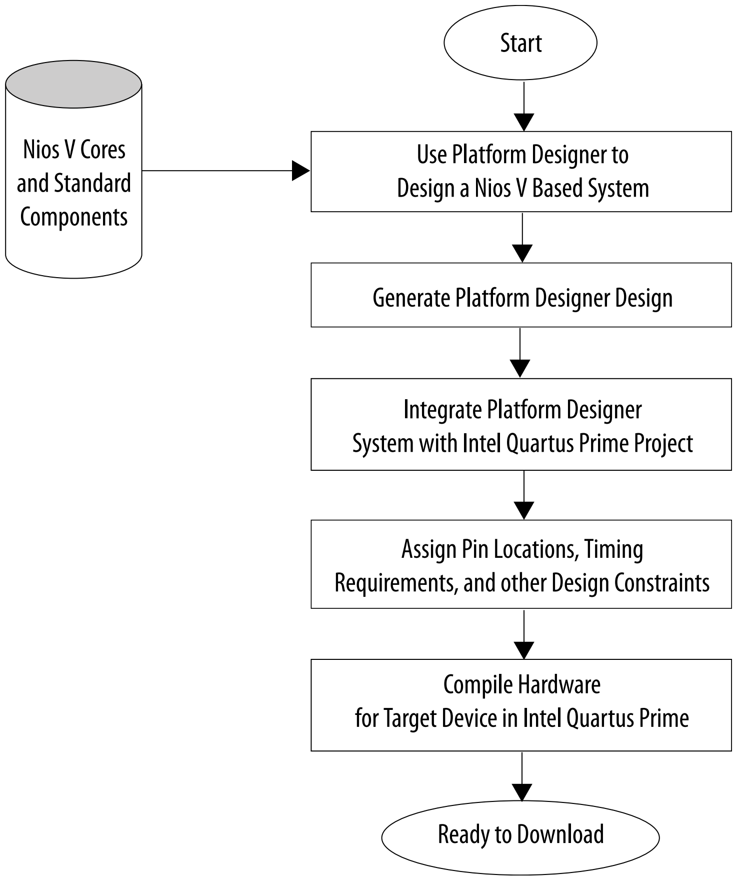
Related Information

- Intel Quartus Prime Pro Edition User Guide: Programmer
- AN 812: Platform Designer System Design Tutorial

2. Nios V Processor Hardware System Design with Intel Quartus Prime Software and Platform Designer

The following diagram illustrates a typical Nios V processor hardware design.

Figure 8. Nios V Processor System Hardware Design Flow


Start
Nios V Cores and Standard Components
Use Platform Designer to Design a Nios V Based System
Generate Platform Designer Design
Integrate Platform Designer System with Intel Quartus Prime Project
Assign Pin Locations, Timing Requirements, and other Design Constraints
Compile Hardware for Target Device in Intel Quartus Prime
Ready to Download

2.1. Creating Nios V Processor System Design with Platform Designer

The Intel Quartus Prime software includes the Platform Designer system integration tool that simplifies the task of defining and integrating Nios V processor IP core and other IPs into an Intel FPGA system design. The Platform Designer automatically creates interconnect logic from the specified high-level connectivity. The interconnect automation eliminates the time-consuming task of specifying system-level HDL connections.

After analyzing the system hardware requirements, you use Intel Quartus Prime to specify the Nios V processor core, memory, and other components your system requires. The Platform Designer automatically generates the interconnect logic to integrate the components in the hardware system.

2.1.1. Instantiating Nios V Processor Intel FPGA IP

You can instantiate any of the processor IP cores in **Platform Designer ➤ IP Catalog ➤ Processors and Peripherals ➤ Embedded Processors**.

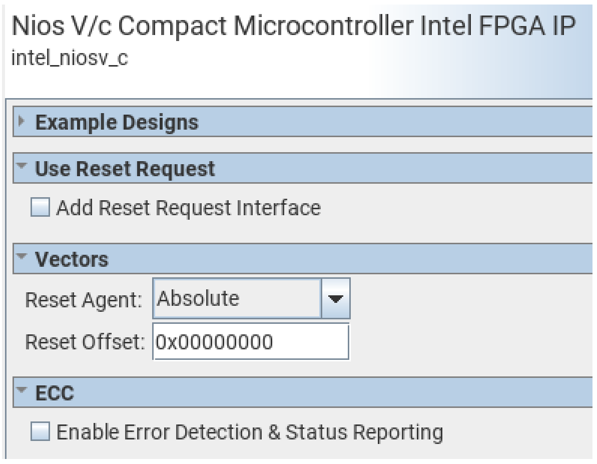
The IP core of each processor supports different configuration options based on its unique architecture. You can define these configurations to better suit your design needs.

Table 3. Configuration Options Across Core Variants

Configuration Options	Nios V/c Processor	Nios V/m Processor	Nios V/g Processor
Debug	—	√	√
Use Reset Request	√	√	√
Vectors	√	√	√
CPU Architecture	—	√	√
ECC	√	√	√
Caches, Peripheral Regions and TCMs	—	—	√
Custom Instructions	—	—	√

2.1.1.1. Instantiating Nios V/c Compact Microcontroller Intel FPGA IP

Figure 9. Nios V/c Compact Microcontroller Intel FPGA IP



Nios V/c Compact Microcontroller Intel FPGA IP
intel_niosv_c

☐ Add Reset Request Interface

Reset Agent: Absolute
Reset Offset: 0x00000000

☐ Enable Error Detection & Status Reporting

2.1.1.1.1. Use Reset Request Tab

Table 4. Use Reset Request Tab Parameter

Use Reset Request Tab	Description
Add Reset Request Interface	• Enable this option to expose local reset ports where a local master can use it to trigger the Nios V processor to reset without affecting other components in a Nios V processor system. • The reset interface consists of an input `resetreq` signal and an output `ack` signal. • You can request a reset to the Nios V processor core by asserting the resetreq signal. • The `resetreq` signal must remain asserted until the processor asserts `ack` signal. Failure for the signal to remain asserted can cause the processor to be in a non-deterministic state. • Assertion of the `resetreq` signal in debug mode has no effect on the processor's state. • The Nios V processor responds that the reset is successful by asserting the `ack` signal. • After the processor is successfully reset, the assertion of the `ack` signal can happen multiple times periodically until the de-assertion of the `resetreq` signal.

2.1.1.1.2. Vectors Tab

Table 5. Vectors Tab Parameters

Vectors	Description
Reset Agent	• The memory hosting the reset vector (the Nios V processor reset address) where the reset code resides. • You can select any memory module connected to the Nios V processor instruction master and supported by a Nios V processor boot flow as the reset agent.
Reset Offset	• Specifies the offset of the reset vector relative to the chosen reset agent's base address. • Platform Designer automatically provides a default value for the reset offset.

Note: Platform Designer provides an **Absolute** option, which allows you to specify an absolute address in Reset Offset. Use this option when the memory storing the reset vector is located outside the processor system and subsystems.

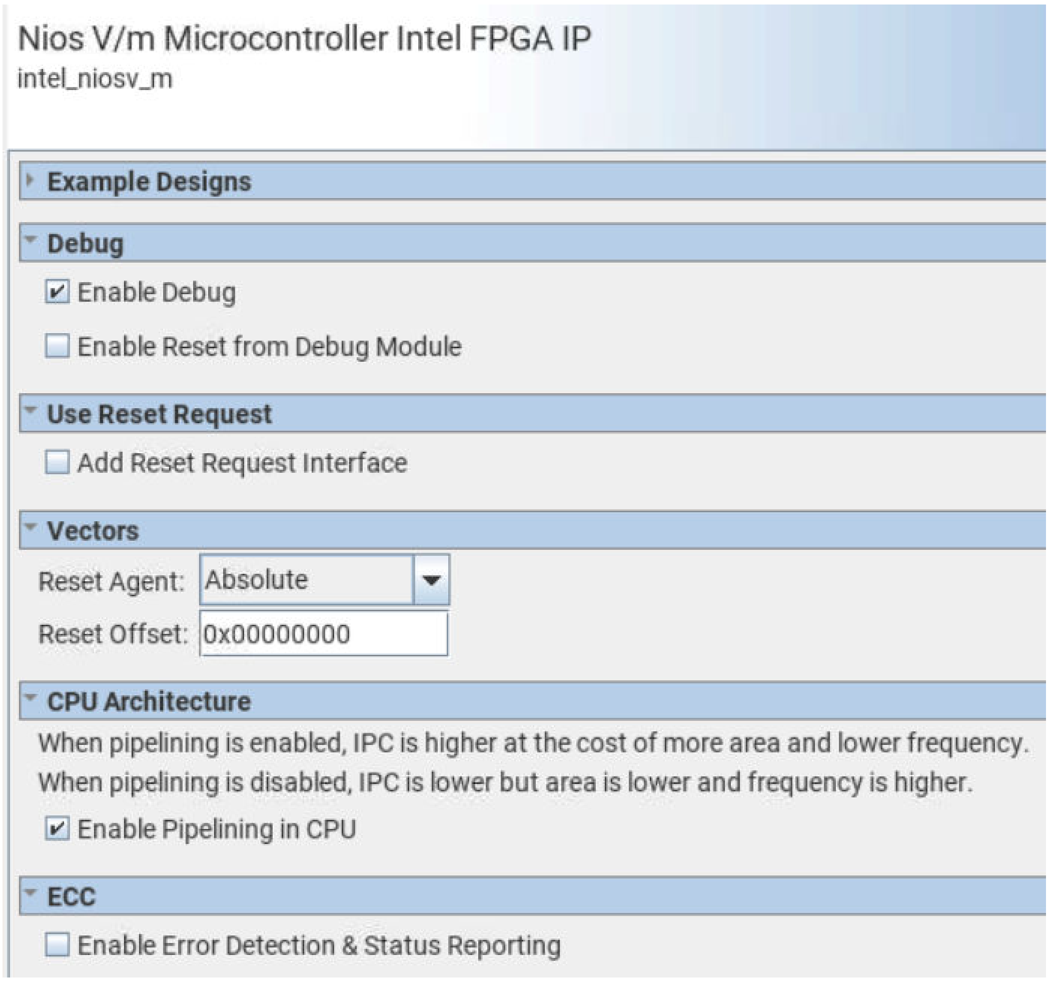
2.1.1.1.3. ECC Tab

Table 6. ECC Tab

ECC	Description
Enable Error Detection and Status Reporting	• Enable this option to apply ECC feature for Nios V processor internal RAM blocks. • ECC feature detects error without correcting it. — If it is a correctable error, the processor continues to operate. — If it is an uncorrectable error, the processor halts its operation.

2.1.1.2. Instantiating Nios V/m Microcontroller Intel FPGA IP

Figure 10. Nios V/m Microcontroller Intel FPGA IP


Nios V/m Microcontroller Intel FPGA IP
intel_niosv_m

Example Designs

Debug

- [✓] Enable Debug
- [ ] Enable Reset from Debug Module

Use Reset Request

- [ ] Add Reset Request Interface

Vectors

Reset Agent: Absolute
Reset Offset: 0x00000000

CPU Architecture

When pipelining is enabled, IPC is higher at the cost of more area and lower frequency.
When pipelining is disabled, IPC is lower but area is lower and frequency is higher.

- [✓] Enable Pipelining in CPU

ECC

- [ ] Enable Error Detection & Status Reporting

2.1.1.2.1. Debug Tab

Table 7. Debug Tab Parameters

Debug Tab	Description
Enable Debug	• Enable this option to add the JTAG target connection module to the Nios V processor. • The JTAG target connection module allows connecting to the Nios V processor through the JTAG interface pins of the FPGA. • The connection provides the following basic capabilities: — Start and stop the Nios V processor — Examine and edit registers and memory. — Download the Nios V application `.elf` file to the processor memory at runtime via niosv-download. — Debug the application running on the Nios V processor • Connect `dm_agent` port to the processor instruction and data bus. Ensure the base address between both buses are the same.
Enable Reset from Debug Module	• Enable this option to expose `dbg_reset_out` and `ndm_reset_in` ports. • JTAG debugger or `niosv-download -r` command trigger the `dbg_reset_out`, which allows the Nios V processor to reset system peripherals connecting to this port. • You must connect the `dbg_reset_out` interface to `ndm_reset_in` instead of `reset` interface to trigger reset to processor core and timer module.

2.1.1.2.2. Use Reset Request Tab

Table 8. Use Reset Request Tab Parameter

Use Reset Request Tab	Description
Add Reset Request Interface	• Enable this option to expose local reset ports where a local master can use it to trigger the Nios V processor to reset without affecting other components in a Nios V processor system. • The reset interface consists of an input `resetreq` signal and an output `ack` signal. • You can request a reset to the Nios V processor core by asserting the resetreq signal. • The `resetreq` signal must remain asserted until the processor asserts `ack` signal. Failure for the signal to remain asserted can cause the processor to be in a non-deterministic state. • Assertion of the `resetreq` signal in debug mode has no effect on the processor's state. • The Nios V processor responds that the reset is successful by asserting the `ack` signal. • After the processor is successfully reset, the assertion of the `ack` signal can happen multiple times periodically until the de-assertion of the `resetreq` signal.

2.1.1.2.3. Vectors Tab

Table 9. Vectors Tab Parameters

Vectors	Description
Reset Agent	• The memory hosting the reset vector (the Nios V processor reset address) where the reset code resides. • You can select any memory module connected to the Nios V processor instruction master and supported by a Nios V processor boot flow as the reset agent.
Reset Offset	• Specifies the offset of the reset vector relative to the chosen reset agent's base address. • Platform Designer automatically provides a default value for the reset offset.

Note: Platform Designer provides an **Absolute** option, which allows you to specify an absolute address in Reset Offset. Use this option when the memory storing the reset vector is located outside the processor system and subsystems.

2.1.1.2.4. CPU Architecture

Table 10. CPU Architecture Tab Parameters

CPU Architecture	Description
Enable Pipelining in CPU	• Enable this option to instantiate pipelined Nios V/m processor. — IPC is higher at the cost of higher logic area and lower Fmax frequency. • Disable this option to instantiate non-pipelined Nios V/m processor. — Has similar core performance with the Nios V/c processor. — Supports debugging and interrupt capability — Lower logic area and higher Fmax frequency at the cost of lower IPC.

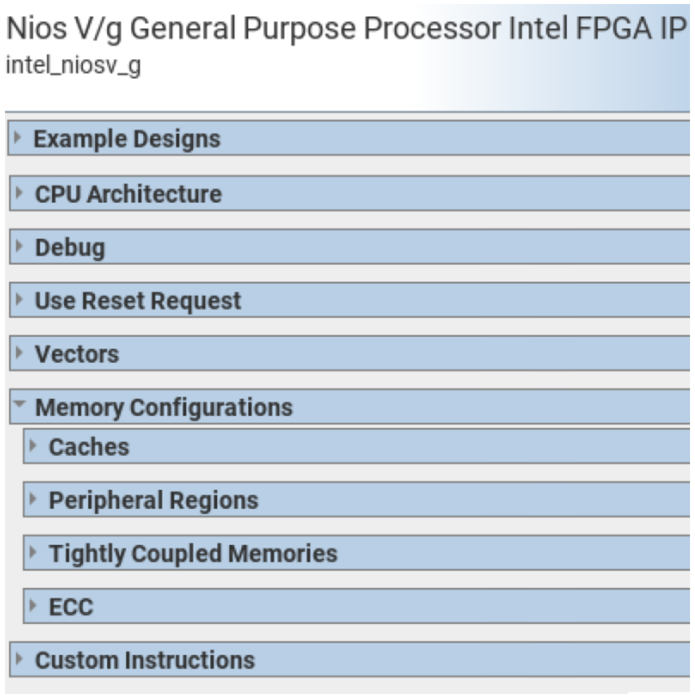
2.1.1.2.5. ECC Tab

Table 11. ECC Tab

ECC	Description
Enable Error Detection and Status Reporting	• Enable this option to apply ECC feature for Nios V processor internal RAM blocks. • ECC feature detects error without correcting it. — If it is a correctable error, the processor continues to operate. — If it is an uncorrectable error, the processor halts its operation.

2.1.1.3. Instantiating Nios V/g General Purpose Processor Intel FPGA IP

Figure 11. Nios V/g General Purpose Processor Intel FPGA IP



2.1.1.3.1. Debug Tab

Table 12. Debug Tab Parameters

Debug Tab	Description
Enable Debug	• Enable this option to add the JTAG target connection module to the Nios V processor. • The JTAG target connection module allows connecting to the Nios V processor through the JTAG interface pins of the FPGA. • The connection provides the following basic capabilities: — Start and stop the Nios V processor — Examine and edit registers and memory. — Download the Nios V application `.elf` file to the processor memory at runtime via niosv-download. — Debug the application running on the Nios V processor • Connect `dm_agent` port to the processor instruction and data bus. Ensure the base address between both buses are the same.
Enable Reset from Debug Module	• Enable this option to expose `dbg_reset_out` and `ndm_reset_in` ports. • JTAG debugger or `niosv-download -r` command trigger the `dbg_reset_out`, which allows the Nios V processor to reset system peripherals connecting to this port. • You must connect the `dbg_reset_out` interface to `ndm_reset_in` instead of `reset` interface to trigger reset to processor core and timer module.

2.1.1.3.2. Use Reset Request Tab

Table 13. Use Reset Request Tab Parameter

Use Reset Request Tab	Description
Add Reset Request Interface	• Enable this option to expose local reset ports where a local master can use it to trigger the Nios V processor to reset without affecting other components in a Nios V processor system. • The reset interface consists of an input `resetreq` signal and an output `ack` signal. • You can request a reset to the Nios V processor core by asserting the resetreq signal. • The `resetreq` signal must remain asserted until the processor asserts `ack` signal. Failure for the signal to remain asserted can cause the processor to be in a non-deterministic state. • Assertion of the `resetreq` signal in debug mode has no effect on the processor's state. • The Nios V processor responds that the reset is successful by asserting the `ack` signal. • After the processor is successfully reset, the assertion of the `ack` signal can happen multiple times periodically until the de-assertion of the `resetreq` signal.

2.1.1.3.3. Vectors Tab

Table 14. Vectors Tab Parameters

Vectors	Description
Reset Agent	• The memory hosting the reset vector (the Nios V processor reset address) where the reset code resides. • You can select any memory module connected to the Nios V processor instruction master and supported by a Nios V processor boot flow as the reset agent.
Reset Offset	• Specifies the offset of the reset vector relative to the chosen reset agent's base address. • Platform Designer automatically provides a default value for the reset offset.

Note: Platform Designer provides an **Absolute** option, which allows you to specify an absolute address in Reset Offset. Use this option when the memory storing the reset vector is located outside the processor system and subsystems.

2.1.1.3.4. Memory Configurations Tab

Table 15. Memory Configuration Tab Parameters

Category	Memory Configuration Tab	Description
Caches	Data Cache Size	• Specifies the size of the data cache. • Valid sizes are from 1 kilobytes (KB) to 16 KB.
	Instruction Cache Size	• Specifies the size of the instruction cache. • Valid sizes are from 1 KB to 16 KB.
Peripheral Region A and B	Size	• Specifies the size of the peripheral region. • Valid sizes are from 64 KB to 2 gigabytes (GB), or None. Choosing None disables the peripheral region.
	Base Address	• Specifies the base address of peripheral region after you select the size. • All addresses in the peripheral region produce uncacheable data accesses. • Peripheral region base address must be aligned to the peripheral region size.
Tightly Coupled Memories	Size	• Specifies the size of the tightly-coupled memory. — Valid sizes are from 0 MB to 512 MB.
	Base Address	• Specifies the base address of tightly-coupled memory.
	Initialization File	• Specifies the initialization file for tightly-coupled memory.

Note: In a Nios V processor system with cache enabled, you must place system peripherals within a peripheral region. You can use peripheral regions to define a non-cacheable transaction for peripherals such as UART, PIO, DMA, and others.

2.1.1.3.5. Custom Instruction Tab

Note: This tab is only available for the Nios V/g processor core.

Custom Instruction	Description
Nios V Custom Instruction Hardware Interface Table	• Nios V processor uses this table to define its custom instruction manager interfaces. • Defined custom instruction manager interfaces are uniquely encoded by an Opcode (CUSTOM0-3) and 3 bits of `funct7[6:4]`. • You can define up to a total of 32 individual custom instruction manager interfaces.
Nios V Custom Instruction Software Macro Table	• Nios V processor uses this table is used to define custom instruction software encodings for defined custom instruction manager interfaces. • For each defined custom instruction software encoding, the Opcode (CUSTOM0-3) and 3 bits of `funct7[6:4]` encoding must correlate to a defined custom instruction manager interface encoding in the Custom Instruction Hardware Interface Table. • You can use `funct7[6:4]`, `funct7[3:0]`, and `funct3[2:0]` to define additional encoding for a given custom instruction, or specified as Xs to be passed in as additional instruction arguments. • Nios V processor provides defined custom instruction software encodings as generated C-macros in `system.h`, and follow the R-type RISC-V instruction format. • Mnemonics may be used to define custom names for: — The generated C-Macros in `system.h`. — The generated GDB debug mnemonics in `custom_instruction_debug.xml`.

Related Information

AN 977: Nios V Processor Custom Instruction
For more information about custom instructions that allow you to customize the Nios® V processor to meet the needs of a particular application.

2.1.1.3.6. CPU Architecture

Table 16. CPU Architecture Parameters

CPU Architecture Tab	Description
Enable Floating Point Unit	• Enable this option to add the floating-point unit ("F" extension) in the processor core.

2.1.1.3.7. ECC Tab

Table 17. ECC Tab

ECC	Description
Enable Error Detection and Status Reporting	• Enable this option to apply ECC feature for Nios V processor internal RAM blocks. • ECC feature detects error without correcting it. — If it is a correctable error, the processor continues to operate. — If it is an uncorrectable error, the processor halts its operation.

2.1.2. Defining System Component Design

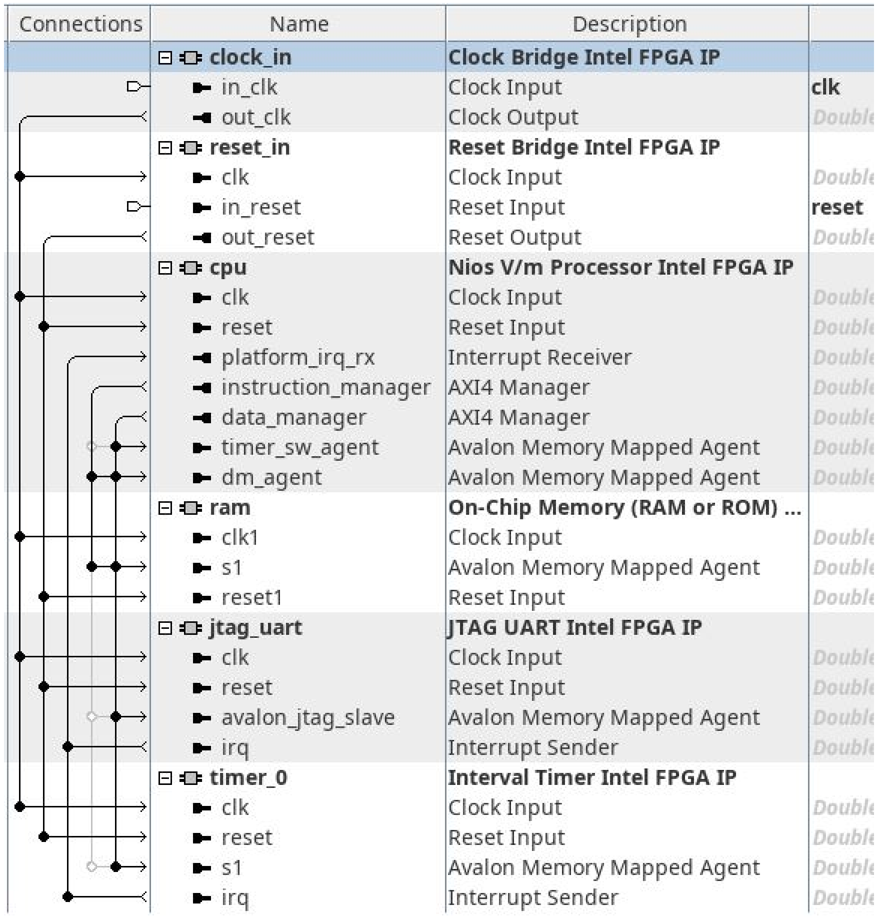
Use the Platform Designer to define the hardware characteristics of the Nios V processor system and add in the desired components. The following diagram demonstrates a basic Nios V processor system design with the following components:

- Nios V processor core
- On-Chip Memory
- JTAG UART
- Interval Timer (optional)[1]

When a new On-Chip Memory is added to a Platform Designer system, perform **Sync System Infos** to reflect the added memory components in reset. Alternatively, you can enable **Auto Sync** in Platform Designer to automatically reflect the latest component changes

[1] You have the option to use the Nios V Internal Timer features to replace the external Interval Timer in Platform Designer.

Figure 12. Example connection of Nios V processor with other peripherals in Platform Designer



Connections	Name	Description	
	clock_in	**Clock Bridge Intel FPGA IP**	
	in_clk	Clock Input	**clk**
	out_clk	Clock Output	Double
	reset_in	**Reset Bridge Intel FPGA IP**	
	clk	Clock Input	Double
	in_reset	Reset Input	**reset**
	out_reset	Reset Output	Double
	cpu	**Nios V/m Processor Intel FPGA IP**	
	clk	Clock Input	Double
	reset	Reset Input	Double
	platform_irq_rx	Interrupt Receiver	Double
	instruction_manager	AXI4 Manager	Double
	data_manager	AXI4 Manager	Double
	timer_sw_agent	Avalon Memory Mapped Agent	Double
	dm_agent	Avalon Memory Mapped Agent	Double
	ram	**On-Chip Memory (RAM or ROM) ...**	
	clk1	Clock Input	Double
	s1	Avalon Memory Mapped Agent	Double
	reset1	Reset Input	Double
	jtag_uart	**JTAG UART Intel FPGA IP**	
	clk	Clock Input	Double
	reset	Reset Input	Double
	avalon_jtag_slave	Avalon Memory Mapped Agent	Double
	irq	Interrupt Sender	Double
	timer_0	**Interval Timer Intel FPGA IP**	
	clk	Clock Input	Double
	reset	Reset Input	Double
	s1	Avalon Memory Mapped Agent	Double
	irq	Interrupt Sender	Double

You must also define operation pins to export as conduit in your Platform Designer system. For example, a proper FPGA system operation pin list is defined as below but not limited to:

- Clock
- Reset
- I/O signals

2.1.3. Specifying Base Addresses and Interrupt Request Priorities

To specify how the components added in the design interact to form a system, you need to assign base addresses for each agent component and assign interrupt request (IRQ) priorities for the JTAG UART and the interval timer. The Platform Designer provides a command - `Assign Base Addresses` - which automatically assigns proper base addresses to all components in a system. However, you can adjust the base addresses based on your needs.

The following are some guidelines for assigning base addresses:

- Nios V processor core has a 32-bit address span. To access agent components, their base address must range between 0x00000000 and 0xFFFFFFFF.
- Nios V programs use symbolic constants to refer to addresses. You do not have to choose address values that are easy to remember.
- Address values that differentiate components with only a one-bit address difference produce more efficient hardware. You do not have to compact all base addresses into the smallest possible address range because compacting can create less efficient hardware.
- Platform Designer does not attempt to align separate memory components in a contiguous memory range. For example, if you want multiple On-Chip Memory components addressable as one contiguous memory range, you must explicitly assign base addresses.

The Platform Designer also provides automation command - `Assign Interrupt Numbers` which connects IRQ signals to produce valid hardware results. However, assigning IRQs effectively requires an understanding to the overall software respond behavior.The Platform Designer cannot make educated guesses about the best IRQ assignment.

To interpret the IRQ priority, the lowest IRQ value has the highest priority. In an ideal system, the timer component must have the highest IRQ priority to maintain the accuracy of the system clock tick.

Related Information

Intel Quartus Prime Pro Edition User Guide:
More information about creating a System with Platform Designer.

2.2. Integrating Platform Designer System into the Intel Quartus Prime Project

After generating the Nios V system design in Platform Designer, perform the following tasks to integrate the Nios V system module into the Intel Quartus Prime FPGA design project.

- Instantiate the Nios V system module in the Intel Quartus Prime project
- Connect signals from Nios V system module to other signals in the FPGA logic
- Assign physical pins location
- Constrain the FPGA design

2.2.1. Instantiating the Nios V Processor System Module in the Intel Quartus Prime Project

The Platform Designer generates a system module design entity in which you can use to instantiate in Intel Quartus Prime. How you instantiate the system module depends on the design entry method for the overall Intel Quartus Prime project. For example, if you were using Verilog HDL for design entry, instantiate the Verilog based system module and if you prefer to use the block diagram method for design entry, instantiate a system module symbol `.bdf` file.

2.2.2. Connecting Signals and Assigning Physical Pin Locations

To connect your Intel FPGA design to your board-level design, perform the following tasks:

- Identify the top-level file for your design and signals to connect to external Intel FPGA device pins.
- Understand which pins to connect through your board-level design user guide or schematics.
- Assign signals in the top-level design to pin your Intel FPGA device with pin assignment tools.

The top-level Intel FPGA IP-based design can be your Platform Designer system. However, the Intel FPGA can also include additional design logic based on your needs and thus introduces a custom top-level file. The top-level file connects the Nios V processor system module signals to other Intel FPGA design logic.

Related Information

Intel Quartus Prime Pro Edition User Guide: Design Constraints

2.2.3. Constraining the Intel FPGA Design

A proper Intel FPGA system design includes design constraints to ensure the design meets timing closure and other logic constraint requirements. You must constrain your Intel FPGA design to meet these requirements explicitly using tools provided in the Intel Quartus Prime software or third-party EDA providers. The Intel Quartus Prime software uses the constraint settings that you provide during the compilation phase to get the optimum placement results.

Related Information

- Intel Quartus Prime Pro Edition User Guide: Design Constraints
- Third-party EDA Partners
- Intel Quartus Prime Pro Edition User Guide: Timing Analyzer

2.3. Designing a Nios V Processor Memory System

This section describes the best practices for selecting memory devices in a Platform Designer embedded system with a Nios V processor and achieving optimum performance. Memory devices play a critical role in improving the overall performance of an Intel FPGA embedded system. Embedded systems memory stores the running instructions and constantly manipulates data.

2.3.1. Volatile Memory

A primary distinction in a memory type is volatility. Volatile memory only holds its contents while you supply power to the memory device. As soon you remove the power, the memory loses its contents. Thus, do not use volatile memory if you need to retain the data when switching off the memory.

Examples of volatile memory are RAM, cache, and registers. These are fast memory types that increases running performance. Intel recommends you to load and execute Nios V processor instruction in RAM and pair Nios V IP core with On-Chip Memory IP or External Memory Interface IP for optimum performance.

To improve the performance, you can eliminate additional Platform Designer adaptation components by matching Nios V processor data manager interface type or width with boot RAM. For example, you can configure On-Chip Memory II with a 32-bits AXI-4 interface, which matches the Nios V data manager interface.

Related Information

- External Memory Interfaces IP Support Center
- On-Chip Memory (RAM or ROM) Intel FPGA IP
- On-Chip Memory II (RAM or ROM) Intel FPGA IP
- Nios V Processor Application Execute-In-Place from OCRAM on page 45

2.3.2. Non-Volatile Memory

Non-volatile memory retains its contents when the power switches off, making it a good choice for storing information that the system must retrieve after a system power cycle. Non-volatile memory commonly stores processor boot-code, persistent application settings, and Intel FPGA configuration data. Although non-volatile memory has the advantage of retaining its data when you remove the power, it is much slower to write compared with volatile memory, and often has more complex writing and erasing procedures. Non-volatile memory is also usually only guaranteed to be erasable a given number of times, after which it may fail.

Examples of non-volatile memory include all types of flash, EPROM, and EEPROM. Intel recommends you to store Intel FPGA bitstreams and Nios V program images in a non-volatile memory, and use serial flash as the boot device for Nios V processors.

Related Information

- Generic Serial Flash Interface Intel FPGA IP User Guide
- Mailbox Client Intel FPGA IP User Guide

2.3.3. On-Chip Memory Configuration – RAM or ROM

You can configure Intel FPGA On-Chip Memory IPs as RAM or ROM.

- RAM provides read and write capability and has a volatile nature. If you are booting the Nios V processor from an On-Chip RAM, you must make sure boot content is preserved and not corrupted in the event of a reset during run time.
- If a Nios V processor is booting from ROM, any software bug on the Nios V processor cannot erroneously overwrite the contents of On-Chip Memory. Thus, reducing the risk of boot software corruption.

Related Information

- On-Chip Memory (RAM or ROM) Intel FPGA IP
- On-Chip Memory II (RAM or ROM) Intel FPGA IP
- Nios V Processor Application Execute-In-Place from OCRAM on page 45

2.4. Clocks and Resets Best Practices

Understanding how the Nios V processor clock and reset domain interacts with every peripheral it connects to is important. A simple Nios V processor system starts with a single clock domain, and it can get complicated with a multi-clock domain system when a fast clock domain collides with a slow clock domain. You need to take note and understand how these different domains sequence out of reset and make sure there aren't any subtle problems.

For best practice, Intel recommends placing the Nios V processor and boot memory in the same clock domain. Do not release the Nios V processor from reset in a fast clock domain when it boots from a memory that resides in a very slow clock domain, which may cause an instruction fetch error. You may require some manual sequencing beyond what Platform Designer provides by default, and plan out reset release topology accordingly based on your use case. If you want to reset your system after it comes up and runs for a while, apply the same considerations to system reset sequencing and post reset initialization requirement.

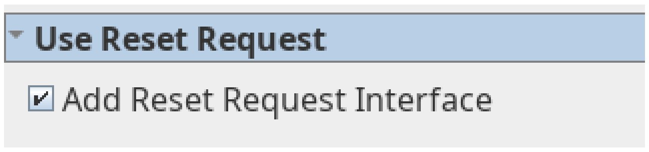
2.4.1. Reset Request Interface

Nios V processor includes an optional reset request facility. The reset request facility consists of `reset_req` and `reset_req_ack` signals.

To enable the reset request in Platform Designer:

1. Launch the **Nios V Processor IP Parameter Editor**.
2. On the **Use Reset Request** setting, turn on the **Add Reset Request Interface** option.

Figure 13. Enable Nios V Processor Reset Request



Use Reset Request

☑ Add Reset Request Interface

The `reset_req` signal acts like an interrupt. When you assert the `reset_req`, you are requesting reset to the core. The core waits for any outstanding bus transaction to complete its operation. For example, if there is a pending memory access transaction, core waits for a complete response. Similarly, core accepts any pending instruction response but does not issue instruction request after receiving the `reset_req` signal.

The reset operation consists of the following flow:

1. Complete all pending operation
2. Flush internal pipeline
3. Set the instruction Program Counter to reset vector
4. Reset core

The whole reset operation takes a few clock cycles. The `reset_req` must remain asserted until `reset_req_ack` is asserted indicating core reset operation has successfully completed. Failure to do so results in core's state being non-deterministic.

2.4.1.1. Typical Use Cases

- You can assert the `reset_req` signal from power-on to prevent the Nios V processor core from starting program execution from its reset vector until other FPGA hosts in the system initialize the Nios V processor boot memory. In this case, the entire subsystem can experience a clean hardware reset. The Nios V processor is held indefinitely in a reset request state until the other FPGA hosts initialize the processor boot memory.
- In a system where you must reset the Nios V processor core without disrupting the rest of the system, you can assert the `reset_req` signal to cleanly halt the current operation of the core. Restart the processor from the reset vector once the system releases the `reset_req_ack` signal.
- An external host may use the reset request interface to ease the implementations of the following tasks:
  - Halt the current Nios V processor program.
  - Load a new program into the Nios V processor boot memory.
  - Allow the processor to begin executing the new program.

Intel recommends you to implement a timeout mechanism to monitor the state of `reset_req_ack` signal. If Nios V processor core falls into an infinite wait state condition and stall with an unknown reason, `reset_req_ack` cannot assert indefinitely. The timeout mechanism enables you to:

- Define a recovery timeout period and perform system recovery with system level reset.
- Perform a hardware level reset.

2.5. Assigning a Default Agent

Platform Designer allows you to specify default agent which act as the error response default agent. The default agent you designate provides an error response service for hosts that attempt non-decoded accesses into the address map.

The following scenarios trigger a non decoded event:

- Bus transaction security state violation
- Transaction access to undefined memory region
- Exception event and etc.

A default agent should be assigned to handle such event, where undefined transaction is rerouted to the default agent and subsequently responds to Nios V processor with an error response.

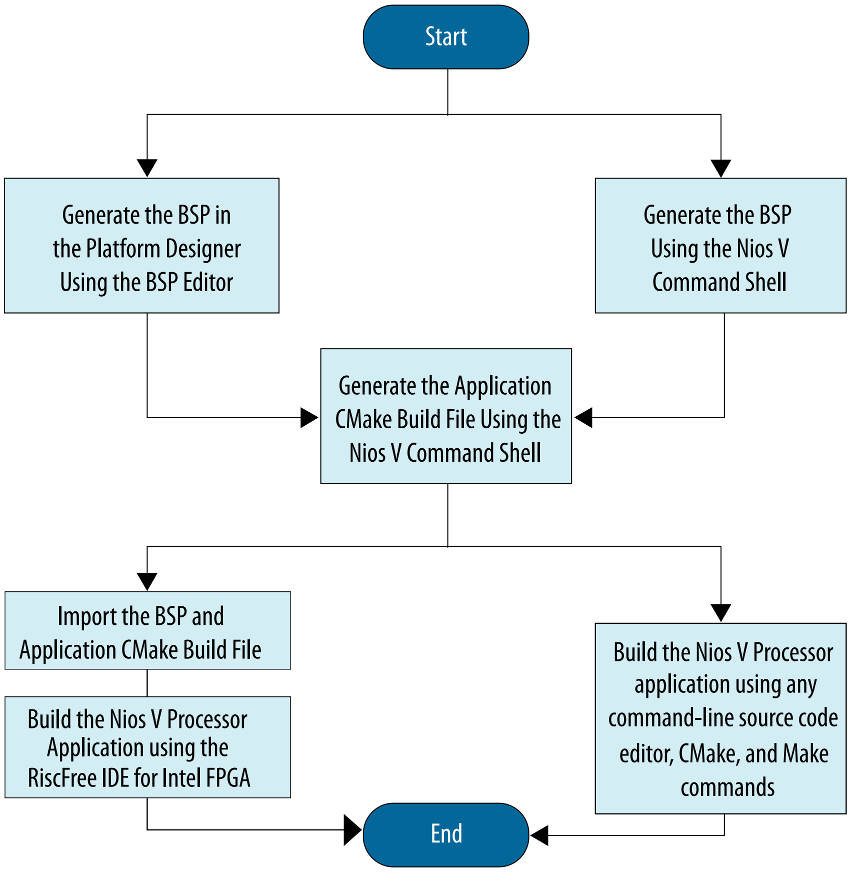
Related Information

- Intel Quartus Prime Pro Edition User Guide: Platform Designer. Designating a Default Agent
- Intel Quartus Prime Pro Edition User Guide: Platform Designer. Error Response Slave Intel FPGA IP
- Github - Supplemental Reset Components for Qsys

3. Nios V Processor Software System Design

This chapter describes the Nios V processor software development flow and the software tools that you can use in developing your embedded design system. The content serves as an overview before developing a Nios V processor software system.

Figure 14. Software Design Flow


Start
Generate the BSP in the Platform Designer Using the BSP Editor
Generate the BSP Using the Nios V Command Shell
Generate the Application CMake Build File Using the Nios V Command Shell
Import the BSP and Application CMake Build File
Build the Nios V Processor Application using the RiscFree IDE for Intel FPGA
Build the Nios V Processor application using any command-line source code editor, CMake, and Make commands
End

Note: Intel recommends that you use an Intel FPGA development kit or a custom prototype board for software development and debugging. Many peripherals and system-level features are available only when your software runs on an actual board.

Related Information

Software Design Flow on page 11

3.1. Nios V Processor Software Development Flow

3.1.1. Board Support Package Project

A Nios V Board Support Package (BSP) project is a specialized library containing system-specific support code. A BSP provides a software runtime environment customized for one processor in a Nios V processor hardware system.

The Intel Quartus Prime software provides **Nios V Board Support Package Editor** and **niosv-bsp** utility tools to modify settings that control the behavior of the BSP.

A BSP contains the following elements:

- Hardware abstraction layer
- Device drivers
- Optional software packages
- Optional real-time operating system

3.1.2. Application Project

A Nios V C/C++ application project has the following features:

- Consists of a collection of source code and a `CMakeLists.txt`.
  - The CMakeLists.txt compiles the source code and links it with a BSP and one or more optional libraries, to create one `.elf` file
- One of the source files contains function `main()`.
- Includes code that calls functions in libraries and BSPs.

Intel provides **niosv-app** utility tool in the Intel Quartus Prime software utility tools to create the Application CMakeLists.txt, and RiscFree IDE for Intel FPGAs to modify the source code in an Eclipse-based environment.

3.2. Intel FPGA Embedded Development Tools

The Nios V processor supports the following tools for software development:

- Graphical User Interface (GUI) - Graphical development tools that are available in both Windows* and Linux* Operating System (OS).
  - Nios V Board Support Package Editor (Nios V BSP Editor)
  - Ashling RiscFree IDE for Intel FPGAs
  - Eclipse CDT for Embedded C/C++ Developers (Eclipse Embedded CDT)
- Command-Line Tools (CLI) - Development tools that are initiated from the Nios V Command Shell. Each tool provides its own documentation in the form of help accessible from the command line. Open the Nios V Command Shell and type the following command: `<name of tool> --help` to view the **Help** menu.
  - Nios V Utilities Tools
  - File Format Conversion Tools
  - Other Utilities Tools

Table 18. GUI Tools and Command-line Tools Tasks Summary

Task	GUI Tool	Command-line Tool
Creating a BSP	Nios V BSP Editor	• In Intel Quartus Prime Pro Edition software: `niosv-bsp -c -s=<.qsys file> -t=<bsp type> [OPTIONS] settings.bsp` • In Intel Quartus Prime Standard Edition software: `-t=<bsp type>[OPTIONS] settings.bsp`
Generating a BSP using existing `.bsp` file	Nios V BSP Editor	`niosv-bsp -g [OPTIONS] settings.bsp`
Updating a BSP	Nios V BSP Editor	`niosv-bsp -u [OPTIONS] settings.bsp`
Examining a BSP	Nios V BSP Editor	`niosv-bsp -q -E=<tcl script> [OPTIONS] settings.bsp`
Creating an application	-	`niosv-app -a=<application directory> -b=<bsp directory> -s=<source files directory> [OPTIONS]`
Creating a user library	-	`niosv-app -l=<library directory> -s=<source files directory> -p=<public includes directory> [OPTIONS]`
Modifying an application	• RiscFree IDE for Intel FPGAs • Eclipse CDT for Embedded C/C++ Developers	Any command-line source editor
Modifying a user library	• RiscFree IDE for Intel FPGAs • Eclipse CDT for Embedded C/C++ Developers	Any command-line source editor
Building an application	• RiscFree IDE for Intel FPGAs • Eclipse CDT for Embedded C/C++ Developers	• `make` • `cmake`
Building a user library	• RiscFree IDE for Intel FPGAs • Eclipse CDT for Embedded C/C++ Developers	• `make` • `cmake`
Downloading an application ELF	• RiscFree IDE for Intel FPGAs • Eclipse CDT for Embedded C/C++ Developers	`niosv-download`
Converting the `.elf` file	-	• `elf2flash` • `elf2hex`

Related Information

- Nios V Tool setup for Eclipse CDT and OpenOCD
  For more information about Eclipse Embedded CDT
- Ashling RiscFree Integrated Development Environment (IDE) for Intel FPGAs User Guide

3.2.1. Nios V Processor Board Support Package Editor

You can use the Nios V processor BSP Editor to perform the following tasks:

- Create or modify a Nios V processor BSP project
- Edit settings, linker regions, and section mappings
- Select software packages and device drivers.

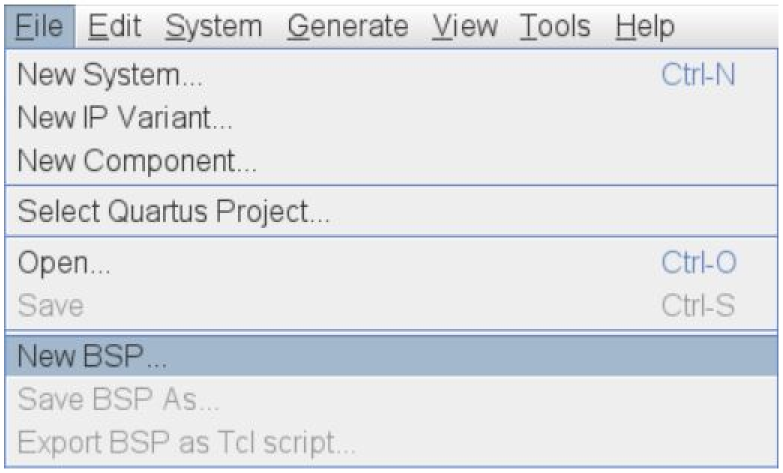
The capabilities of the BSP Editor include the capabilities of the **niosv-bsp** utilities. Any project created in the BSP Editor can also be created using the command-line utilities.

Note: For Intel Quartus Prime Standard Edition software, refer to *AN 980: Nios V Processor Intel Quartus Prime Software Support* for the steps to invoke the BSP Editor GUI.

To launch the **BSP Editor**, follow these steps:

1. Open Platform Designer, and navigate to the **File** menu.
    a. To open an existing BSP setting file, click **Open...**
    b. To create a new BSP, click **New BSP...**
2. Select the **BSP Editor** tab and provide the appropriate details.

Figure 15. Launch BSP Editor



Related Information

AN 980: Nios V Processor Intel Quartus Prime Software Support

3.2.2. RiscFree IDE for Intel FPGAs

The RiscFree IDE for Intel FPGAs is an Eclipse-based IDE for the Nios V processor. Intel recommends that you develop the Nios V processor software in this IDE for the following reasons:

- The features are developed and verified to be compatible with the Nios V processor build flow.
- Equipped with all the necessary toolchains and supporting tools which enables you to easily start Nios V development.
- You do not need to install Eclipse CDT for Embedded C/C++ Developer when using the RiscFree IDE for Intel FPGAs.

Related Information

- Ashling RiscFree Integrated Development Environment (IDE) for Intel FPGAs User Guide
- Building the Application Project on page 14

3.2.3. Eclipse CDT for Embedded C/C++ Developer

The Eclipse CDT for Embedded C/C++ Developer (Eclipse Embedded CDT) is an IDE that can include additional Eclipse plug-ins and open-source tools for RISC-V development. You can modify an application or a user library with the Eclipse Embedded CDT.

The Eclipse Embedded CDT setup consists of:

1. Open-source tools installation and setup
2. Project setting configuration
3. Software build flow

Note: You must update the `hal.toolchain.prefix` in **BSP Editor** based on the open-source RISC-V toolchain.

Related Information

- Nios V Tool setup for Eclipse CDT and OpenOCD
  For more information about Eclipse Embedded CDT
- Building the Application Project on page 14

3.2.4. Nios V Utilities Tools

You can create, modify, and build Nios V programs with commands typed at a command line or embedded in a script. The Nios V command-line tools described in this section are in the `<Intel Quartus Prime software installation directory>/niosv/bin` directory.

Table 19. Nios V Utilities Tools

Command-Line Tools	Summary
niosv-app	To generate and configure an application project.
niosv-bsp	To create or update a BSP settings file and create the BSP files.
niosv-download	To download the ELF file to a Nios® V processor.
niosv-shell	To open the Nios V Command Shell.
niosv-stack-report	To inform you of the left-over memory space available to your application `.elf` for stack or heap usage.

3.2.5. File Format Conversion Tools

File format conversion is sometimes necessary when passing data from one utility to another. The file format conversion tools are in the `<Intel Quartus Prime software installation directory>/niosv/bin` directory.

Table 20. File Format Conversion Tools

Command-Line Tools	Summary
elf2flash	To translate the `.elf` file to `.srec` format for flash memory programming.
elf2hex	To translate the `.elf` file to `.hex` format for memory initialization.

3.2.6. Other Utilities Tools

You might require the following command-line tools when building a Nios V processor based system. These command-line tools are either provided by Intel in `<Intel Quartus Prime installation directory>/quartus/bin` or acquired from open-source tools.

Table 21. Other Command-Line Tools

Command-Line Tools	Type	Summary
juart-terminal	Intel-provided	To monitor stdout and stderr, and to provide input to a Nios® V processor subsystem through stdin. This tool only applies to the JTAG UART IP when it is connected to the Nios® V processor.
openocd	Intel-provided	To execute OpenOCD.
openocd-cfg-gen	Intel-provided	• To generate the OpenOCD configuration file. • To display JTAG chain device index.

Related Information

Building the Application Project on page 14

4. Nios V Processor Configuration and Booting Solutions

You can configure the Nios V processor to boot and execute software from different memory locations. The boot memory is the Quad Serial Peripheral Interface (QSPI) flash, On-Chip Memory (OCRAM), or Tightly Coupled Memory (TCM).

4.1. Introduction

The Nios V processor supports two types of boot processes:

- Execute-in-Place (XIP) using alt_load() function
- Program copied to RAM using boot copier.

The Nios V embedded programs development is based on the hardware abstraction layer (HAL). The HAL provides a small boot loader program (also known as boot copier) that copies relevant linker sections from the boot memory to their run time location at boot time. You can specify the program and data memory run time locations by manipulating the Board Support Package (BSP) Editor settings.

This section describes:

- Nios V processor boot copier that boots your Nios V processor system according to the boot memory selection
- Nios V processor booting options and general flow
- Nios V programming solutions for the selected boot memory

4.2. Linking Applications

When you generate the Nios V processor project, the **BSP Editor** generates two linker related files:

- `linker.x`: The linker command file that the generated application's makefile uses to create the `.elf` binary file.
- `linker.h`: Contains information about the linker memory layout.

All linker setting modifications you make to the BSP project affect the contents of these two linker files.

Every Nios V processor application contains the following linker sections:

Table 22. Linker Sections

Linker Sections	Descriptions
`.text`	Executable code.
`.rodata`	Any read-only data used in the execution of the program.

continued...

Linker Sections	Descriptions
`.rwdata`	Stores read-write data used in the execution of the program.
`.bss`	Contains uninitialized static data.
`.heap`	Contains dynamically allocated memory.
`.stack`	Stores function-call parameters and other temporary data.

You can add additional linker sections to the `.elf` file to hold custom code and data. These linker sections are placed in named memory regions, defined to correspond with physical memory devices and addresses. By default, **BSP Editor** automatically generates these linker sections. However, you can control the linker sections for a particular application.

4.2.1. Linking Behavior

This section describes the **BSP Editor** default linking behavior and how to control the linking behavior.

4.2.1.1. Default BSP Linking

During BSP configuration, the tools perform the following steps automatically:

1. Assign memory region names: Assign a name to each system memory device and add each name to the linker file as a memory region.
2. Find largest memory: Identify the largest read-and-write memory region in the linker file.
3. Assign linker sections: Place the default linker sections (`.text`, `.rodata`, `.rwdata`, `.bss`, `.heap`, and `.stack`) in the memory region identified in the previous step.
4. Write files: Write the `linker.x` and `linker.h` files.

Typically, the linker section allocation scheme works during the software development process because the application is guaranteed to function if the memory is large enough.

The rules for the default linking behavior are contained in the Intel-generated Tcl scripts `bsp-set-defaults.tcl` and `bsp-linker-utils.tcl` found in the `<Intel Quartus Prime installation directory>/niosv/scripts/bsp-defaults` directory. The `niosv-bsp` command invokes these scripts. Do not modify these scripts directly.

4.2.1.2. Configurable BSP Linking

You can manage the default linking behavior in the **Linker Script** tab of the **BSP Editor**. Manipulate the linker script using the following methods:

- Add a memory region: Maps a memory region name to a physical memory device.
- Add a section mapping: Maps a section name to a memory region. The **BSP Editor** allows you to view the memory map before and after making changes.

4.3. Nios V Processor Booting Methods

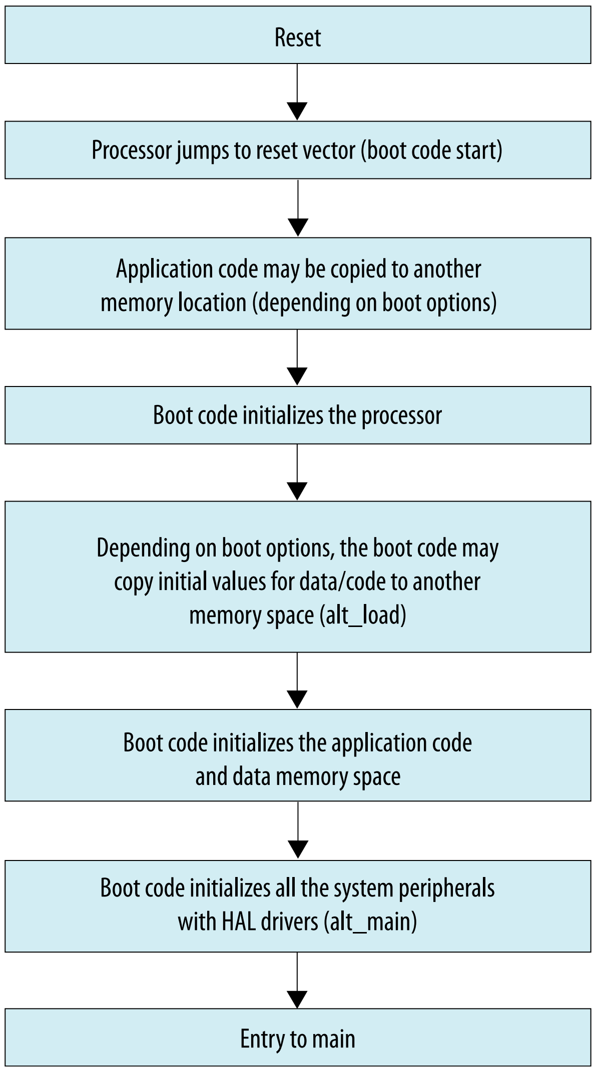
There are a few methods to boot up the Nios V processor in Intel FPGA devices. The methods to boot up Nios V processor vary according to the flash memory selection and device families.

Table 23. Supported Flash Memories with Respective Boot Options

Supported Boot Memories	Device	Nios V Processor Booting Methods	Application Runtime Location	Boot Copier
Configuration QSPI Flash (for Active Serial configuration)	Control block-based devices (with Generic Serial Flash Interface Intel FPGA IP)[2]	Nios V processor application execute-in-place from configuration QSPI flash	Configuration QSPI flash (XIP) + OCRAM/ External RAM (for writable data sections)	`alt_load()` function
		Nios V processor application copied from configuration QSPI flash to RAM using boot copier	OCRAM/ External RAM	GSFI bootloader
	SDM-based devices (with Mailbox Client Intel FPGA IP). [2]	Nios V processor application copied from configuration QSPI flash to RAM using boot copier	OCRAM/ External RAM	SDM bootloader
On-chip Memory (OCRAM)	All supported Intel FPGA devices [2]	Nios V processor application execute-in-place from OCRAM	OCRAM	`alt_load()` function
Tightly Coupled Memory (TCM)	All supported Intel FPGA devices[2]	Nios V processor application execute-in-place from TCM	Instruction TCM (XIP) + Data TCM (for writable data sections)	None

(2) Refer to *AN 980: Nios V Processor Intel Quartus Prime Software Support* for the device list.

Figure 16. Nios V Processor Boot Flow


Reset
Processor jumps to reset vector (boot code start)
Application code may be copied to another memory location (depending on boot options)
Boot code initializes the processor
Depending on boot options, the boot code may copy initial values for data/code to another memory space (alt_load)
Boot code initializes the application code and data memory space
Boot code initializes all the system peripherals with HAL drivers (alt_main)
Entry to main

Related Information

- Generic Serial Flash Interface Intel FPGA IP User Guide
- Mailbox Client Intel FPGA IP User Guide
- AN 980: Nios V Processor Intel Quartus Prime Software Support

4.4. Introduction to Nios V Processor Booting Methods

Nios V processor systems require the software images to be configured in system memory before the processor can begin executing the application program. Refer to Linker Sections for the default linker sections.

The **BSP Editor** generates a linker script that performs the following functions:

- Ensures that the processor software is linked in accordance with the linker settings of the BSP editor and determines where the software resides in memory.
- Positions the processor's code region in the memory component according to the assigned memory components.

The following section briefly describes the available Nios V processor booting methods.

4.4.1. Nios V Processor Application Execute-In-Place from Boot Flash

Intel designed the *Generic Serial Flash Interface Intel FPGA IP* so that the boot flash address space is immediately accessible to the Nios V processor upon system reset, without the need to initialize the memory controller or memory devices. This enables the Nios V processor to execute application code stored on the boot devices directly without using a boot copier to copy the code to another memory type.

When the Nios V processor application execute-in-place from boot flash, the **BSP Editor** performs the following functions:

- Sets the `.text` linker sections to the boot flash memory region.
- Sets the `.bss`,`.rodata`, `.rwdata`, `.stack` and `.heap` linker sections to the RAM memory region.

You must enable the `alt_load()` function in the **BSP Settings** to copy the data sections (`.rodata`, `.rwdata`,, `.exceptions`) to the RAM upon system reset. The code section (`.text`) remains in the boot flash memory region.

Related Information

Generic Serial Flash Interface Intel FPGA IP User Guide

4.4.1.1. alt_load()

You can enable the `alt_load()` function in the HAL code using the **BSP Editor**.

When used in the execute-in-place boot flow, the `alt_load()` function performs the following tasks:

- Operates as a mini boot copier that copies the memory sections to RAM based on the BSP settings.
- Copies data sections (`.rodata`, `.rwdata`, `.exceptions`) to RAM but not the code sections (`.text`).The code section (`.text`) section is a read-only section and remains in the booting flash memory region. This partitioning helps to minimize the RAM usage but may limit the code execution performance because accesses to flash memory are slower than accesses to the on-chip RAM.

The following table lists the BSP Editor settings and functions:

Table 24. BSP Editor Settings

BSP Editor Setting	Function
`hal.linker.enable_alt_load`	Enables `alt_load()` function.
`hal.linker.enable_alt_load_copy_rodata`	`alt_load()` copies `.rodata` section to RAM.
`hal.linker.enable_alt_load_copy_rwdata`	`alt_load()` copies `.rwdata` section to RAM.
`hal.linker.enable_alt_load_copy_exceptions`	`alt_load()` copies `.exceptions` section to RAM.

4.4.2. Nios V Processor Application Copied from Boot Flash to RAM Using Boot Copier

The Nios V processor and HAL include a boot copier that provides sufficient functionality for most Nios V processor applications and is convenient to implement with the Nios V software development flow.

When the application uses a boot copier, it sets all linker sections (`.text`, `.heap` , `.rwdata`, `.rodata` , `.bss`, `.stack`) to an internal or external RAM. Using the boot copier to copy a Nios V processor application from the boot flash to the internal or external RAM for execution helps to improve the execution performance.

For this boot option, the Nios V processor starts executing the boot copier software upon system reset. The software copies the application from the boot flash to the internal or external RAM. Once the process is complete, the Nios V processor transfers the program control over to the application.

Note: If the boot copier is in flash, then the `alt_load()` function does not need to be called because they both serve the same purpose.

4.4.2.1. GSFI Bootloader

The GSFI bootloader is the Nios V processor boot copier that supports QSPI flash memory in control block-based devices. The GSFI bootloader includes the following features:

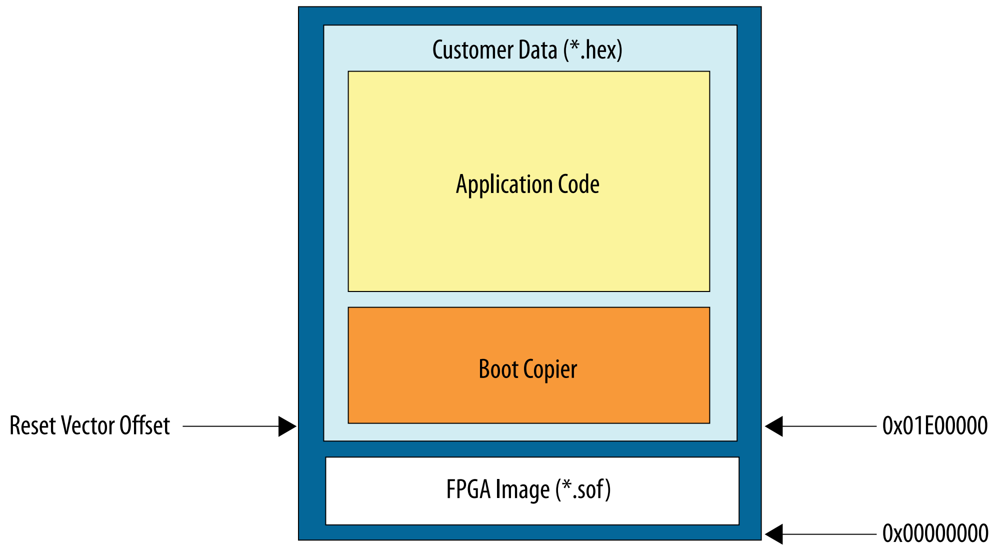
- Locates the software application in non-volatile memory.
- Unpacks and copies the software application image to RAM.
- Automatically switches processor execution to application code in RAM after copy completes.

The boot image is located right after the boot copier. You need to ensure the Nios V processor reset offset points to the start of the boot copier. The Figure: Memory Map for QSPI Flash with GSFI Bootloader memory map for QSPI Flash with GSFI Bootloader shows the flash memory map for QSPI flash when using a boot copier. This memory map assumes the flash memory memory stores the FPGA image and the application software.

Table 25. GSFI Bootloader for Nios V Processor Core

Nios V Processor Core	GSFI Bootloader File Location
Nios V/m processor	`<Intel Quartus Installation Directory>/niosv/components/bootloader/niosv_m_bootloader.srec`
Nios V/g processor	`<Intel Quartus Installation Directory>/niosv/components/bootloader/niosv_g_bootloader.srec`

Figure 17. Memory Map for QSPI Flash with GSFI Bootloader



Note:

1. At the start of the memory map is the FPGA image followed by your data, which consists of boot copier and application code.
2. You must set the Nios V processor reset offset in Platform Designer and point it to the start of the boot copier.
3. The size of the FPGA image is unknown.You can only know the exact size after the Intel Quartus Prime project compilation. You must determine an upper bound for the size of the Intel FPGA image. For example, if the size of the FPGA image is estimated to be less than 0x01E00000, set the Reset Offset to 0x01E00000 in Platform Designer, which is also the start of the boot copier.
4. A good design practice consists of setting the reset vector offset at a flash sector boundary to ensure no partial erase of the FPGA image occurs in case the software application is updated.

4.4.2.2. SDM Bootloader

The SDM bootloader is a HAL application code utilizing the Mailbox Client Intel FPGA IP HAL driver for processor booting. Intel recommends this bootloader application when using the configuration QSPI flash in SDM-based devices to boot the Nios V processor.

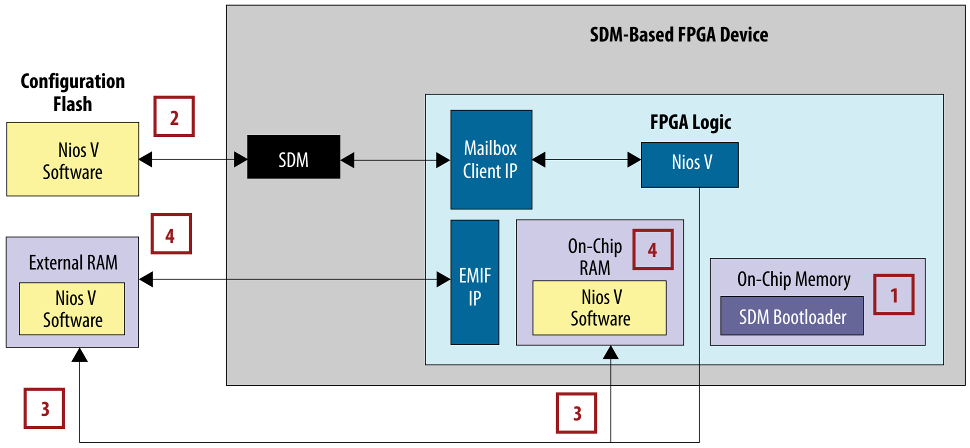
Upon system reset, the Nios V processor first boots the SDM bootloader from a tiny on-chip memory and executes the SDM bootloader to communicate with the configuration QSPI flash using the Mailbox Client IP.

The SDM bootloader performs the following tasks:

- Locates the Nios V software in the configuration QSPI flash.
- Copies the Nios V software into the on-chip RAM or external RAM.
- Switches the processor execution to the Nios V software within the on-chip RAM or external RAM.

Once the process is complete, the SDM bootloader transfers program control over to the user application. Intel recommends the memory organization as outlined in Memory Organization for SDM Bootloader.

Figure 18. SDM Bootloader Process Flow



1. Nios V processor runs the SDM bootloader from the on-chip memory.
2. SDM bootloader communicates with the configuration flash and locate the Nios V software.
3. SDM bootloader copies the Nios V software from the Configuration Flash into on-chip RAM / external RAM.
4. SDM bootloader switches the Nios V processor execution to the Nios V software in the on-chip RAM / external RAM.

4.4.3. Nios V Processor Application Execute-In-Place from OCRAM

In this method, the Nios V processor reset address is set to the base address of the on-chip memory (OCRAM). The application binary (`.hex`) file is loaded into the OCRAM when the FPGA is configured, after the hardware design is compiled in the Intel Quartus Prime software. Once the Nios V processor resets, the application begins executing and branches to the entry point.

Note:
- Execute-In-Place from OCRAM does not require boot copier because Nios V processor application is already in place at system reset.
- Intel recommends enabling `alt_load()` for this booting method so that the embedded software behaves identically when reset without reconfiguring the FPGA device image.
- You must enable the `alt_load()` function in the **BSP Settings** to copy the `.rwdata` section upon system reset. In this method, the initial values for initialized variables are stored separately from the corresponding variables to avoid overwriting on program execution.

4.4.4. Nios V Processor Application Execute-In-Place from TCM

The execute-in-place method sets the Nios V processor reset address to the base address of the tightly coupled memory (TCM). The application binary (`.hex`) file is loaded into the TCM when you configure the FPGA after you compile the hardware design in the Intel Quartus Prime software. Once the Nios V processor resets, the application begins executing and branches to the entry point.

Note: Execute-In-Place from TCM does not require boot copier because Nios V processor application is already in place at system reset.

4.5. Nios V Processor Booting from Configuration QSPI Flash

The Nios V processor supports the following two boot options using configuration QSPI flash under Active Serial configuration mode:

- Nios V processor application executes in-place from configuration QSPI flash.
- Nios V processor application is copied from configuration QSPI flash to RAM using boot copier.

Based on the related Intel FPGA devices, refer to the following sections:

- Control block-based devices:
  - Nios V Processor Application Executes-In-Place from Configuration QSPI Flash
  - Processor Application Copied from Configuration QSPI Flash to RAM Using Boot Copier (GSFI Bootloader)
- SDM-based devices:
  - Nios V Processor Application Copied from Configuration QSPI Flash to RAM Using Boot Copier (SDM Bootloader)

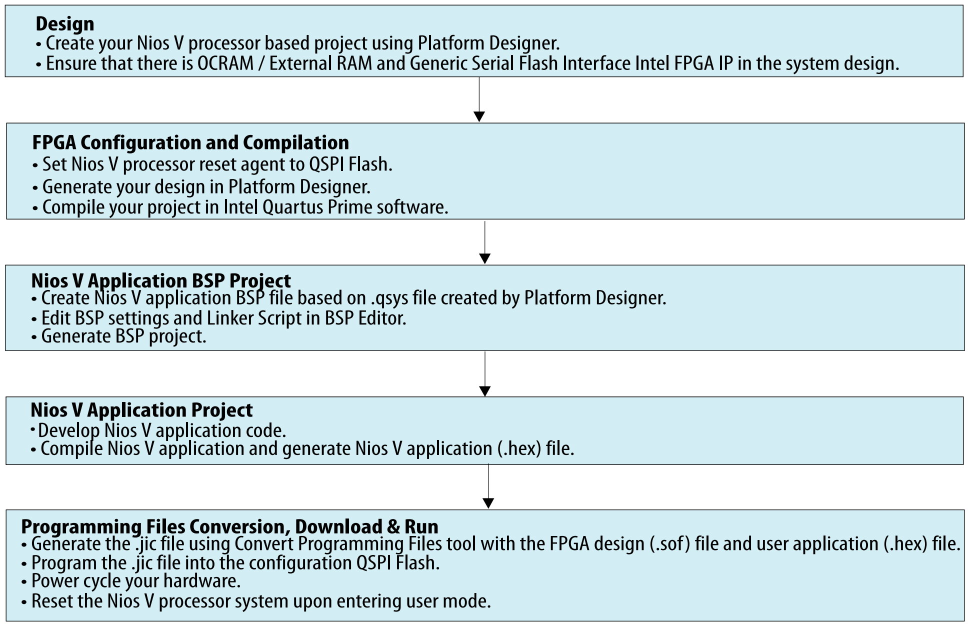
Table 26. Supported Flash Memories with respective Boot Options

Supported Boot Memories	Nios V Booting Methods	Application Runtime Location	Boot Copier
Control block-based devices[3] (with Generic Serial Flash Interface Intel FPGA IP)	Nios V processor application execute-in-place from configuration QSPI flash	Configuration QSPI flash (XIP) + OCRAM/ External RAM (for writable data sections)	alt_load() function
	Nios V processor application copied from configuration QSPI flash to RAM using boot copier	OCRAM/ External RAM	GSFI bootloader
SDM-based devices[3] (with Mailbox Client Intel FPGA IP	Nios V processor application copied from configuration QSPI flash to RAM using boot copier	OCRAM/ External RAM	SDM bootloader

[3] Refer to *AN 980: Nios V Processor Intel Quartus Prime Software Support* for the device list.

4.5.1. Nios V Processor Design, Configuration and Boot Flow (Control Block-based Device)

Figure 19. Design, Configuration and Booting Flow (Control Block-based Device)


Design
• Create your Nios V processor based project using Platform Designer.
• Ensure that there is OCRAM / External RAM and Generic Serial Flash Interface Intel FPGA IP in the system design.
FPGA Configuration and Compilation
• Set Nios V processor reset agent to QSPI Flash.
• Generate your design in Platform Designer.
• Compile your project in Intel Quartus Prime software.
Nios V Application BSP Project
• Create Nios V application BSP file based on .qsys file created by Platform Designer.
• Edit BSP settings and Linker Script in BSP Editor.
• Generate BSP project.
Nios V Application Project
• Develop Nios V application code.
• Compile Nios V application and generate Nios V application (.hex) file.
Programming Files Conversion, Download & Run
• Generate the .jic file using Convert Programming Files tool with the FPGA design (.sof) file and user application (.hex) file.
• Program the .jic file into the configuration QSPI Flash.
• Power cycle your hardware.
• Reset the Nios V processor system upon entering user mode.

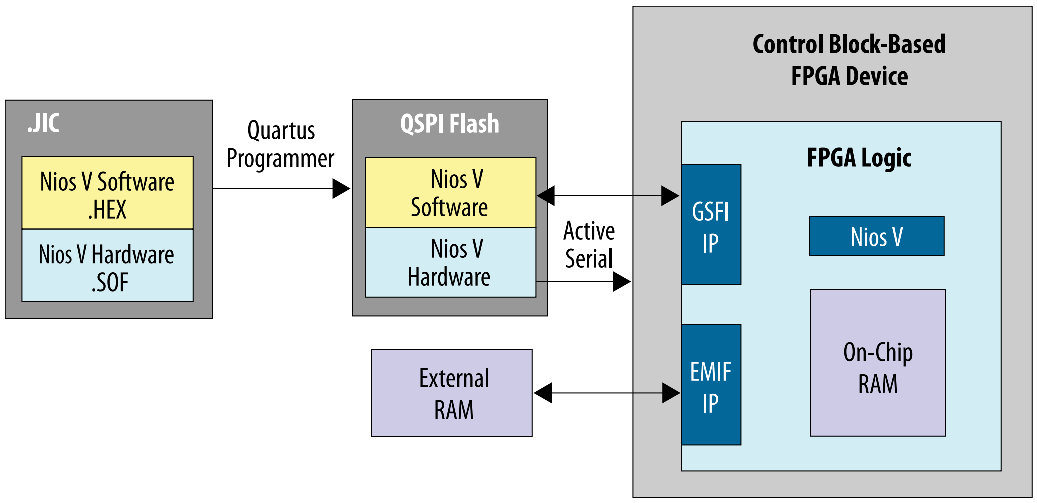
4.5.1.1. Nios V Processor Application Executes-In-Place from Configuration QSPI Flash

The execute-in-place (XIP) option is suitable for Nios V processor application, when only a limited amount of on-chip memory is available to the processor. This boot option is only available for control block-based devices.

The `alt_load()` function operates as a mini boot copier that initializes and copies the writable memory sections only to OCRAM or external RAM. The code section (`.text`), which is a read-only section, remains in the configuration QSPI flash memory region. Retaining the read-only section in configuration QSPI minimizes RAM usage but may limit the code execution performance.

The Nios V processor application is programmed into the configuration QSPI flash. The Nios V processor reset the agent points to the configuration QSPI flash to allow code execution after the system resets.

Figure 20. Nios V Processor Application Executes-In-Place from Configuration QSPI Flash



Related Information

GSFI Bootloader Example Design on page 70

4.5.1.1.1. Hardware Design Flow

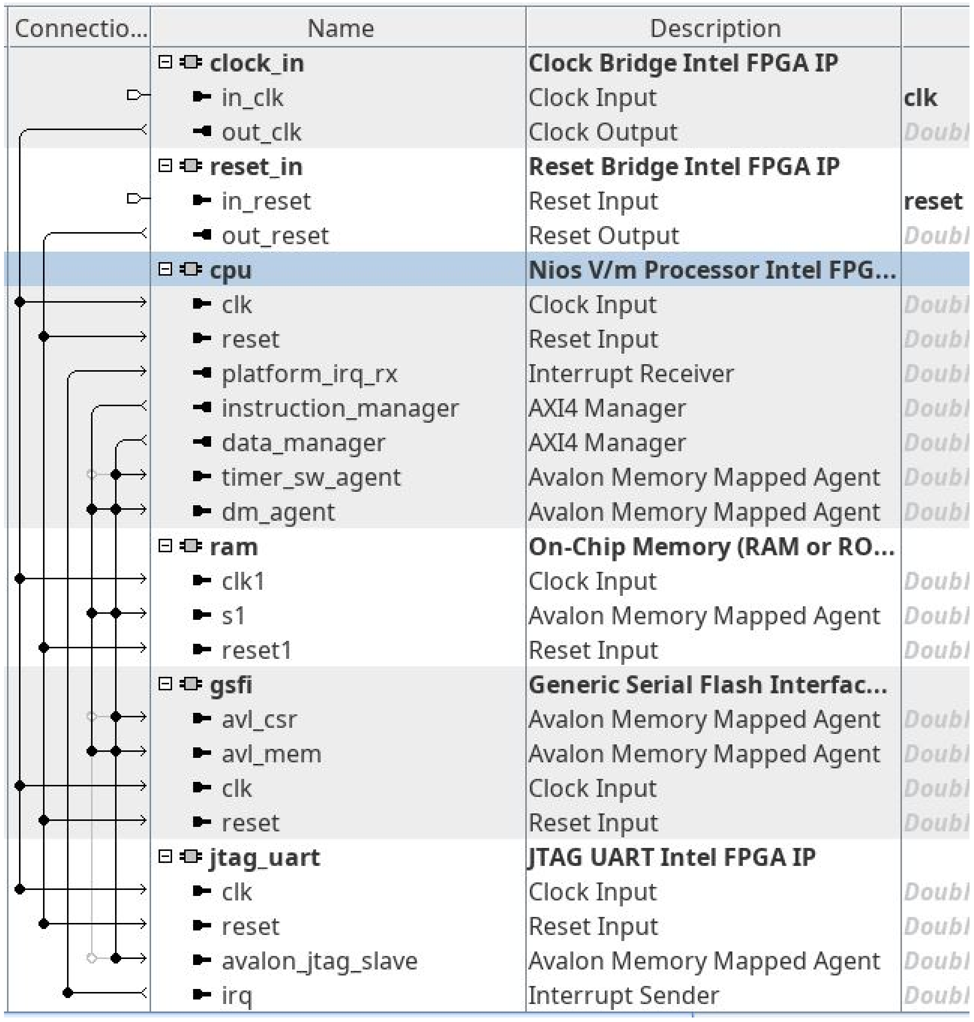
The following sections describe the steps for building a bootable system for a Nios V processor application, which executes in place from the configuration QSPI flash.

The following example is built using an Intel Arria 10 SoC Development Kit.

IP Component Settings

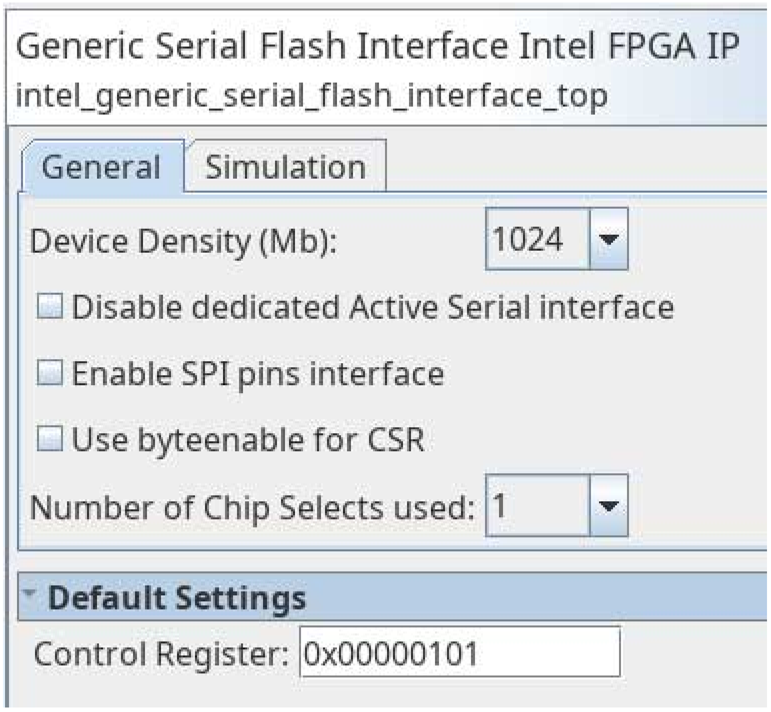
1. Create your Nios V processor project using Intel Quartus Prime and Platform Designer.
2. Add Generic Serial Flash Interface Intel FPGA IP into your Platform Designer.

Figure 21. Connections for Nios V Processor Project



Connectio...	Name	Description	
	clock_in	**Clock Bridge Intel FPGA IP**	
	in_clk	Clock Input	**clk**
	out_clk	Clock Output	Doubl
	reset_in	**Reset Bridge Intel FPGA IP**	
	in_reset	Reset Input	**reset**
	out_reset	Reset Output	Doubl
	cpu	**Nios V/m Processor Intel FPG...**	
	clk	Clock Input	Doubl
	reset	Reset Input	Doubl
	platform_irq_rx	Interrupt Receiver	Doubl
	instruction_manager	AXI4 Manager	Doubl
	data_manager	AXI4 Manager	Doubl
	timer_sw_agent	Avalon Memory Mapped Agent	Doubl
	dm_agent	Avalon Memory Mapped Agent	Doubl
	ram	**On-Chip Memory (RAM or RO...**	
	clk1	Clock Input	Doubl
	s1	Avalon Memory Mapped Agent	Doubl
	reset1	Reset Input	Doubl
	gsfi	**Generic Serial Flash Interfac...**	
	avl_csr	Avalon Memory Mapped Agent	Doubl
	avl_mem	Avalon Memory Mapped Agent	Doubl
	clk	Clock Input	Doubl
	reset	Reset Input	Doubl
	jtag_uart	**JTAG UART Intel FPGA IP**	
	clk	Clock Input	Doubl
	reset	Reset Input	Doubl
	avalon_jtag_slave	Avalon Memory Mapped Agent	Doubl
	irq	Interrupt Sender	Doubl

Figure 22. Generic Serial Flash Interface Intel FPGA IP Parameter Settings


Generic Serial Flash Interface Intel FPGA IP
intel_generic_serial_flash_interface_top

Device Density (Mb): 1024
☐ Disable dedicated Active Serial interface
☐ Enable SPI pins interface
☐ Use byteenable for CSR
Number of Chip Selects used: 1

Default Settings
Control Register: 0x00000101

3. Change the **Device Density (Mb)** according to the QSPI flash size.
4. Change the addressing mode by modifying bit 8 of the **Control Register** value in the **Default Settings** parameter section. Changing bit 8 to 0x0 enables 3-byte addressing, or 0x1 enables 4-byte addressing.

Note: Refer to **Intel Supported Configuration Devices tab ➤ Intel Supported Third Party Configuration Devices** in the *Device Configuration Support Center* to check the byte addressing mode supported for each flash device in each Intel FPGA device.

For example, Intel Arria 10 devices support the 4-byte addressing mode when used with Micron flash devices .

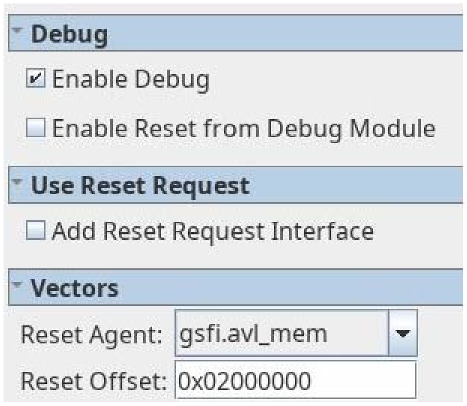
Reset Agent Settings for Nios V Processor Execute-In-Place Method

1. In the Nios V processor IP parameter editor, set the **Reset Agent** to QSPI Flash.
   a. Your (`.sof`) image size influences your reset offset configuration. The reset offset is the start address of the HEX file in QSPI flash and it must point to a location after the (`.sof`) image. If the (`.sof`) image space and the reset offset location overlap, Intel Quartus Prime software displays an overlap error. You can determine the minimum reset offset by using the configuration bitstream size from the device datasheet.

      Refer to the following example:

- The uncompressed configuration bitstream size for Intel Arria 10 GX 660 is 252,959,072 bits (31,619,884 bytes).
- If the SOF image starts at address 0x0, the SOF image can extend up to address 0x1E27B2C. In this case, the minimum reset offset you can select to avoid overlap errors is 0x1E27B30.
- Intel recommends you to use a flash sector boundary address for the reset offset. Doing so allows you to update the application software image at a later time without interfering with the FPGA image.

Figure 23. Parameter Editor Settings



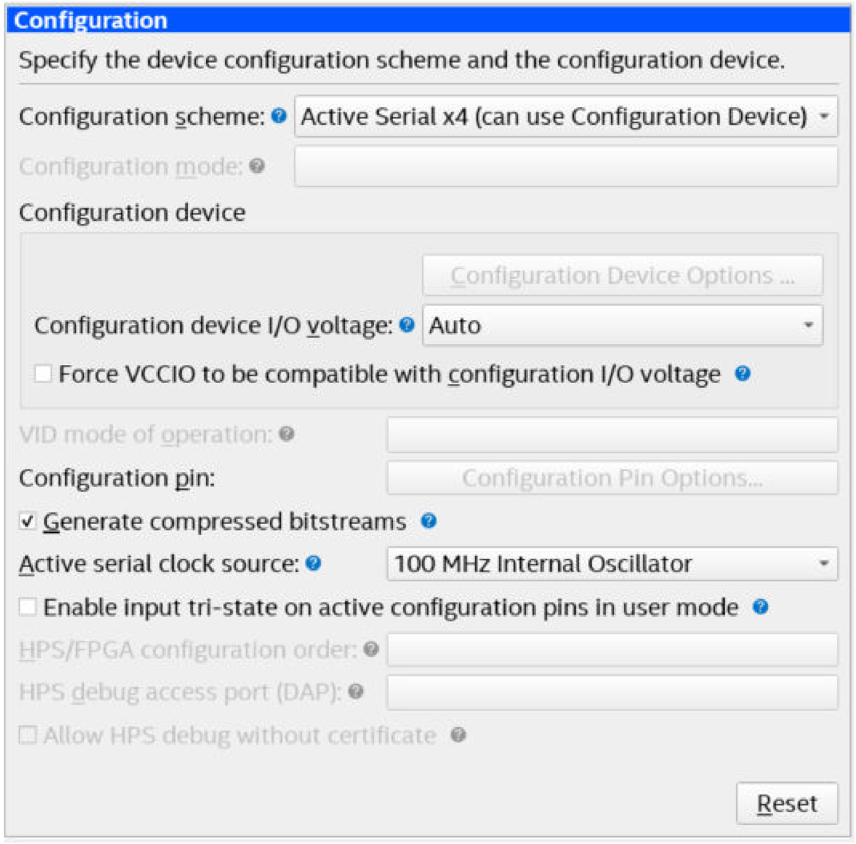
Debug
☑ Enable Debug
☐ Enable Reset from Debug Module
Use Reset Request
☐ Add Reset Request Interface
Vectors
Reset Agent: gsfi.avl_mem
Reset Offset: 0x02000000

2. Click **Generate HDL**, the Generation dialog box appears.
3. Specify output file generation options and then click **Generate**.

Intel Quartus Prime Software Settings

1. In the Intel Quartus Prime software, click **Assignment ➤ Device ➤ Device and Pin Options ➤ Configuration**.
2. Set **Configuration scheme** to **Active Serial x4 (can use Configuration Device)**.
3. Set the **Active serial clock source** to **100 MHz Internal Oscillator**.

Figure 24. Device and Pin Options


Configuration

Specify the device configuration scheme and the configuration device.

Configuration scheme: Active Serial x4 (can use Configuration Device)

Configuration mode:

Configuration device

Configuration Device Options ...

Configuration device I/O voltage: Auto

☐ Force VCCIO to be compatible with configuration I/O voltage

VID mode of operation:

Configuration pin: Configuration Pin Options...

☑ Generate compressed bitstreams

Active serial clock source: 100 MHz Internal Oscillator

☐ Enable input tri-state on active configuration pins in user mode

HPS/FPGA configuration order:

HPS debug access port (DAP):

☐ Allow HPS debug without certificate

Reset

4. Click **OK** to exit the **Device and Pin Options** window.
5. Click **OK** to exit the **Device** window.
6. Click **Start Compilation** to compile your project.

Related Information

Intel Arria 10 SoC Development Kit

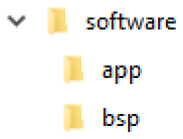
4.5.1.1.2. Software Design Flow

This section provides the design flow to generate and build a Nios V processor software project. To ensure a streamlined build flow, you are encouraged to create a similar directory tree in your design project. The software design flow is based on this directory tree.

Use the following steps to create the software project directory tree:

1. In your design project folder, create a folder called `software`.
2. In the `software` folder, create two folders called app and bsp.

Figure 25. Software Project Directory Tree



Creating the BSP Project Application

You must edit the BSP editor settings according to the selected Nios V processor boot options.

To launch the BSP Editor, perform the following steps:

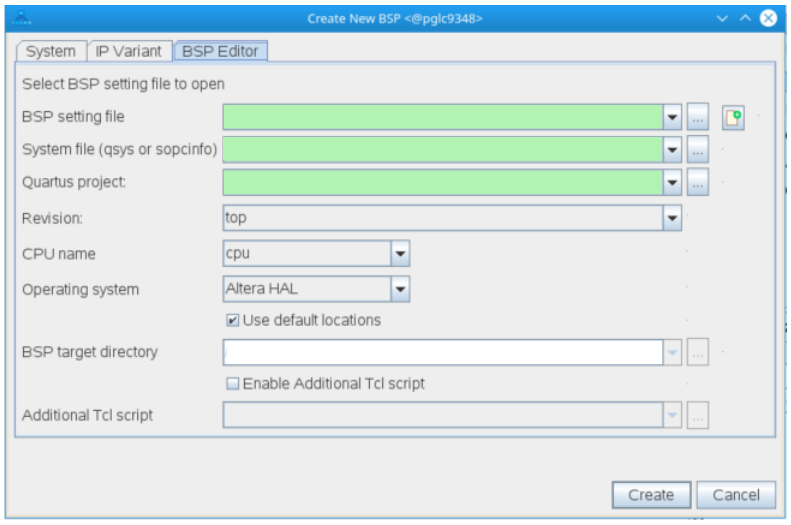
1. In the Platform Designer window, select **File ➤ New BSP**. The **Create New BSP** windows appears.
2. For **BSP setting file**, navigate to the `software/bsp` folder and name the BSP as `settings.bsp`.

BSP path: `<project directory>/software/bsp/settings.bsp`

1. For **System file (qsys or sopcinfo)**, select the Nios V processor Platform Designer system (`*.qsys`).

   Note: For Intel Quartus Prime Standard Edition software, generate the BSP file using SOPCINFO file. Refer to *AN 980: Nios V Processor Intel Quartus Prime Software Support* for more information.
2. For **Quartus project**, select the Intel Quartus Prime Project File.
3. For **Revision**, select the correct revision.
4. For **CPU name**, select the Nios V processor.
5. Select the **Operating system** as **Altera HAL**.
6. Click **Create** to create the BSP file.

Figure 26. Create New BSP window



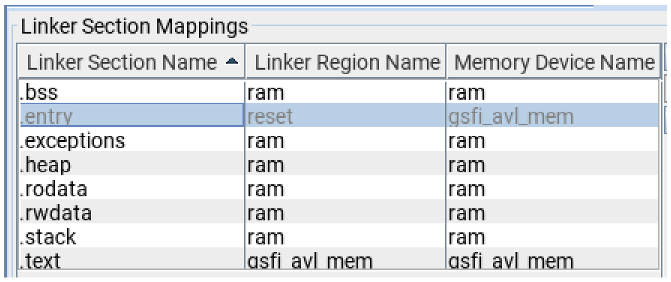
Select BSP setting file to open

BSP setting file	
System file (qsys or sopcinfo)	
Quartus project:	
Revision:	top
CPU name	cpu
Operating system	Altera HAL
	☑ Use default locations
BSP target directory	
	☐ Enable Additional Tcl script
Additional Tcl script	

Configuring BSP Editor and Generating the BSP Project

1. In the **BSP Editor**, click **BSP Linker Script**.
2. In the **Linker Section Name** perform the following settings:
   a. Set `.text` to the QSPI flash in the **Linker Region Name**.
   b. Set `.exceptions` to OCRAM/ External RAM or QSPI Flash according to your design preference.
   c. Set the rest of the items to the OCRAM or external RAM.

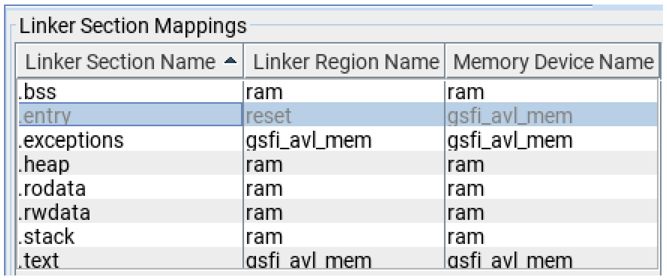
Figure 27. Linker Region Settings When Exceptions is set to OCRAM/ External RAM



Linker Section Mappings

Linker Section Name	Linker Region Name	Memory Device Name
.bss	ram	ram
.entry	reset	qsfi_avl_mem
.exceptions	ram	ram
.heap	ram	ram
.rodata	ram	ram
.rwdata	ram	ram
.stack	ram	ram
.text	qsfi_avl_mem	qsfi_avl_mem

Figure 28. Linker Region Settings When Exceptions is set to QSPI Flash

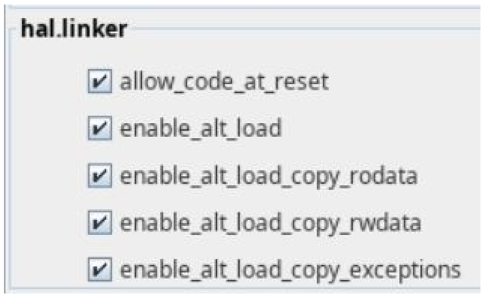


Linker Section Mappings

Linker Section Name	Linker Region Name	Memory Device Name
.bss	ram	ram
.entry	reset	qsfi_avl_mem
.exceptions	qsfi_avl_mem	qsfi_avl_mem
.heap	ram	ram
.rodata	ram	ram
.rwdata	ram	ram
.stack	ram	ram
.text	qsfi_avl_mem	qsfi_avl_mem

3. Go to **Main ➤ Settings ➤ Advanced ➤ hal.linker**.
4. If exception is set to OCRAM or External RAM, enable the following:
   - `allow_code_at_reset`
   - `enable_alt_load`
   - `enable_alt_load_copy_rodata`
   - `enable_alt_load_copy_rwdata`
   - `enable_alt_load_copy_exceptions`

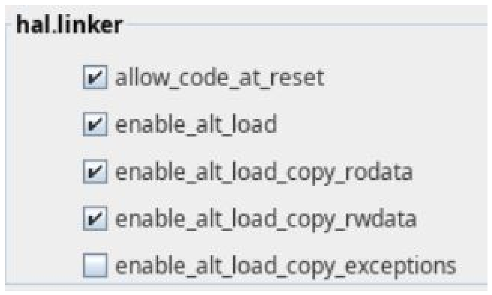
Figure 29. hal.linker Settings for Exception Agent OCRAM or External RAM



hal.linker
- [✓] allow_code_at_reset
- [✓] enable_alt_load
- [✓] enable_alt_load_copy_rodata
- [✓] enable_alt_load_copy_rwdata
- [✓] enable_alt_load_copy_exceptions

5. If exception is set to QSPI flash, enable the following:
   - `allow_code_at_reset`
   - `enable_alt_load`
   - `enable_alt_load_copy_rodata`
   - `enable_alt_load_copy_rwdata`

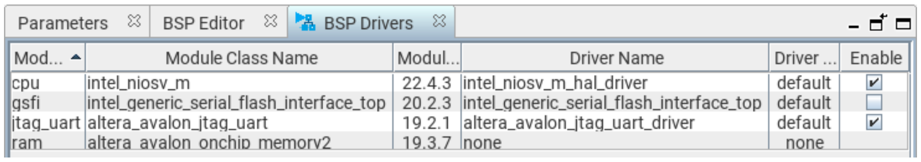
Figure 30. hal.linker Settings for QSPI Flash



hal.linker
- [✓] allow_code_at_reset
- [✓] enable_alt_load
- [✓] enable_alt_load_copy_rodata
- [✓] enable_alt_load_copy_rwdata
- [ ] enable_alt_load_copy_exceptions

6. Navigate to the **BSP Drivers** tab.
7. Disable the Generic Serial Flash Interface driver (`intel_generic_serial_flash_interface_top`).

Figure 31. BSP Drivers



Parameters | BSP Editor | BSP Drivers

Mod...	Module Class Name	Modul...	Driver Name	Driver ...	Enable
cpu	intel_niosv_m	22.4.3	intel_niosv_m_hal_driver	default	✓
gsfi	intel_generic_serial_flash_interface_top	20.2.3	intel_generic_serial_flash_interface_top	default	
jtag_uart	altera_avalon_jtag_uart	19.2.1	altera_avalon_jtag_uart_driver	default	✓
ram	altera_avalon_onchip_memory2	19.3.7	none	none	

8. Return to the **BSP Editor** tab and click **Generate BSP**. Make sure the BSP generation is successful.
9. Close the **BSP Editor**.

Generating the Application Project File

1. Navigate to the `software/app` folder and create your Nios V application source code.
2. Launch the Nios V Command Shell.
3. Execute the command below to generate the application `CMakeLists.txt`.

```
niosv-app --app-dir=software/app --bsp-dir=software/bsp \
  --srcs=software/app/<Nios V application source code>
```

Building the Application Project

You can choose to build the application project using the RiscFree IDE for Intel FPGAs, Eclipse Embedded CDT or through the command line interface (CLI).

If you prefer using CLI, you can build the application using the following command:

```
cmake -G "Unix Makefiles" -DCMAKE_BUILD_TYPE=Debug -B \
  software/app/debug -S software/app
```

```
 make -C software/app/debug
```

The application (`.elf`) file is created in `software/app/debug` folder.

Generating HEX File

You must generate a `.hex` file from your application `.elf` file, so you can create a `.jic` file suitable for programming flash devices.

1. Launch the Nios V Command Shell.
2. For Nios V processor application execute-in-place (XIP) from configuration QSPI flash, use the following commands line to convert the ELF to HEX for your application. The commands create the application (`.hex`) file.

```
elf2flash --input software/app/debug/<Nios V application>.elf \
  --output flash.srec --reset <reset offset + base address of GSFI AVL MEM> \
  --base <base address of GSFI AVL MEM> \
  --end <end address of GSFI AVL MEM>
```

```
riscv32-unknown-elf-objcopy --input-target srec --output-target ihex \
  flash.srec <Nios V application>.hex
```

Related Information

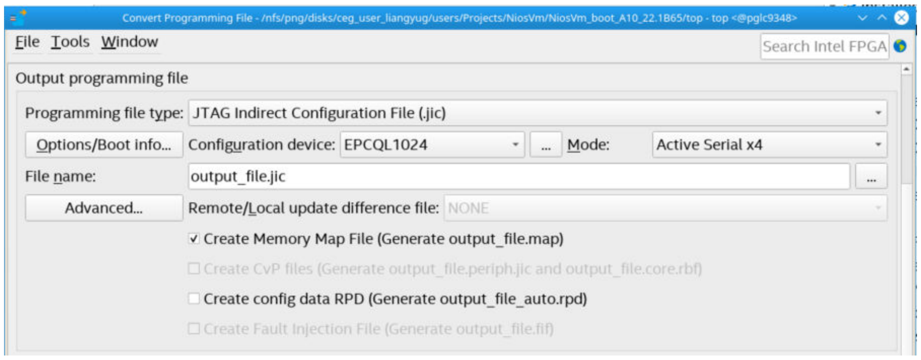
Summary of Nios V Processor Vector Configuration and BSP Settings on page 105

4.5.1.1.3. Programming Files Generation

JTAG Indirect Configuration File (`.jic`) Generation

1. In the Intel Quartus Prime software, go to **File ➤ Convert Programming Files**.
2. For **Programming file type**, select **JTAG Indirect Configuration File (.jic)**
3. For **Mode** select **Active Serial x4**.

Figure 32. Convert Programming File Window


Convert Programming File - /nfs/png/disks/ceg_user_liangyug/users/Projects/NiosVm/NiosVm_boot_A10_22.1B65/top - top <@pglc9348>

Output programming file

Programming file type: JTAG Indirect Configuration File (.jic)

Options/Boot info... Configuration device: EPCQL1024 ... Mode: Active Serial x4

File name: output_file.jic

Advanced... Remote/Local update difference file: NONE

☑ Create Memory Map File (Generate output_file.map)

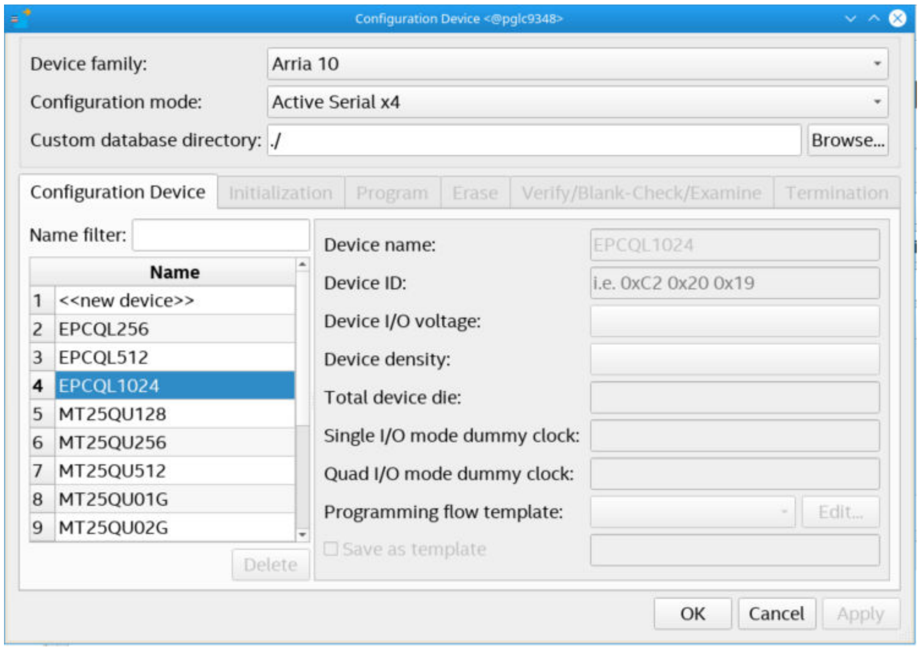
☐ Create CvP files (Generate output_file.periph.jic and output_file.core.rbf)

☐ Create config data RPD (Generate output_file_auto.rpd)

☐ Create Fault Injection File (Generate output_file.fif)

4. Click **"..."** to enter the **Configuration Device** tab and select the available options. The **Configuration Device** allows for choosing a specific supported device or alternatively an unsupported device.

Figure 33. Configuration Device Window


Configuration Device <@pglc9348>

Device family: Arria 10

Configuration mode: Active Serial x4

Custom database directory: ./ Browse...

Name filter:

	Name
1	<<new device>>
2	EPCQL256
3	EPCQL512
4	EPCQL1024
5	MT25QU128
6	MT25QU256
7	MT25QU512
8	MT25QU01G
9	MT25QU02G

Device name: EPCQL1024

Device ID: i.e. 0xC2 0x20 0x19

Device I/O voltage:

Device density:

Total device die:

Single I/O mode dummy clock:

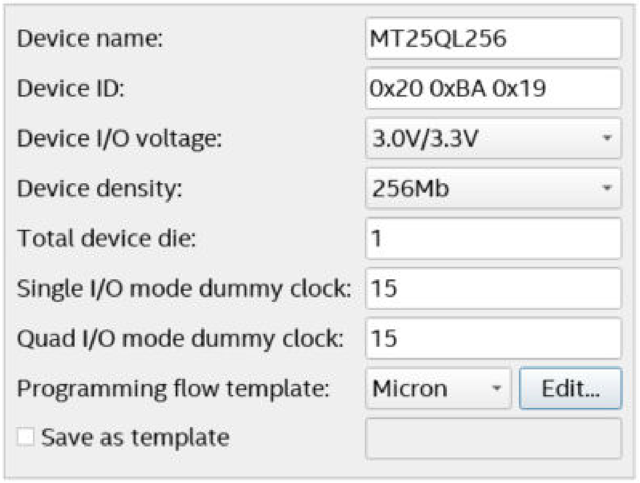
Quad I/O mode dummy clock:

Programming flow template:

☐ Save as template

5. If you are using a supported device, make your selection, and click **OK**. Else, proceed with the following steps:
   a. Select **<<new device>>**.
   b. Enter the information about **Device name**, **Device ID**, **Device I/O voltage**, **Device density**, **Total device die**, **Dummy clock (Single I/O or Quad I/O mode)** and **Programming flow template**.
   c. Click **Apply**.

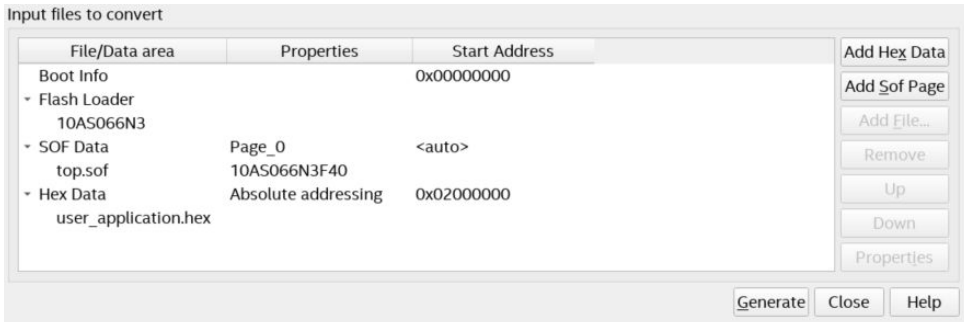
Note: The **Programming flow template** helps you define a template for flash operation in Initialization, Program, Erase, Verify/Blank-Check/ Examine and Termination. If the device is not available for selection, refer to **Modifying Programming Flows** in *Generic Flash Programmer User Guide* to modify the programming flow. For details about memory parameters like dummy clock cycles, please contact the related vendor.



Device name:	MT25QL256
Device ID:	0x20 0xBA 0x19
Device I/O voltage:	3.0V/3.3V
Device density:	256Mb
Total device die:	1
Single I/O mode dummy clock:	15
Quad I/O mode dummy clock:	15
Programming flow template:	Micron
Save as template	

6. Under the **Input files to convert** tab,
   a. Choose the Flash Loader for the FPGA used by selecting **Flash Loader** and click **Add Device**.
   b. Add the `.sof` file to the SOF Data by selecting **SOF Data** and click **Add File**.
   c. Click **Add Hex Data** to add Nios V application (`.hex`) file. Select the **Absolute addressing** and **Big-endian** button. Browse to the `.hex` file location. Click **OK**.
7. Click **Generate** to generate the JIC file.

Figure 34. Input files to convert tab



Input files to convert

File/Data area	Properties	Start Address
Boot Info		0x00000000
Flash Loader		
10AS066N3		
SOF Data	Page_0	<auto>
top.sof	10AS066N3F40	
Hex Data	Absolute addressing	0x02000000
user_application.hex		

4.5.1.1.4. QSPI Flash Programming

Intel FPGA Device QSPI Flash Programming

1. Ensure that the Intel FPGA device's Active Serial (AS) pin is routed to the QSPI flash. This routing allows the flash loader to load into the QSPI flash and configure the board correctly.
2. Ensure the MSEL pin setting on the board is configured for AS programming.
3. Open the Intel Quartus Prime Programmer and make sure JTAG was detected under the **Hardware Setup**.
4. Select **Auto Detect** and choose the FPGA device according to your board.
5. Right-click the selected Intel FPGA device and select **Edit ➤ Change File**. Next, select the generated JIC file.
6. Select the **Program/ Configure** check boxes for the FPGA and QSPI devices.
7. Click **Start** to start programming.

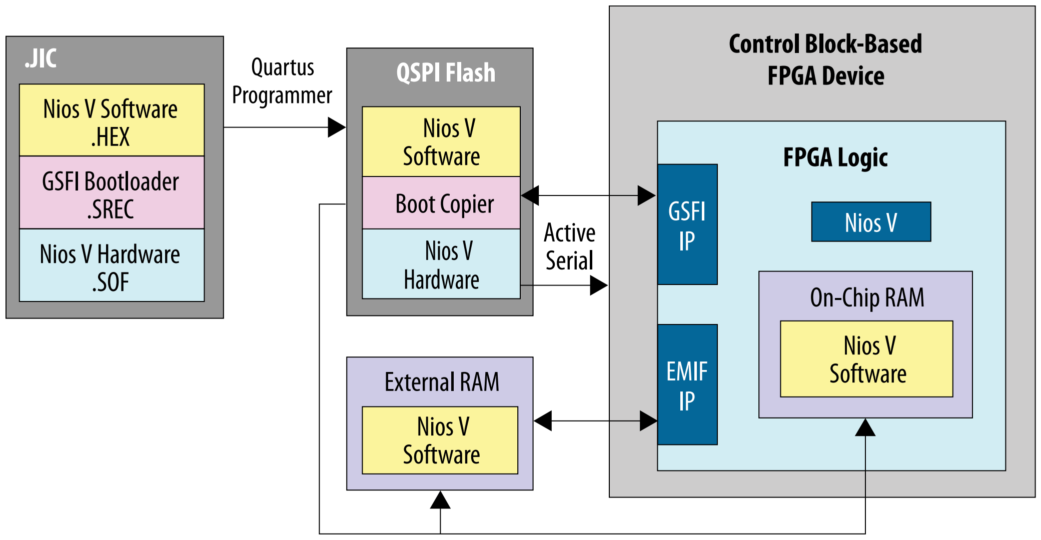
Note: Power cycle the device to begin Active Serial configuration scheme, and reset the Nios V processor system upon entering user mode.

4.5.1.2. Nios V Processor Application Copied from Configuration QSPI Flash to RAM Using Boot Copier (GSFI Bootloader)

You can use a boot copier to copy the Nios V processor application from the configuration QSPI flash to RAM when you require multiple iterations of the application software development and high system performance.

The boot copier is located at the Nios V processor reset address in flash, and is immediately followed by the application. For this boot option, the Nios V processor starts executing the boot copier software upon system reset, which copies the application from the configuration QSPI to the internal or external RAM. Once copying is complete, the Nios V processor transfers the program control over to the application.

Figure 35. Nios V Processor Application Copied from Configuration QSPI Flash to RAM Using Boot Copier (GSFI Bootloader)



Related Information

GSFI Bootloader Example Design on page 70

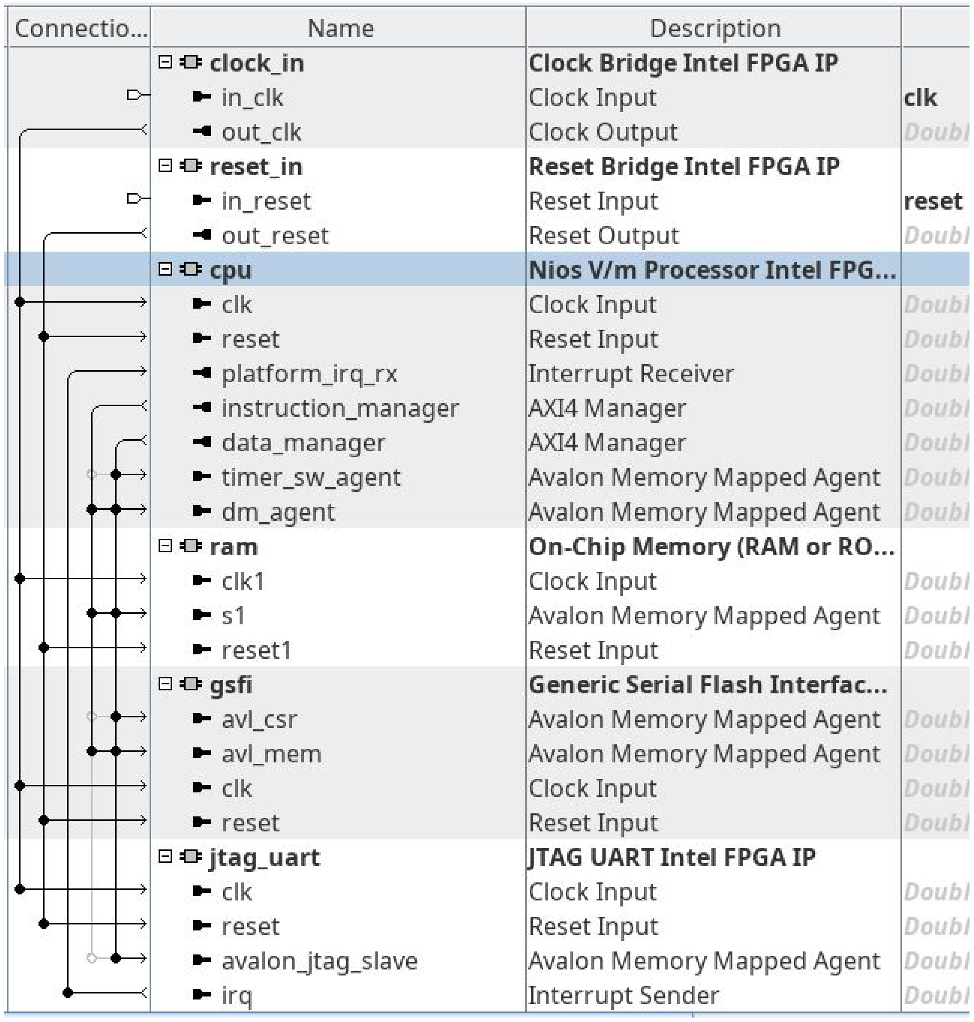
4.5.1.2.1. Hardware Design Flow

The following sections describe a step-by-step method for building a bootable system for a Nios V processor application copied from configuration QSPI flash to RAM using GSFI Bootloader. The following example is built using Intel Arria 10 SoC Development Kit.

IP Component Settings

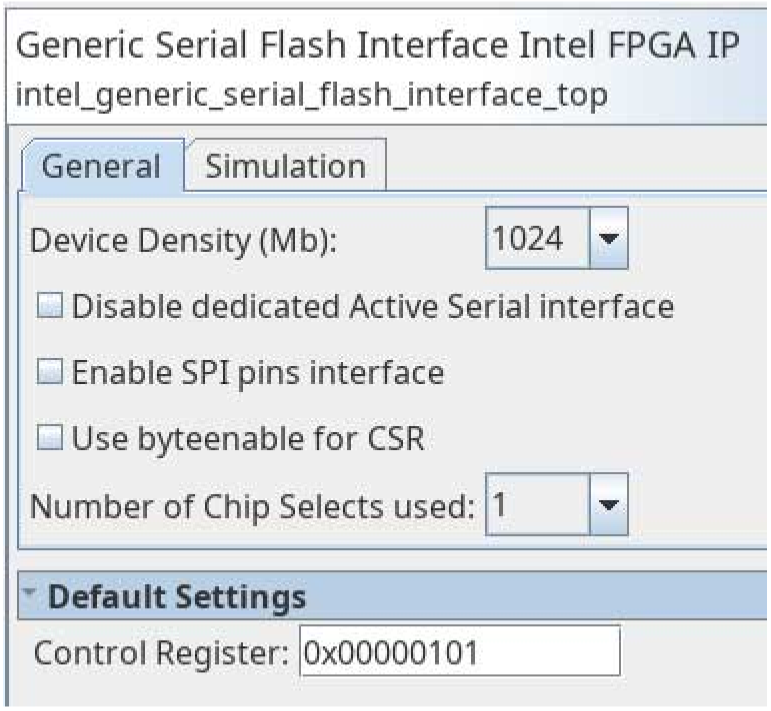
1. Create your Nios V processor project using Intel Quartus Prime and Platform Designer.
2. Add the Generic Serial Flash Interface Intel FPGA IP is into your Platform Designer system.

Figure 36. Connections for Nios V Processor Project



Connectio...	Name	Description	
	clock_in	**Clock Bridge Intel FPGA IP**	
	in_clk	Clock Input	**clk**
	out_clk	Clock Output	Doubl
	reset_in	**Reset Bridge Intel FPGA IP**	
	in_reset	Reset Input	**reset**
	out_reset	Reset Output	Doubl
	cpu	**Nios V/m Processor Intel FPG...**	
	clk	Clock Input	Doubl
	reset	Reset Input	Doubl
	platform_irq_rx	Interrupt Receiver	Doubl
	instruction_manager	AXI4 Manager	Doubl
	data_manager	AXI4 Manager	Doubl
	timer_sw_agent	Avalon Memory Mapped Agent	Doubl
	dm_agent	Avalon Memory Mapped Agent	Doubl
	ram	**On-Chip Memory (RAM or RO...**	
	clk1	Clock Input	Doubl
	s1	Avalon Memory Mapped Agent	Doubl
	reset1	Reset Input	Doubl
	gsfi	**Generic Serial Flash Interfac...**	
	avl_csr	Avalon Memory Mapped Agent	Doubl
	avl_mem	Avalon Memory Mapped Agent	Doubl
	clk	Clock Input	Doubl
	reset	Reset Input	Doubl
	jtag_uart	**JTAG UART Intel FPGA IP**	
	clk	Clock Input	Doubl
	reset	Reset Input	Doubl
	avalon_jtag_slave	Avalon Memory Mapped Agent	Doubl
	irq	Interrupt Sender	Doubl

Figure 37. Generic Serial Flash Interface Intel FPGA IP Parameter Settings


Generic Serial Flash Interface Intel FPGA IP
intel_generic_serial_flash_interface_top

Device Density (Mb): 1024

☐ Disable dedicated Active Serial interface

☐ Enable SPI pins interface

☐ Use byteenable for CSR

Number of Chip Selects used: 1

Default Settings

Control Register: 0x00000101

3. Change the **Device Density (Mb)** according to the QSPI flash size.
4. Change the addressing mode by modifying bit 8 of the **Control Register** value in the **Default Settings** parameter section. Changing bit 8 to 0x0 enables 3-byte addressing, or 0x1 enables 4-byte addressing

Note: Refer to **Intel Supported Configuration Devices tab ➤ Intel Supported Third Party Configuration Devices** in *Device Configuration Support Center* to check the byte addressing mode supported for each flash device in each Intel FPGA device.

For example, Intel Arria 10 devices when used with Micron flash devices support the 4-byte addressing mode.

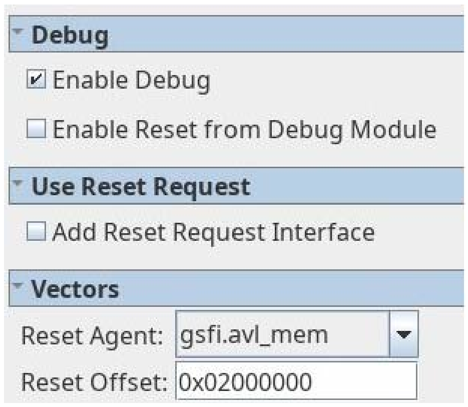
Reset Agent Settings for Nios V Processor Boot-copier Method

1. In the Nios V processor parameter editor, set the **Reset Agent** to QSPI Flash.

   Note: Your SOF image size influences your reset offset configuration. The reset offset is the start of the address of the HEX file in QSPI flash and it must point to a location after the SOF image. If the SOF image space and the reset offset location overlap, Intel Quartus Prime software displays and overlap error. You can determine the minimum reset offset by using the configuration bitstream size from the device datasheet.

   For example, the uncompressed configuration bitstream size for Intel Arria 10 GX 660 is 252,959,072 bits (31,619,884 bytes). If the SOF image starts at address 0x0, the SOF image can extend up to address 0x1E27FFF (0x1E27B2C). In this case, the minimum reset offset you can select is 0x2000000.

Figure 38. Nios V Parameter Editor Settings



Debug

☑ Enable Debug

☐ Enable Reset from Debug Module

Use Reset Request

☐ Add Reset Request Interface

Vectors

Reset Agent: gsfi.avl_mem

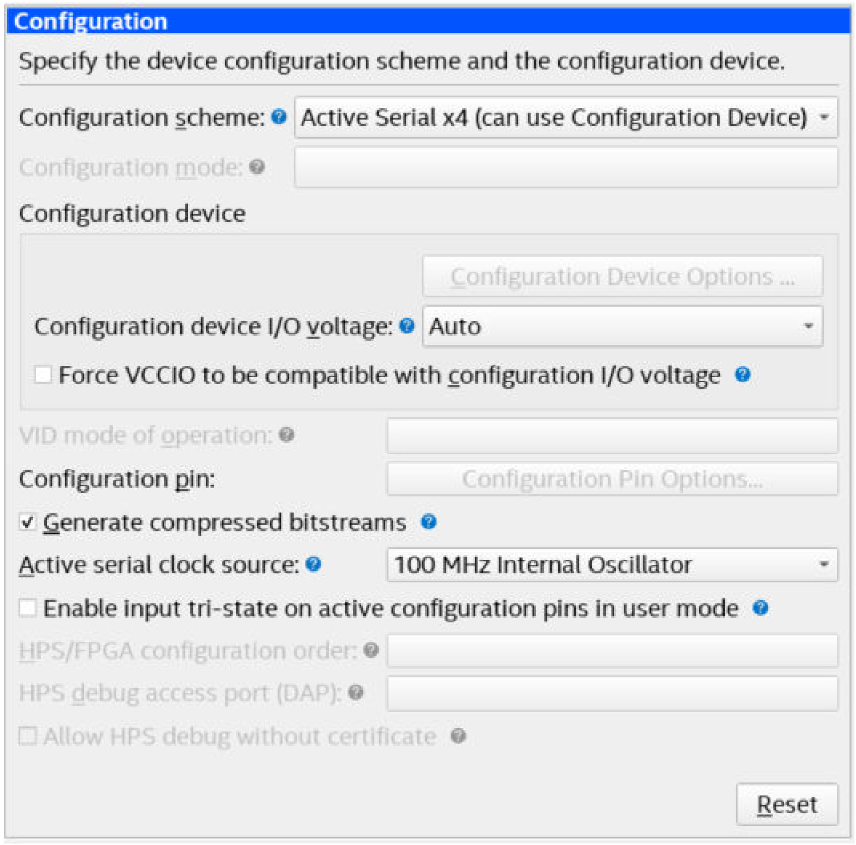
Reset Offset: 0x02000000

2. Click **Generate HDL**, the Generation dialog box appears.
3. Specify output file generation options and then click **Generate**.

Intel Quartus Prime Software Settings

1. In the Intel Quartus Prime software, click **Assignment ➤ Device ➤ Device and Pin Options ➤ Configuration** .
2. Set **Configuration scheme** to **Active Serial x4 (can use Configuration Device)**.
3. Set the **Active serial clock source** to **100 MHz Internal Oscillator**.

Figure 39. Device and Pin Options


Configuration

Specify the device configuration scheme and the configuration device.

Configuration scheme: Active Serial x4 (can use Configuration Device)

Configuration mode:

Configuration device

Configuration Device Options ...

Configuration device I/O voltage: Auto

☐ Force VCCIO to be compatible with configuration I/O voltage

VID mode of operation:

Configuration pin: Configuration Pin Options...

☑ Generate compressed bitstreams

Active serial clock source: 100 MHz Internal Oscillator

☐ Enable input tri-state on active configuration pins in user mode

HPS/FPGA configuration order:

HPS debug access port (DAP):

☐ Allow HPS debug without certificate

Reset

4. Click **OK** to exit the **Device and Pin Options** window.
5. Click **OK** to exit the **Device** window.
6. Click **Start Compilation** to compile your project.

Related Information

- Intel Arria 10 SoC Development Kit
- Intel Arria 10 Device Datasheet

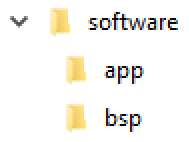
4.5.1.2.2. Software Design Flow

This section provides the software design flow to generate and build the Nios V processor software project. To ensure a streamline build flow, you are encouraged to create similar directory tree in your design project. The following software design flow is based on this directory tree.

To create the software project directory tree, follow these steps:

1. In your design project folder, create a folder named `software`.
2. In the `software` folder, create two folders named `app` and `bsp`.

Figure 40. Software Project Directory Tree



Creating the BSP Project Application

You must edit the BSP editor settings according to the selected Nios V processor boot options.

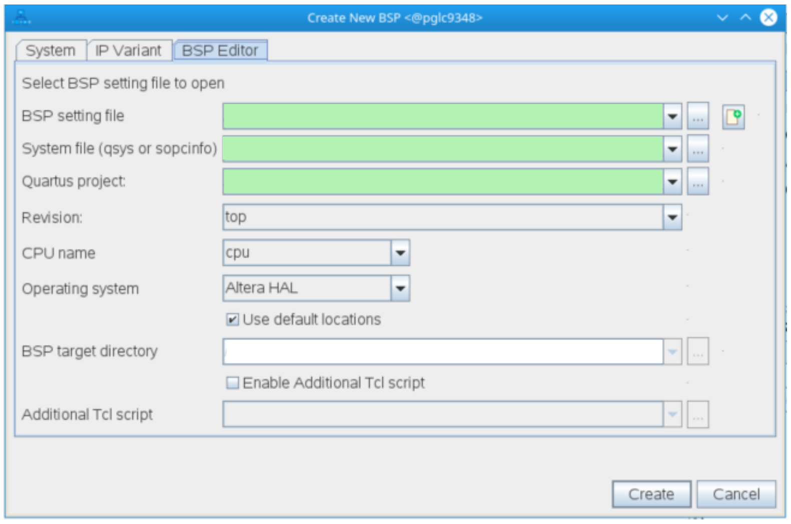
To launch the BSP Editor, perform the following steps:

1. In the Platform Designer window, select **File ➤ New BSP**. The **Create New BSP** windows appears.
2. For **BSP setting file**, navigate to the `software/bsp` folder and name the BSP as `settings.bsp`.

   BSP path: `<project directory>/software/bsp/settings.bsp`
3. For **System file (qsys or sopcinfo)**, select the Nios V processor Platform Designer system (`.qsys`) file.

   Note: For Intel Quartus Prime Standard Edition software, generate the BSP file using SOPCINFO file. Refer to *AN 980: Nios V Processor Intel Quartus Prime Software Support* for more information.
4. For **Quartus project**, select the Quartus Project File.
5. For **Revision**, select the correct revision.
6. For **CPU name**, select the Nios V processor.
7. Select the **Operating system** as **Altera HAL**.
8. Click **Create** to create the BSP file.

Figure 41. Create New BSP window



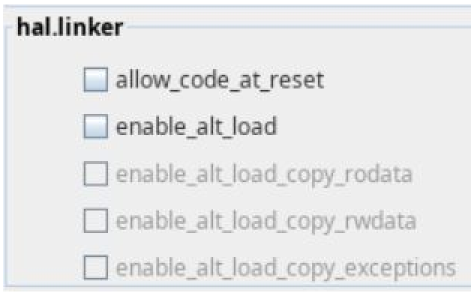
Select BSP setting file to open

BSP setting file	
System file (qsys or sopcinfo)	
Quartus project:	
Revision:	top
CPU name	cpu
Operating system	Altera HAL
	☑ Use default locations
BSP target directory	
	☐ Enable Additional Tcl script
Additional Tcl script	

Configuring BSP Editor and Generating the BSP Project

1. Go to **Main ➤ Settings ➤ Advanced ➤ hal.linker**.
2. Leave all settings unchecked.

Figure 42. hal.linker Settings

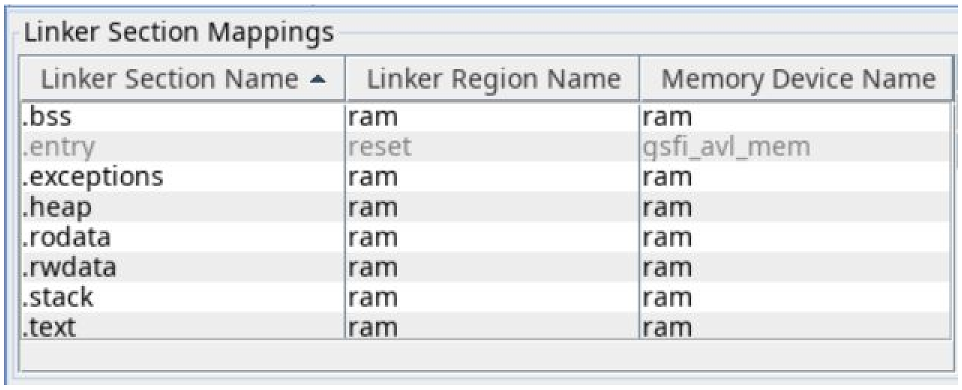


hal.linker

- [ ] allow_code_at_reset
- [ ] enable_alt_load
- [ ] enable_alt_load_copy_rodata
- [ ] enable_alt_load_copy_rwdata
- [ ] enable_alt_load_copy_exceptions

3. Click the **BSP Linker Script** tab in the **BSP Editor**.
4. Set all the Linker Section Name list to the OCRAM or external RAM.

Figure 43. Linker Region Settings



Linker Section Mappings

Linker Section Name ▲	Linker Region Name	Memory Device Name
.bss	ram	ram
.entry	reset	qsfi_avl_mem
.exceptions	ram	ram
.heap	ram	ram
.rodata	ram	ram
.rwdata	ram	ram
.stack	ram	ram
.text	ram	ram

5. Click **Generate BSP**. Make sure the BSP generation is successful.
6. Close the **BSP Editor**.

Generating the Application Project File

1. Navigate to the `software/app` folder and create your Nios V application source code.
2. Launch the Nios V Command Shell.
3. Execute the command below to generate the application `CMakeLists.txt`.

```
niosv-app --app-dir=software/app --bsp-dir=software/bsp \
--srcs=software/app/<Nios V application source code>
```

Building the Application Project

You can choose to build the application project using the RiscFree IDE for Intel FPGAs, Eclipse Embedded CDT or through the command line interface (CLI).

With the CLI, you can build the user project using the following commands:

```
cmake -G "Unix Makefiles" -DCMAKE_BUILD_TYPE=Debug \
-B software/app/debug -S software/app
```

```
make -C software/app/debug
```

The application (`.elf`) file is created in `software/app/debug` folder.

Generating HEX File

You must generate a `.hex` file from your application `.elf` file, so you can create a `.jic` file suitable for programming flash devices.

1. Launch the Nios V Command Shell.
2. For Nios V processor application copied from QSPI flash using boot copier, use the following command line to generate the `.hex` file for your application.
3. Refer to the table *GSFI Bootloader for Nios V Processor Core* in the topic *GSFI Bootloader* for the suitable GSFI bootloader that you can use in the `elf2flash` command.

```
elf2flash
  --boot <Intel Quartus Prime installation directory>/
      niosv/components/bootloader/<GSFI bootloader>
  --input software/app/debug/<Nios V application>.elf \
  --output flash.srec --reset <reset offset + base address of GSFI AVL MEM> \
  --base <base address of GSFI AVL MEM> \
  --end <end address of GSFI AVL MEM>
```

```
riscv32-unknown-elf-objcopy --input-target srec --output-target ihex \
  flash.srec <Nios V application>.hex
```

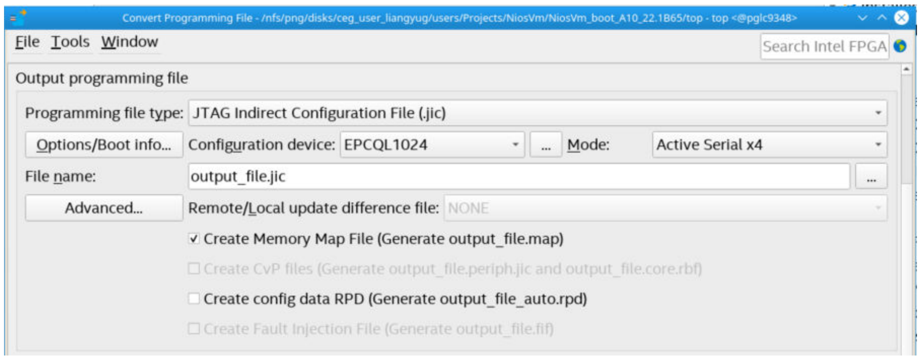
Related Information

- Summary of Nios V Processor Vector Configuration and BSP Settings on page 105
- GSFI Bootloader on page 43
  Refer to the table GSFI Bootloader for Nios V Processor Core for more information about the suitable GSFI bootloader that you can use in the elf2flash command.

4.5.1.2.3. Programming Files Generation

1. In the Intel Quartus Prime software, go to **File ➤ Convert Programming Files**.
2. For **Programming file type**, select **JTAG Indirect Configuration File (.jic)**
3. For **Mode** select **Active Serial x4**.

Figure 44. Convert Programming File Window


Convert Programming File - /nfs/png/disks/ceg_user_liangyug/users/Projects/NiosVm/NiosVm_boot_A10_22.1B65/top - top <@pglc9348>
File Tools Window
Search Intel FPGA
Options/Boot info...
Advanced...

Output programming file

Programming file type: JTAG Indirect Configuration File (.jic)

Configuration device: EPCQL1024

Mode: Active Serial x4

File name: output_file.jic

Remote/Local update difference file: NONE

✓ Create Memory Map File (Generate output_file.map)

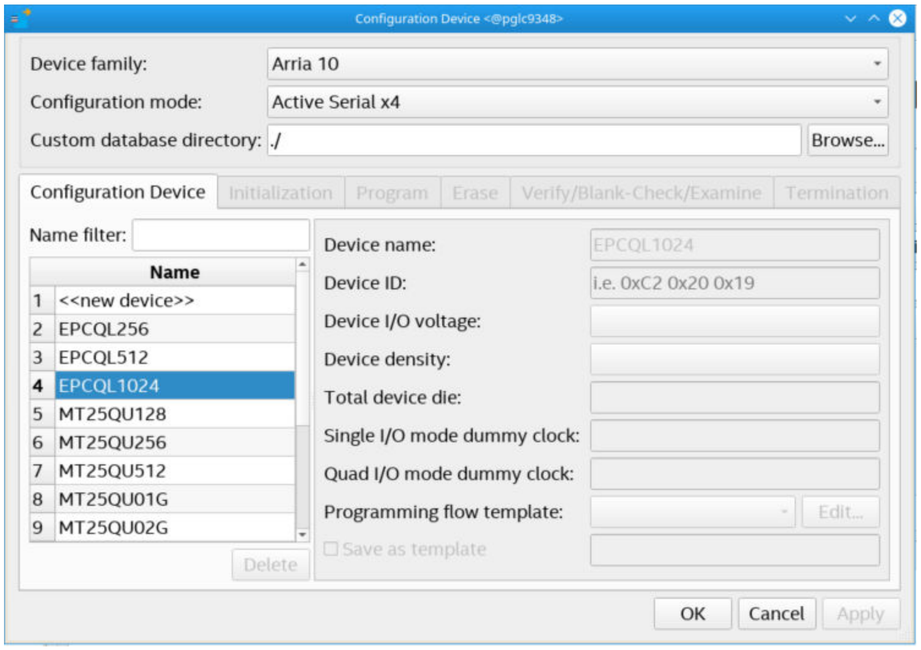
☐ Create CvP files (Generate output_file.periph.jic and output_file.core.rbf)

☐ Create config data RPD (Generate output_file_auto.rpd)

☐ Create Fault Injection File (Generate output_file.fif)

4. Click **...** to enter the **Configuration Device** tab and select the available options. The **Configuration Device** allows for choosing a specific supported device or alternatively an unsupported device.

Figure 45. Configuration Device Window



Device family: Arria 10

Configuration mode: Active Serial x4

Custom database directory: ./

Name filter:

	Name
1	<<new device>>
2	EPCQL256
3	EPCQL512
4	EPCQL1024
5	MT25QU128
6	MT25QU256
7	MT25QU512
8	MT25QU01G
9	MT25QU02G

Device name: EPCQL1024

Device ID: i.e. 0xC2 0x20 0x19

Device I/O voltage:

Device density:

Total device die:

Single I/O mode dummy clock:

Quad I/O mode dummy clock:

Programming flow template:

☐ Save as template

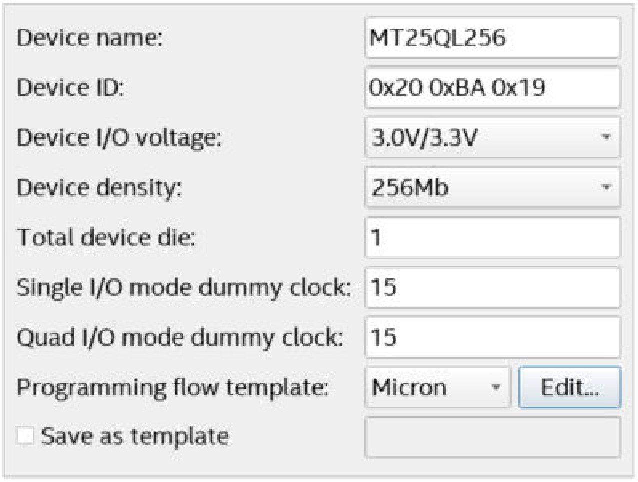
5. If you are using a supported device, make your selection, and click **OK**. Else, proceed with the following steps:

   a. Select **<<new device>>**.

   b. Enter the information about **Device name**, **Device ID**, **Device I/O voltage**, **Device density**, **Total device die**, **Dummy clock (Single I/O or Quad I/O mode)** and **Programming flow template**.

   c. Click **Apply**.

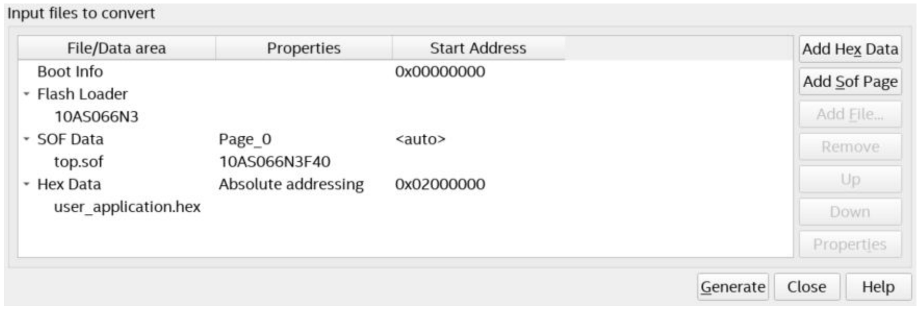
Note: The **Programming flow template** helps you to define a template for flash operation in Initialization, Program, Erase, Verify/Blank-Check/Examine and Termination. If the device is not available for selection, refer to **Modifying Programming Flows** in *Generic Flash Programmer User Guide* to modify the programming flow. Please contact the related vendor for details about memory parameters, such as dummy clock cycles.



Device name:	MT25QL256
Device ID:	0x20 0xBA 0x19
Device I/O voltage:	3.0V/3.3V
Device density:	256Mb
Total device die:	1
Single I/O mode dummy clock:	15
Quad I/O mode dummy clock:	15
Programming flow template:	Micron
Save as template	

6. Under the **Input files to convert** tab,
    a. Select the **Flash Loader** and click **Add Device**.
    b. Add the `.sof` file to the SOF Data by selecting **SOF Data** and click **Add File**.
    c. Click **Add Hex Data** to add Nios V application (`.hex`) file. Select the **Absolute addressing** and **Big-endian** button. Browse to the `.hex` file location. Click **OK**.
7. Click **Generate** to generate the JIC file.

Figure 46. Input files to convert tab



Input files to convert

File/Data area	Properties	Start Address
Boot Info		0x00000000
Flash Loader		
10AS066N3		
SOF Data	Page_0	<auto>
top.sof	10AS066N3F40	
Hex Data	Absolute addressing	0x02000000
user_application.hex		

4.5.1.2.4. QSPI Flash Programming

Intel FPGA Device QSPI Flash Programming

1. Ensure that the Active Serial (AS) pin of the Intel FPGA device is routed to the QSPI flash. This routing allows the flash loader to load into the QSPI flash and configure the board correctly.
2. Ensure the MSEL pin setting on the board is configured for AS programming.
3. Open the Intel Quartus Prime Programmer and make sure JTAG was detected under the **Hardware Setup**.
4. Select **Auto Detect** and choose the FPGA device according to your board.
5. Right-click the selected FPGA device and select **Edit ➤ Change File**. Next, select the generated JIC file.
6. Select the **Program/ Configure** check boxes for FPGA and QSPI devices. Click **Start** to start programming.

Note: Power cycle the device to begin Active Serial configuration scheme, and reset the Nios V processor system upon entering user mode.

4.5.1.3. GSFI Bootloader Example Design

Note: For Intel Quartus Prime Standard Edition software, refer to the topic Intel Quartus Prime Software Support to generate the example design.

You can download the GSFI bootloader example design from the Intel FPGA Design Store. The example design is based on the Intel Arria 10 SoC Development Kit. Using the provided scripts, the hardware and software design are generated, and programmed respectively as SRAM Object Files (`.sof`) and JTAG Indirect Configuration Files (`.jic`) into the device.

Follow the steps below to generate the GSFI bootloader example design:

1. Go to Intel FPGA Design Store.
2. Search for *Arria10 - Bootloader GSFI Design* package.
3. Click on the link at the title.
4. Accept the *Software License Agreement*.
5. Download the package according to the Intel Quartus Prime software version of your host machine.
6. Double-click to run the `top.par` file.
7. `top_project` folder is created by default after running the PAR file.
8. Open the `top_project` and refer to the `readme.txt` for how-to guide.

Table 27. Example Design File Description

File	Description
`hw/`	Contains files necessary to run the hardware project.
`ready_to_test/`	Contains pre-built hardware and software binaries to run the design on the target hardware. For this package, the target hardware is Intel Arria 10 SoC development kit.

continued...

File	Description
`scripts/`	Consists of scripts to build the design.
`sw/`	Contains software application files.
`readme.txt`	Contains description and steps to apply the pre-bulit binaries or rebuild the binaries from scratch.

Figure 47. GSFI Bootloader Example Design

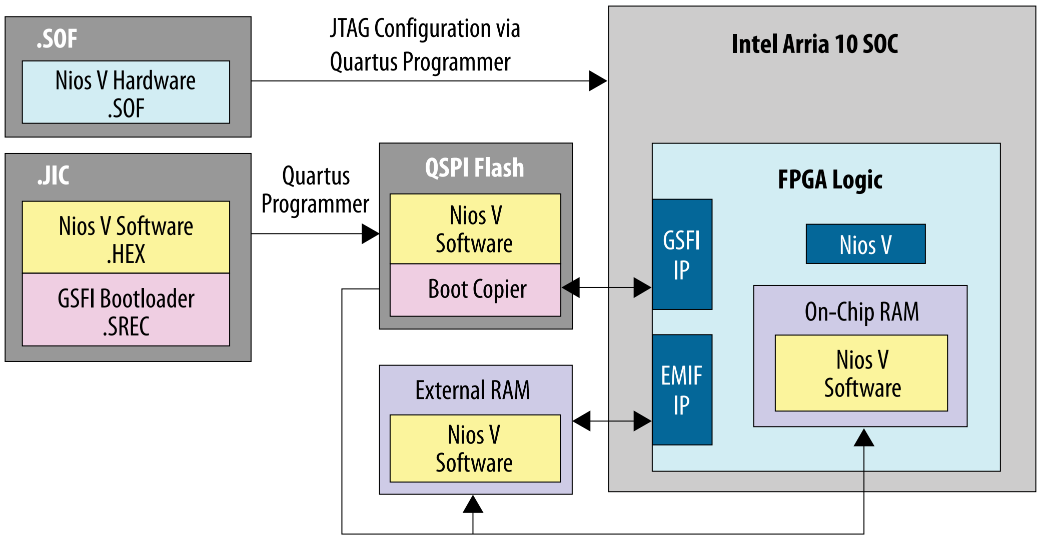
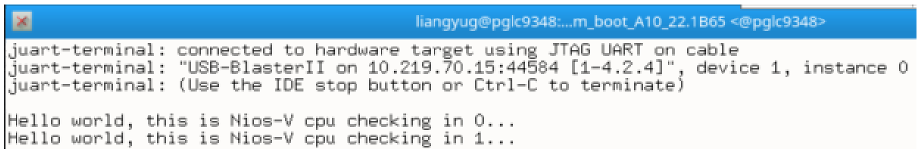


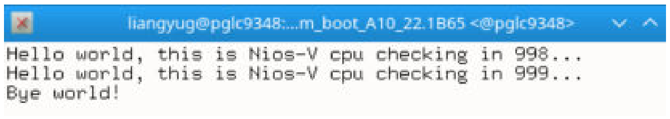
Figure 48. JUART Terminal Output

1. In the beginning, the window displays the following message:



```
liangyug@pglc9348:...m_boot_A10_22.1B65 <@pglc9348>
juart-terminal: connected to hardware target using JTAG UART on cable
juart-terminal: "USB-BlasterII on 10.219.70.15:44584 [1-4.2.4]", device 1, instance 0
juart-terminal: (Use the IDE stop button or Ctrl-C to terminate)

Hello world, this is Nios-V cpu checking in 0...
Hello world, this is Nios-V cpu checking in 1...
```

2. Reaching the end, the window displays the following message:



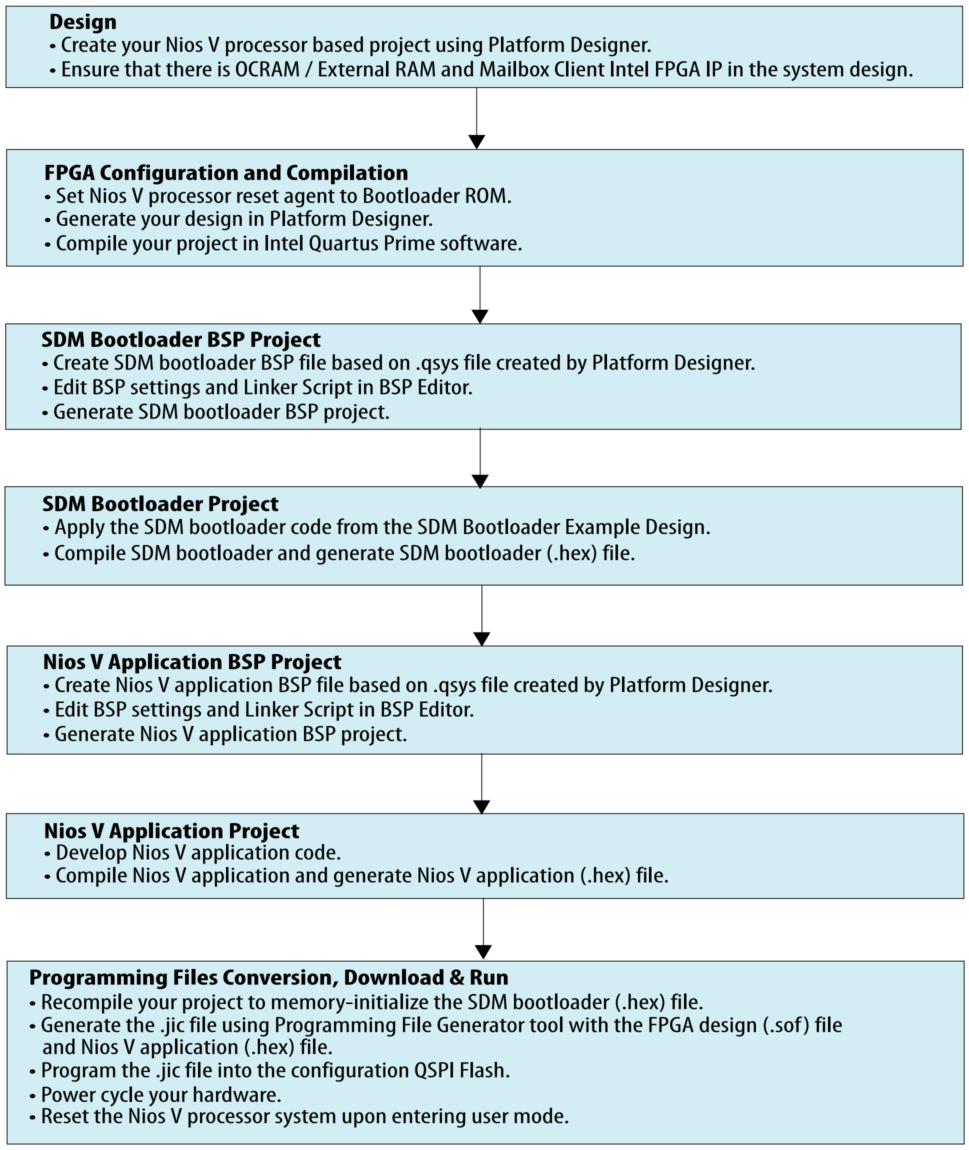
```
liangyug@pglc9348:...m_boot_A10_22.1B65 <@pglc9348>
Hello world, this is Nios-V cpu checking in 998...
Hello world, this is Nios-V cpu checking in 999...
Bye world!
```

Related Information

- Nios V Processor Application Executes-In-Place from Configuration QSPI Flash on page 47
  For more information about booting the Nios V processor-based system from control-block based FPGA devices.
- Nios V Processor Application Copied from Configuration QSPI Flash to RAM Using Boot Copier (GSFI Bootloader) on page 59
  For more information about booting the Nios V processor-based system from control-block based FPGA devices.
- Intel FPGA Design Store

4.5.2. Nios V Processor Design, Configuration and Boot Flow (SDM-based Devices)

Figure 49. Design, Configuration and Booting Flow (SDM-based Devices)



Design
- Create your Nios V processor based project using Platform Designer.
- Ensure that there is OCRAM / External RAM and Mailbox Client Intel FPGA IP in the system design.

FPGA Configuration and Compilation
- Set Nios V processor reset agent to Bootloader ROM.
- Generate your design in Platform Designer.
- Compile your project in Intel Quartus Prime software.

SDM Bootloader BSP Project
- Create SDM bootloader BSP file based on .qsys file created by Platform Designer.
- Edit BSP settings and Linker Script in BSP Editor.
- Generate SDM bootloader BSP project.

SDM Bootloader Project
- Apply the SDM bootloader code from the SDM Bootloader Example Design.
- Compile SDM bootloader and generate SDM bootloader (.hex) file.

Nios V Application BSP Project
- Create Nios V application BSP file based on .qsys file created by Platform Designer.
- Edit BSP settings and Linker Script in BSP Editor.
- Generate Nios V application BSP project.

Nios V Application Project
- Develop Nios V application code.
- Compile Nios V application and generate Nios V application (.hex) file.

Programming Files Conversion, Download & Run
- Recompile your project to memory-initialize the SDM bootloader (.hex) file.
- Generate the .jic file using Programming File Generator tool with the FPGA design (.sof) file and Nios V application (.hex) file.
- Program the .jic file into the configuration QSPI Flash.
- Power cycle your hardware.
- Reset the Nios V processor system upon entering user mode.

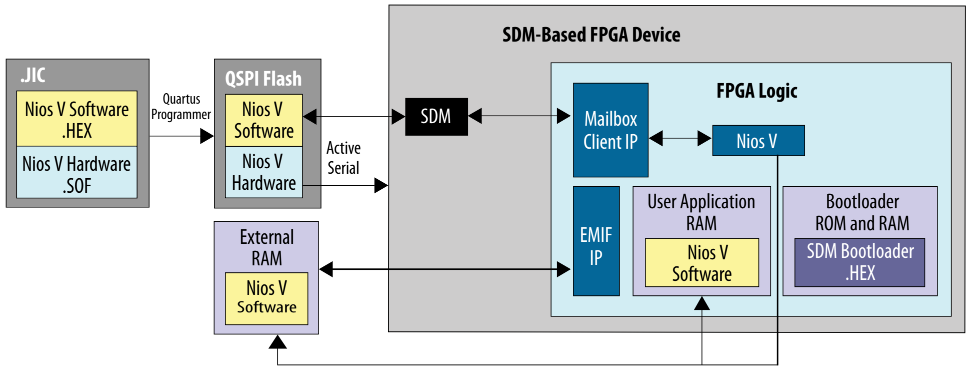
4.5.2.1. Nios V Processor Application Copied from Configuration QSPI Flash to RAM Using Boot Copier (SDM Bootloader)

You can use a boot copier to copy the Nios V application from configuration QSPI flash to RAM when multiple iterations of application software development and high system performance are required. Intel recommends applying the memory organization in Memory Organization for SDM Bootloader to use the SDM Bootloader. The following section covers the description of each memory and the steps required to create them.

The boot copier is memory-initialized in the Bootloader ROM. For this boot option, the Nios V processor starts executing the boot copier upon system reset, which copies the application from the configuration QSPI to the internal or external RAM. Once this completes, the Nios V processor transfers the program control over to the application.

Note: In SDM-based FPGA device, Nios V software booting from configuration QSPI Flash is not supported when the FPGA device is configured using Avalon-ST scheme.

Figure 50. Nios V Processor Application Copied from Configuration QSPI Flash to RAM Using Boot Copier (SDM Bootloader)



Memory Organization for SDM Bootloader

The Nios V processor system should comprise of the following memory spaces to implement the SDM bootloader and Nios V application. The memories are implemented as such:

- Bootloader ROM and RAM (used by SDM bootloader only).
- Nios V processor application RAM (used by user application only).

Table 28. Description of Memory Organization

Memory	Memory Type	Application Use	Linker Section	Notes
Bootloader ROM (Internal ROM)	Read-Only Memory (ROM)	SDM Bootloader	`.text`	Set as the Nios V processor reset agent. Perform memory initialization with SDM bootloader (`.hex`) file.
Bootloader RAM (Internal RAM)	Random Access Memory (RAM)	SDM Bootloader	`.rodata`, `.rwdata`, `.bss`, `.stack`, `.heap`, `.exceptions`	Initialize SDM bootloader using `alt_load()`.
User Application RAM (Internal or External RAM)	Random Access Memory (RAM)	Nios V Application	`.text`, `.rodata`, `.rwdata`, `.bss`, `.stack`, `.heap`, `.exceptions`	The Nios V processor application is loaded into RAM from flash by the SDM Bootloader.

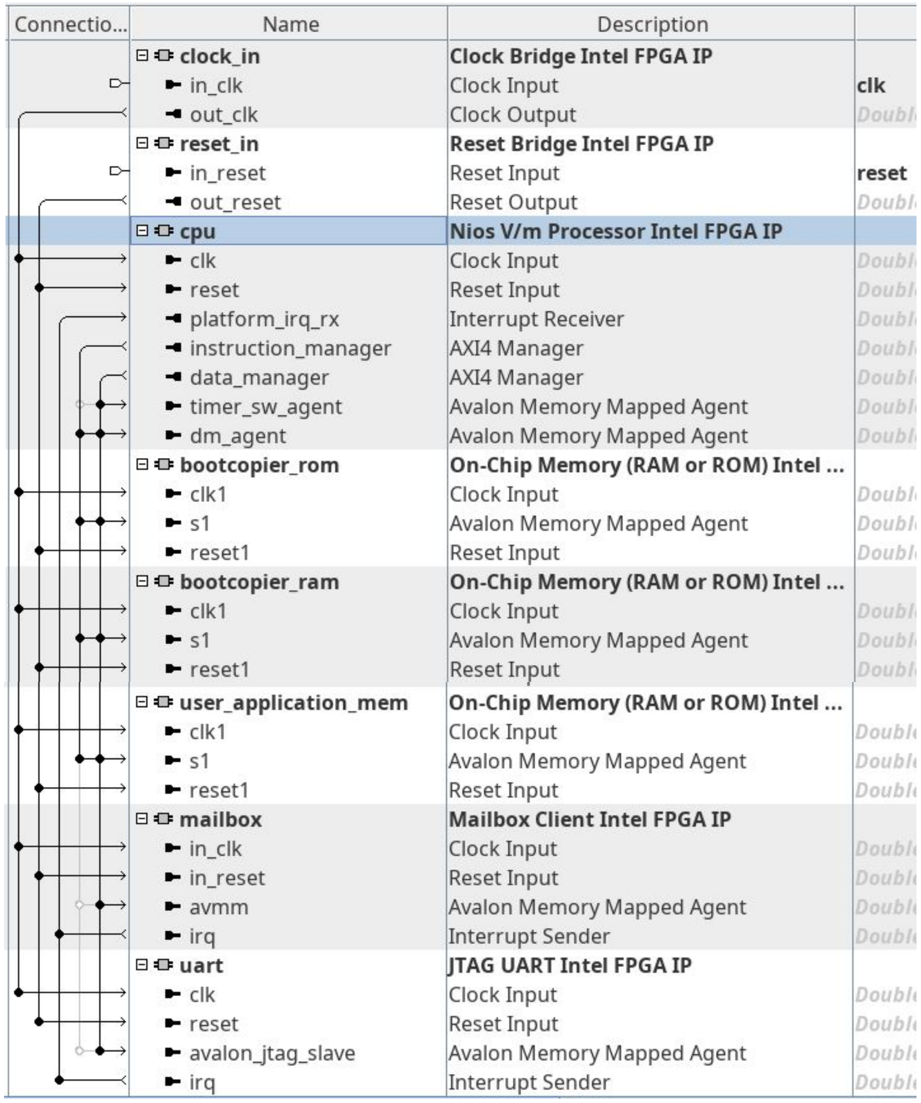
4.5.2.1.1. Hardware Design Flow

The following sections describe a step-by-step method for building a bootable system for a Nios V processor application copied from configuration QSPI flash to RAM using SDM Bootloader. The example below is built using Intel Stratix® 10 SX SoC L-Tile.

IP Component Settings

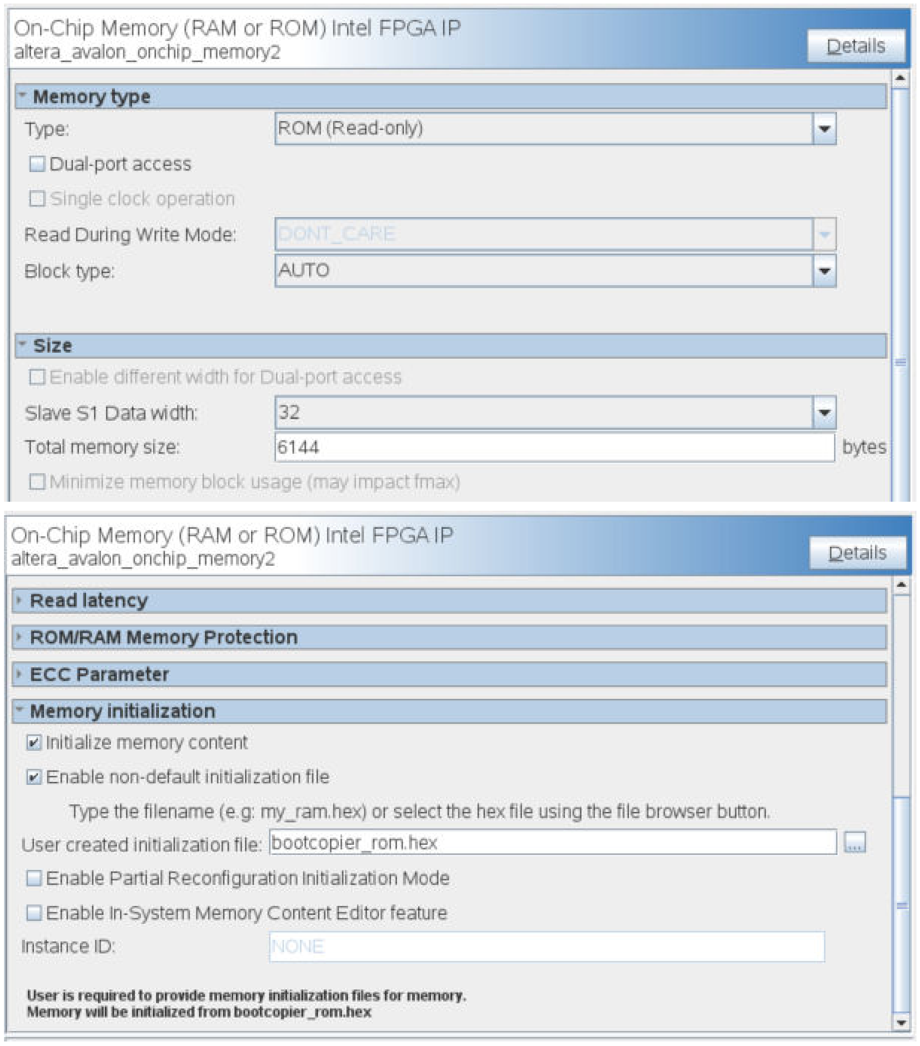
1. Create your Nios V processor project using Intel Quartus Prime and Platform Designer.
2. Add the Mailbox Client Intel FPGA IP into your Platform Designer system.

Figure 51. Connections for Nios V Processor Project



Connectio...	Name	Description	
	clock_in	**Clock Bridge Intel FPGA IP**	
	in_clk	Clock Input	**clk**
	out_clk	Clock Output	Doubl
	reset_in	**Reset Bridge Intel FPGA IP**	
	in_reset	Reset Input	**reset**
	out_reset	Reset Output	Doubl
	cpu	**Nios V/m Processor Intel FPGA IP**	
	clk	Clock Input	Doubl
	reset	Reset Input	Doubl
	platform_irq_rx	Interrupt Receiver	Doubl
	instruction_manager	AXI4 Manager	Doubl
	data_manager	AXI4 Manager	Doubl
	timer_sw_agent	Avalon Memory Mapped Agent	Doubl
	dm_agent	Avalon Memory Mapped Agent	Doubl
	bootcopier_rom	**On-Chip Memory (RAM or ROM) Intel ...**	
	clk1	Clock Input	Doubl
	s1	Avalon Memory Mapped Agent	Doubl
	reset1	Reset Input	Doubl
	bootcopier_ram	**On-Chip Memory (RAM or ROM) Intel ...**	
	clk1	Clock Input	Doubl
	s1	Avalon Memory Mapped Agent	Doubl
	reset1	Reset Input	Doubl
	user_application_mem	**On-Chip Memory (RAM or ROM) Intel ...**	
	clk1	Clock Input	Doubl
	s1	Avalon Memory Mapped Agent	Doubl
	reset1	Reset Input	Doubl
	mailbox	**Mailbox Client Intel FPGA IP**	
	in_clk	Clock Input	Doubl
	in_reset	Reset Input	Doubl
	avmm	Avalon Memory Mapped Agent	Doubl
	irq	Interrupt Sender	Doubl
	uart	**JTAG UART Intel FPGA IP**	
	clk	Clock Input	Doubl
	reset	Reset Input	Doubl
	avalon_jtag_slave	Avalon Memory Mapped Agent	Doubl
	irq	Interrupt Sender	Doubl

Figure 52. On-Chip Memory (RAM or ROM) Intel FPGA IP Parameter Settings


On-Chip Memory (RAM or ROM) Intel FPGA IP
altera_avalon_onchip_memory2

Memory type
Type: ROM (Read-only)
☐ Dual-port access
☐ Single clock operation
Read During Write Mode: DONT_CARE
Block type: AUTO

Size
☐ Enable different width for Dual-port access
Slave S1 Data width: 32
Total memory size: 6144 bytes
☐ Minimize memory block usage (may impact fmax)

On-Chip Memory (RAM or ROM) Intel FPGA IP
altera_avalon_onchip_memory2

Read latency
ROM/RAM Memory Protection
ECC Parameter
Memory initialization
☑ Initialize memory content
☑ Enable non-default initialization file
Type the filename (e.g: my_ram.hex) or select the hex file using the file browser button.
User created initialization file: bootcopier_rom.hex
☐ Enable Partial Reconfiguration Initialization Mode
☐ Enable In-System Memory Content Editor feature
Instance ID: NONE

User is required to provide memory initialization files for memory.
Memory will be initialized from bootcopier_rom.hex

3. Change the **On-Chip Memory (RAM or ROM) Intel FPGA IP Parameter Settings** according to the memory function. Ensure that you have the following memories in the system.

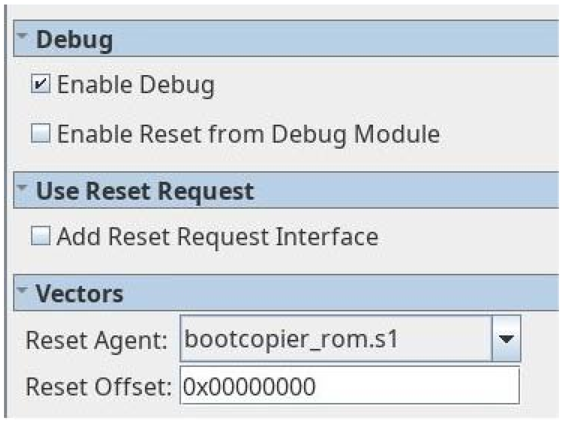
Memory	Memory Type	Total Memory Size	Memory initialization
Bootloader ROM	ROM (Read-only)	6144 bytes or more	Enable the following settings: • **Initialize memory content** • **Enable non-default initialization file** with `bootcopier_rom.hex`
Bootloader RAM	RAM (Writable)	6144 bytes or more	Leave all settings unchecked.
User Application RAM	RAM (Writable)	Depends on your application [4]	Leave all settings unchecked.

[4] Your application size varies according to the usage. Set the memory size according to your design.

Reset Agent Settings for Nios Processor

1. In the Nios V processor parameter editor, set the **Reset Agent** to Bootloader ROM.

Figure 53. Nios V Processor Parameter Editor Settings



Debug

☑ Enable Debug

☐ Enable Reset from Debug Module

Use Reset Request

☐ Add Reset Request Interface

Vectors

Reset Agent: bootcopier_rom.s1

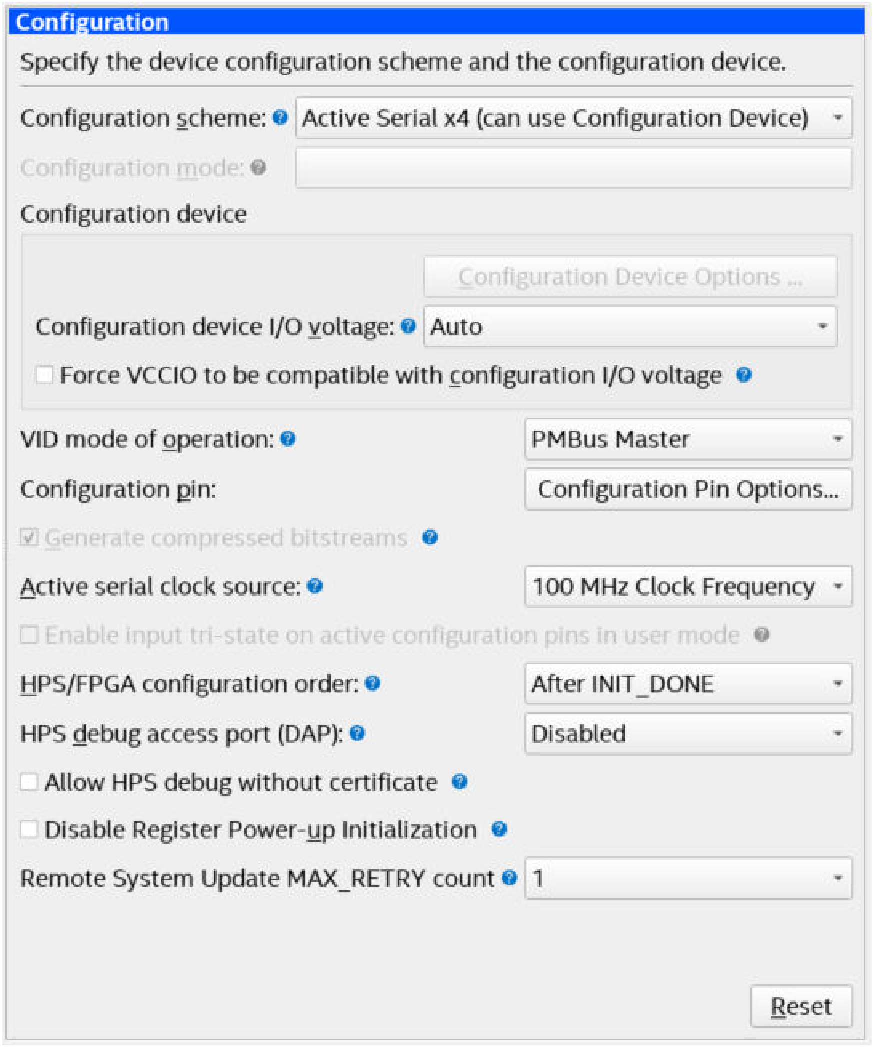
Reset Offset: 0x00000000

2. Click **Generate HDL**, the Generation dialog box appears.
3. Specify output file generation options and then click **Generate**.

Intel Quartus Prime Software Settings

1. In the Intel Quartus Prime software, click **Assignment ➤ Device ➤ Device and Pin Options ➤ Configuration**.
2. Set **Configuration scheme** to **Active Serial x4 (can use Configuration Device)**.
3. Set **VID mode of operation** according to your board design.
4. Set the **Active serial clock source** to **100 MHz Internal Oscillator**.

Figure 54. Device and Pin Options



Configuration

Specify the device configuration scheme and the configuration device.

Configuration scheme: Active Serial x4 (can use Configuration Device)

Configuration mode:

Configuration device

Configuration Device Options ...

Configuration device I/O voltage: Auto

☐ Force VCCIO to be compatible with configuration I/O voltage

VID mode of operation: PMBus Master

Configuration pin: Configuration Pin Options...

☑ Generate compressed bitstreams

Active serial clock source: 100 MHz Clock Frequency

☐ Enable input tri-state on active configuration pins in user mode

HPS/FPGA configuration order: After INIT_DONE

HPS debug access port (DAP): Disabled

☐ Allow HPS debug without certificate

☐ Disable Register Power-up Initialization

Remote System Update MAX_RETRY count 1

Reset

5. Click **OK** to exit the **Device and Pin Options** window.
6. Click **OK** to exit the **Device** window.
7. Click **Start Compilation** to compile your project.

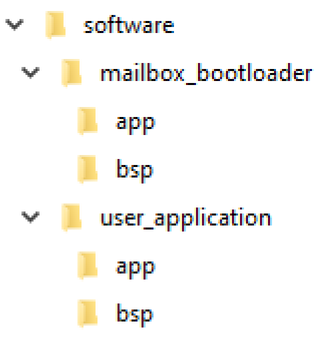
4.5.2.1.2. Software Design Flow

This section provides the software design flow to generate and build the Nios V processor software project for the SDM bootloader and Nios V application. To ensure a streamlined build flow, you are encouraged to create a similar directory tree in your design project. The following software design flow is based on the following directory tree.

To create the software project directory tree, follow these steps:

1. In your design project folder, create a folder named `software`.
2. In the `software` folder, create two folders named `mailbox_bootloader` and `user_application`.
3. In the `mailbox_bootloader` folder, create two folders named app and `bsp`.
4. In the `user_application` folder, create two folders named app and `bsp`.

Figure 55. Software Project Directory Tree



4.5.2.1.3. Software Design Flow (SDM Bootloader Project)

This section provides the design flow to generate and build the SDM Bootloader project.

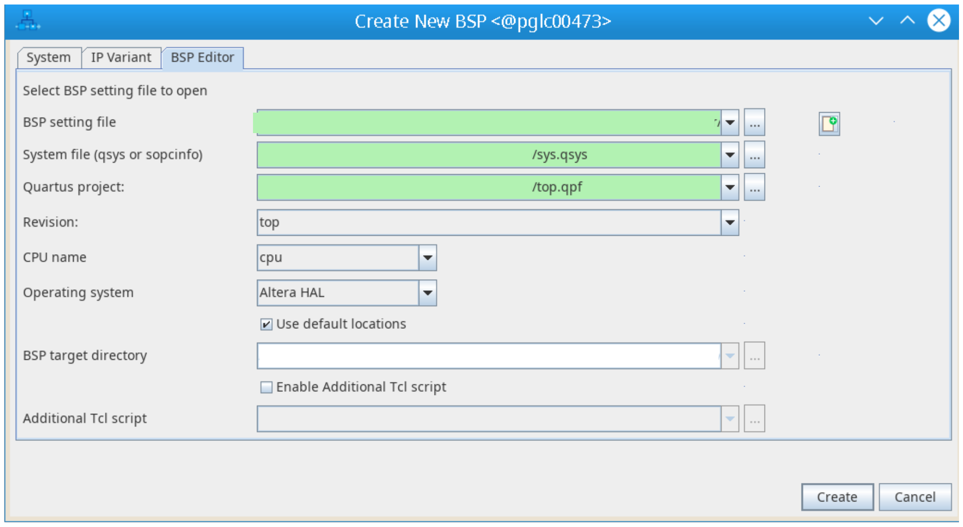
Creating the SDM Bootloader BSP Project

To launch the BSP Editor, follow these steps:

1. In the Platform Designer window, select **File ➤ New BSP**. The **Create New BSP** windows appears.
2. For **BSP setting file**, navigate to the `software/mailbox_bootloader/bsp` folder and name the BSP as `settings.bsp`.

   BSP path: `<project directory>/software/mailbox_bootloader/bsp/settings.bsp`
3. For **System file (qsys or sopcinfo)**, select the Nios V processor Platform Designer system (`.qsys`) file.
4. For **Quartus project**, select the Quartus Project File.
5. For **Revision**, select the correct revision.
6. For **CPU name**, select the Nios V processor.
7. Select the **Operating system** as **Altera HAL**.
8. Click **Create** to create the BSP file.

Figure 56. Create New BSP Window


Create New BSP <@pglc00473>

System | IP Variant | BSP Editor

Select BSP setting file to open

BSP setting file:
System file (qsys or sopcinfo): /sys.qsys
Quartus project: /top.qpf
Revision: top
CPU name: cpu
Operating system: Altera HAL
☑ Use default locations
BSP target directory:
☐ Enable Additional Tcl script
Additional Tcl script:

Configuring BSP Editor and Generating the BSP Project

1. Go to **BSP Editor ➤ Main ➤ Settings**
2. Configure the settings per the following table:

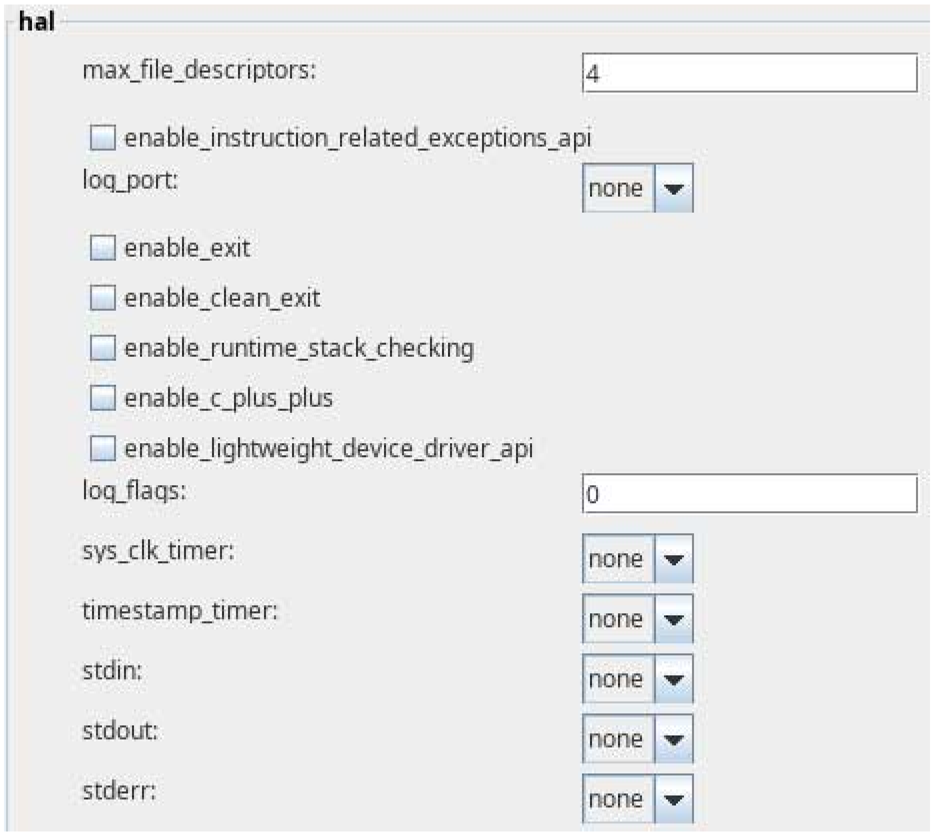
Table 29. Settings for BSP Editor

Settings	Action
`hal.max_file_descriptors`(5)	Input as **4**
`hal.log_port`(5)	Select as **None**
`hal.enable_exit`(5) `hal.enable_clean_exit`(5) `hal.c_plus_plus`(5)	**Unchecked** to disable the feature.
`hal.sys_clk_timer`(5) `hal.timerstamp_timer`(5) `hal.stdin`(5) `hal.stdout`(5) `hal.stderr`(5)	Select as **None**
`hal.linker`	Enable the following settings: • **allow_code_at_reset** • **enable_alt_load** • **enable_alt_load_copy_rodata** • **enable_alt_load_copy_rwdata** • **enable_alt_load_copy_exceptions**
`hal.make.cflags_user_flags`(5)	Input as **-ffunction-sections -fdata-sections**
`hal.make.link_flags`(5)	Input as **-Wl,--gc-sections**
`hal.make.cflags_optimization`(5)	Input as **-Os**

(5) Intel recommends you to use these settings to reduce the SDM bootloader code footprint.

:

Figure 57. hal Settings



hal

max_file_descriptors: 4

☐ enable_instruction_related_exceptions_api

log_port: none

☐ enable_exit

☐ enable_clean_exit

☐ enable_runtime_stack_checking

☐ enable_c_plus_plus

☐ enable_lightweight_device_driver_api

log_flags: 0

sys_clk_timer: none

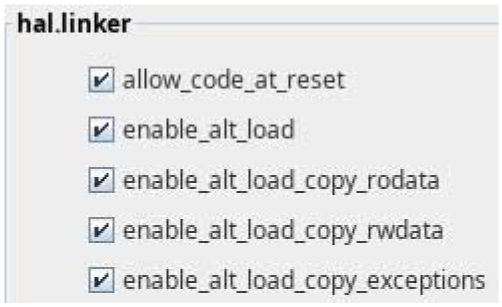
timestamp_timer: none

stdin: none

stdout: none

stderr: none

Figure 58. hal.linker Settings



hal.linker

☑ allow_code_at_reset

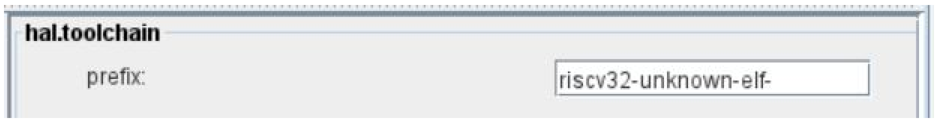
☑ enable_alt_load

☑ enable_alt_load_copy_rodata

☑ enable_alt_load_copy_rwdata

☑ enable_alt_load_copy_exceptions

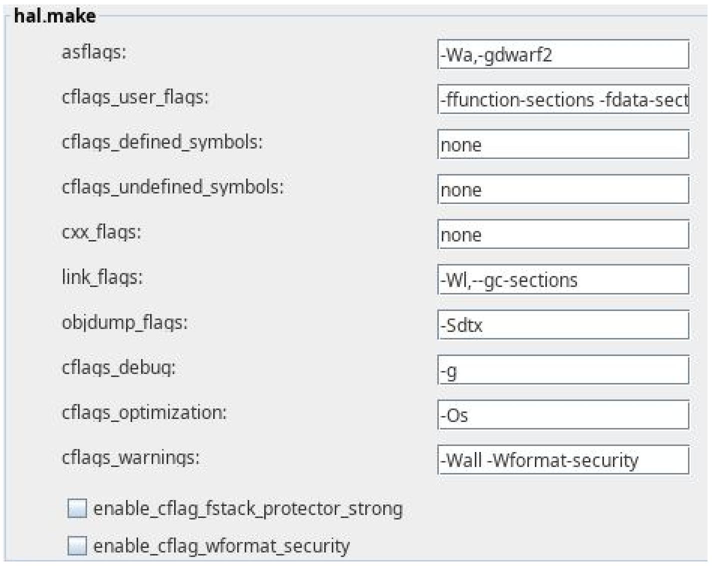
Figure 59. hal.toolchain Settings



hal.toolchain

prefix: riscv32-unknown-elf-

Figure 60. hal.make Settings



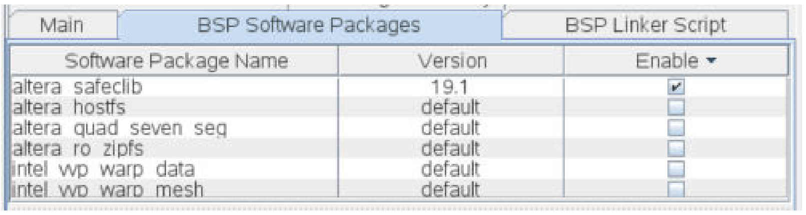
hal.make

asflags:	-Wa,-gdwarf2
cflags_user_flags:	-ffunction-sections -fdata-sect
cflags_defined_symbols:	none
cflags_undefined_symbols:	none
cxx_flags:	none
link_flags:	-Wl,--gc-sections
objdump_flags:	-Sdtx
cflags_debug:	-g
cflags_optimization:	-Os
cflags_warnings:	-Wall -Wformat-security

☐ enable_cflag_fstack_protector_strong
☐ enable_cflag_wformat_security

3. Go to **BSP Software Package** and enable `altera_safeclib`

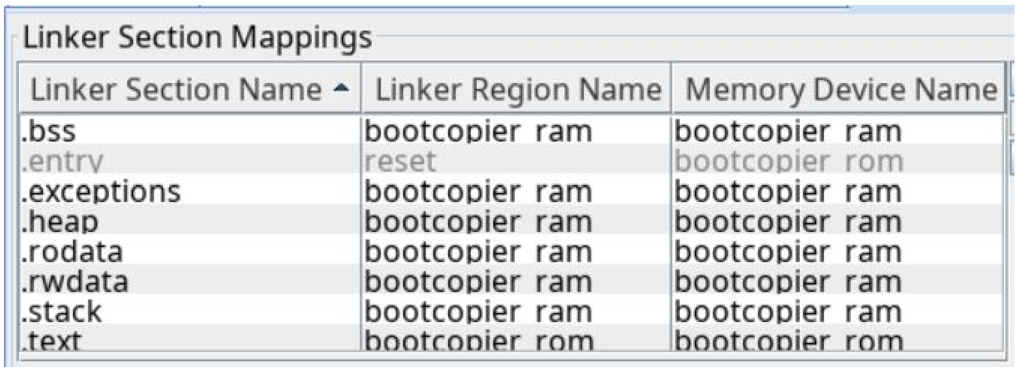
Figure 61. BSP Software Package



Software Package Name	Version	Enable
altera safeclib	19.1	✓
altera hostfs	default	☐
altera quad seven seg	default	☐
altera ro zipfs	default	☐
intel vvp warp data	default	☐
intel vvp warp mesh	default	☐

4. Click the **BSP Linker Script** tab in the BSP Editor.
5. Set the .text item in the **Linker Section Name** to the Bootloader ROM in the **Linker Region Name**. Set the rest of the items in the **Linker Section Name** list to the Bootloader RAM.

Figure 62. Linker Region Settings

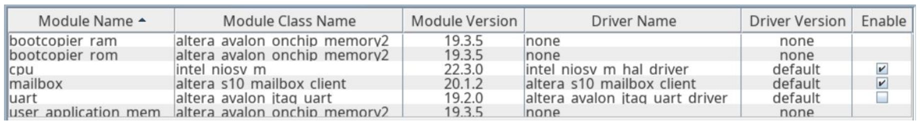


Linker Section Mappings

Linker Section Name	Linker Region Name	Memory Device Name
.bss	bootcopier ram	bootcopier ram
.entry	reset	bootcopier rom
.exceptions	bootcopier ram	bootcopier ram
.heap	bootcopier ram	bootcopier ram
.rodata	bootcopier ram	bootcopier ram
.rwdata	bootcopier ram	bootcopier ram
.stack	bootcopier ram	bootcopier ram
.text	bootcopier rom	bootcopier rom

6. Navigate to the **BSP Driver** tab and disable all drivers (except the Nios V Processor and Mailbox Client Intel FPGA IP).

Figure 63. BSP Driver tab



Module Name ▲	Module Class Name	Module Version	Driver Name	Driver Version	Enable
bootcopier ram	altera avalon onchip memory2	19.3.5	none	none	
bootcopier rom	altera avalon onchip memory2	19.3.5	none	none	
cpu	intel niosv m	22.3.0	intel niosv m hal driver	default	✓
mailbox	altera s10 mailbox client	20.1.2	altera s10 mailbox client	default	✓
uart	altera avalon jtag uart	19.2.0	altera avalon jtag uart driver	default	☐
user application mem	altera avalon onchip memory2	19.3.5	none	none	

7. Click **Generate BSP**. Make sure the BSP generation is successful.
8. Close the BSP Editor.

Create the SDM Bootloader Application Project

1. Download the example design files (Refer to *SDM Bootloader Example Design* section). You do not need to build the example design.
2. Navigate to the `sw/mailbox_bootloader/app` folder in the **SDM Bootloader Example Design** project.
3. Copy the SDM bootloader (`mailbox_bootloader.c`) into the `software/mailbox_bootloader/app` folder in your project.
4. Redefine the PAYLOAD_OFFSET in `mailbox_bootloader.c`.

   Note: The SOF image size influences the PAYLOAD_OFFSET. The PAYLOAD_OFFSET is the start address of the Nios V application HEX file in QSPI flash and must point to a location after the SOF image. You can determine the minimum PAYLOAD_OFFSET by using the configuration bitstream size from the device datasheet.

   For example, the estimated compressed configuration bitstream size for Intel Stratix 10 SX 2800 is 577 Mbits (72.125 MBytes). The actual size can be equal or smaller than this bitstream size. If the SOF image starts at address 0x0, the SOF image should reached until address 0x44C8FFF (0x44C8A48). With that, the minimum PAYLOAD_OFFSET you can select is 0x4500000.
5. Launch the Nios V Command Shell.
6. Execute the command below to generate the SDM bootloader application `CMakeLists.txt`.

```
niosv-app --app-dir=software/mailbox_bootloader/app\
        --bsp-dir=software/mailbox_bootloader/bsp\
        --srcs=software/mailbox_bootloader/app/mailbox_bootloader.c
```

Building the SDM Bootloader Project

You can choose to build the SDM bootloader project using the RiscFree IDE for Intel FPGAs, Eclipse Embedded CDT, or through the command line interface (CLI).

With the CLI , you can build the SDM bootloader using the following commands:

```
cmake -G "Unix Makefiles" -DCMAKE_BUILD_TYPE=Release -B \
        software/mailbox_bootloader/app/release -S \
        software/mailbox_bootloader/app
```

```
make -C software/mailbox_bootloader/app/release
```

The SDM bootloader (`.elf`) file is created in

`software/mailbox_bootloader/app/release` folder.

Generating the HEX File and Initializing the Memory

A HEX file must be generated from the ELF file so that the HEX file can be used for memory initialization.

1. Launch the Nios V Command Shell.
2. For SDM bootloader, use the following command line to convert the ELF to HEX. This command creates the SDM bootloader (`bootcopier_rom.hex`) file.

```
elf2hex software/mailbox_bootloader/app/release/app.elf \
  -o bootcopier_rom.hex \
  -b <base address of Bootloader ROM> \
  -w <data width of Bootloader ROM in bits> \
  -e <end address of Bootloader ROM> \
  -r <data width of Bootloader ROM in bytes>
```

Recompile the hardware design to memory-initialize the `bootcopier_rom.hex` into the Bootloader ROM.

Related Information

- Summary of Nios V Processor Vector Configuration and BSP Settings on page 105
- SDM Bootloader Example Design on page 90

4.5.2.1.4. Software Design Flow (User Application Project)

This section provides the design flow to generate and build the Nios V processor user application.

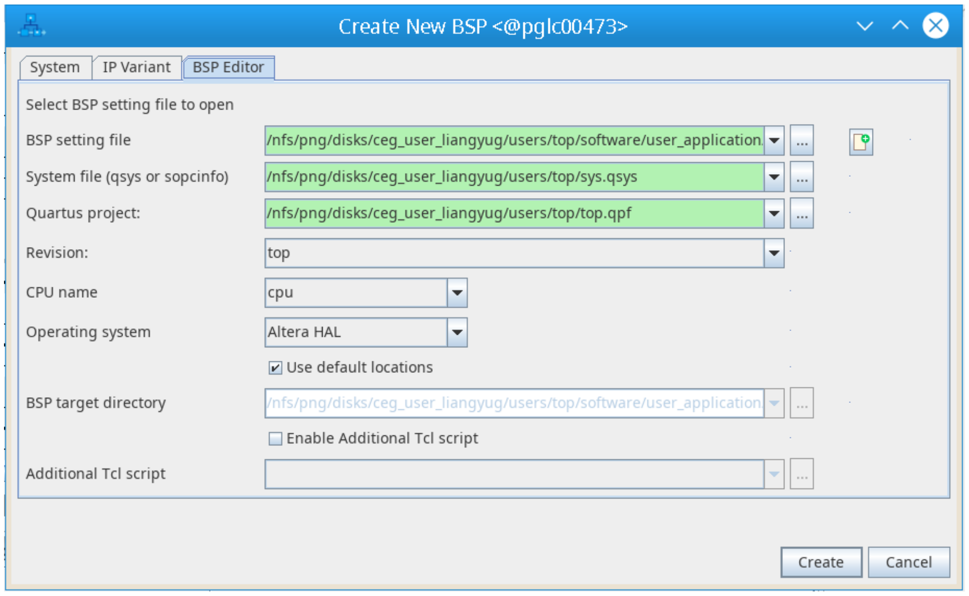
Creating the User Application BSP Project

To launch the BSP Editor, follow these steps:

1. In the Platform Designer window, select **File ➤ New BSP** . The **Create New BSP** windows appears.
2. For **BSP setting file**, navigate to the `software/user_application/bsp` folder and name the BSP as `settings.bsp`.

   BSP path: `<project directory>/software/user_application/bsp/settings.bsp`
3. For **System file (qsys or sopcinfo)**, select the Nios V processor Platform Designer system (`.qsys`).
4. For **Quartus project**, select the Quartus Project File.
5. For **Revision**, select the correct revision.
6. For **CPU name**, select the Nios V processor.
7. Select the **Operating system** as **Altera HAL**.
8. Click **Create** to create the BSP file.

Figure 64. Create New BSP Window



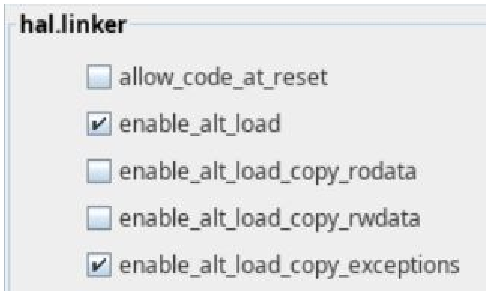
Select BSP setting file to open

BSP setting file	/nfs/png/disks/ceg_user_liangyug/users/top/software/user_application
System file (qsys or sopcinfo)	/nfs/png/disks/ceg_user_liangyug/users/top/sys.qsys
Quartus project:	/nfs/png/disks/ceg_user_liangyug/users/top/top.qpf
Revision:	top
CPU name	cpu
Operating system	Altera HAL
	☑ Use default locations
BSP target directory	/nfs/png/disks/ceg_user_liangyug/users/top/software/user_application
	☐ Enable Additional Tcl script
Additional Tcl script	

Configure BSP Editor and Generate the BSP Project

1. Go to **Main ➤ Settings ➤ Settings ➤ Advanced ➤ hal.linker**.
2. Enable the following settings:
   a. **enable_alt_load**
   b. **enable_alt_load_copy_exceptions**

Figure 65. hal.linker Settings

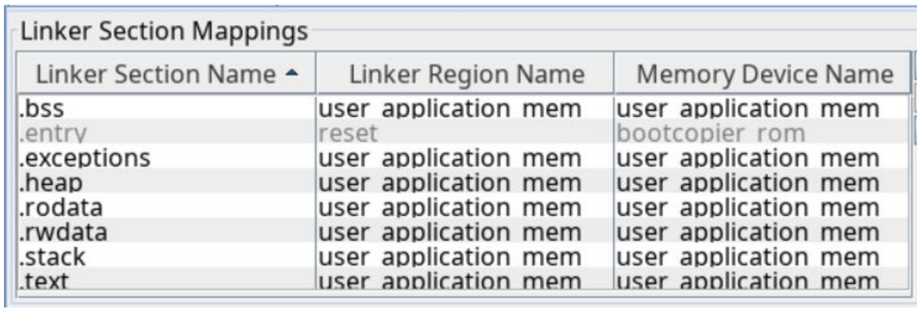


hal.linker

- ☐ allow_code_at_reset
- ☑ enable_alt_load
- ☐ enable_alt_load_copy_rodata
- ☐ enable_alt_load_copy_rwdata
- ☑ enable_alt_load_copy_exceptions

3. Click the **BSP Linker Script** tab in the **BSP Editor**.

Figure 66. Linker Region Settings



Linker Section Mappings

Linker Section Name	Linker Region Name	Memory Device Name
.bss	user_application_mem	user_application_mem
.entry	reset	bootcopier_rom
.exceptions	user_application_mem	user_application_mem
.heap	user_application_mem	user_application_mem
.rodata	user_application_mem	user_application_mem
.rwdata	user_application_mem	user_application_mem
.stack	user_application_mem	user_application_mem
.text	user_application_mem	user_application_mem

4. Set all the **Linker Section Name** list to the User Application RAM.
5. Click **Generate BSP**. Make sure the BSP generation is successful.
6. Close the **BSP Editor**.

Creating the User Application Project

1. Navigate to the `software/user_application/app` folder and create your user application source code.
2. Launch the Nios V Command Shell.
3. Execute the command below to generate the user application `CMakeLists.txt`.

```
niosv-app --app-dir=software/user_application/app \
  --bsp-dir=software/user_application/bsp \
  --srcs=software/user_application/app/<user application>
```

Building the Application Project

You can choose to build the application project using the RiscFree IDE for Intel FPGAs, Eclipse Embedded CDT, or through the command line interface (CLI).

With the CLI, you can build the user application using the following commands:

```
cmake -G "Unix Makefiles" -DCMAKE_BUILD_TYPE=Debug \
 -B software/user_application/app/debug -S software/user_application/app
```

```
make -C software/user_application/app/debug
```

The user application (`.elf`) file is created in `software/user_application/app/debug` folder.

Generating the HEX File

You must generate a `.hex` file from your application `.elf` file, so you can create a `.jic` file suitable for programming flash devices.

1. Launch the Nios V Command Shell.
2. For Nios V processor application copied from QSPI flash using SDM bootloader, use the following commands to convert the ELF to HEX for your application. These commands creates the user application (`.hex`) file.

```
elf2flash --input software/user_application/app/debug/<user application>.elf \
 --output flash.srec –epcs --offset 0x0
```

```
riscv32-unknown-elf-objcopy --input-target srec \
--output-target ihex flash.srec \
 <user application>.hex
```

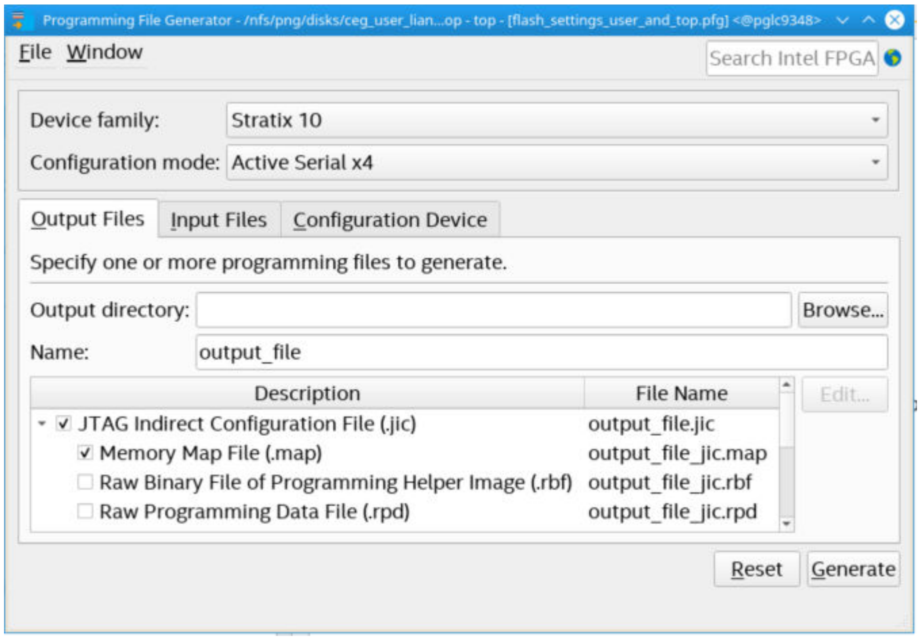
Related Information

Summary of Nios V Processor Vector Configuration and BSP Settings on page 105

4.5.2.1.5. Programming Files Generation

1. Go to **File ➤ Programming File Generator**.
2. Select the **Configuration mode** to be **Active Serial x4**.
3. In the **Output Files** tab, select **JTAG Indirect Configuration File (.jic)**.

Figure 67. Programming File Generator (Output Files)


Programming File Generator - /nfs/png/disks/ceg_user_lian...op - top - [flash_settings_user_and_top.pfg] <@pglc9348>
File
Window
Search Intel FPGA
Output Files
Input Files
Configuration Device
Browse...
Edit...
Reset
Generate

Device family: Stratix 10
Configuration mode: Active Serial x4

Specify one or more programming files to generate.

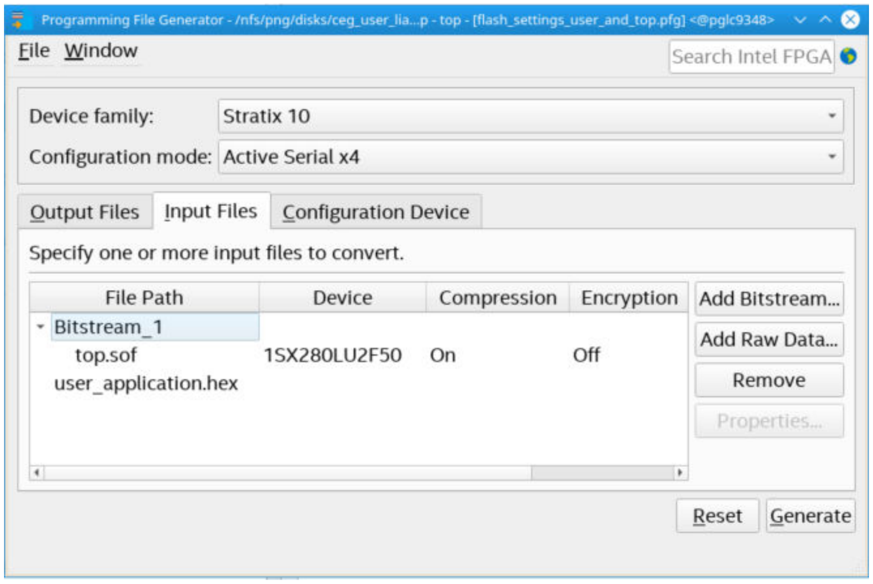
Output directory:
Name: output_file

Description	File Name
✓ JTAG Indirect Configuration File (.jic)	output_file.jic
✓ Memory Map File (.map)	output_file_jic.map
☐ Raw Binary File of Programming Helper Image (.rbf)	output_file_jic.rbf
☐ Raw Programming Data File (.rpd)	output_file_jic.rpd

4. In the **Input Files** tab, perform the following steps:

a. Add the SOF file by clicking **Add Bitstream**.
b. Add the user application (`.hex`) file by clicking **Add Raw Data**.
c. Select the HEX file and click **Properties**.
d. Select **Bit Swap : On**.

Figure 68. Programming File Generator (Input Files)



Programming File Generator - /nfs/png/disks/ceg_user_lia...p - top - [flash_settings_user_and_top.pfg] <@pglc9348>

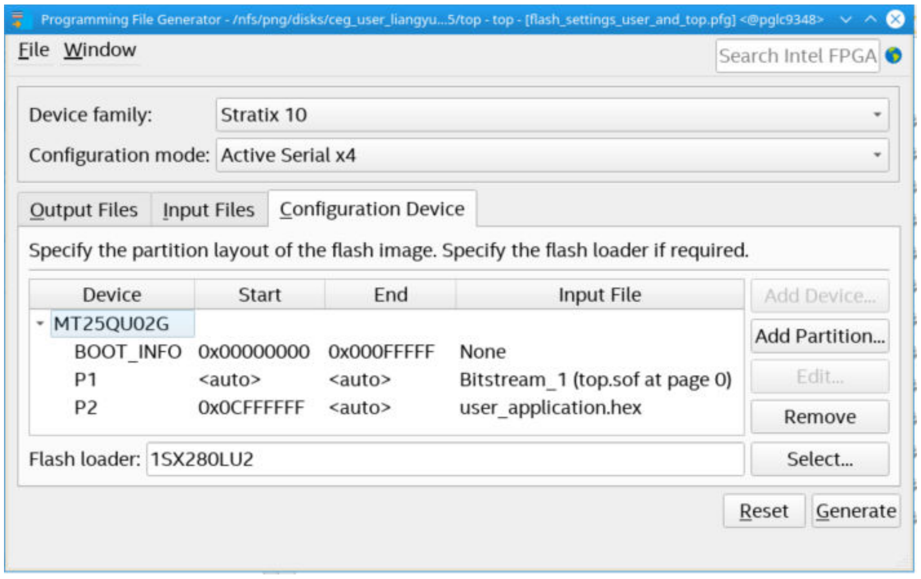
Device family: Stratix 10
Configuration mode: Active Serial x4

Specify one or more input files to convert.

File Path	Device	Compression	Encryption
Bitstream_1			
top.sof	1SX280LU2F50	On	Off
user_application.hex			

5. In the **Configuration Device** tab,
   a. Add the flash device by clicking **Add Device**.
      i. If you are using a supported device, you may make your selection, and click **OK**. Else, proceed to **Apply the Configuration Device window**.
   b. Add the SOF file by selecting the flash device and click **Add Partition**.
   c. Add the user application (`.hex`) file by selecting the flash device and click **Add Partition**. Select the **Address Mode** to **Start** and set the **Start address** to the value set for PAYLOAD_OFFSET in `mailbox_bootcopier.c`.
   d. Select the **Flash loader** according to the Intel FPGA device.

Figure 69. Programming File Generator (Configuration Device)



Device family: Stratix 10

Configuration mode: Active Serial x4

Specify the partition layout of the flash image. Specify the flash loader if required.

Device	Start	End	Input File
MT25QU02G			
BOOT_INFO	0x00000000	0x000FFFFF	None
P1	<auto>	<auto>	Bitstream_1 (top.sof at page 0)
P2	0x0CFFFFFF	<auto>	user_application.hex

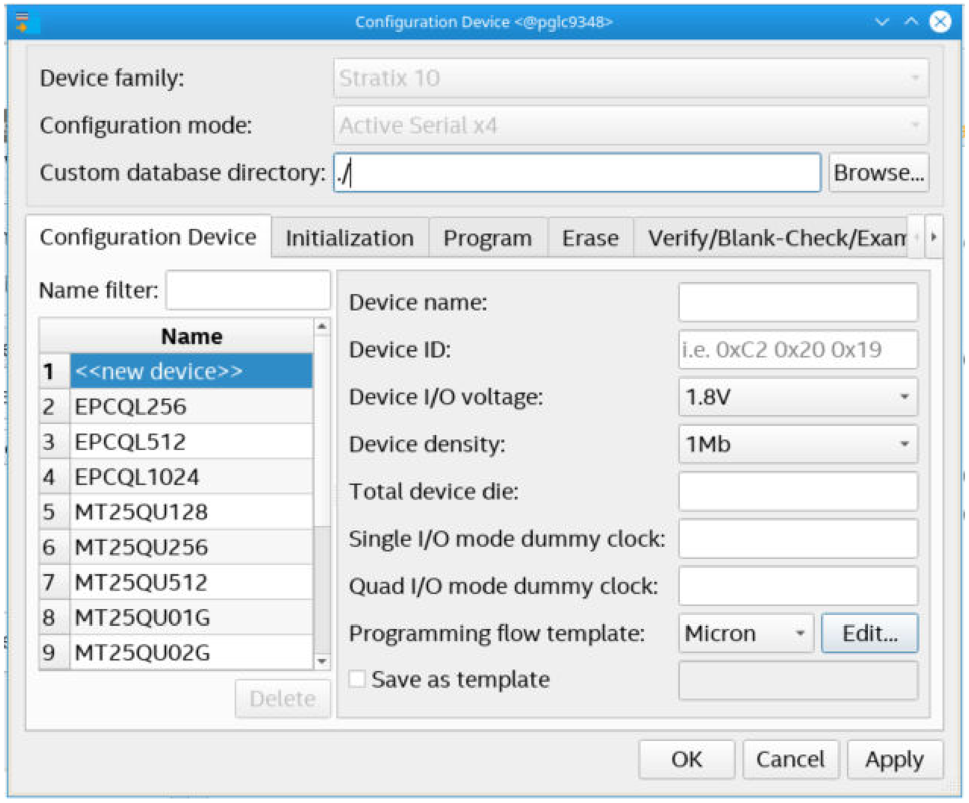
Flash loader: 1SX280LU2

6. Click **Generate** to generate the JIC file.

Applying the Configuration Device Window

The **Configuration Device** allows for choosing a specific supported device or alternatively an unsupported device.

Figure 70. Configuration Device Window



Configuration Device <@pglc9348>

Device family: Stratix 10
Configuration mode: Active Serial x4
Custom database directory: ./

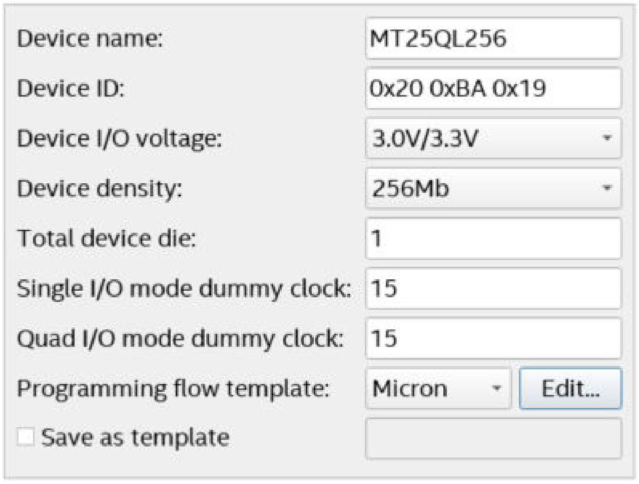
Name filter:

	Name
1	<<new device>>
2	EPCQL256
3	EPCQL512
4	EPCQL1024
5	MT25QU128
6	MT25QU256
7	MT25QU512
8	MT25QU01G
9	MT25QU02G

Device name:
Device ID: i.e. 0xC2 0x20 0x19
Device I/O voltage: 1.8V
Device density: 1Mb
Total device die:
Single I/O mode dummy clock:
Quad I/O mode dummy clock:
Programming flow template: Micron
Save as template

1. If you are using a supported device, make your selection, and click **OK**. Else, proceed with the following steps:
   a. Select **<<new device>>**.
   b. Enter the information about **Device name**, **Device ID**, **Device I/O voltage**, **Device density**, **Total device die**, **Dummy clock (Single I/O or Quad I/O mode)** and **Programming flow template**.
   c. Click **Apply**.

   Note: The **Programming flow template** helps you to define a template for flash operation in Initialization, Program, Erase, Verify/Blank-Check/ Examine and Termination. If the device is not available for selection, refer to **Modifying Programming Flows** in *Generic Flash Programmer User Guide* to modify the programming flow.



Device name:	MT25QL256
Device ID:	0x20 0xBA 0x19
Device I/O voltage:	3.0V/3.3V
Device density:	256Mb
Total device die:	1
Single I/O mode dummy clock:	15
Quad I/O mode dummy clock:	15
Programming flow template:	Micron
Save as template	

4.5.2.1.6. QSPI Flash Programming SDM

Intel FPGA Device QSPI Flash Programming

1. Ensure that the Intel FPGA device's Active Serial (AS) pin is routed to the QSPI flash. This routing allows the flash loader to load into the QSPI flash and configure the board correctly.
2. Ensure the MSEL pin setting on the board is configured for AS programming.
3. Open the Intel Quartus Prime Programmer and make sure JTAG is detected under the **Hardware Setup**.
4. Select **Auto Detect** and choose the FPGA device according to your board.
5. Right-click the selected Intel FPGA device and select **Edit ➤ Change File**. Next, select the generated JIC file.
6. Select the **Program/ Configure** check boxes for FPGA and QSPI devices.
7. Click **Start** to start programming.

4.5.2.2. SDM Bootloader Example Design

You can download the SDM bootloader example design from the *Intel FPGA Design Store*. The example design is based on the Intel Stratix 10 SX SoC L-Tile development kit.

Using the provided scripts, the hardware and software design are generated and programmed respectively as SRAM Object Files (`.sof`) and JTAG Indirect Configuration Files (`.jic`) into the device.

Follow the steps below to generate the SDM bootloader example design:

1. Go to Intel FPGA Design Store.
2. Search for *Stratix10 - Bootloader SDM Design* package.
3. Click on the link at the title.

4. Accept the *Software License Agreement*.
5. Download the package according to the Intel Quartus Prime software version of your host machine.
6. Refer to the `readme.txt` for how-to guide.

Table 30. Example Design File Description

File	Description
`hw/`	Contains files necessary to run the hardware project.
`ready_to_test/`	Contains pre-built hardware and software binaries to run the design on the target hardware. For this package, the target hardware is Intel Stratix 10 SX 10 SoC L-tile development kit.
`scripts/`	Consists of scripts to build the design.
`sw/`	Contains software application files.
`readme.txt`	Contains description and steps to apply the pre-bulit binaries or rebuild the binaries from scratch.

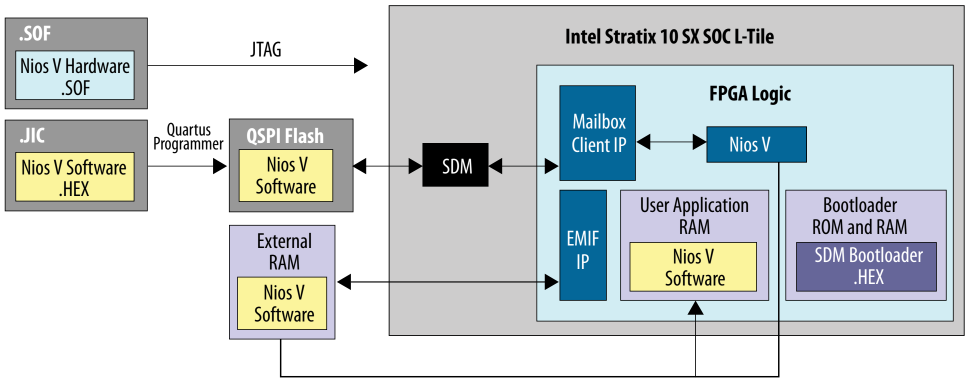
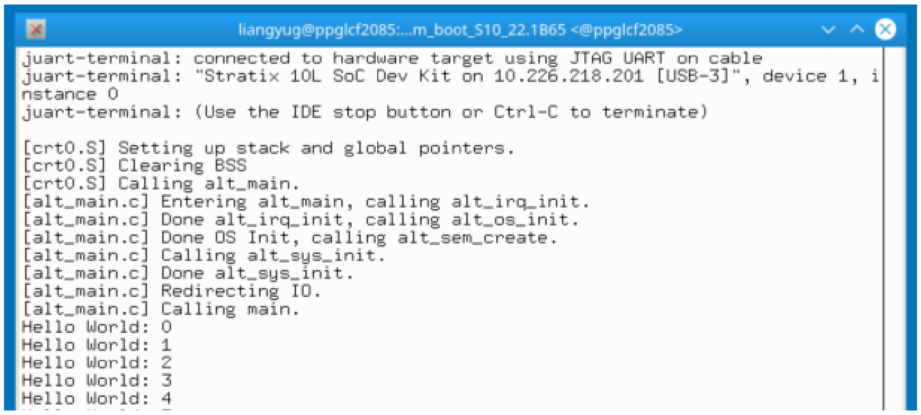


Figure 71. SDM Bootloader Example Design

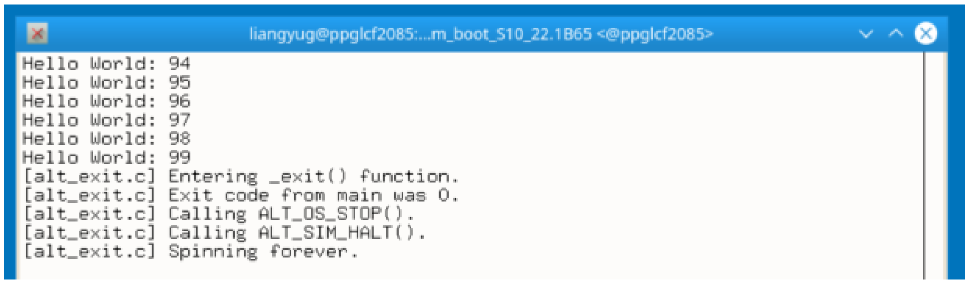
Figure 72. JUART Terminal Output

1. In the beginning, the window displays the following message



```
juart-terminal: connected to hardware target using JTAG UART on cable
juart-terminal: "Stratix 10L SoC Dev Kit on 10.226.218.201 [USB-3]", device 1, i
nstance 0
juart-terminal: (Use the IDE stop button or Ctrl-C to terminate)

[crt0.S] Setting up stack and global pointers.
[crt0.S] Clearing BSS
[crt0.S] Calling alt_main.
[alt_main.c] Entering alt_main, calling alt_irq_init.
[alt_main.c] Done alt_irq_init, calling alt_os_init.
[alt_main.c] Done OS Init, calling alt_sem_create.
[alt_main.c] Calling alt_sys_init.
[alt_main.c] Done alt_sys_init.
[alt_main.c] Redirecting IO.
[alt_main.c] Calling main.
Hello World: 0
Hello World: 1
Hello World: 2
Hello World: 3
Hello World: 4
```

2. Reaching the end, the window displays the following message:



```
Hello World: 94
Hello World: 95
Hello World: 96
Hello World: 97
Hello World: 98
Hello World: 99
[alt_exit.c] Entering _exit() function.
[alt_exit.c] Exit code from main was 0.
[alt_exit.c] Calling ALT_OS_STOP().
[alt_exit.c] Calling ALT_SIM_HALT().
[alt_exit.c] Spinning forever.
```

Related Information

- Software Design Flow (SDM Bootloader Project) on page 78
- Intel FPGA Design Store

4.6. Nios V Processor Booting from On-Chip Memory (OCRAM)

This section describes the Nios V processor booting and executing software from On-Chip Memory (OCRAM) available in all supported Intel FPGA devices.

4.6.1. Nios V Processor Application Executes in-place from OCRAM

The on-chip memory is initialized during FPGA configuration with data from a Nios V processor application image. This data is built into the FPGA configuration bitstream. This process eliminates the need for a boot copier, as the Nios V processor application is already in place at system reset.

Figure 73. Nios V Processor Application Executes In-Place from OCRAM when FPGA Device Configured from QSPI Flash

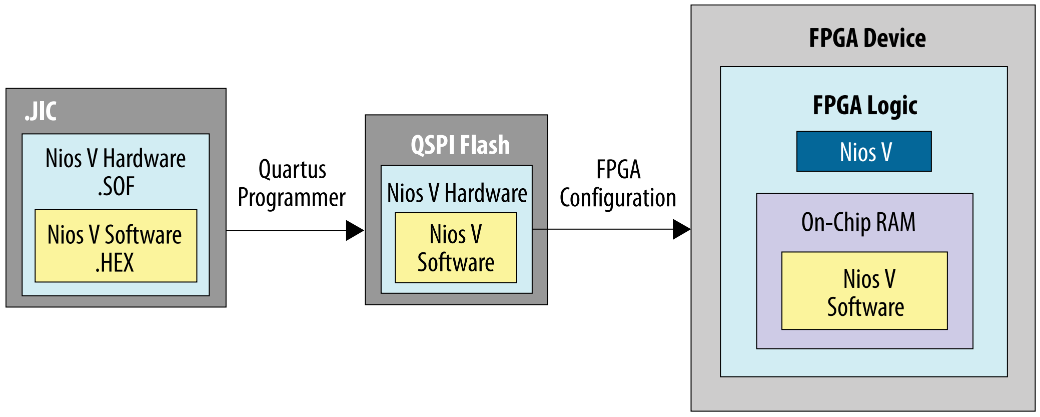
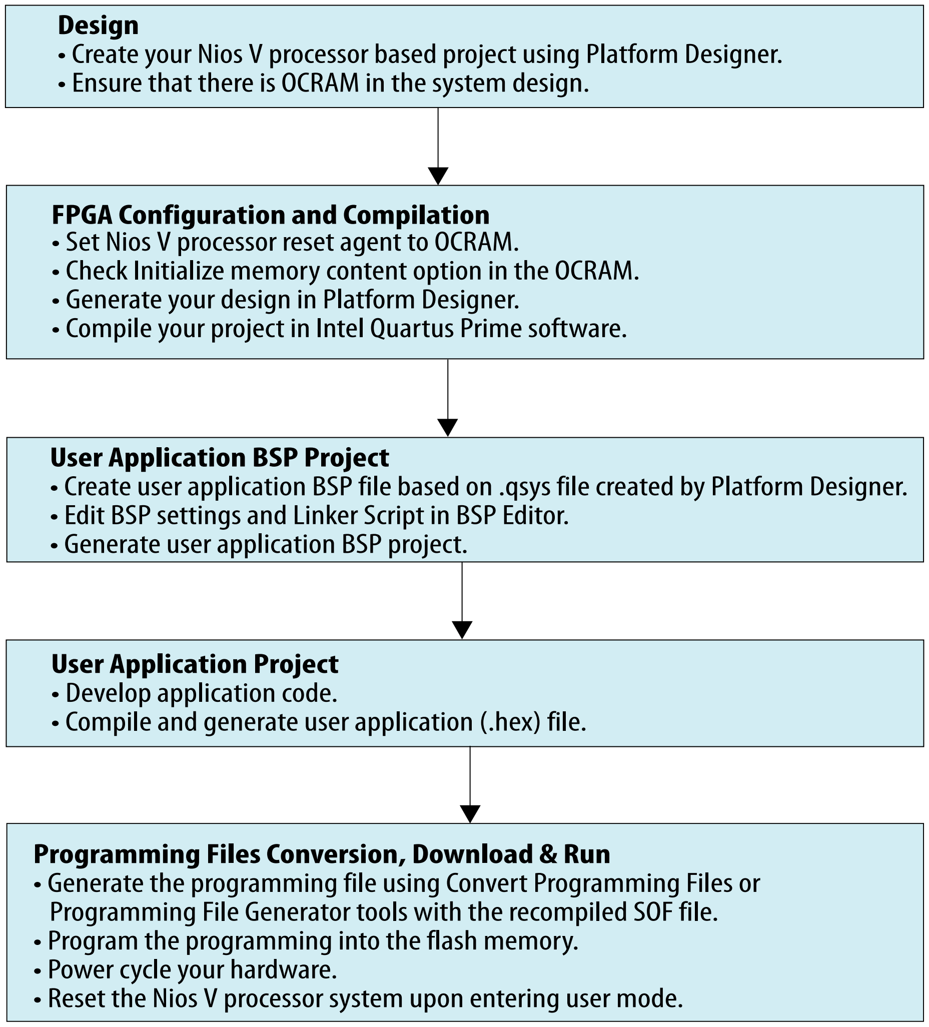


Figure 74. Design, Configuration and Booting Flow



Design
- Create your Nios V processor based project using Platform Designer.
- Ensure that there is OCRAM in the system design.

FPGA Configuration and Compilation
- Set Nios V processor reset agent to OCRAM.
- Check Initialize memory content option in the OCRAM.
- Generate your design in Platform Designer.
- Compile your project in Intel Quartus Prime software.

User Application BSP Project
- Create user application BSP file based on .qsys file created by Platform Designer.
- Edit BSP settings and Linker Script in BSP Editor.
- Generate user application BSP project.

User Application Project
- Develop application code.
- Compile and generate user application (.hex) file.

Programming Files Conversion, Download & Run
- Generate the programming file using Convert Programming Files or Programming File Generator tools with the recompiled SOF file.
- Program the programming into the flash memory.
- Power cycle your hardware.
- Reset the Nios V processor system upon entering user mode.

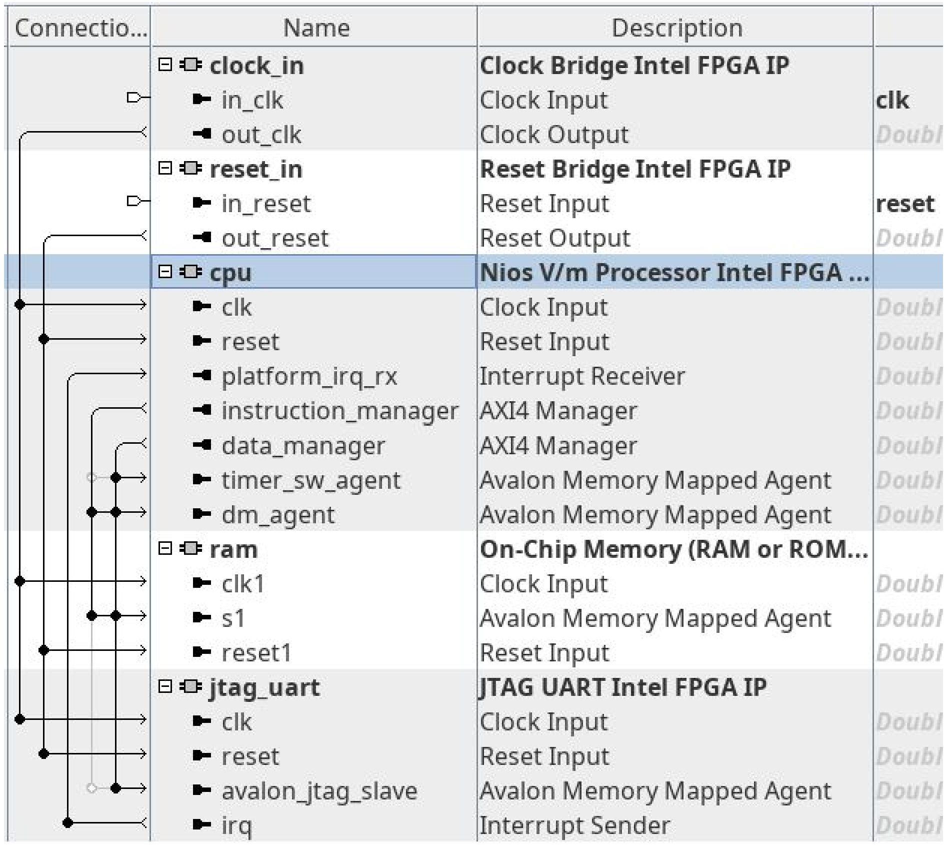
4.6.1.1. Hardware Design Flow

The following sections describe a step-by-step method for building a bootable system for a Nios V processor application from OCRAM. The example below is built using Intel Arria 10 SoC development kit.

IP Component Settings

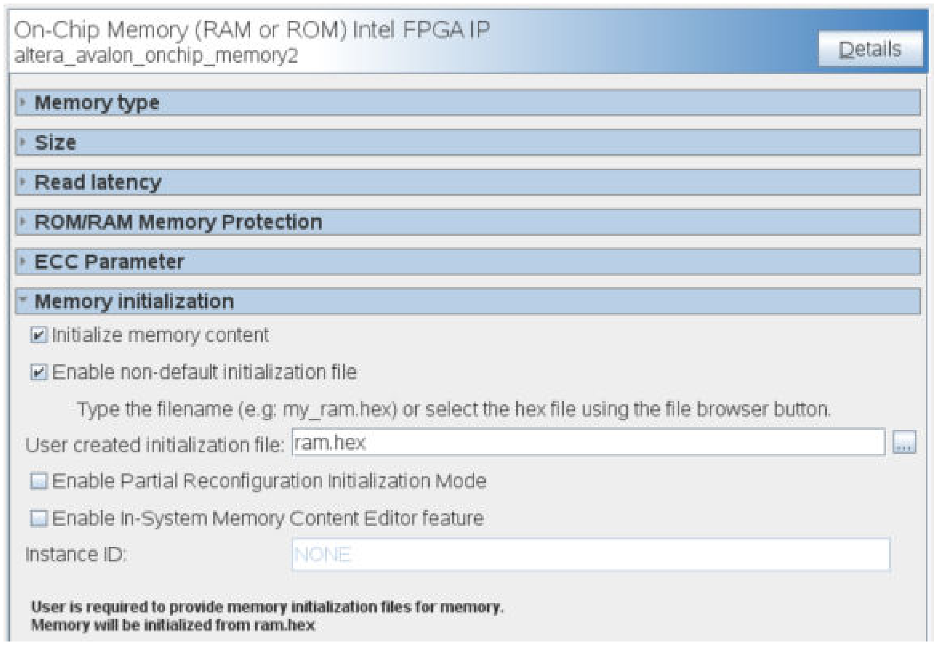
1. Create your Nios V processor project using Intel Quartus Prime and Platform Designer.
2. Ensure the On-Chip Memory (RAM or ROM) Intel FPGA is added into your Platform Designer system.
3. Enable **Initialize memory content** and **Enable non-default initialization file** with `ram.hex` in the on-chip memory.

Figure 75. Connections for Nios V Processor Project



Connectio...	Name	Description	
	⊟ **clock_in**	**Clock Bridge Intel FPGA IP**	
	in_clk	Clock Input	**clk**
	out_clk	Clock Output	Doubl
	⊟ **reset_in**	**Reset Bridge Intel FPGA IP**	
	in_reset	Reset Input	**reset**
	out_reset	Reset Output	Doubl
	⊟ **cpu**	**Nios V/m Processor Intel FPGA ...**	
	clk	Clock Input	Doubl
	reset	Reset Input	Doubl
	platform_irq_rx	Interrupt Receiver	Doubl
	instruction_manager	AXI4 Manager	Doubl
	data_manager	AXI4 Manager	Doubl
	timer_sw_agent	Avalon Memory Mapped Agent	Doubl
	dm_agent	Avalon Memory Mapped Agent	Doubl
	⊟ **ram**	**On-Chip Memory (RAM or ROM...**	
	clk1	Clock Input	Doubl
	s1	Avalon Memory Mapped Agent	Doubl
	reset1	Reset Input	Doubl
	⊟ **jtag_uart**	**JTAG UART Intel FPGA IP**	
	clk	Clock Input	Doubl
	reset	Reset Input	Doubl
	avalon_jtag_slave	Avalon Memory Mapped Agent	Doubl
	irq	Interrupt Sender	Doubl

Figure 76. On-Chip Memory (RAM or ROM) Intel FPGA IP Parameter Settings


On-Chip Memory (RAM or ROM) Intel FPGA IP
altera_avalon_onchip_memory2
Details
Memory type
Size
Read latency
ROM/RAM Memory Protection
ECC Parameter
Memory initialization

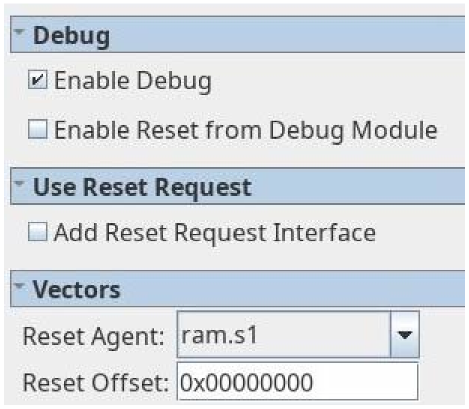
☑ Initialize memory content
☑ Enable non-default initialization file
Type the filename (e.g: my_ram.hex) or select the hex file using the file browser button.
User created initialization file: ram.hex
☐ Enable Partial Reconfiguration Initialization Mode
☐ Enable In-System Memory Content Editor feature
Instance ID: NONE

User is required to provide memory initialization files for memory.
Memory will be initialized from ram.hex

Reset Agent Settings for Nios V Processor

1. In the Nios V processor parameter editor, set the **Reset Agent** to OCRAM

Figure 77. Nios V Processor Parameter Editor Settings



☑ Enable Debug
☐ Enable Reset from Debug Module
☐ Add Reset Request Interface
Reset Agent: ram.s1
Reset Offset: 0x00000000

2. Click **Generate HDL**, the Generation dialog box appears.
3. Specify output file generation options and then click **Generate**.

Intel Quartus Prime Settings

1. In the Intel Quartus Prime software, click **Assignment ➤ Device ➤ Device and Pin Options ➤ Configuration**.
2. Set **Configuration scheme** according to your FPGA configuration scheme
3. Click **OK** to exit the **Device and Pin Options** window.
4. Click **OK** to exit the **Device** window.
5. Click **Start Compilation** to compile your project.

Related Information

Intel Arria 10 SoC Development Kit

4.6.1.2. Software Design Flow

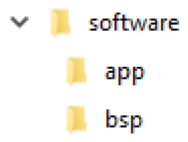
This section provides the design flow to generate and build the Nios V processor software project. To ensure a streamlined build flow, you are encouraged to create similar directory tree in your design project. The following software design flow is based on this directory tree.

To create the software project directory tree, follow these steps:

1. In your design project folder, create a folder called `software`.

2. In the `software` folder, create two folders called `app` and `bsp`.

Figure 78. Software Project Directory Tree



Creating the Application BSP Project

Note: For Intel Quartus Prime Standard Edition software, refer to the topic *AN 980: Nios V Processor Intel Quartus Prime Software Support* for the steps to invoke the BSP Editor GUI.

To launch the BSP Editor, follow these steps:

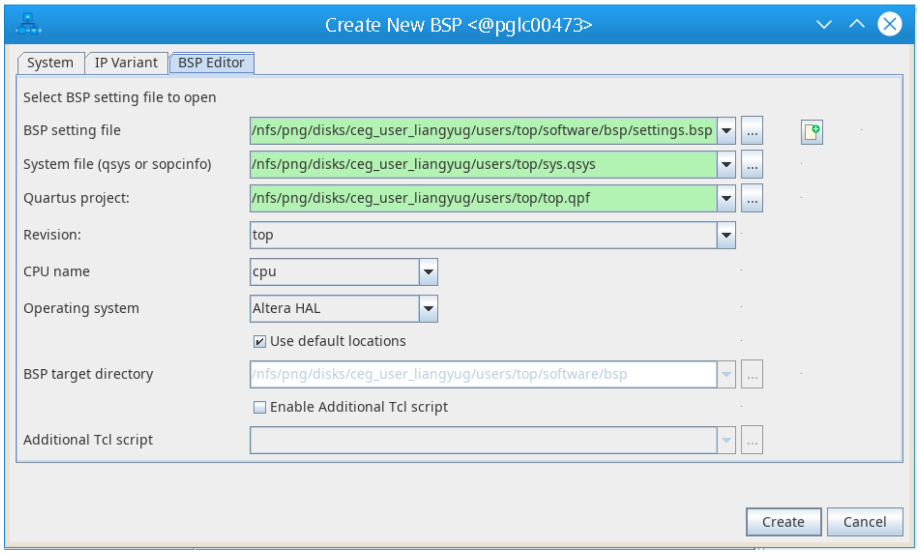
1. In the Platform Designer window, select **File ➤ New BSP**. The **Create New BSP** windows appears.
2. For **BSP setting file**, navigate to the `software/bsp` folder and name the BSP as `settings.bsp`.

   BSP path: `<project directory>/software/bsp/settings.bsp`
3. For **System file (qsys or sopcinfo)**, select the Nios V processor Platform Designer system (`.qsys`) file.

   Note: For Intel Quartus Prime Standard Edition software, generate the BSP file using SOPCINFO file. Refer to *AN 980: Nios V Processor Intel Quartus Prime Software Support* for more information.
4. For **Quartus project**, select the Quartus Project File.

5. For **Revision**, select the correct revision.
6. For **CPU name**, select the Nios V processor.
7. Select the **Operating system** as **Altera HAL**.
8. Click **Create** to create the BSP file.

Figure 79. Create New BSP Window



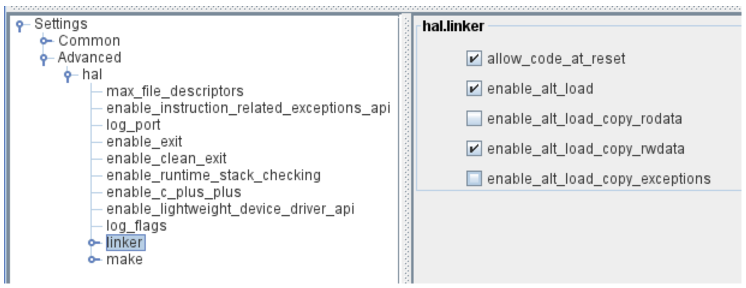
Select BSP setting file to open

BSP setting file	/nfs/png/disks/ceg_user_liangyug/users/top/software/bsp/settings.bsp
System file (qsys or sopcinfo)	/nfs/png/disks/ceg_user_liangyug/users/top/sys.qsys
Quartus project:	/nfs/png/disks/ceg_user_liangyug/users/top/top.qpf
Revision:	top
CPU name	cpu
Operating system	Altera HAL
	✓ Use default locations
BSP target directory	/nfs/png/disks/ceg_user_liangyug/users/top/software/bsp
	☐ Enable Additional Tcl script
Additional Tcl script	

Configuring the BSP Editor and Generating the BSP Project

1. Go to **Main ➤ Settings ➤ Advanced ➤ hal.linker**.
2. Enable the following settings:
    - **allow_code_at_reset**
    - **enable_alt_load**
    - **enable_alt_load_copy_rwdata**

Figure 80. hal.linker Settings



✓ allow_code_at_reset
✓ enable_alt_load
☐ enable_alt_load_copy_rodata
✓ enable_alt_load_copy_rwdata
☐ enable_alt_load_copy_exceptions

3. Click the **BSP Linker Script** tab in the **BSP Editor**
4. Set all the **Linker Section Name** list to the OCRAM.
5. Click **Generate BSP**. Make sure the BSP generation is successful.
6. Close the **BSP Editor**

Generating the Application Project File

1. Navigate to the `software/app` folder and create your user application source code.
2. Launch the Nios V Command Shell.
3. Execute the command below to generate the user application `CMakeLists.txt`.

```
niosv-app --app-dir=software/app --bsp-dir=software/bsp \
    --srcs=software/app/<user application>
```

Building the Application Project

You can choose to build the application project using RiscFree IDE for Intel FPGAs, Eclipse Embedded CDT, or through the command line interface (CLI).

If you prefer using CLI, you can build the application using the following command:

```
cmake -G "Unix Makefiles" -B software/app/build -S software/app
```

```
make -C software/app/build
```

The user application (`.elf`) file is created in `software/app/build` folder.

Generating the HEX File

You must generate a `.hex` file from your application .elf file, so you can create a `.jic` file suitable for programming flash devices.

1. Launch the Nios V Command Shell.
2. For Nios V processor application boot from OCRAM, use the following command line to convert the ELF to HEX for your application. This command creates the user application (`ram.hex`) file.

   ```
   elf2hex software/app/build/<user_application>.elf -o ram.hex \
       -b <base address of OCRAM> \
       -w <data width of OCRAM in bits> \
       -e <end address of OCRAM> \
       -r <data width of OCRAM in bytes>
   ```

3. Recompile the hardware design to memory-initialize the `ram.hex` into the OCRAM.

Related Information

- Summary of Nios V Processor Vector Configuration and BSP Settings on page 105
- AN 980: Nios V Processor Intel Quartus Prime Software Support

4.6.1.3. Programming

The Nios V processor application file is built into the Intel FPGA configuration bitstream. Based on your Intel FPGA configuration scheme, program your device with the programming file containing `.sof` file. The Nios V processor application runs once the Nios V processor system is reset upon entering user mode.

4.7. Nios V Processor Booting from Tightly Coupled Memory (TCM)

This section describes the Nios V processor booting and executing software from Tightly Coupled Memory (TCM) available in all supported Intel FPGA devices.

4.7.1. Nios V Processor Application Executes in-place from TCM

The tightly coupled memories are initialized during FPGA configuration with data from a Nios V processor application image. This data is built into the FPGA configuration bitstream. This process eliminates the need for a boot copier, as the Nios V processor application is already in place at system reset.

Figure 81. Nios V Processor Application Executes In-Place from OCRAM when FPGA Device Configured from QSPI Flash

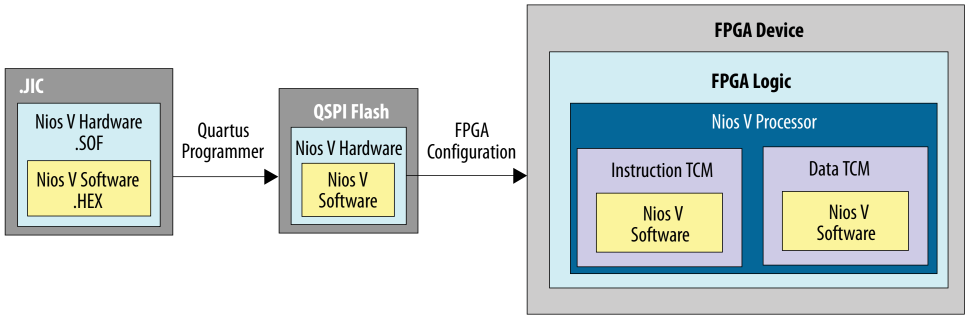
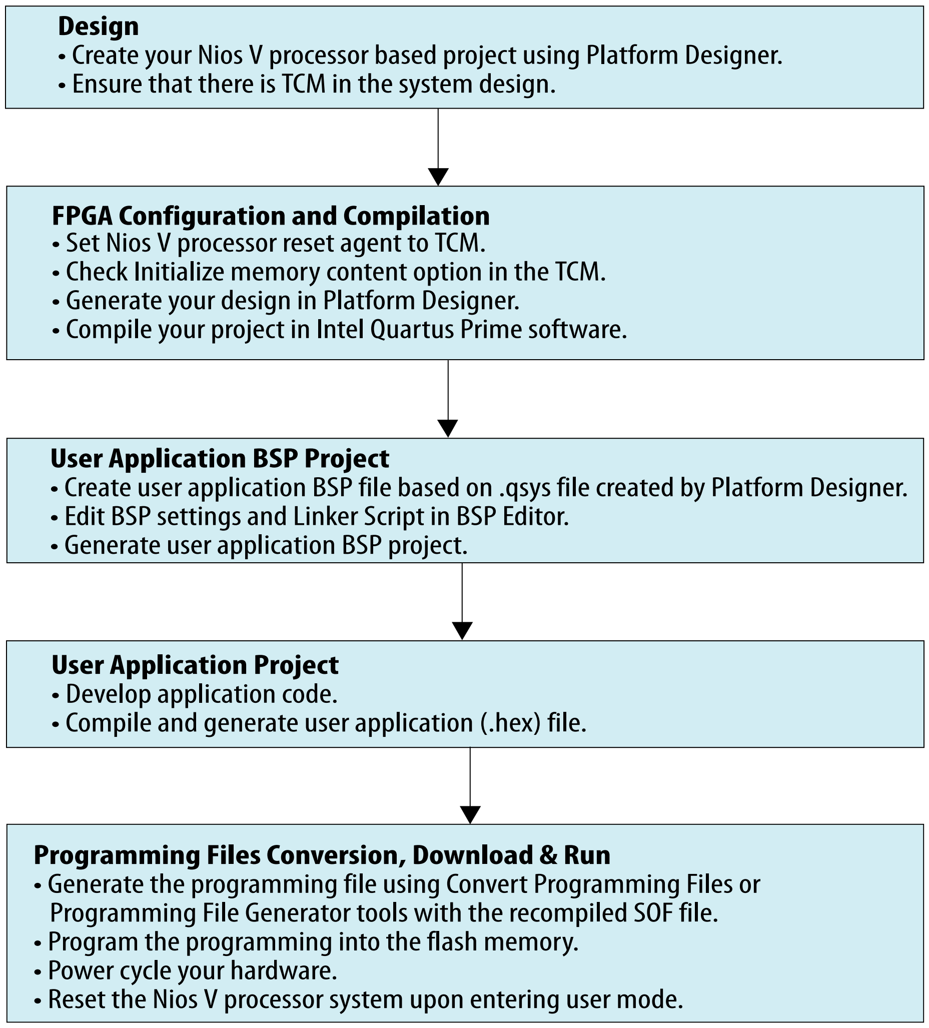


Figure 82. Design, Configuration, and Booting Flow


Design
• Create your Nios V processor based project using Platform Designer.
• Ensure that there is TCM in the system design.
FPGA Configuration and Compilation
• Set Nios V processor reset agent to TCM.
• Check Initialize memory content option in the TCM.
• Generate your design in Platform Designer.
• Compile your project in Intel Quartus Prime software.
User Application BSP Project
• Create user application BSP file based on .qsys file created by Platform Designer.
• Edit BSP settings and Linker Script in BSP Editor.
• Generate user application BSP project.
User Application Project
• Develop application code.
• Compile and generate user application (.hex) file.
Programming Files Conversion, Download & Run
• Generate the programming file using Convert Programming Files or Programming File Generator tools with the recompiled SOF file.
• Program the programming into the flash memory.
• Power cycle your hardware.
• Reset the Nios V processor system upon entering user mode.

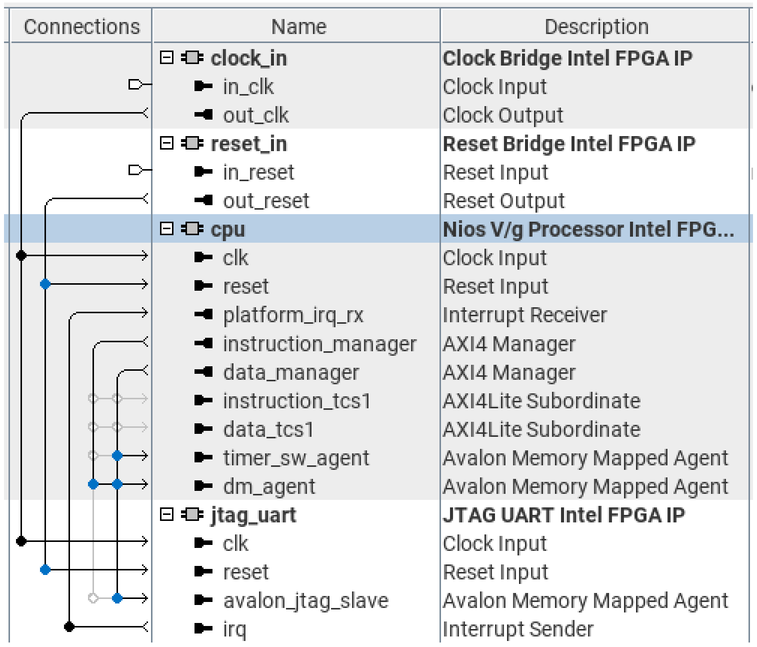
4.7.1.1. Hardware Design Flow

The following sections describe a step-by-step method for building a bootable system for a Nios V processor application from TCM. The example below is built using Intel Arria 10 SoC development kit.

IP Component Settings

1. Create your Nios V processor project using Intel Quartus Prime and Platform Designer.

Figure 83. Connections for Nios V Processor Project

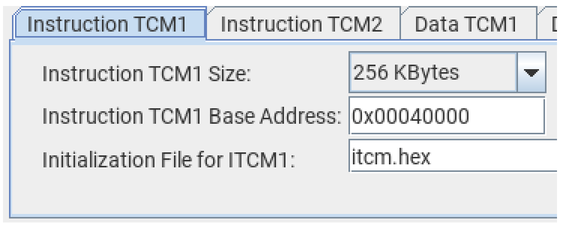


Connections	Name	Description
	clock_in	Clock Bridge Intel FPGA IP
	in_clk	Clock Input
	out_clk	Clock Output
	reset_in	Reset Bridge Intel FPGA IP
	in_reset	Reset Input
	out_reset	Reset Output
	cpu	Nios V/g Processor Intel FPG...
	clk	Clock Input
	reset	Reset Input
	platform_irq_rx	Interrupt Receiver
	instruction_manager	AXI4 Manager
	data_manager	AXI4 Manager
	instruction_tcs1	AXI4Lite Subordinate
	data_tcs1	AXI4Lite Subordinate
	timer_sw_agent	Avalon Memory Mapped Agent
	dm_agent	Avalon Memory Mapped Agent
	jtag_uart	JTAG UART Intel FPGA IP
	clk	Clock Input
	reset	Reset Input
	avalon_jtag_slave	Avalon Memory Mapped Agent
	irq	Interrupt Sender

TCM Settings for Nios V Processor

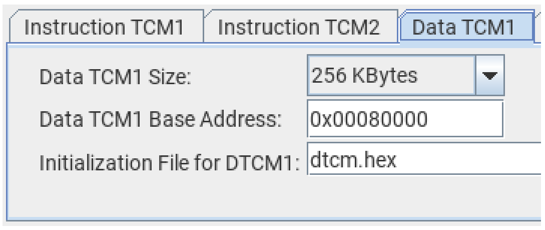
1. In the Nios V processor parameter editor, enable the **Instruction TCM1** and **Data TCM1**.
2. Initialize **Instruction TCM1** with `itcm.hex`.
3. Initialize **Data TCM1** with `dtcm.hex`.

Figure 84. Instruction TCM1 Settings



Instruction TCM1 Size:	256 KBytes
Instruction TCM1 Base Address:	0x00040000
Initialization File for ITCM1:	itcm.hex

Figure 85. Data TCM1 Settings

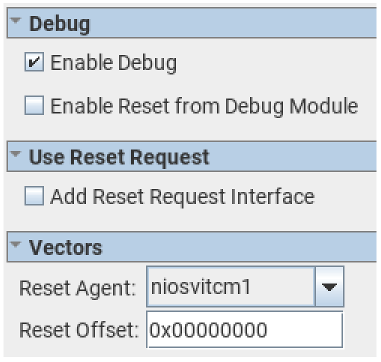


Data TCM1 Size:	256 KBytes
Data TCM1 Base Address:	0x00080000
Initialization File for DTCM1:	dtcm.hex

Reset Agent Settings for Nios V Processor

1. In the Nios V processor parameter editor, set the **Reset Agent** to Instruction TCM1.

Figure 86. Reset Agent Settings for Nios V Processor



Debug
[✓] Enable Debug
[ ] Enable Reset from Debug Module
Use Reset Request
[ ] Add Reset Request Interface
Vectors
Reset Agent: niosvitcm1
Reset Offset: 0x00000000

2. Click **Generate HDL**, the Generation dialog box appears.
3. Specify output file generation options and then click **Generate**.

Intel Quartus Prime Settings

1. In the Intel Quartus Prime software, click **Assignment ➤ Device ➤ Device and Pin Options ➤ Configuration**.
2. Set **Configuration scheme** according to your FPGA configuration scheme
3. Click **OK** to exit the **Device and Pin Options** window.
4. Click **OK** to exit the **Device** window.
5. Click **Start Compilation** to compile your project.

Related Information

Intel Arria 10 SoC Development Kit

4.7.1.2. Software Design Flow

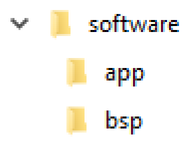
This section provides the design flow to generate and build the Nios V processor software project. To ensure a streamlined build flow, you are encouraged to create similar directory tree in your design project. The following software design flow is based on this directory tree.

To create the software project directory tree, follow these steps:

1. In your design project folder, create a folder called `software`.

2. In the `software` folder, create two folders called `app` and `bsp`.

Figure 87. Software Project Directory Tree



Creating the Application BSP Project

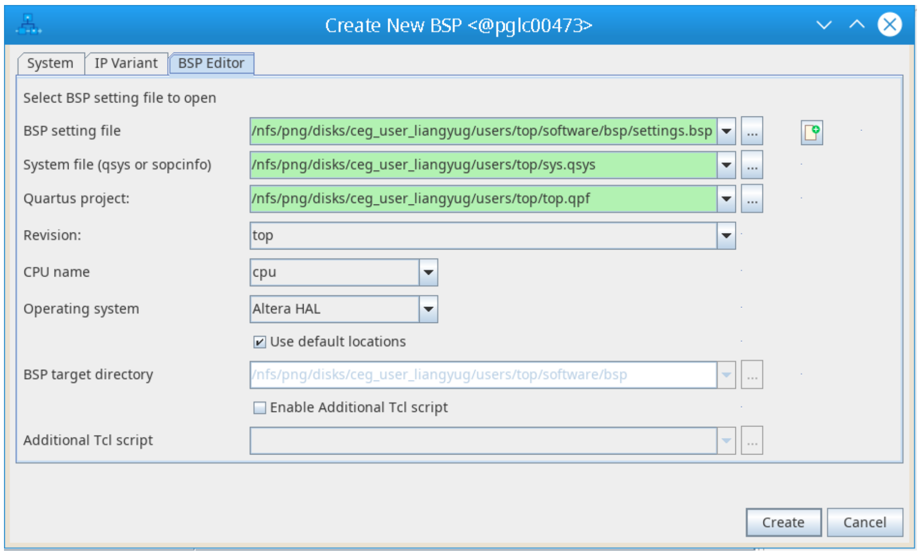
To launch the BSP Editor, follow these steps:

1. In the Platform Designer window, select **File ➤ New BSP**. The **Create New BSP** windows appears.
2. For **BSP setting file**, navigate to the `software/bsp` folder and name the BSP as `settings.bsp`.

   BSP path: `<project directory>/software/bsp/settings.bsp`
3. For **System file (qsys or sopcinfo)**, select the Nios V/g processor Platform Designer system (`.qsys`) file.

   Note: For Intel Quartus Prime Standard Edition software, generate the BSP files using SOPCINFO. Refer to *AN 980: Nios V Processor Intel Quartus Prime Software Support* for more information.
4. For **Quartus project**, select the Intel Quartus Prime Project File.
5. For **Revision**, select the correct revision.
6. For **CPU name**, select the Nios V/g processor.
7. Select the **Operating system** as **Altera HAL**.
8. Click **Create** to create the BSP file.

Figure 88. Create New BSP Window



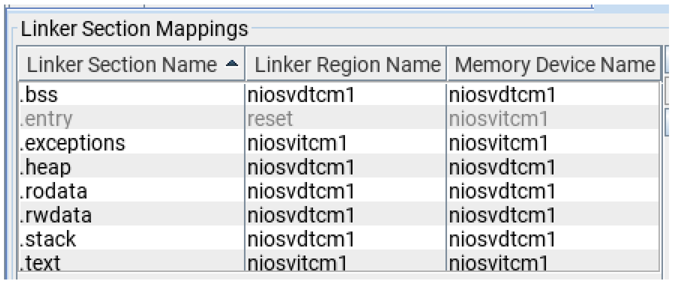
Select BSP setting file to open

BSP setting file: /nfs/png/disks/ceg_user_liangyug/users/top/software/bsp/settings.bsp
System file (qsys or sopcinfo): /nfs/png/disks/ceg_user_liangyug/users/top/sys.qsys
Quartus project: /nfs/png/disks/ceg_user_liangyug/users/top/top.qpf
Revision: top
CPU name: cpu
Operating system: Altera HAL
[✓] Use default locations
BSP target directory: /nfs/png/disks/ceg_user_liangyug/users/top/software/bsp
[ ] Enable Additional Tcl script
Additional Tcl script:

Configuring the BSP Editor and Generating the BSP Project

1. Click the **BSP Linker Script** tab in the **BSP Editor**.
2. In the **Linker Section Name** perform the following settings:
   a. Set `.text` and `.exceptions` to Instruction TCM1.
   b. Set the remaining Linker Section Name to the Data TCM1.

Figure 89. Linker Region Settings for TCM



Linker Section Mappings

Linker Section Name	Linker Region Name	Memory Device Name
.bss	niosvdtcm1	niosvdtcm1
.entry	reset	niosvitcm1
.exceptions	niosvitcm1	niosvitcm1
.heap	niosvdtcm1	niosvdtcm1
.rodata	niosvdtcm1	niosvdtcm1
.rwdata	niosvdtcm1	niosvdtcm1
.stack	niosvdtcm1	niosvdtcm1
.text	niosvitcm1	niosvitcm1

3. Click **Generate BSP**. Make sure the BSP generation is successful.
4. Close the **BSP Editor**

Generating the Application Project File

1. Navigate to the `software/app` folder and create your user application source code.
2. Launch the Nios V Command Shell.
3. Execute the command below to generate the user application `CMakeLists.txt`.

```
niosv-app --app-dir=software/app --bsp-dir=software/bsp \
    --srcs=software/app/<user application>
```

Building the Application Project

You can choose to build the application project using RiscFree IDE for Intel FPGAs, Eclipse Embedded CDT, or through the command line interface (CLI).

If you prefer using CLI, you can build the application using the following command:

```
cmake -G "Unix Makefiles" -B software/app/build -S software/app
```

```
make -C software/app/build
```

The user application (`.elf`) file is created in `software/app/build` folder.

Generating the HEX File

You must generate a `.hex` file from your application .elf file, so you can create a `.jic` file suitable for programming flash devices.

1. Launch the Nios V Command Shell.
2. For Nios V processor application boot from TCM, use the following command line to convert the ELF to HEX for your application. This command creates the user application (`itcm.hex` and `dtcm.hex`) file.

```
elf2hex software/app/build/<user_application>.elf -o itcm.hex \
    -b <base address of ITCM> -w 32  \
    -e <end address of ITCM> -r 4

elf2hex software/app/build/<user_application>.elf -o dtcm.hex \
    -b <base address of DTCM> -w 32 \
    -e <end address of DTCM> -r 4
```

3. Recompile the hardware design to memory-initialize both the HEX files into the Instruction TCM and Data TCM.

Related Information

- Summary of Nios V Processor Vector Configuration and BSP Settings on page 105
- AN 980: Nios V Processor Intel Quartus Prime Software Support

4.7.1.3. Programming

The Nios V processor application file is built into the Intel FPGA configuration bitstream. Based on your Intel FPGA configuration scheme, program your device with the programming file containing `.sof` file. The Nios V processor application runs once the Nios V processor system is reset upon entering user mode.

4.8. Summary of Nios V Processor Vector Configuration and BSP Settings

The following table shows a summary of Nios V processor reset and exception agent configurations, and BSP settings.

Table 31. Summary of Nios V Processor Vector Configurations and BSP Settings

Boot Option	Reset Agent	BSP Editor Setting: Settings	BSP Editor Setting: Linker Script
Nios V processor application executes in-place from configuration QSPI flash	Configuration QSPI Flash	If the `.exceptionn` Linker Section is set to OCRAM/ External RAM, enable the following settings in **Advanced.hal.linker** • allow_code_at_reset • enable_alt_load • enable_alt_load_copy_rodata • enable_alt_load_copy_rwdata • enable_alt_load_copy_exceptions If the `.exception` Linker Section is set to QSPI Flash, enable the following settings in **Advanced.hal.linker**: • allow_code_at_reset • enable_alt_load • enable_alt_load_copy_rodata • enable_alt_load_copy_rwdata	• Set `.text` Linker Section to QSPI flash • Set `.exception` Linker Section to OCRAM/External RAM or QSPI flash. • Set other Linker Sections (`.heap`, `.rwdata`, `rodata`,`.bss`, `.stack`) to OCRAM / External RAM
Nios V processor application copied from configuration QSPI flash to RAM using GSFI bootloader	Configuration QSPI Flash	Uncheck all settings in **Advanced.hal.linker** .	Make sure all Linker Sections are set to OCRAM / External RAM.
Nios V processor application copied from configuration QSPI flash to RAM using SDM bootloader	Bootloader ROM	For SDM bootloader, enable the following settings in **Advanced.hal.linker**: • allow_code_at_reset • enable_alt_load • enable_alt_load_copy_rodata • enable_alt_load_copy_rwdata • enable_alt_load_copy_exceptions	For SDM bootloader, • Set `.text` Linker Section to Bootloader ROM. • Set other Linker Sections (`.heap`, `.rwdata`, `.rodata`, `.bss` , `.stack`, `.exception`) to Bootloader RAM.
		For user application, enable the following settings in **Advanced.hal.linker**:	For user application, make sure all Linker Sections are set to User Application RAM.

continued...

Boot Option	Reset Agent	BSP Editor Setting: Settings	BSP Editor Setting: Linker Script
		• enable_alt_load • enable_alt_load_copy_exceptions	
Nios V processor application execute in-place from On-chip Memory (OCRAM)	OCRAM	Enable **allow_code_at_reset** in **Advanced.hal.linker** and uncheck other settings.	Make sure all Linker Sections are set to OCRAM.
Nios V processor application execute in-place from Tightly Coupled Memory (TCM)	TCM	Enable **allow_code_at_reset** in **Advanced.hal.linker** and uncheck other settings.	• Set `.text` and `.exception` Linker Section to Instruction TCM. • Set other Linker Section (`.heap`, `.rwdata`, `.rodata`, `.bss` , `.stack` to Data TCM.

Related Information

- Software Design Flow on page 52
  Nios V Processor Application Copied from Configuration QSPI Flash to RAM Using Boot Copier (GSFI Bootloader)
- Software Design Flow on page 64
  Nios V Processor Application Executes-In-Place from Configuration QSPI Flash
- Software Design Flow (SDM Bootloader Project) on page 78
  Nios V Processor Application Copied from Configuration QSPI Flash to RAM Using Boot Copier (SDM Bootloader) - SDM Bootloader Project
- Software Design Flow (User Application Project) on page 83
  Nios V Processor Application Copied from Configuration QSPI Flash to RAM Using Boot Copier (SDM Bootloader) - User Application Project
- Software Design Flow on page 96
  Nios V Processor Application Executes in-place from OCRAM
- Software Design Flow on page 102

5. Nios V Processor - Using the MicroC/TCP-IP Stack

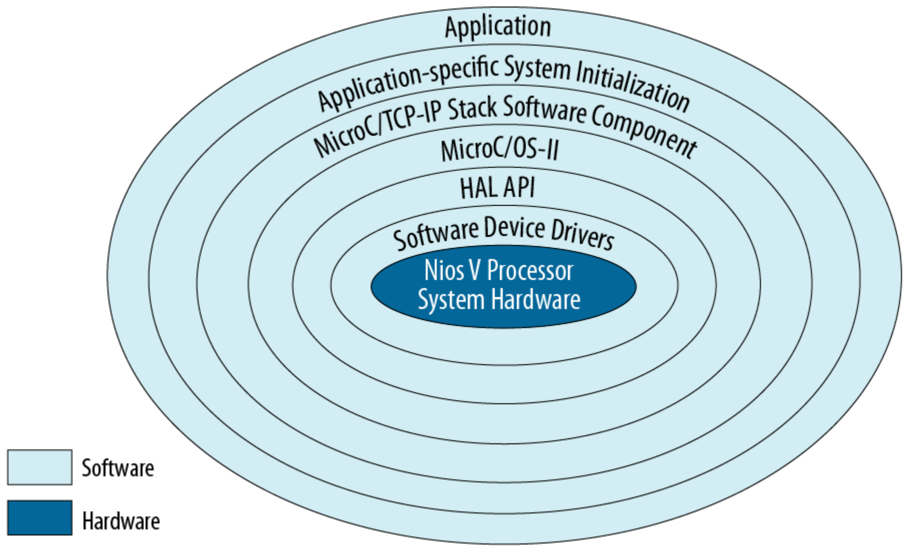
5.1. Introduction

The Nios V processor tools contains the µC/OS-II RTOS and the µC/TCP-IP software component, providing designers with the ability to quickly build networked embedded systems applications for the Nios V processor.

5.2. Software Architecture

The onion diagram shows the architectural layers of a Nios V processor µC/OS-II software application.

Figure 90. Layered Software Model


Application
Application-specific System Initialization
MicroC/TCP-IP Stack Software Component
MicroC/OS-II
HAL API
Software Device Drivers
Nios V Processor System Hardware
Software
Hardware

Each layer encapsulates the specific implementation details of that layer, abstracting the data for the next outer layer. The following list describes each layer:

- **Nios V processor system hardware**: The core of the onion diagram represents the Nios V processor and hardware peripherals implemented in the Intel FPGA.
- **Software device drivers**: The software device drivers layer contains the software functions that manipulate the Ethernet and hardware peripherals. These drivers know the physical details of the peripheral devices, abstracting those details from the outer layers.
- **HAL API**: The Hardware Abstraction Layer (HAL) application programming interface (API) provides a standardized interface to the software device drivers, presenting a POSIX-like API to the outer layers.
- **MicroC/OS-II**: The µC/OS-II RTOS layer provides multitasking and inter-task communication services to the µC/TCP-IP Stack and the Nios V processor.
- **MicroC/TCP-IP Stack software component**: The µC/TCP-IP Stack software component layer provides networking services to the application layer and application-specific system initialization layer through the sockets API.
- **Application-specific system initialization**: The application-specific system initialization layer includes the µC/OS-II and µC/TCP-IP Stack software component initialization functions invoked from main(), as well as creates all application tasks, and all the semaphores, queue, and event flag RTOS inter-task communication resources.
- **Application**: The outermost application layer contains the Nios V µC/TCP-IP Stack application.

5.3. Support and Licensing

Intel distributes µC/OS-II and µC/TCP-IP in the Intel Quartus Prime Design Suite for evaluation purposes only. Commercial version of µC/OS-II and µC/TCP-IP is under Apache 2.0 Open Source Licensing, for more information refer to the Micrium Licensing Website.

Related Information

Micrium Licensing Website
For more information about Commercial version about µC/OS-II and µC/TCP-IP under Apache 2.0 Open Source Licensing.

5.4. MicroC/TCP-IP Example Designs

5.4.1. Hardware and Software Requirements

To use a µC/OS-II and µC/TCP-IP program on an Intel FPGA requires the following hardware and software:

- Intel Quartus Prime software
  - Intel Quartus Prime Pro Edition software version 21.3 or later
  - Intel Quartus Prime Standard Edition software version 22.1 or later
- Ashling RiscFree IDE for Intel FPGAs software version 22.2 or later

  Note: Intel recommends you to install the same software version for all softwares.
- One of the supported Intel FPGA devices
  - The example designs are implemented on Intel Arria 10 10 SoC development kit
- Intel FPGA Download Cable II
- RJ-45 connected Ethernet cable on the same network as the PC development host

You must connect your development board to a host PC on the Ethernet and USB/JTAG ports.

Related Information

Intel Arria 10 SoC Development Kit

5.4.2. Overview

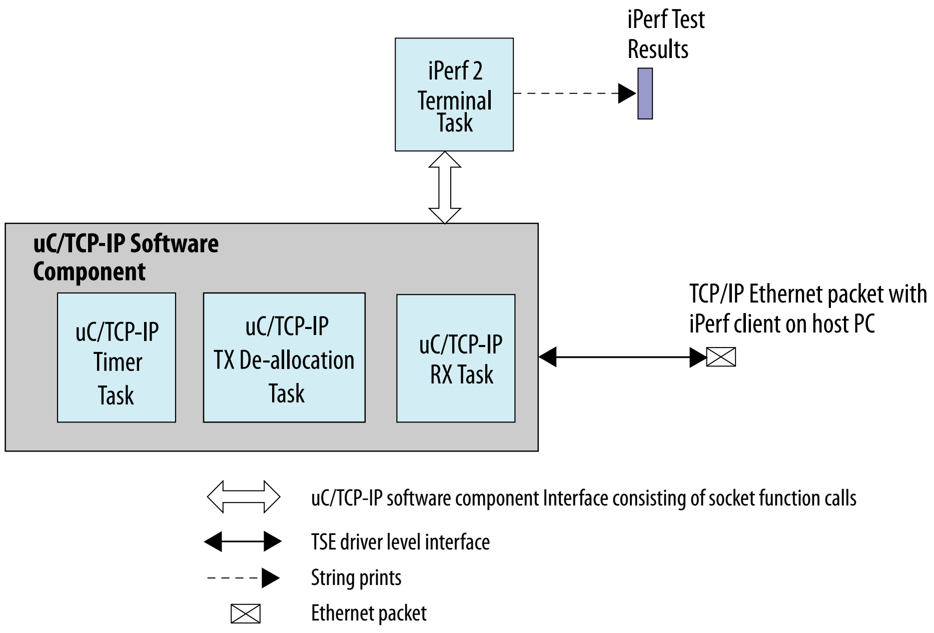
Note: For Intel Quartus Prime Standard Edition software, refer to *AN 980: Nios V Processor Intel Quartus Prime Software Support* for the steps to generate the example design.

You can download the µC/TCP-IP Example Designs from the *Intel FPGA Store*. The example designs are based on the Intel Arria 10 10 SoC development kit. Using the scripts, the hardware and software design are generated, and programmed as SRAM Object Files (`.sof`) into the device. Using the memory-initialized `.sof` file, the Nios V processor boots the µC/TCP-IP application from the On-Chip Memory after resetting the processor during User Mode.

The featured µC/TCP-IP Example Designs are :

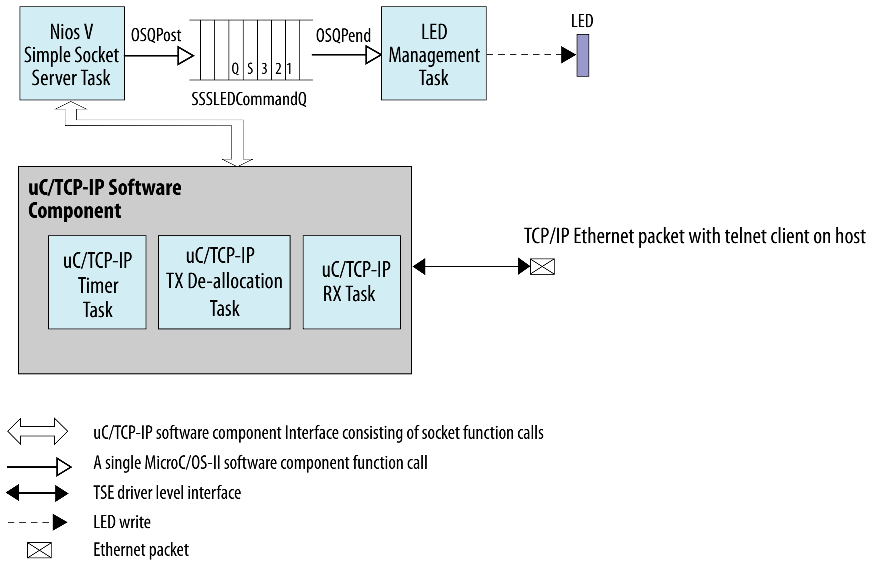
- **µC/TCP-IP IPerf Example Design**
  - This example design incorporated the µC/IPerf, an iPerf 2 server or client developed for the µC/TCP-IP Stack and the µC/OS-II RTOS. iPerf 2 is a benchmarking tool for measuring performance between two systems, and it can be used as a server or a client.
  - An iPerf server receives iPerf request sent over a TCP/IP connection from any iPerf clients, and runs the iPerf test according to the provided arguments. Each test reports the bandwidth, loss and other parameters.

Figure 91. µC/TCP-IP IPerf Data Flow Diagram


iPerf Test Results
iPerf 2 Terminal Task
uC/TCP-IP Software Component
uC/TCP-IP Timer Task
uC/TCP-IP TX De-allocation Task
uC/TCP-IP RX Task
TCP/IP Ethernet packet with iPerf client on host PC
uC/TCP-IP software component Interface consisting of socket function calls
TSE driver level interface
String prints
Ethernet packet

- **µC/TCP-IP Simple Socket Server Example Design**
  - This example design demonstrates communication with a telnet client on a development host PC. The telnet client offers a convenient way of issuing commands over a TCP/IP socket to the Ethernet-connected µC/TCP-IP running on the development board with a simple TCP/IP socket server example.
  - The socket server example receives commands sent over a TCP/IP connection and turns LEDs on and off according to the commands. The example consists of a socket server task that listens for commands on a TCP/IP port and dispatches those commands to a set of LED management tasks.

Figure 92. µC/TCP-IP Simple Socket Server Data Flow Diagram


Nios V Simple Socket Server Task
OSQPost
Q S 3 2 1
SSSLEDCommandQ
OSQPend
LED Management Task
LED
uC/TCP-IP Software Component
uC/TCP-IP Timer Task
uC/TCP-IP TX De-allocation Task
uC/TCP-IP RX Task
TCP/IP Ethernet packet with telnet client on host
uC/TCP-IP software component Interface consisting of socket function calls
A single MicroC/OS-II software component function call
TSE driver level interface
LED write
Ethernet packet

Note: The Nios V target system does not implement a full telnet server.

Related Information

- µC Product Documentation and Release Notes
  For more information about µC/OS-II, µC/TCP-IP and µC/IPerf.
- AN 980: Nios V Processor Intel Quartus Prime Software Support
- AN 980: Nios V Processor Intel Quartus Prime Software Support

5.4.3. Acquiring the Example Design Files

Generating the µC/TCP-IP Example Designs

To generate the µC/TCP-IP Example Designs , perform the following steps:

1. Go to Intel FPGA Design Store.
2. Search for *Arria10 - Simple Socket Server* or *Arria10 - IPerf* package.
3. Click on the link at the title.

4. Accept the *Software License Agreement*.
5. Download the package according to the Intel Quartus Prime software version of your host machine.
6. Refer to the `readme.txt` for how-to guide.

Table 32. Example Design File Description

File	Description
`hw/`	Contains files necessary to run the hardware project.
`ready_to_test/`	Contains pre-built hardware and software binaries to run the design on the target hardware. For this package, the target hardware is Intel Arria 10 SoC development kit.
`scripts/`	Consists of scripts to build the design.
`sw/`	Contains software application files.
`readme.txt`	Contains description and steps to apply the pre-bulit binaries or rebuild the binaries from scratch.

Running the µC/TCP-IP Example Designs

The µC/TCP-IP Example Designs are provided with scripts to facilitate the build flow. The scripts are stored in the scripts folder. You may refer to the readme file (readme.txt) to develop the example designs using the provided scripts, or develop the design manually using the Nios V processor tools.

For more information about the hardware and software development follow, refer to Hardware Development Flow and Software Development Flow.

Related Information

Intel FPGA Design Store

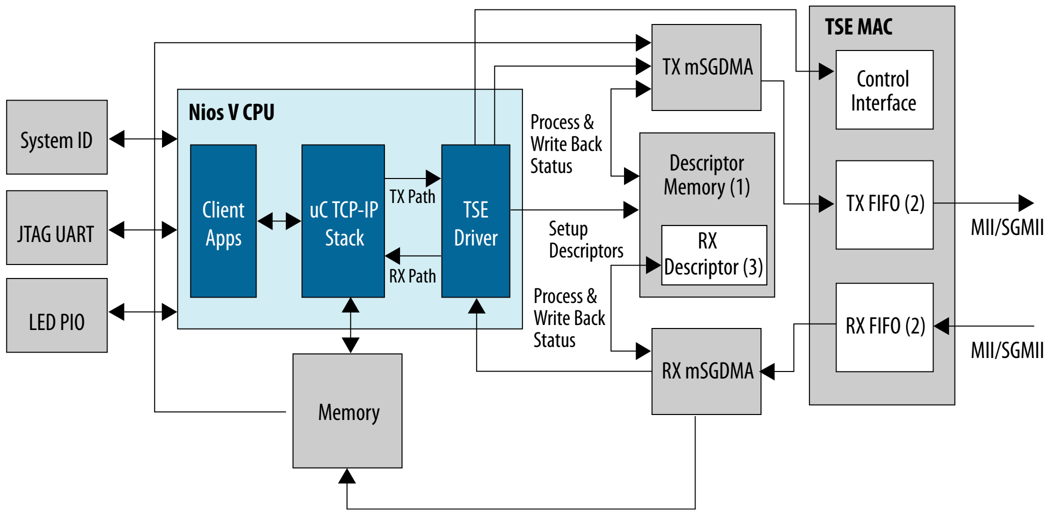
5.4.4. Hardware Design Files

Despite the example designs functioned differently, they share similar hardware design and BSP settings. The only difference lies in their respective Nios V application source code, one for the Simple Socket Server application, while the other for the iPerf 2 application.

The µC/TCP-IP example designs are developed using the Platform Designer. The hardware files can be generated using the `build_sof.py` Python script. The example design consist of:

- Nios V Processor Intel FPGA IP
- On-Chip Memory II Intel FPGA IP for System Memory and Descriptor Memory
- JTAG UART Intel FPGA IP
- System ID Peripheral Intel FPGA IP
- Parallel I/O Intel FPGA IP (PIO)
- Modular Scatter-Gather DMA Intel FPGA IP (mSGDMA)
- Triple-Speed Ethernet Intel FPGA IP (TSE)

Figure 93. Hardware Block Diagram



Note:

- [1] The first *n* bytes are reserved for mSGDMA descriptor buffers, where *n* is the number of bytes taken by the configured RX or TX buffers. Applications must not use this memory region.
- [2] For MAC variations without internet FIFO buffers, the transmit and receive FIFOs are external to the MAC function.
- [3] Only one buffer type (RX or TX buffers) can reside in the descriptor memory.

5.4.5. Software Design Files

5.4.5.1. MicroC/TCP-IP IPerf Example Design

The µC/TCP-IP IPerf example design software files are readily available in the example design zip file. They are stored in the `sw/app` folder.

The following software files constitute the µC/TCP-IP IPerf application:

- **uC-IPerf** folder: Contains µC/IPerf source code.
- **app_iperf.c**: Contains the iPerf reporter application.
- **app_iperf.h**: Contains function prototypes for the reporter application.
- **iperf_cfg.h**: Describe the µC/IPerf module static parameters and run-time configuration structure.
- **log.h**: Contains definitions for logging macros.
- **main.c**: Defines the global structure of type `alt_tse_system_info` which describes the TSE configuration. Defines `main()`, which initializes µC/OS-II, µC/TCP-IP and µC/IPerf, processes the MAC and IP addresses, contains the PHY management tasks, and defines function prototypes.
- **uc_tcp_ip_init.c**: Contains MAC address and IP address routines to manage addressing. Routines are used by µC/TCP-IP during initialization, but are implementation-specific.
- **uc_tcp_ip_init.h**: Contains definitions and function prototypes for µC/TCP-IP initialization.

5.4.5.2. MicroC/TCP-IP Simple Socket Server Example Design

The µC/TCP-IP Simple Socket Server example design software files are readily available in the example design zip file. They are stored in the `sw/app` folder.

The following software files constitute the µC/TCP-IP Simple Socket Server application:

- **alt_error_handler.c**: Contains three error handlers, one each for the Nios V Simple Socket Server, µC/TCP-IP, and µC/OS-II.
- **alt_error_handler.h**: Contains definitions and function prototypes for the three software component-specific error handlers.
- **led.c**: Contains the LED management tasks.
- **led.h** : Contains function prototypes for the LED management tasks.
- **log.h**: Contains definitions for logging macros.
- **main.c**: Defines the global structure of type `alt_tse_system_info` which describes the TSE configuration. Defines `main()`, which initializes µC/OS-II and µC/TCP-IP, processes the MAC and IP addresses, contains the PHY management tasks, and defines function prototypes.
- **simple_socket_server.c**: Defines the tasks and functions that use the µC/TCP-IP sockets interface, and creates all the µC/OS-II resources.
- **simple_socket_server.h**: Defines the task prototypes, task priorities, and other µC/OS-II resources used.
- **uc_tcp_ip_init.c**: Contains MAC address and IP address routines to manage addressing. Routines are used by µC/TCP-IP during initialization, but are implementation-specific.
- **uc_tcp_ip_init.h**: Contains definitions and function prototypes for µC/TCP-IP initialization.

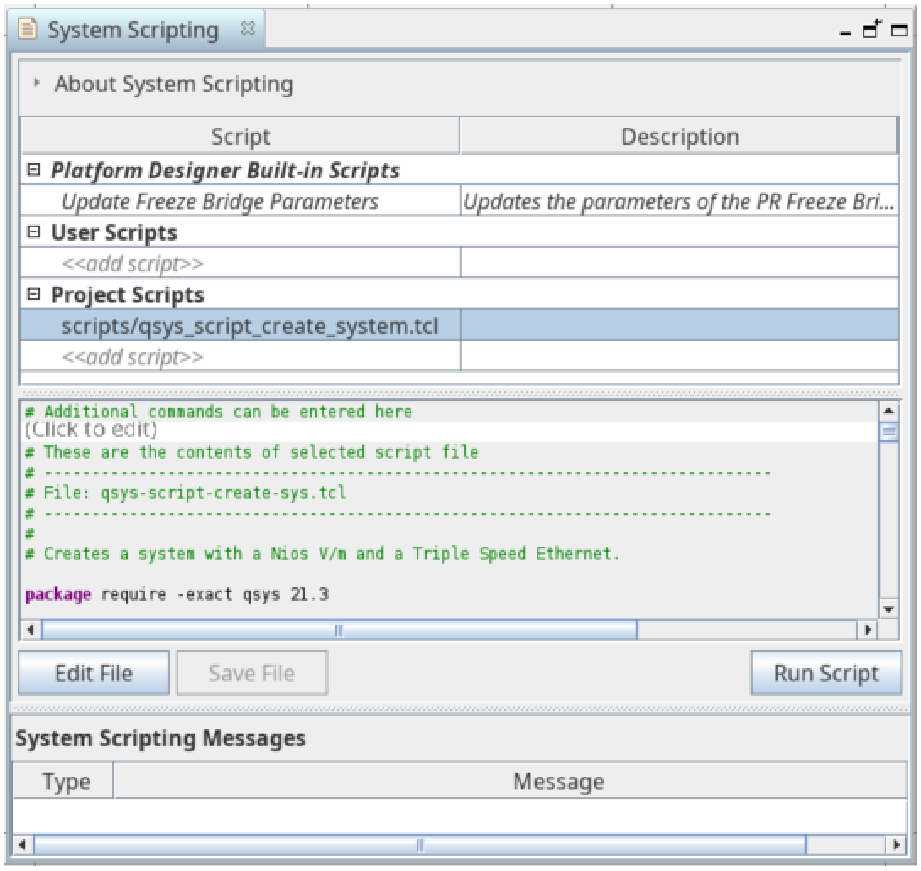
5.5. Development Flow

5.5.1. Hardware Development Flow

You can create the µC/TCP-IP example designs hardware system using the Platform Designer.

1. In the Platform Designer, create a new Platform Designer system (`sys.qsys`).
2. Navigate to **View ➤ System Scripting**.
3. Under the **Project Scripts**, add and run `sys.tcl`.

Figure 94. System Scripting Windows



Script	Description
Platform Designer Built-in Scripts	
Update Freeze Bridge Parameters	Updates the parameters of the PR Freeze Bri...
User Scripts	
<<add script>>	
Project Scripts	
scripts/qsys_script_create_system.tcl	
<<add script>>	

```
# Additional commands can be entered here
(Click to edit)
# These are the contents of selected script file
# ---------------------------------------------------------------------------
# File: qsys-script-create-sys.tcl
# ---------------------------------------------------------------------------
#
# Creates a system with a Nios V/m and a Triple Speed Ethernet.

package require -exact qsys 21.3
```

Type	Message

4. The generated Platform Designer system consist of the Nios V processor, TSE IP, mSGDMA IP and other peripherals. Refer to Hardware Design Files for the complete system.
5. Click **Generate HDL** to generate the system HDL.
6. Click **Processing ➤ Start Compilation** to perform a full hardware compilation and generate the hardware `.sof` file.

   Note: Currently, the hardware `.sof` file is not memory-initialized with the µC/TCP-IP application. Refer to the following section for more information.

Related Information

- Intel Quartus Prime Pro Edition User Guide: Platform Designer
  More information about Creating a Board Support Package with BSP Editor.
- Nios V Processor Software Developer Handbook: Board Support Package Editor

5.5.2. Software Development Flow

Creating a µC/TCP-IP and µC/OS-II software image for the Simple Socket Server or the iPerf example design consist of the following general steps:

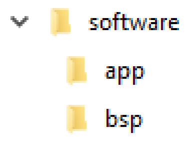
1. Create a board support package (BSP) project, including µC/OS-II and the µC/TCP-IP software component.
2. Creating a Nios V application project with the provided software design files.
3. Building the application project.
4. Running and debugging the application project.

To ensure a streamlined build flow, you are encouraged to create similar directory tree in your design project. The following software design flow is based on this directory tree.

Follow these steps to create the software project directory tree:

1. In your design project folder, create a folder called `software`.
2. In the `software` folder, create two folders called `app` and `bsp`.

Figure 95. Software Project Directory Tree



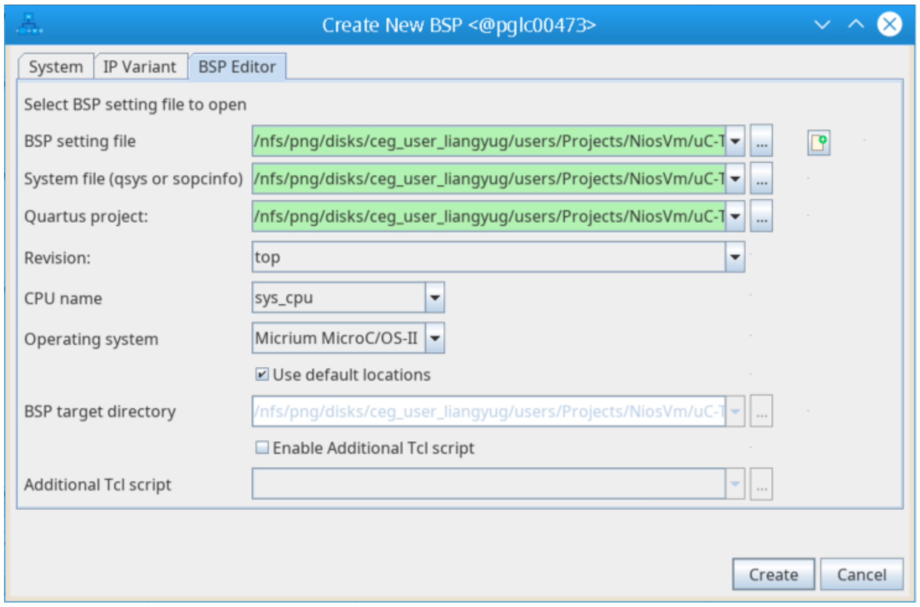
5.5.2.1. Creating a BSP project

Follow these steps to create a BSP project:

1. In the Platform Designer window, go to **File ➤ New BSP**. The **Create New BSP** window appears.
2. For **BSP setting file**, navigate to the software/bsp folder and create a BSP file (settings.bsp).
3. For **System file (qsys or sopcinfo)**, select the Nios V processor Platform Designer system.

   Note: For Intel Quartus Prime Standard Edition software, generate the BSP files using SOPCINFO file. Refer to *AN 980: Nios V Processor Intel Quartus Prime Software Support* for more information.
4. For **Quartus project**, select the example design Quartus Project File.
5. For **Revision**, select the correct revision.
6. For **CPU name**, select the Nios V processor.
7. Select the **Operating system** as **Micrium MicroC/OS II**.
8. Click **Create** to create the BSP file.

Figure 96. Create New BSP windows



Select BSP setting file to open

BSP setting file	/nfs/png/disks/ceg_user_liangyug/users/Projects/NiosVm/uC-T
System file (qsys or sopcinfo)	/nfs/png/disks/ceg_user_liangyug/users/Projects/NiosVm/uC-T
Quartus project:	/nfs/png/disks/ceg_user_liangyug/users/Projects/NiosVm/uC-T
Revision:	top
CPU name	sys_cpu
Operating system	Micrium MicroC/OS-II
	☑ Use default locations
BSP target directory	/nfs/png/disks/ceg_user_liangyug/users/Projects/NiosVm/uC-T
	☐ Enable Additional Tcl script
Additional Tcl script	

5.5.2.2. Configuring the BSP

Follow these steps to configure the BSP project:

1. In the BSP Editor, navigate to **Main ➤ Settings** to configure the BSP Settings as shown in the following table.
2. Both example designs apply the same BSP settings.

Table 33. BSP Editor Settings

BSP Editor Settings	Description
`hal.enable_instruction_related_exceptions_api`	**Enable** by checking the option box.
`hal.log_flags`	Set value as **0**
`hal.log_port`	Set value as **sys_jtag_uart**
`hal.make.cflags_defined_symbols`	Set value as **-DTSE_MY_SYSTEM**
`hal.make.cflags_user_flags`	Set value as **-ffunction-sections -fdata-sections -fno-tree-vectorize**
`hal.make.cflags_warnings`	Set value as **-Wall -Wextra -Wformat -Wformat-security**
`hal.make.link_flags`	Set value as **-Wl,--gc-sections**
`ucosii.miscellaneous.os_max_events`	Set value as **80**
`ucosii.os_tmr_en`	**Enable** by checking the option box.

3. Go to **BSP Software Package** tab and enable the `uc_tcp_ip` software package.

Figure 97. BSP Software Package

4. Click **Generate BSP** to generate the BSP file.

5.5.2.3. Creating an Application Project

You need to use the niosv-app utility to create the application `CMakeLists.txt` and source it to the application source code. Due to different application source code between the two example designs, the niosv-app commands are sourced different.

1. Copy the **Software Design Files** to the `software/app` folder.
2. Launch the Nios V Command Shell.
3. Based on your example design, execute the following command to generate the user application `CMakeLists.txt`.
    - µC/TCP-IP Simple Socket Server example design

    ```
    niosv-app --app-dir=software/app --bsp-dir=software/bsp \
        --srcs=software/app/alt_error_handler.c \
        --srcs=software/app/led.c \
        --srcs=software/app/main.c \
        --srcs=software/app/simple_socket_server.c \
        --srcs=software/app/uc_tcp_ip_init.c
    ```

    - µC/TCP-IP IPerf example design

    ```
    niosv-app --app-dir=software/app --bsp-dir=software/bsp \
        --srcs=software/app/app_iperf.c \
        --srcs=software/app/main.c \
        --srcs=software/app/uC-IPerf/OS/uCOS-II/iperf_os.c \
        --srcs=software/app/uC-IPerf/Reporter/Terminal/iperf_rep.c \
        --srcs=software/app/uC-IPerf/Source/iperf-c.c \
        --srcs=software/app/uC-IPerf/Source/iperf-s.c \
        --srcs=software/app/uC-IPerf/Source/iperf.c \
        --srcs=software/app/uc_tcp_ip_init.c \
        --incs=software/app/uC-IPerf
    ```

5.5.2.4. Building the Application Project

You can choose to build the application project using RiscFree IDE for Intel FPGAs, Eclipse Embedded CDT, or through the command line interface (CLI).

You can configure the source files such as enabling DHCP or setting MAC and IP addresses. Refer to Optional Configuration for more details.

If you prefer CLI, you can build the application using the following commands:

```
cmake -G "Unix Makefiles" -DCMAKE_BUILD_TYPE=Release \
 -B software/app/build -S software/app
```

```
make -j4 -C software/app/build
```

```
The user application (.elf) file is created in software/app/build folder.
```

5.5.3. Device Programming

To program Nios V processor based system into the FPGA and to run your application, use Intel Quartus Prime Programmer tool.

1. To create the Nios V processor inside the FPGA device, program the `.sof` file onto the board with the following command.

Table 34. Command

Operating System	Command
Windows	`quartus_pgm -c 1 -m JTAG -o p;top.sof@1`
Linux	`quartus_pgm -c 1 -m JTAG -o p\;top.sof@1`

Note:
- -c 1 is referring to cable number connected to the Host Computer.
- @1 is referring to device index on the JTAG Chain and may differ for your board.

2. Download the `.elf` using the `niosv-download` command.

```
niosv-download -g <elf file>
```

3. Use the JTAG UART terminal to print the stdout and stderr of the Nios V processor system.

```
juart-terminal
```

5.6. Operating the Example Designs

5.6.1. Operating the MicroC/TCP-IP IPerf

To display the µC/TCP-IP IPerf application messages, the example design utilizes the JTAG UART Intel FPGA IP. You can begin the display message by using the following command:

```
juart-terminal
```

The JTAG UART terminal displays the booting message logs, followed by the µC/TCP-IP setup logs from the µC/TCP-IP IPerf example design.

Within the µC/TCP-IP setup logs, there is a message stating the current IP address adopted by the system (as configured in `main.c` source code). If DHCP is enabled, the DHCP server-supplied IP address displays the message that indicates the DHCP client for the Ethernet interface acquires a DHCP IP address.

The message "`TEST ID : <test number>`" is displayed along with its IP address and other information, when the µC/TCP-IP IPerf server is initialized and ready for connection.

After the iPerf server is ready, you can start an iPerf client on your host computer to interact with the iPerf server. To start the iPerf client, follow these steps:

1. From your operating system, open a command shell or a terminal.

   Note: On Windows, you can also use **Run** on the Start menu.

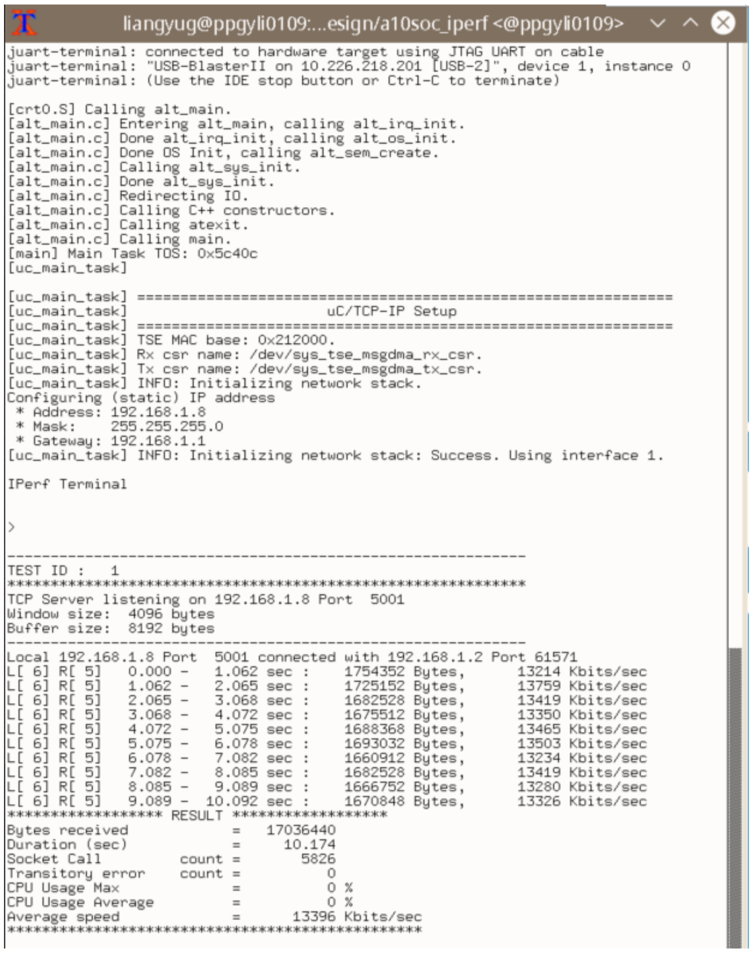
2. Type the following command, specifying either the static IP address or the DHCP server-provided IP address:

```
iperf -c <IP Address>
```

If the connection to the development board is successful, the iPerf test result displays in both the iPerf server and client.

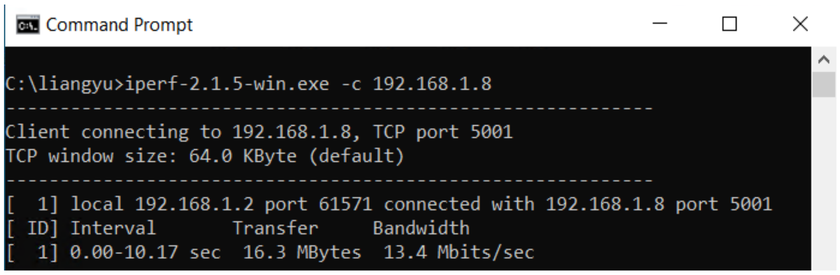
In the following examples, the configured IP address for the iPerf server is 192.168.1.45 at port 5001.

Figure 98. iPerf Server on Intel FPGA device



```
juart-terminal: connected to hardware target using JTAG UART on cable
juart-terminal: "USB-BlasterII on 10.226.218.201 [USB-2]", device 1, instance 0
juart-terminal: (Use the IDE stop button or Ctrl-C to terminate)

[crt0.S] Calling alt_main.
[alt_main.c] Entering alt_main, calling alt_irq_init.
[alt_main.c] Done alt_irq_init, calling alt_os_init.
[alt_main.c] Done OS Init, calling alt_sem_create.
[alt_main.c] Calling alt_sys_init.
[alt_main.c] Done alt_sys_init.
[alt_main.c] Redirecting IO.
[alt_main.c] Calling C++ constructors.
[alt_main.c] Calling atexit.
[alt_main.c] Calling main.
[main] Main Task TOS: 0x5c40c
[uc_main_task]

[uc_main_task] ==============================================================
[uc_main_task]                       uC/TCP-IP Setup
[uc_main_task] ==============================================================
[uc_main_task] TSE MAC base: 0x212000.
[uc_main_task] Rx csr name: /dev/sys_tse_msgdma_rx_csr.
[uc_main_task] Tx csr name: /dev/sys_tse_msgdma_tx_csr.
[uc_main_task] INFO: Initializing network stack.
Configuring (static) IP address
 * Address: 192.168.1.8
 * Mask:    255.255.255.0
 * Gateway: 192.168.1.1
[uc_main_task] INFO: Initializing network stack: Success. Using interface 1.

IPerf Terminal


>

------------------------------------------------------------
TEST ID :   1
************************************************************
TCP Server listening on 192.168.1.8 Port  5001
Window size:  4096 bytes
Buffer size:  8192 bytes
------------------------------------------------------------
Local 192.168.1.8 Port  5001 connected with 192.168.1.2 Port 61571
L[ 6] R[ 5]   0.000 -   1.062 sec :    1754352 Bytes,      13214 Kbits/sec
L[ 6] R[ 5]   1.062 -   2.065 sec :    1725152 Bytes,      13759 Kbits/sec
L[ 6] R[ 5]   2.065 -   3.068 sec :    1682528 Bytes,      13419 Kbits/sec
L[ 6] R[ 5]   3.068 -   4.072 sec :    1675512 Bytes,      13350 Kbits/sec
L[ 6] R[ 5]   4.072 -   5.075 sec :    1688368 Bytes,      13465 Kbits/sec
L[ 6] R[ 5]   5.075 -   6.078 sec :    1693032 Bytes,      13503 Kbits/sec
L[ 6] R[ 5]   6.078 -   7.082 sec :    1660912 Bytes,      13234 Kbits/sec
L[ 6] R[ 5]   7.082 -   8.085 sec :    1682528 Bytes,      13419 Kbits/sec
L[ 6] R[ 5]   8.085 -   9.089 sec :    1666752 Bytes,      13280 Kbits/sec
L[ 6] R[ 5]   9.089 -  10.092 sec :    1670848 Bytes,      13326 Kbits/sec
****************** RESULT ******************
Bytes received              =   17036440
Duration (sec)              =     10.174
Socket Call           count =       5826
Transitory error      count =          0
CPU Usage Max               =          0 %
CPU Usage Average           =          0 %
Average speed               =      13396 Kbits/sec
************************************************
```

Figure 99. iPerf Client on Host Computer



```
C:\liangyu>iperf-2.1.5-win.exe -c 192.168.1.8
------------------------------------------------------------
Client connecting to 192.168.1.8, TCP port 5001
TCP window size: 64.0 KByte (default)
------------------------------------------------------------
[  1] local 192.168.1.2 port 61571 connected with 192.168.1.8 port 5001
[ ID] Interval       Transfer     Bandwidth
[  1] 0.00-10.17 sec  16.3 MBytes  13.4 Mbits/sec
```

Related Information

IPerf Homepage

5.6.2. Operating the MicroC/TCP-IP Simple Socket Server

To display the µC/TCP-IP Simple Socket Server application messages, the example design utilizes the JTAG UART Intel FPGA IP. You can begin the display message by using the following command:

```
juart-terminal
```

The JTAG UART terminal displays the booting message logs, followed by the µC/TCP-IP setup logs from the µC/TCP-IP Simple Socket Server example design.

Within the µC/TCP-IP setup logs, find a message stating the system adopts the current IP address (as configured in `main.c` source code). If DHCP is enabled, the DHCP server-supplied IP address displays the message that indicates the DHCP client for the Ethernet interface acquires a DHCP IP address.

The message "`[sss_task] Simple Socket Server listening on port <port number>`" is displayed when the µC/TCP-IP Stack is ready for connection.

After the µC/TCP-IP Stack is ready, you can start a telnet session to interact with the stack. To start a telnet session, follow these steps:

1. From your operating system, open a command shell or a terminal.

   Note: On Windows, you can also use **Run** on the Start menu.

2. Type the following command, specifying either the static IP address or the DHCP server-provided IP address:

   ```
   telnet <IP Address> <Port>
   ```

If the connection to the development board is successful, the menu of available commands display in a command window.

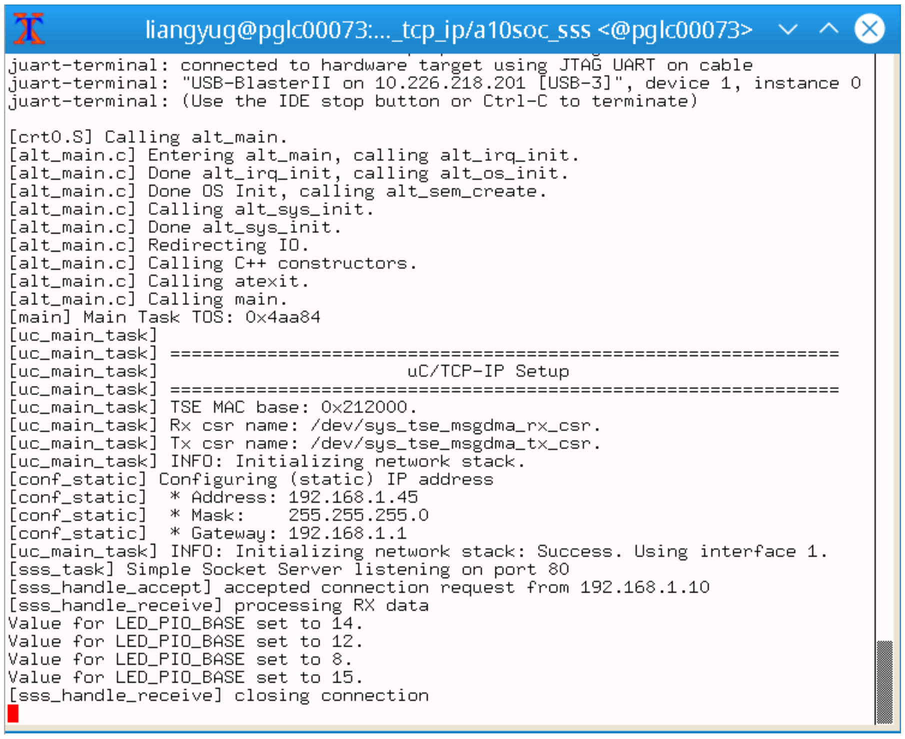
Telnet Session to Intel FPGA Device Figure shows the Nios V Simple Socket Server Menu, along with the following entered commands:

- **0 to 3**: Toggle board LEDs D0 to D3
- **S**: Board LED Light Show
- **Q**: Terminate Session

When you enter commands at the command prompt, Ethernet sends the commands over the telnet connection to a task waiting on a socket for commands. The task responds to those commands by sending instructions to another task that manipulates the LED.

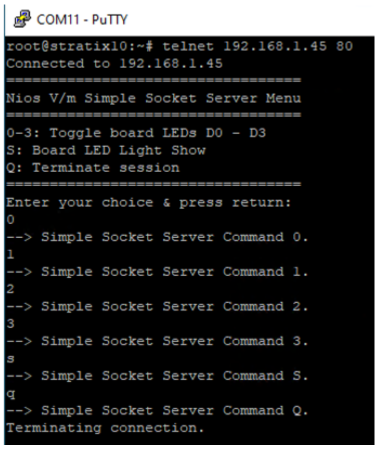
In the following examples, the configured IP address for the Simple Socket Server is 192.168.1.45 at port 80.

Figure 100. Display Message from Intel FPGA device



```
juart-terminal: connected to hardware target using JTAG UART on cable
juart-terminal: "USB-BlasterII on 10.226.218.201 [USB-3]", device 1, instance 0
juart-terminal: (Use the IDE stop button or Ctrl-C to terminate)

[crt0.S] Calling alt_main.
[alt_main.c] Entering alt_main, calling alt_irq_init.
[alt_main.c] Done alt_irq_init, calling alt_os_init.
[alt_main.c] Done OS Init, calling alt_sem_create.
[alt_main.c] Calling alt_sys_init.
[alt_main.c] Done alt_sys_init.
[alt_main.c] Redirecting IO.
[alt_main.c] Calling C++ constructors.
[alt_main.c] Calling atexit.
[alt_main.c] Calling main.
[main] Main Task TOS: 0x4aa84
[uc_main_task]
[uc_main_task] ===============================================================
[uc_main_task]                       uC/TCP-IP Setup
[uc_main_task] ===============================================================
[uc_main_task] TSE MAC base: 0x212000.
[uc_main_task] Rx csr name: /dev/sys_tse_msgdma_rx_csr.
[uc_main_task] Tx csr name: /dev/sys_tse_msgdma_tx_csr.
[uc_main_task] INFO: Initializing network stack.
[conf_static] Configuring (static) IP address
[conf_static]  * Address: 192.168.1.45
[conf_static]  * Mask:    255.255.255.0
[conf_static]  * Gateway: 192.168.1.1
[uc_main_task] INFO: Initializing network stack: Success. Using interface 1.
[sss_task] Simple Socket Server listening on port 80
[sss_handle_accept] accepted connection request from 192.168.1.10
[sss_handle_receive] processing RX data
Value for LED_PIO_BASE set to 14.
Value for LED_PIO_BASE set to 12.
Value for LED_PIO_BASE set to 8.
Value for LED_PIO_BASE set to 15.
[sss_handle_receive] closing connection
```

Figure 101. Telnet Session to Intel FPGA Device



```
root@stratix10:~# telnet 192.168.1.45 80
Connected to 192.168.1.45
=================================
Nios V/m Simple Socket Server Menu
=================================
0-3: Toggle board LEDs D0 - D3
S: Board LED Light Show
Q: Terminate session
=================================
Enter your choice & press return:
0
--> Simple Socket Server Command 0.
1
--> Simple Socket Server Command 1.
2
--> Simple Socket Server Command 2.
3
--> Simple Socket Server Command 3.
s
--> Simple Socket Server Command S.
q
--> Simple Socket Server Command Q.
Terminating connection.
```

5.7. Optional Configuration

The default configuration of the two µC/TCP-IP example designs is open for modifications. The configuration is only meant to fulfill the basic requirements to build a working µC/TCP-IP Simple Socket Server and iPerf design. This section introduces some of the common configuration that you can apply on your own designs.

After the configuration is complete, you must rebuild the uC/TCP-IP application files using the same build commands described in the Building the Application Project section.

5.7.1. Configuring Hardware Name

The global structure of type "`alt_tse_system_info`" (named "`tse_mac_device`") reflects the IP names according to the `system.h` file. If you change the default IP names or not using the default hardware project, you must update the following names in `main.c` source code. You can find the source code in the `software/apps folder`.

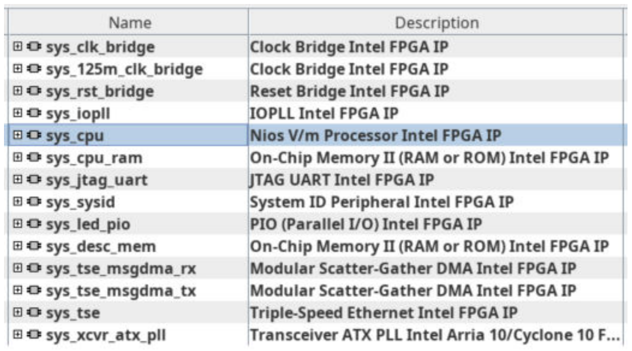
The latest IP names can be found in the `system.h` file after the hardware compilation in Intel Quartus Prime software. The header file is located in the BSP folder.

The following table lists the example design and the default IP names.

Table 35. Default IP Names

Example Design	IP Name
TSE	SYS_TSE
TX MSGDMA	SYS_TSE_MSGDMA_TX
RX MSGDMA	SYS_TSE_MSGDMA_RX
Descriptor Memory	SYS_DESC_MEM

Figure 102. Example Design Platform Designer System



Name	Description
sys_clk_bridge	Clock Bridge Intel FPGA IP
sys_125m_clk_bridge	Clock Bridge Intel FPGA IP
sys_rst_bridge	Reset Bridge Intel FPGA IP
sys_iopll	IOPLL Intel FPGA IP
sys_cpu	Nios V/m Processor Intel FPGA IP
sys_cpu_ram	On-Chip Memory II (RAM or ROM) Intel FPGA IP
sys_jtag_uart	JTAG UART Intel FPGA IP
sys_sysid	System ID Peripheral Intel FPGA IP
sys_led_pio	PIO (Parallel I/O) Intel FPGA IP
sys_desc_mem	On-Chip Memory II (RAM or ROM) Intel FPGA IP
sys_tse_msgdma_rx	Modular Scatter-Gather DMA Intel FPGA IP
sys_tse_msgdma_tx	Modular Scatter-Gather DMA Intel FPGA IP
sys_tse	Triple-Speed Ethernet Intel FPGA IP
sys_xcvr_atx_pll	Transceiver ATX PLL Intel Arria 10/Cyclone 10 F...

Example 1. Default Hardware Names in main.c

```
alt_tse_system_info tse_mac_device[MAXNETS] = {
    TSE_SYSTEM_EXT_MEM_NO_SHARED_FIFO(
        SYS_TSE,              // tse_name
        0,                    // offset
        SYS_TSE_MSGDMA_TX,    // msgdma_tx_name
        SYS_TSE_MSGDMA_RX,    // msgdma_rx_name
        TSE_PHY_AUTO_ADDRESS, // phy_addr
        NULL,                 // phy_cfg_fp
        SYS_DESC_MEM          // desc_mem_name
    )
};
```

5.7.2. Configuring MAC and IP Addresses

You can configure the MAC and IP addresses of the µC/TCP-IP module by editing the `struct network_conf conf` in `software/app/main.c` source code.

If a DHCP server is available on your network, enable the DHCP feature by modifying the `use_dhcp` field to true (`DEF_TRUE`). If the DHCP feature is enabled, the provided IP addresses, network mask, and gateway are left unused. It is not required to clear their contents.

If the development board is connected directly to your PC with a crossover Ethernet cable, or no DHCP server is available, disable the DHCP feature (`!DEF_TRUE`) and specify the IP addresses, network mask, and gateway.

Example 2. Default "struct network_conf conf" in main.c

```
struct network_conf conf = {
    .tse_sys_info = tse_sys_info,
    .mac_addr = "00:07:ed:ff:8c:05",
    .use_dhcp = !DEF_TRUE,
    .ipv4_addr_str    = "192.168.1.45",
    .ipv4_mask_str    = "255.255.255.0",
    .ipv4_gateway_str = "192.168.1.1"
};
```

Note:

- Choose your default IP and gateway addresses carefully. Some secure router configurations block DHCP request packets on local subnetworks such as the 192.168.X.X subnetwork. If you encounter problems, try using 0.0.0.0 as your default IP and gateway addresses.
- You can configure the DHCP waiting time (`DHCP_WAIT_MS`) in the `uc_tcp_ip_init.c` source code. The DHCP waiting time is the amount of time delayed before verifying that a valid IP address is acquired by the TSE IP.

5.7.3. Configuring MicroC/TCP-IP Initialization

You can configure the µC/TCP-IP application specific settings according to your preference. These settings are configurable in `software/app/uc_tcp_ip_init.c` source code.

5.7.3.1. Network Task Configuration

In this example design, the µC/TCP-IP stack has three configurable tasks, the Receive task, the Transmit De-allocation task and the Timer task. Each task is configured with its own task priority and task stack size.

In order to place a task at higher priority, you have to register it with a lower value, and vice versa. The third-party vendor recommends configuring the task priorities as listed below for optimum performance.

- Network Transmit (TX) De-allocation task (Highest priority)
- Network timer task
- Network Receive (RX) task (Lowest Priority)

As for the task stack size, it is dependent on the processor architecture and compiler used. Configuring the stack size to 4,096 bytes is deemed sufficient for most applications.

Table 36. Network task configuration

Settings	Description	Default Value
TX_TASK_PRIO	Network TX De-allocation Task Priority	1u
RX_TASK_PRIO	Network RX Task Priority	3u
TMR_TASK_PRIO	Network Timer Task Priority	5u
TX_TASK_SIZE	Network TX De-allocation Task Stack Size	4096u
RX_TASK_SIZE	Network RX Task Stack Size	4096u
TMR_TASK_SIZE	Network Timer Task Stack Size	4096u

Example 3. Default network task configuration in `uc_tcp_ip_init.c`

```
#define TX_TASK_SIZE  (4096u)
#define RX_TASK_SIZE  (4096u)
#define TMR_TASK_SIZE (4096u)

static const unsigned TX_TASK_PRIO  = 1u;
static const unsigned RX_TASK_PRIO  = 3u;
static const unsigned TMR_TASK_PRIO = 5u;
```

5.7.3.2. Network Interface Configuration

µC/TCP-IP stores received and transmitted data in network buffers (also referred as receive buffers and transmit buffers). The size of these network buffers should fulfill the minimum and maximum packet frame sizes of the network interfaces.

Based on the µC/TCP-IP application requirements, configure the network buffers to better suit your needs in `software/app/uc_tcp_ip_init.c` source code. The following are the configurable network buffers:

- Receive Large Buffer
- Transmit Large Buffer
- Transmit Small Buffer

While it is best to leave the size of large buffers at maximum frame size, you can lower the size of the small buffers to reduce the system's RAM usage, further improving the system performance. Additionally, you can configure the number of receive and transmit buffers allocated to the device. Keep in mind that you need to have at least one buffer given for each network.

After configuring the network buffers, register them to a valid memory locations, either the main system memory or a dedicated descriptor memory. If you are using a dedicated descriptor memory, define the starting address and memory span of the descriptor memory in the source code.

Refer to Network Buffers Configuration Table for the µC/TCP-IP example designs default configuration. A dedicated descriptor memory is provided in the hardware system, and registered to the receive buffers. Transmit buffers are routed to the main memory.

Note: Intel recommends receive buffers to hold up to the maximum frame size because the size of data received is unknown to the device. Alternatively, the size of data transmitted is known to the device, making it possible to use small transmit buffers when the transmitted data is smaller than the maximum frame size. Thus, allowing receive buffers to use large buffers only, while transmit buffers use both large and small buffers.

Table 37. Network Buffers Configuration

Settings	Description	Default Value
`.RxBufPoolType`	Memory location for the receive data buffers. [6][7]	`NET_IF_MEM_TYPE_DEDICATED`
`.RxBufLargeNbr`	Number of receive buffers allocated to the device.	`NUM_RX_BUFFERS`[8]
`.TxBufPoolType`	Memory location for the transmit data buffers. [6][7]	`NET_IF_MEM_TYPE_MAIN`
`.TxBufLargeNbr`	Number of transmit buffers allocated to the device.	`5u`
`.TxBufSmallSize`	Size of the small transmit buffers.	`60u`
`.TxBufSmallNbr`	Number of small transmit buffers allocated to the device.	`5u`
`.MemAddr`	Starting address of the dedicated descriptor memory.[9]	`SYS_DESC_MEM_BASE`
`.MemSize`	Size of the dedicated descriptor memory (in bytes).	`SYS_DESC_MEM_SPAN`

Example 4. Default network interface configuration in `uc_tcp_ip_init.c`

```
static const CPU_INT08U NUM_RX_LISTS = 2;
static const NET_BUF_QTY NUM_RX_BUFFERS =
    2 * NUM_RX_LISTS * ALTERA_TSE_MSGDMA_RX_DESC_CHAIN_SIZE;

static NET_DEV_CFG_ETHER NetDev_Cfg_Ether_TSE = {

    .RxBufPoolType    = NET_IF_MEM_TYPE_DEDICATED,
    .RxBufLargeSize   = 1536u,
    .RxBufLargeNbr    = NUM_RX_BUFFERS,
    .RxBufAlignOctets = 4u,
    .RxBufIxOffset    = 2u,

    .TxBufPoolType    = NET_IF_MEM_TYPE_MAIN,
    .TxBufLargeSize   = 1518u,
    .TxBufLargeNbr    = 5u,
    .TxBufSmallSize   = 60u,
    .TxBufSmallNbr    = 5u,
    .TxBufAlignOctets = 4u,
    .TxBufIxOffset    = 0u,

    .MemAddr    = SYS_DESC_MEM_BASE,
    .MemSize    = SYS_DESC_MEM_SPAN,

    .Flags =  NET_DEV_CFG_FLAG_NONE,
```

[6] This field must be set to either `NET_IF_MEM_TYPE_MAIN` for main memory or `NET_IF_MEM_TYPE_DEDICATED` for dedicated descriptor memory.

[7] Only one buffer type (receive or transmit buffers) can be set to `NET_IF_MEM_TYPE_DEDICATED`.

[8] This field is derived based on `NUM_RX_LISTS` and `ALTERA_TSE_MSGDMA_RX_DESC_CHAIN_SIZE`.

[9] If there is no dedicated descriptor memory in the system, this field should be set to `NULL`

```
    .RxDescNbr = 8u, // NOTE: Not configurable.
    .TxDescNbr = 1u, // NOTE: Not configurable.

    .BaseAddr           = 0,
    .DataBusSizeNbrBits = 0,

    .HW_AddrStr = "",
};
```

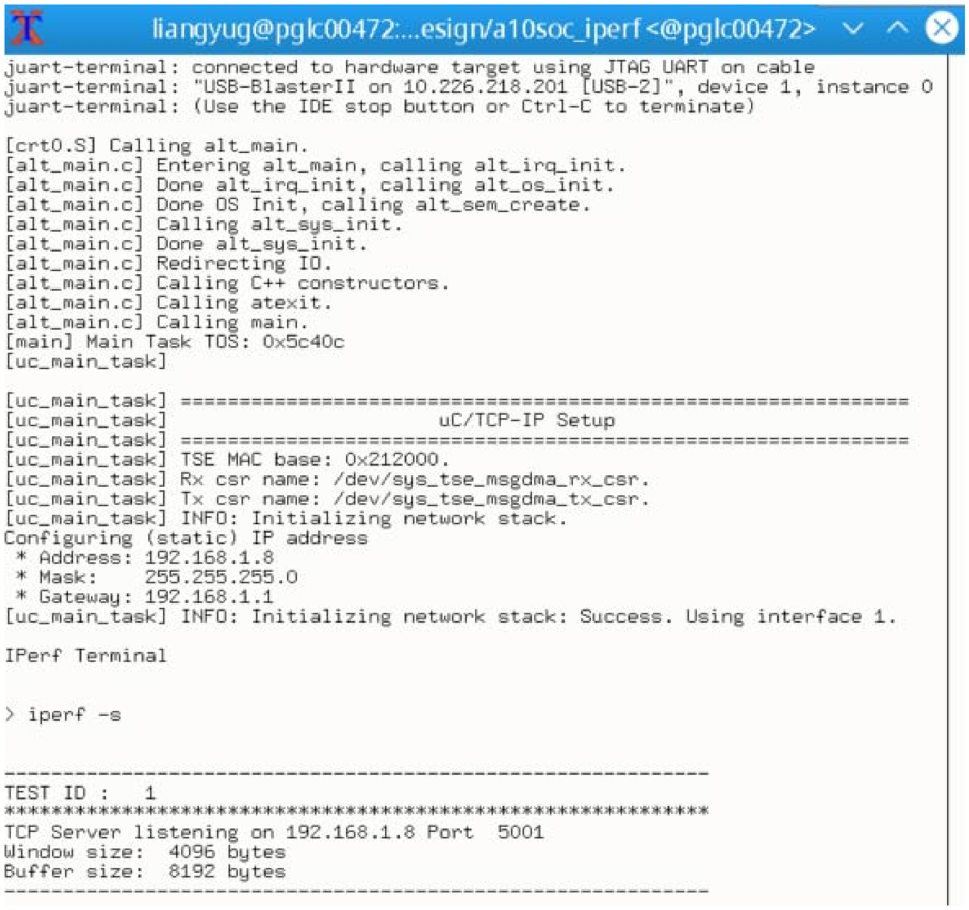
5.7.4. Configuring iPerf Server Auto-Initialization

Note: This configuration is only available for µC/TCP-IP IPerf Example Design.

The example design is capable of initializing the iPerf server using pre-determined arguments upon running the Nios V applications. This is developed for ease of use purpose, and it can be disabled if other iPerf utility is required.

To disable this feature, you need to provide the **0** argument to the `App_IPerf_TaskTerminal()`, and the iPerf terminal begins acquiring the custom `iperf` commands after iPerf is successfully initialized. The `iperf` command must end with an ENTER key to complete the acquisition process.

Figure 103. iPerf Terminal



```
juart-terminal: connected to hardware target using JTAG UART on cable
juart-terminal: "USB-BlasterII on 10.226.218.201 [USB-2]", device 1, instance 0
juart-terminal: (Use the IDE stop button or Ctrl-C to terminate)

[crt0.S] Calling alt_main.
[alt_main.c] Entering alt_main, calling alt_irq_init.
[alt_main.c] Done alt_irq_init, calling alt_os_init.
[alt_main.c] Done OS Init, calling alt_sem_create.
[alt_main.c] Calling alt_sys_init.
[alt_main.c] Done alt_sys_init.
[alt_main.c] Redirecting IO.
[alt_main.c] Calling C++ constructors.
[alt_main.c] Calling atexit.
[alt_main.c] Calling main.
[main] Main Task TOS: 0x5c40c
[uc_main_task]

[uc_main_task] ==============================================================
[uc_main_task]                         uC/TCP-IP Setup
[uc_main_task] ==============================================================
[uc_main_task] TSE MAC base: 0x212000.
[uc_main_task] Rx csr name: /dev/sys_tse_msgdma_rx_csr.
[uc_main_task] Tx csr name: /dev/sys_tse_msgdma_tx_csr.
[uc_main_task] INFO: Initializing network stack.
Configuring (static) IP address
 * Address: 192.168.1.8
 * Mask:    255.255.255.0
 * Gateway: 192.168.1.1
[uc_main_task] INFO: Initializing network stack: Success. Using interface 1.

IPerf Terminal


> iperf -s


------------------------------------------------------------
TEST ID :   1
************************************************************
TCP Server listening on 192.168.1.8 Port  5001
Window size:  4096 bytes
Buffer size:  8192 bytes
------------------------------------------------------------
```

Example 5. iPerf server auto-initialization feature in `main.c`

```
//To enable auto-initialization
App_IPerf_TaskTerminal( 1 );

//To disable auto-initialization
App_IPerf_TaskTerminal( 0 );
```

5.8. MicroC/TCP-IP Simple Socket Server Concepts

5.8.1. MicroC/OS-II Resources

This section describes the tasks, queue, event flag, and semaphores that implement the µC/TCP-IP Simple Socket Server application.

Tasks

The following table lists the µC/OS-II tasks that implements the µC/TCP-IP Simple Socket Server application.

Table 38. µC/OS-II tasks for the µC/TCP-IP Simple Socket Server

Tasks	Description
`SSSCreateOSDataStructs()`	Creates an instance of all the µC/OS-II resources.
`SSSCreateTasks()`	Initializes tasks that do not use the networking services.
`SSSSimpleSocketServerTask()`	Manages the socket server connection, and calls relevant subroutines to manage the socket connection.
`LEDManagementTask()`	Manages the LEDs, driven by commands received from a µC/OS-II queue, named SSSLEDCommandQ.
`LEDLightshowTask()`	Manages the LED light show, once enabled by the `LEDManagementTask()`.

Inter-Task Communication Resources

The following global handles (or pointers) create and manipulate your µC/OS-II inter-task communication resources. All the resources begin with Simple Socket Server, indicating a public resource provided by the Nios V Simple Socket Server that is shared between software modules.

The `SSSCreateOSDataStructs()` function declares and creates these resources in `simple_socket_server.c`.

- **SSSLEDCommandQ**: A µC/OS-II queue that sends commands from the simple socket server task, `SSSSimpleSocketServerTask()` to the development board LED control task, `LEDManagementTask()`.
- **SSSLEDLightshowSem**: A µC/OS-II semaphore that is referred by the `LEDLightshowTask()` before the LEDs update.
- **SSSLEDEventFlag**: A µC/OS-II flag that corresponds to one of the LEDs.

Note: The µC/TCP-IP Simple Socket Server uses capitalized acronym prefixes to identify public resources for each software module, and lowercase letters with underscores to indicate a private resource or function used internally to a software module.

The following are the software module acronym identifiers:

- **SSS**: µC/TCP-IP Simple Socket Server software module
- **LED**: LED management software module
- **OS**: µC/OS-II RTOS software component

5.8.2. Error Handling

A suite of error-handling functions defined in alt_error_handler() check error handling of the µC/TCP-IP Simple Socket Server application, µC/TCP-IP Stack, and µC/OS-II system call error-codes. All system, socket, and application calls check for error conditions whenever an error exist.

5.8.3. MicroC/TCP-IP Stack Default Configuration

The µC/TCP-IP Stack creates one or more system level tasks during system initialization, when you call the network_init() function. Users have complete control over these system level tasks through a global configuration file named net_cfg.h, located in the directory structure for the BSP project, in the **uC-TCP-IP/uC-Conf** folder.

You can edit the #define statements in `net_cfg.h` to configure the following options for the µC/TCP-IP Stack:

- Module Inclusion: Identifies which built-in µC/TCP-IP modules should be started.
- Module Configuration: Configure how built-in µC/TCP-IP modules should be started.

Related Information

µC/TCP-IP Documentation
For more information about the µC/TCP-IP Stack configuration.

6. Nios V Processor Debugging, Verifying, and Simulating

Debugging and verifying an embedded system involves hardware and software components. To successfully debug an embedded system requires expertise in both hardware and software. This chapter helps you understand several tools and techniques that are useful in debugging, verifying, and bring up the embedded system.

6.1. Debugging Nios V/c Processor

Nios V/c processor implements the compact architecture to achieve a smaller logic size by applying the following trait:

- Non-pipelined datapath
- No debug module
- No processor CSR
- No interrupts and exceptions
- No internal timer module

The Nios V/c processor core is limited to hardware debugging without the debug module. Software debugging is not applicable for the Nios V/c processor core.

Intel recommends to use the non-pipelined Nios V/m processor to allow full debugging capabilities. The architecture performance of a non-pipelined Nios V/m processor is similar to the Nios V/c processor, at the expense of bigger logic size.

Table 39. Nios V/c and Nios V/m Processor Core

Feature	Nios V/c Processor	Non-pipelined Nios V/m Processor
Debug Module	—	Supported
Processor CSR	—	Supported
Interrupt and Exceptions	—	Supported
Logic Size (ALM)(10)	x1	x1.5
DMIPS/Mhz Performance(10)	x1	x1
CoreMark/MHz Performance(10)	x1	x1
Internal Timer	—	Supported

6.1.1. Steps to Debug Nios V/c Processor

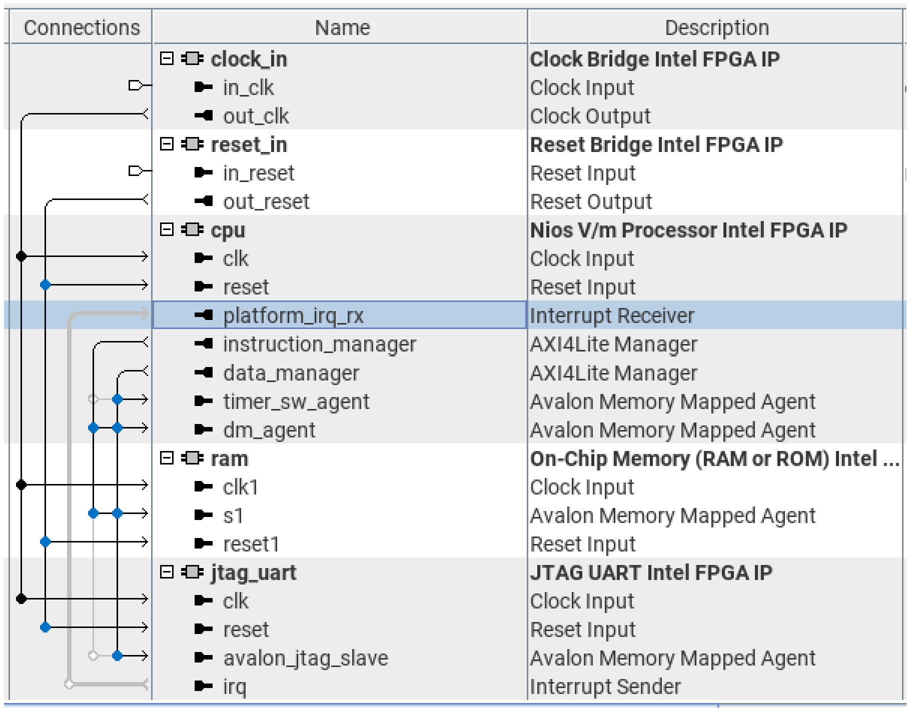
You can utilize Nios V/m processor to debug Nios V/c processor:

(10) Relative to the Nios V/c processor.

1. Start the processor system using non-pipelined Nios V/m processor.
   a. Turn on **Enable Debug**
   b. Turn off **Enable Pipelining in CPU**
2. Ensure there is no interrupt or exception in the Nios V/m processor system. Do not connect to the **Interrupt Receiver** on the processor.

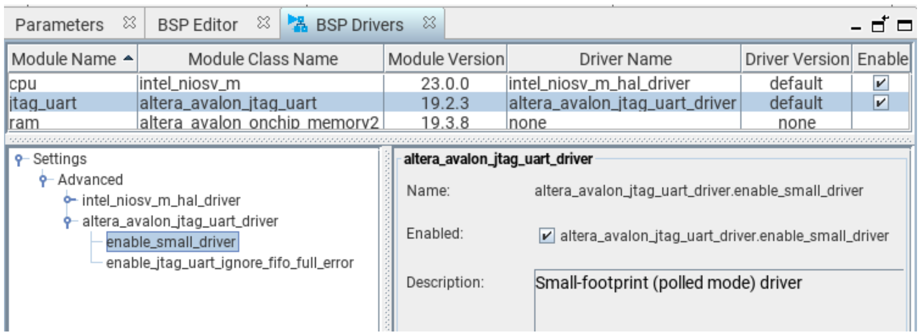
   Note: To implement a JTAG UART Intel FPGA IP without interrupt, you can enable the **small JTAG UART driver** in the BSP Editor to apply polled operation. Ensure that the compile definition (`ALTERA_AVALON_JTAG_UART_SMALL`) is found in the `ttoolchain.cmake`.

Figure 104. Nios V/m Processor System with No Interrupt



Connections	Name	Description
	clock_in	Clock Bridge Intel FPGA IP
	in_clk	Clock Input
	out_clk	Clock Output
	reset_in	Reset Bridge Intel FPGA IP
	in_reset	Reset Input
	out_reset	Reset Output
	cpu	Nios V/m Processor Intel FPGA IP
	clk	Clock Input
	reset	Reset Input
	platform_irq_rx	Interrupt Receiver
	instruction_manager	AXI4Lite Manager
	data_manager	AXI4Lite Manager
	timer_sw_agent	Avalon Memory Mapped Agent
	dm_agent	Avalon Memory Mapped Agent
	ram	On-Chip Memory (RAM or ROM) Intel ...
	clk1	Clock Input
	s1	Avalon Memory Mapped Agent
	reset1	Reset Input
	jtag_uart	JTAG UART Intel FPGA IP
	clk	Clock Input
	reset	Reset Input
	avalon_jtag_slave	Avalon Memory Mapped Agent
	irq	Interrupt Sender

Figure 105. Enable Small JTAG UART Driver in BSP Editor



Module Name	Module Class Name	Module Version	Driver Name	Driver Version	Enable
cpu	intel_niosv_m	23.0.0	intel_niosv_m_hal_driver	default	✓
jtag_uart	altera_avalon_jtag_uart	19.2.3	altera_avalon_jtag_uart_driver	default	✓
ram	altera_avalon_onchip_memory2	19.3.8	none	none	

Settings
- Advanced
  - intel_niosv_m_hal_driver
  - altera_avalon_jtag_uart_driver
    - enable_small_driver
    - enable_jtag_uart_ignore_fifo_full_error

altera_avalon_jtag_uart_driver

Name: altera_avalon_jtag_uart_driver.enable_small_driver

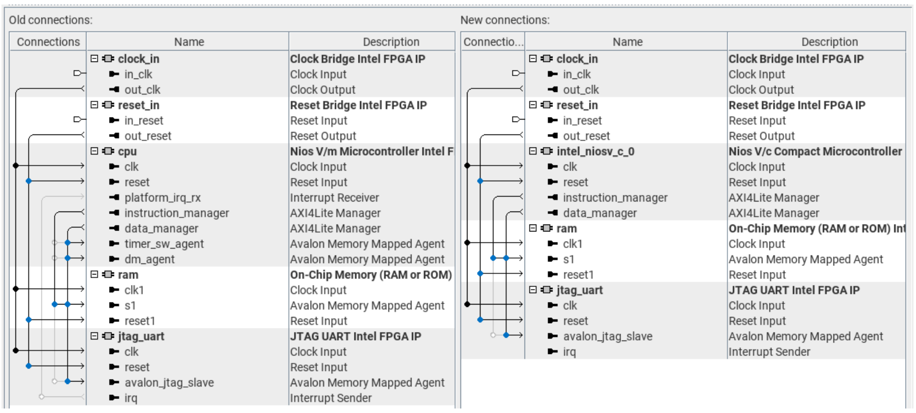
Enabled: ✓ altera_avalon_jtag_uart_driver.enable_small_driver

Description: Small-footprint (polled mode) driver

3. Develop the Nios V processor software application in baremetal (Intel HAL).
4. Program the design SOF file onto the Intel FPGA device.

5. Download the application ELF file into the Nios V processor system.
6. Perform design verification and debugging with the Nios V/m processor core.
7. Verify that the Nios V/m processor is working successfully, then replace the Nios V/m processor with Nios V/c processor.
    a. Right-click the Nios V/m processor, click **Replace ➤ Nios V/c Processor Intel FPGA IP**.
    b. Reconfigure the same assignment in the **IP Parameter Editor**.
    c. Address any possible errors.
    d. Click **Sync System Infos**.

Figure 106. Nios V/c Processor Replacement



Old connections:

Connections	Name	Description
	clock_in	**Clock Bridge Intel FPGA IP**
	in_clk	Clock Input
	out_clk	Clock Output
	reset_in	**Reset Bridge Intel FPGA IP**
	in_reset	Reset Input
	out_reset	Reset Output
	cpu	**Nios V/m Microcontroller Intel F**
	clk	Clock Input
	reset	Reset Input
	platform_irq_rx	Interrupt Receiver
	instruction_manager	AXI4Lite Manager
	data_manager	AXI4Lite Manager
	timer_sw_agent	Avalon Memory Mapped Agent
	dm_agent	Avalon Memory Mapped Agent
	ram	**On-Chip Memory (RAM or ROM)**
	clk1	Clock Input
	s1	Avalon Memory Mapped Agent
	reset1	Reset Input
	jtag_uart	**JTAG UART Intel FPGA IP**
	clk	Clock Input
	reset	Reset Input
	avalon_jtag_slave	Avalon Memory Mapped Agent
	irq	Interrupt Sender

New connections:

Connectio...	Name	Description
	clock_in	**Clock Bridge Intel FPGA IP**
	in_clk	Clock Input
	out_clk	Clock Output
	reset_in	**Reset Bridge Intel FPGA IP**
	in_reset	Reset Input
	out_reset	Reset Output
	intel_niosv_c_0	**Nios V/c Compact Microcontroller**
	clk	Clock Input
	reset	Reset Input
	instruction_manager	AXI4Lite Manager
	data_manager	AXI4Lite Manager
	ram	**On-Chip Memory (RAM or ROM) Int**
	clk1	Clock Input
	s1	Avalon Memory Mapped Agent
	reset1	Reset Input
	jtag_uart	**JTAG UART Intel FPGA IP**
	clk	Clock Input
	reset	Reset Input
	avalon_jtag_slave	Avalon Memory Mapped Agent
	irq	Interrupt Sender

8. Implement booting Nios V/c processor from On-Chip Memory.
9. Recreate the application BSP, APP, and ELF.
10. Program the memory-initialized design SOF file onto the Intel FPGA device.
11. Power cycle the Intel FPGA device.

6.2. Debugging Nios V Processor Software Designs

6.2.1. OpenOCD and Eclipse Embedded CDT

The Open On-Chip Debugger (OpenOCD) is an open-source gdb server that provides debugging, in-system programming, and boundary-scan testing for embedded target devices with hardware debugger's JTAG port. The Eclipse Embedded CDT is a tool to create, build, debug and manage RISC-V projects and application. Refer to the Related Information about how to use OpenOCD to debug Nios V processor application.

6.2.2. Objdump File

The Nios V processor build process always generate an object dump text file (`.objdump`) from your application `.elf` file. The `.objdump` file contains information about the memory sections and their layout, the addresses of functions, and the original C source code interleaved with the assembly code. The `.objdump` file generates in the `<project_directory>/software/app/build` folder .

6.2.3. Show Make Commands

The individual Makefile commands appear in the display as they run. To enable a verbose mode for the make command, execute the following build command in the Nios V Command Shell when building the application:

```
make VERBOSE=1
```

6.3. Debugging Tools

The Intel Quartus Prime software allows you to use the debugging tools to exercise and analyze the logic under test and maximize closure. Below are the list of debugging tools in the Intel Quartus Prime software:

- Signal Tap Logic Analyzer
- Logic Analyzer Interface
- Signal Probe
- In-system Sources and Probes
- Virtual JTAG Interface
- System Console
- In-System Memory Content Editor

Related Information

Intel Quartus Prime Pro Edition User Guide: Debug Tools

6.4. Additional Embedded Design Considerations

Consider the following topics as you design your system:

- JTAG signal integrity
- Additional memory space for prototyping

6.4.1. JTAG Signal Integrity

The JTAG signal integrity on your system is very important. Poor signal integrity on the JTAG interface can prevent you from debugging over the JTAG connection or cause inconsistent debugger behavior.

Related Information

Intel Quartus Prime Timing Analyzer Cookbook
For more information about JTAG Signals.

6.4.2. Additional Memory Space for System Prototyping

Even if your final product includes no off-chip memory, Intel recommends that your prototype board include a connection to some region of off-chip memory. This component in your system provides additional memory capacity that enables you to focus on refining code functionality without worrying about code size. Later in the design process, you can substitute a smaller memory device to store your software.

6.5. Simulating Nios V Processor Designs

This section describes the following tasks:

- Generating an RTL simulation environment with Nios V processor example designs and Platform Designer.
- Running the RTL simulation in the Questa* Intel FPGA Edition simulator.

The increasing pressure to deliver robust products to market timely has amplified the importance of comprehensively verifying embedded processor designs. Therefore, consider the verification solution supplied with the processor when choosing an embedded processor. Nios V embedded processor designs support a broad range of verification solutions, including the following:

- Board Level Verification—Intel offers several development boards that provide a versatile platform for verifying both the hardware and software of a Nios V embedded processor system. You can further debug the hardware components that interact with the processor with the Signal Tap embedded logic analyzer.
- Register Transfer Level (RTL) Simulation—RTL simulation is a powerful means of debugging the interaction between a processor and its peripheral set. When debugging a target board, it is often difficult to view signals buried deep in the system. RTL simulation alleviates this problem by enabling you to probe every register and signal in the design. You can easily simulate Nios V based systems in the Questa Intel FPGA Edition simulator with an automatically generated simulation environment, that is Platform Designer.

Note: Due to a limitation with embedded memory blocks, the simulation model of Nios V processor does not support ECC on Intel Arria 10 devices.

Related Information

Embedded Memory (RAM: 1-PORT, RAM: 2-PORT, ROM: 1-PORT, and ROM: 2-PORT) User Guide

6.5.1. Prerequisites

You must have experience using Platform Designer and are familiar with the QuestaSim simulator. To simulate the Nios V processor design using the instructions in this handbook, you must install the following softwares:

- Intel Quartus Prime
- QuestaSim Simulator

6.5.2. Setting Up and Generating Your Simulation Environment in Platform Designer

To generate simulation files, perform the following steps:

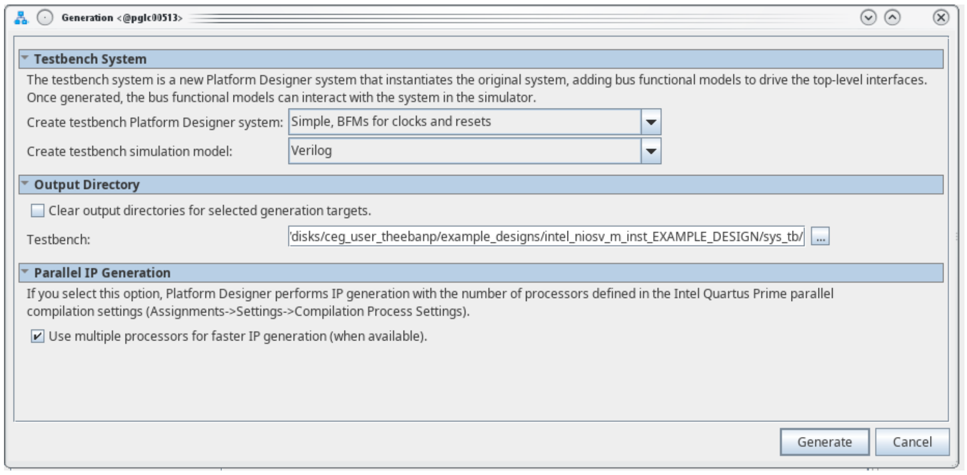
1. Start the **Intel Quartus Prime software** and open the **Platform Designer** from the **Tools** menu.
2. Open the `<your project design>.qsys` file.

Note: Ensure that you have completed building your Platform Designer system before generating the simulation models

3. In **Platform Designer**, navigate to **Generate ➤ Generate Testbench System**.
4. On the **Generation** window, set the following parameters to these values:
   a. Create testbench Platform Designer system— **Standard, BFMs for standard Platform Designer interfaces.**

      Note: If your system has exported ports other than the clock and reset, choose Standard, BFMs for standard Avalon interfaces.
   b. Create testbench simulation model—**Verilog**
   c. Select **Use multiple processors for faster IP generation (when available)**.
5. Click **Generate**, and **Save**, if prompted.

Figure 107. Testbench Generation


Generation <@pglc00513>

Testbench System

The testbench system is a new Platform Designer system that instantiates the original system, adding bus functional models to drive the top-level interfaces. Once generated, the bus functional models can interact with the system in the simulator.

Create testbench Platform Designer system: Simple, BFMs for clocks and resets

Create testbench simulation model: Verilog

Output Directory

☐ Clear output directories for selected generation targets.

Testbench: 'disks/ceg_user_theebanp/example_designs/intel_niosv_m_inst_EXAMPLE_DESIGN/sys_tb/

Parallel IP Generation

If you select this option, Platform Designer performs IP generation with the number of processors defined in the Intel Quartus Prime parallel compilation settings (Assignments->Settings->Compilation Process Settings).

☑ Use multiple processors for faster IP generation (when available).

6.5.2.1. Using IP and Platform Designer Simulation Setup Scripts

Intel IP cores and Platform Designer systems generate simulation setup scripts. Modify these scripts to set up supported simulators. The script, `msim_setup.tcl` is located in the path: `<project_directory>/sys_tb/sim/mentor`.

6.5.3. Creating Nios V Processor Software

6.5.3.1. Generating the Board Support Package

Generate the BSP file using the following steps:

1. Launch the Nios V Command Shell
2. Based on your Intel Quartus Prime version, execute the following command to generate the BSP file. Select the **type** as **hal** or **ucosii**.

- The command for Intel Quartus Prime Pro Edition:

```
niosv-bsp -c --quartus-project=top.qpf --qsys=sys.qsys \
--type=<hal, ucosii, or freertos> software/bsp/settings.bsp
```

- The command for Intel Quartus Prime Standard Edition:

```
niosv-bsp -c --quartus-project=hw/top.qpf --sopcinfo=hw/sys.sopcinfo \
--type=<hal, ucosii, or freertos> software/bsp/settings.bsp
```

6.5.3.2. Generating the Application Project File

Generate the application file using the following steps:

1. Launch the Nios V Command Shell.
2. Execute the command below to generate an application `CMakeLists.txt`.

```
niosv-app --bsp-dir=software/bsp --app-dir=software/app \
--srcs=software/app/<source code 1> \
--srcs=software/app/<source code 2>
```

6.5.3.3. Building the Application Project

You can choose to build the application project using the RiscFree IDE for Intel FPGAs, Eclipse Embedded CDT or through the command line interface (CLI).

If you prefer using CLI, you can build the application using the following command:

```
cmake -G "Unix Makefiles" -DCMAKE_BUILD_TYPE=Debug -B \
software/app/build -S software/app
```

```
make -C software/app/build
```

The application (`.elf`) file is created in `software/app/build` folder.

6.5.4. Generating Memory Initialization File

To generate application image .hex file using the elf2hex command:

```
elf2hex <elf input file> -b <On-chip memory start address> -w <On-chip memory
data width in bits> -e < On-chip memory end address> -r 4 <hex output file>
```

```
e.g. elf2hex software/app/build/hello.elf -b 0x0 -w 32 -e 0x4FFFF -r 4 -o
ram.hex
```

6.5.5. Generating System Simulation Files

At this point in the design flow, you have generated your system and created all the files necessary for simulation listed in the table below. These are the necessary files required to run the simulation.

Table 40. Files Generated for Nios V Processor Simulation

File	Description
`<Project directory>/sys_tb/`	Platform Designer generates a testbench system when you enable the **Create testbench Platform Designer** system option. Platform Designer connects the corresponding Avalon Bus Functional Models to all exported interfaces of your system. For more information about Platform Designer, refer to the *Intel Quartus Prime Pro Edition User Guide: Platform Designer*.
`<Project directory>/sys_tb/sys_tb/sim/mentor/msim_setup.tcl`	Sets up a QuestaSim simulation environment and creates alias commands to compile the required device libraries and system design files in the correct order and loads the toplevel design for simulation.
`<Project directory>/<Memory Initialization Files>.hex`	Memory Initialization Files (`.hex`) is required to initialize memory components in your system. Use elf2hex utility to create Nios V processor program to populate the `.hex` file.

Related Information

Intel Quartus Prime Pro Edition User Guide: Platform Designer

6.5.6. Running Simulation in the QuestaSim Simulator Using Command Line

You can launch the QuestaSim simulator using command `vsim`, in the Nios V Command Shell. The `msim_setup.tcl` script in the package generated creates alias commands for each step. For the list of commands, refer to the following table:

Macros	Description
dev_com	Compile device library files.
com	Compiles the design files in correct order.
elab	Elaborates the top-level design.
elab_debug	Elaborates the top-level design with the novopt option.
ld	Compiles all the design files and elaborates the top-level design.
ld_debug	Compiles all the design files and elaborates the top-level design with the **vopt** option.

Note: The **vopt** option is to run optimization before elaborating the top-level design in the simulator.

You can run the simulation in the QuestaSim simulator by performing the following steps.

1. In the transcript window, change your working directory to mentor by using the following command.
   ```
   cd <Project directory>/sys_tb/sys_tb/sim/mentor
   ```
2. Copy the memory initialization file generated into the current path (Mentor folder)
   `file copy -force <Project directory>/ram.hex ./`
3. Run the `msim_setup.tcl` by using the following command.
   ```
   do msim_setup.tcl
   ```

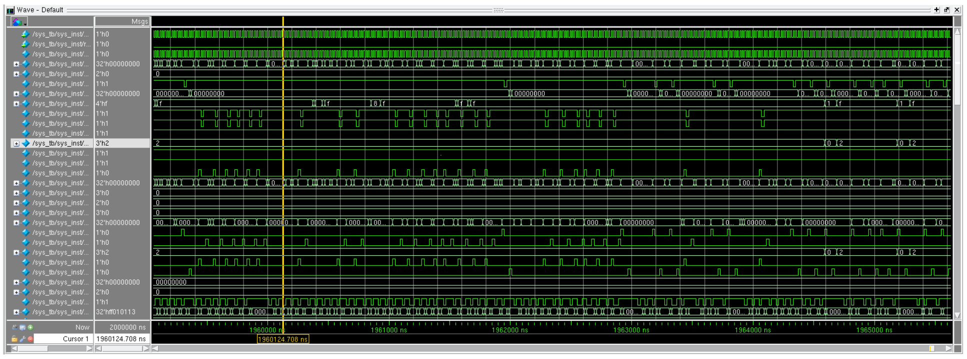
4. Compiles all the design files and elaborates the top-level design with **vopt** option by using the following command

```
ld_debug
```

5. Type `run 2ms` to start the simulation for 2 milliseconds.

At the end of the simulation, “Hello world, this is the Nios V/m cpu checking in …” message prints in the Transcript window. You can observe the simulation results from the waveform viewer as well. The following figure shows the simulation result.

Figure 108. Simulation Result



7. Nios V Processor — Remote System Update

7.1. Overview

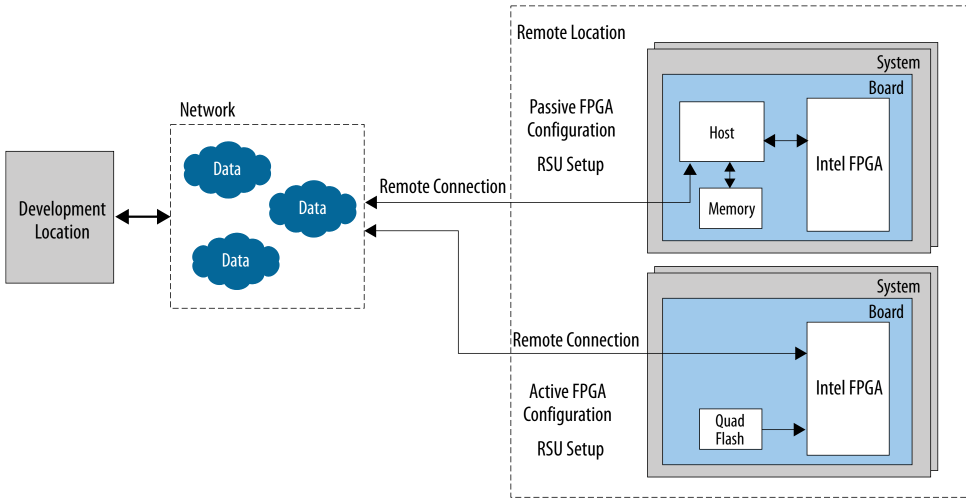
Intel FPGA devices support Remote System Update (RSU) feature to allow you to update the FPGA image and reconfigure the device remotely. RSU has the following advantages:

- Provides a mechanism to deliver feature enhancements and bug fixes without recalling your products
- Reduces time-to-market
- Extends product life

In control block-based devices[11], you need the Remote Update Intel FPGA IP to implement the RSU. Refer to *Remote Update Intel FPGA IP User Guide* for more information.

In SDM-based devices[11], you can write configuration bitstreams to the configuration flash device using RSU and Mailbox Client Intel FPGA IP. A single configuration device can store multiple application images and a single factory image. After that, you can perform FPGA reconfiguration from the RSU image through a host. The RSU can implement JTAG-to-Avalon® Master Bridge IP, Nios V processor, or Hard Processor System (HPS) as the RSU host.

Figure 109. Typical Remote System Update Process



(11) Refer to *AN 980: Nios V Processor Intel Quartus Prime Software Support* for the device list.

Related Information

- Intel Stratix 10 Configuration User Guide: Remote System Update
  Functional description on RSU and implementing RSU feature with JTAG-to-Avalon Master Bridge IP in Intel Stratix 10 devices
- Intel Agilex® 7 Configuration User Guide: Remote System Update
  Functional description on RSU and implementing RSU feature with JTAG-to-Avalon Master Bridge IP in Intel Agilex® 7 devices.
- Mailbox Client Intel FPGA IP User Guide
  More information about the LibRSU HAL API that performs RSU operations in SDM-based devices (Intel Stratix 10 and Intel Agilex® 7 devices).
- Intel Stratix 10 Hard Processor System Remote System Update User Guide
  More information about using the HPS to drive RSU in Intel Stratix 10 SoC devices.
- Intel Agilex® 7 Hard Processor System Remote System Update User Guide
  More information on using the HPS to drive RSU in Intel Agilex® 7 SoC devices.
- Remote Update Intel FPGA IP User Guide
  Functional description on implementing Remote Update Intel FPGA IP in control block-based devices (Intel Cyclone® 10 GX and Intel Arria 10 devices).
- AN 980: Nios V Processor Intel Quartus Prime Software Support

7.2. Intel Quartus Prime Pro Edition Software and Tool Support

7.2.1. Intel® Quartus® Prime Pro Edition Software

You must use the Intel Quartus Prime Pro Edition software to compile the hardware projects for remote system update in SDM-based devices.

7.2.1.1. Setting Max Retry Parameter

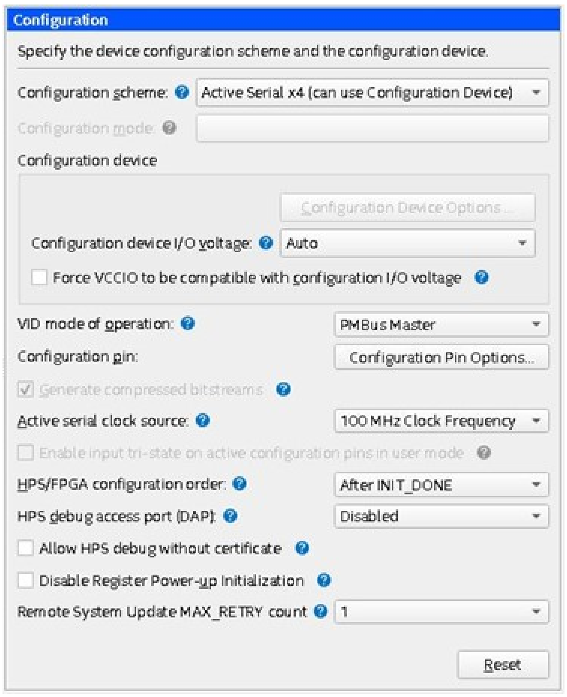
The `max retry` parameter specifies how many times the application and factory images are tried when configuration failures occur.

- The default value is one, which means each image is tried only once.
- The maximum possible value is three, which means each image can be tried up to three times.

The `max retry` parameter is stored in the decision firmware data area. The decision firmware data can also be updated by a decision firmware update image, or by a combined application image.

The max retry parameter is specified for the hardware project used to create the factory image, from the Intel Quartus Prime GUI by navigating to **Assignments ➤ Device ➤ Device and Pin Options ➤ Configuration** and selecting the value for the **Remote System Update MAX_RETRY count** field.

Figure 110. Configuration Window


Configuration

Specify the device configuration scheme and the configuration device.

Configuration scheme: Active Serial x4 (can use Configuration Device)

Configuration mode:

Configuration device

Configuration Device Options...

Configuration device I/O voltage: Auto

☐ Force VCCIO to be compatible with configuration I/O voltage

VID mode of operation: PMBus Master

Configuration pin: Configuration Pin Options...

✓ Generate compressed bitstreams

Active serial clock source: 100 MHz Clock Frequency

☐ Enable input tri-state on active configuration pins in user mode

HPS/FPGA configuration order: After INIT_DONE

HPS debug access port (DAP): Disabled

☐ Allow HPS debug without certificate

☐ Disable Register Power-up Initialization

Remote System Update MAX_RETRY count 1

You can also specify the parameter directly by editing the project Intel Quartus Prime settings file (.qsf) and adding the following line or changing the value if it is already there:

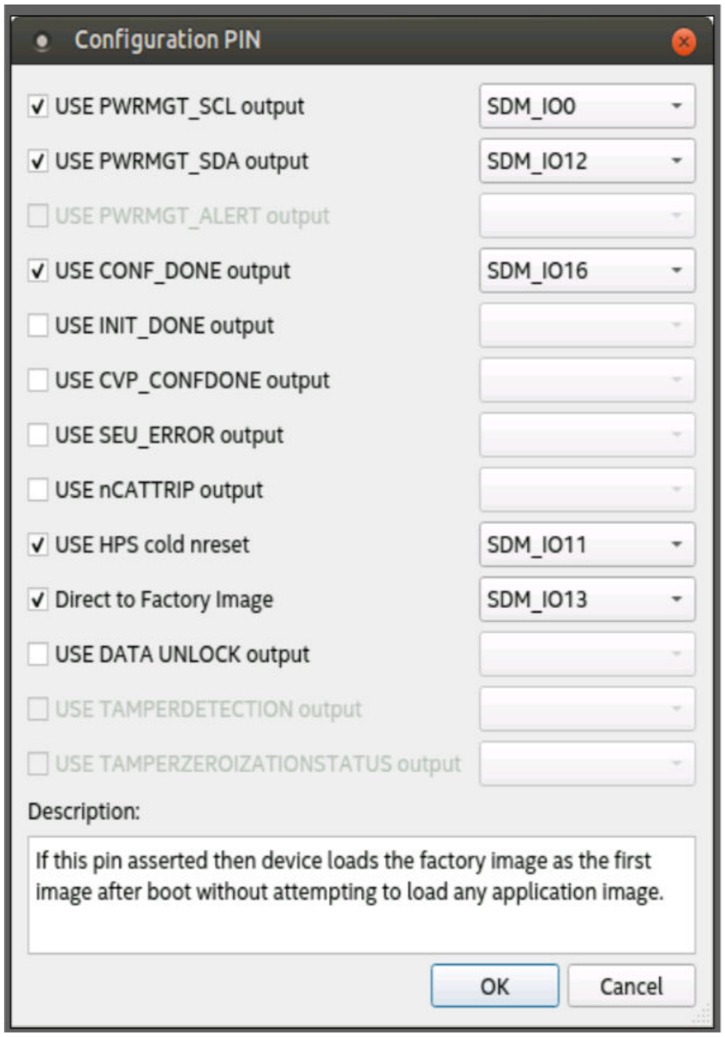
```
set_global_assignment -name RSU_MAX_RETRY_COUNT 3
```

7.2.1.2. Selecting Factory Load Pin

Intel Quartus Prime software offers the option to select the pin to use to force the factory application to load on a reset.

1. Navigate to **Assignments ➤ Device ➤ Device and Pin Options ➤ Configuration ➤ Configuration Pin Options**.
2. Check the **Direct to Factory Image** check box.
3. Select the desired pin from the drop box.

Figure 111. Configuration PIN GUI



✓ USE PWRMGT_SCL output	SDM_IO0
✓ USE PWRMGT_SDA output	SDM_IO12
USE PWRMGT_ALERT output	
✓ USE CONF_DONE output	SDM_IO16
USE INIT_DONE output	
USE CVP_CONFDONE output	
USE SEU_ERROR output	
USE nCATTRIP output	
✓ USE HPS cold nreset	SDM_IO11
✓ Direct to Factory Image	SDM_IO13
USE DATA UNLOCK output	
USE TAMPERDETECTION output	
USE TAMPERZEROIZATIONSTATUS output	

Description:

If this pin asserted then device loads the factory image as the first image after boot without attempting to load any application image.

7.2.2. Programming File Generator

Intel Quartus Prime Programming File Generator, is part of Intel Quartus Prime Pro Edition software. The tool creates programming files for all RSU scenarios as follows:

- Initial flash images
- Application images
- Factory update images
- Decision firmware update images
- Combined application images

Related Information

- Intel Quartus Prime Pro Edition User Guide: Programmer
- Intel Quartus Prime Standard Edition User Guide: Platform Designer

7.2.2.1. Programming File Generator File Types

The following table lists the most important file types created by the Programming File Generator for RSU:

Table 41. File Extension

File Extension	File Type	Description
`.jic`	JTAG Indirect Configuration File	These files are intended to be written to the flash by using the Intel Quartus Prime Programmer tool. They contain the actual flash data, and also a flash loader, which is a small FPGA design used by the Intel Quartus Prime Programmer to write the data.
`.rpd`	Raw Programming Data File	These files contain actual binary content for the flash and no additional metadata. They can contain the full content of the flash, similar with the `.jic` file—this is typically used in the case where an external tool is used to program the initial flash image. They can also contain an application image, or a factory update image.
`.map`	Memory Map File	These files contain details about where the input data was placed in the output file. This file is human readable.
`.rbf`	Raw Binary File	These files are binary files which can be used typically to configure the FPGA fabric for HPS first use cases. They can also be used for passively configuring the FPGA device through Avalon streaming interface, but that is not supported with RSU.

7.2.2.2. Bitswap Option

The Intel Quartus Prime Programmer assumes by default that the binary files have the bits in the reversed order for each byte. Because of this, you need to enable the **bitswap=on** option as follows:

- For each input binary file (`.bin` and `.hex` files are supported).
- For each output RPD file:
  - — Full flash images
  - — Application images
  - — Factory update images
  - — Decision firmware update images
  - — Combined application images

You can use the **bitswap** option following the examples presented in this document.

7.2.2.3. Intel Quartus Prime Programmer

Use the Intel Quartus Prime Programmer to program the initial flash image.

7.2.2.4. Supported QSPI Flash Devices

For a list of supported QSPI Flash Devices, refer to the *Intel® Supported Configuration Devices*.

Related Information

Device Configuration - Support Center

7.3. Nios V Processor RSU Quick Start Guide in SDM-based Devices

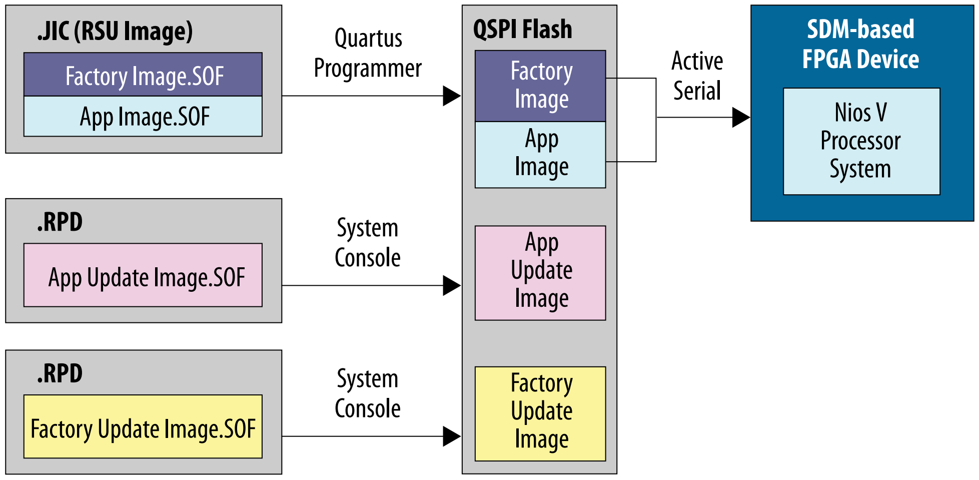
You can perform the remote system update in the SDM-based devices using the Nios V processor system. The example demonstrates the following operations:

Table 42. Remote System Update Example using Nios V Processor System

Operation	Supported Image
Creating the flash images	• Initial RSU image (containing bitstreams for factory and application image) • Application update image • Factory update image
Reconfiguring the FPGA device	• Factory image • Application image
Updating the RSU flash image	• Factory update image • Application update image

The block diagram below shows the processor system along with the configuration QSPI flash layout. Intel builds the system using the Intel Stratix 10 SX SoC L-Tile development kit. The Nios V processor boots the processor software from the memory-initialized on-chip memory.

Figure 112. Remote System Update Example Design



In a normal RSU use case, each image can be unique and performs different functions. The initial RSU JIC image contains the factory image and application image, while the update images are generated as `.rpd` file.

The Nios V processor system is incapable of performing the device reconfiguration directly with both update images because they are not registered within the initial RSU image. To perform the RSU image update, the processor reads and writes the update images into the initial RSU image and then initiates the device reconfiguration.

Note: Besides storing the updated images in a non-volatile flash, they can be transferred to the processor system through a network.

7.3.1. Individual Factory, Application, and Update Images

The example requires four images to demonstrate the RSU feature. You can modify a Nios V processor project and create four different systems with distinctive functions. However, you need to perform multiple compilation to achieve that.

To simplify the build flow, this example implements two processor systems (factory and application system) and makes three copies of the latter `.SOF` file and named them respectively as below:

- `factory.sof` (Factory Image .SOF)
- `application-0.sof` (App Image .SOF)
- `application-1.sof` (App Update Image .SOF)
- `application-2.sof` (Factory Update Image .SOF)

Even if the application images contain the same bitstreams, you can identify the images using the RSU status log.

Table 43. Nios V Processor System Details

System	Factory	Application
Platform Designer System	To create a Platform Designer system, follow the steps in section *Hardware Design Flow* with OCRAM size of 6 Mbytes.	To create a Platform Designer system, follow the steps in section *Hardware Design Flow* with OCRAM size of 1 Mbytes.
Board Support Package	Apply the BSP settings using the steps in section *Software Design Flow*.	
Nios V Processor Source Code	Uses `factory.c` that features basic RSU operations from *Example source code for Nios V Processor LibRSU application*. Refer to the link in Related Information.	Uses `application.c` that features a simplified RSU operations from *Example source code for Nios V Processor LibRSU application*. Refer to the link in Related Information.
Processor Boot Method	Software boots from OCRAM.	
Image	• Factory Image .SOF	• App Image .SOF • App Update Image .SOF • Factory Update Image .SOF

Related Information

- Example source code for Nios V Processor LibRSU application
- Hardware Design Flow on page 145
- Software Design Flow on page 148

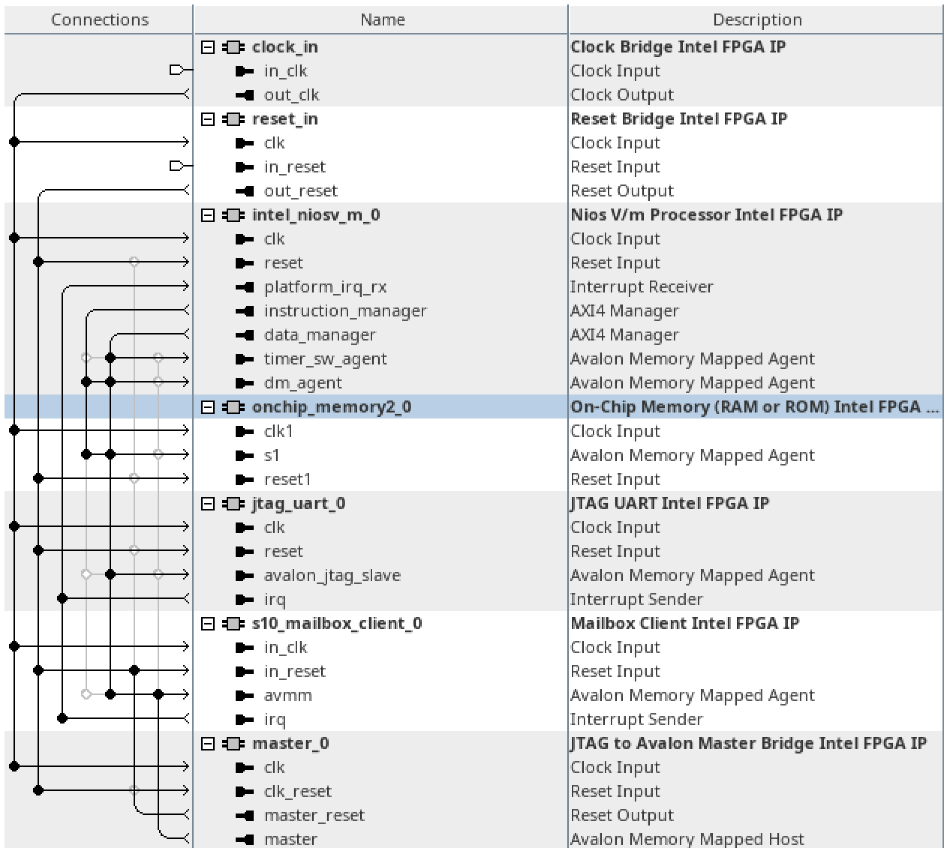
7.3.2. Hardware Design Flow

7.3.2.1. Create a Platform Designer System

1. Add the Nios V processor and the following peripherals into the Platform Designer system:

- Nios V/m Processor Intel FPGA IP
- On-Chip Memory (RAM) Intel FPGA IP
- JTAG UART Intel FPGA IP
- Mailbox Client Intel FPGA IP
- JTAG to Avalon Master Bridge Intel FPGA IP

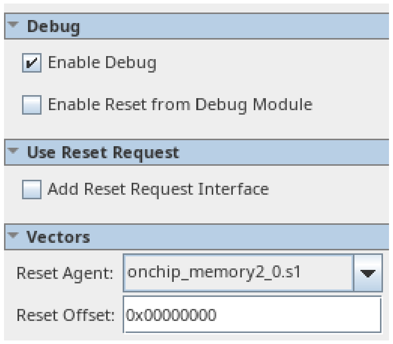
Figure 113. Connections in Platform Designer System



Connections	Name	Description
	clock_in	**Clock Bridge Intel FPGA IP**
	in_clk	Clock Input
	out_clk	Clock Output
	reset_in	**Reset Bridge Intel FPGA IP**
	clk	Clock Input
	in_reset	Reset Input
	out_reset	Reset Output
	intel_niosv_m_0	**Nios V/m Processor Intel FPGA IP**
	clk	Clock Input
	reset	Reset Input
	platform_irq_rx	Interrupt Receiver
	instruction_manager	AXI4 Manager
	data_manager	AXI4 Manager
	timer_sw_agent	Avalon Memory Mapped Agent
	dm_agent	Avalon Memory Mapped Agent
	onchip_memory2_0	**On-Chip Memory (RAM or ROM) Intel FPGA ...**
	clk1	Clock Input
	s1	Avalon Memory Mapped Agent
	reset1	Reset Input
	jtag_uart_0	**JTAG UART Intel FPGA IP**
	clk	Clock Input
	reset	Reset Input
	avalon_jtag_slave	Avalon Memory Mapped Agent
	irq	Interrupt Sender
	s10_mailbox_client_0	**Mailbox Client Intel FPGA IP**
	in_clk	Clock Input
	in_reset	Reset Input
	avmm	Avalon Memory Mapped Agent
	irq	Interrupt Sender
	master_0	**JTAG to Avalon Master Bridge Intel FPGA IP**
	clk	Clock Input
	clk_reset	Reset Input
	master_reset	Reset Output
	master	Avalon Memory Mapped Host

2. In the Nios V processor **Parameters** tab
   - Enable the **Enable Debug** feature.
   - Set the **Reset Agent** to OCRAM.

Figure 114. Nios V Processor Intel FPGA IP Parameter Editor

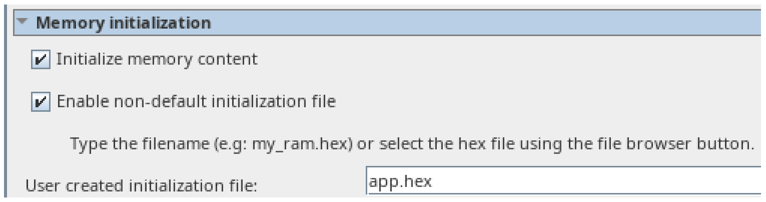

Debug
- [✓] Enable Debug
- [ ] Enable Reset from Debug Module

Use Reset Request
- [ ] Add Reset Request Interface

Vectors
- Reset Agent: onchip_memory2_0.s1
- Reset Offset: 0x00000000

3. In the On-Chip Memory (RAM or ROM) Intel FPGA Parameters tab **Total memory size** box, specify the memory size as below:
    - 1 Mbytes for application system
    - 6 Mbytes for factory system.
4. Enable **Initialize memory content** and **Enable non-default initialization file** with `app.hex` in the OCRAM.

Figure 115. On-Chip Memory Intel FPGA IP Parameter Editor


Memory initialization
- [✓] Initialize memory content
- [✓] Enable non-default initialization file

Type the filename (e.g: my_ram.hex) or select the hex file using the file browser button.

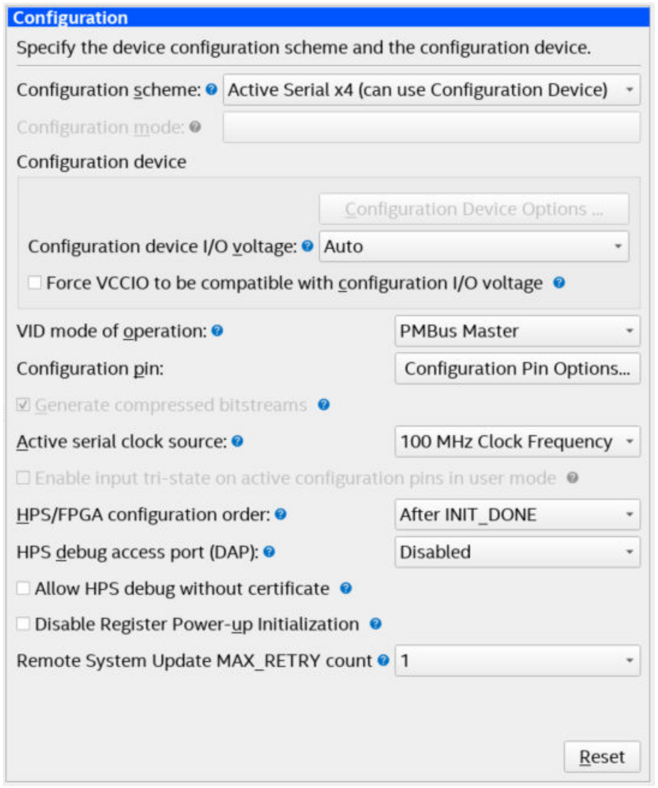
User created initialization file: app.hex

5. Click **Generate HDL**, the Generation dialog box appears.
6. Specify output file generation options and then click **Generate**.

7.3.2.2. Intel Quartus Prime Software Settings

1. In the Intel Quartus Prime software, click **Assignment ➤ Device ➤ Device and Pin Options ➤ Configuration**.
2. Set **Configuration scheme** to **Active Serial x4 (can use Configuration Device)**.
3. Set **VID mode of operation** according to your board design.
4. Set the **Active serial clock source** to **100 MHz Internal Oscillator**

Figure 116. Device and Pin Options


Configuration

Specify the device configuration scheme and the configuration device.

Configuration scheme: Active Serial x4 (can use Configuration Device)

Configuration mode:

Configuration device

Configuration Device Options ...

Configuration device I/O voltage: Auto

☐ Force VCCIO to be compatible with configuration I/O voltage

VID mode of operation: PMBus Master

Configuration pin: Configuration Pin Options...

☑ Generate compressed bitstreams

Active serial clock source: 100 MHz Clock Frequency

☐ Enable input tri-state on active configuration pins in user mode

HPS/FPGA configuration order: After INIT_DONE

HPS debug access port (DAP): Disabled

☐ Allow HPS debug without certificate

☐ Disable Register Power-up Initialization

Remote System Update MAX_RETRY count 1

Reset

5. Click **OK** to exit the **Device and Pin Options** window.
6. Click **OK** to exit the **Device** window.
7. Click **Start Compilation** to compile your project.

7.3.2.3. Software Design Flow

Creating a Nios V processor software image for RSU consists of the following general steps:

1. Generate the ZLIB libraries.
2. Create a board support package (BSP) project.
3. Creating a Nios V processor application project.
4. Building the application project using the provided source codes.
5. Running and debugging the application project.

To ensure a streamline build flow, Intel encourages you to create a similar directory tree in your design project. The following software design flow is based on this directory tree.

To create the software project directory tree, follow these steps:

1. In your design project folder, create a new folder named `software`.
2. In the `software` folder, create another two folders named `app` and `bsp`.

7.3.2.3.1. Generating the ZLIB libraries

The LibRSU HAL API requires the ZLIB libraries. Follow these steps to generate the ZLIB libraries:

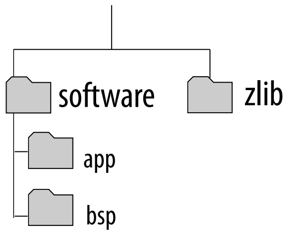
1. Acquire the latest version number of ZLIB libraries from the ZLIB home page.
2. Navigate to the design project folder.
3. Copy the following code and replace `<version>` with the latest version number:

```
wget http://zlib.net/zlib-<version>.tar.gz
tar xf zlib-<version>.tar.gz
mv zlib-<version> zlib
```

4. Run the command.
5. The `zlib` folder is ready with the ZLIB libraries.

   Note: One of the LibRSU software component, `librsu_ll_qspi.c` includes the ZLIB libraries under a specific path. If the project directory tree is different than the following figure, modify the ZLIB libraries path in `librsu_ll_qspi.c`.

Figure 117. Software Project Directory Tree

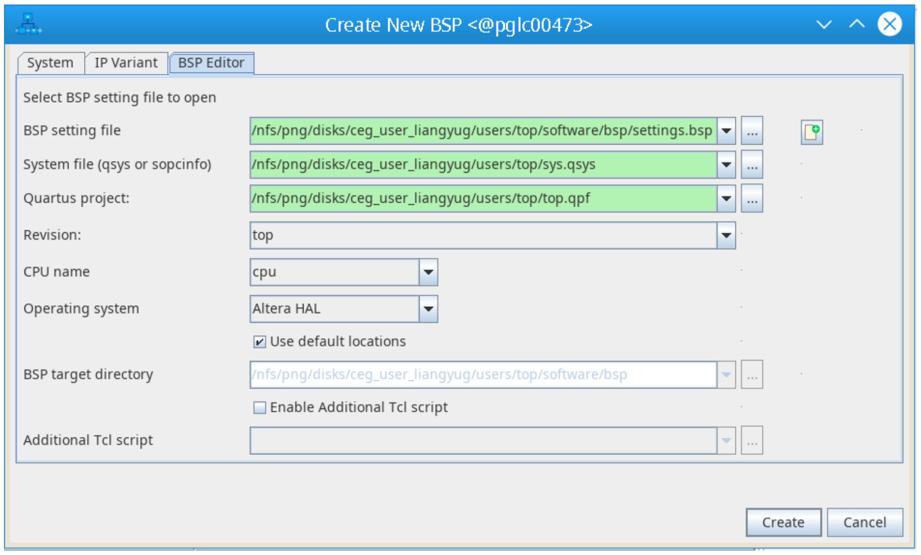


7.3.2.3.2. Creating a Board Support Package Project

Follow these steps to create a BSP project:

1. In the Platform Designer window,, go to **File ➤ New BSP** . The **Create New BSP** window appears.
2. For **BSP setting file**, navigate to the `software/bsp` folder and create a BSP file (`settings.bsp`).
3. For **System file (qsys or sopcinfo)**, select the Nios V processor Platform Designer system.
4. For **Quartus project**, select the example design Quartus Project File.
5. For **Revision**, select the correct revision.
6. For **CPU name**, select the Nios V processor.
7. Select the **Operating system** as **Altera HAL**.
8. Click **Create** to create the BSP file.

Figure 118. Create New BSP Window



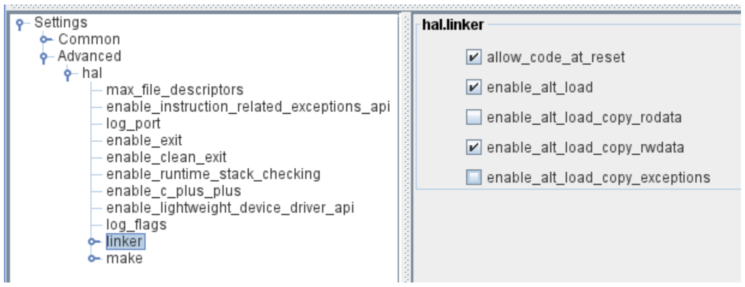
Select BSP setting file to open

BSP setting file	/nfs/png/disks/ceg_user_liangyug/users/top/software/bsp/settings.bsp
System file (qsys or sopcinfo)	/nfs/png/disks/ceg_user_liangyug/users/top/sys.qsys
Quartus project:	/nfs/png/disks/ceg_user_liangyug/users/top/top.qpf
Revision:	top
CPU name	cpu
Operating system	Altera HAL
	☑ Use default locations
BSP target directory	/nfs/png/disks/ceg_user_liangyug/users/top/software/bsp
	☐ Enable Additional Tcl script
Additional Tcl script	

7.3.2.3.3. Configuring and Generating the BSP Project

1. In the **BSP Editor**, go to **Main ➤ Settings ➤ Advanced ➤ hal.linker**.
2. Enable the following settings:
   - **allow_code_at_reset**
   - **enable_alt_load**
   - **enable_alt_load_copy_rwdata**

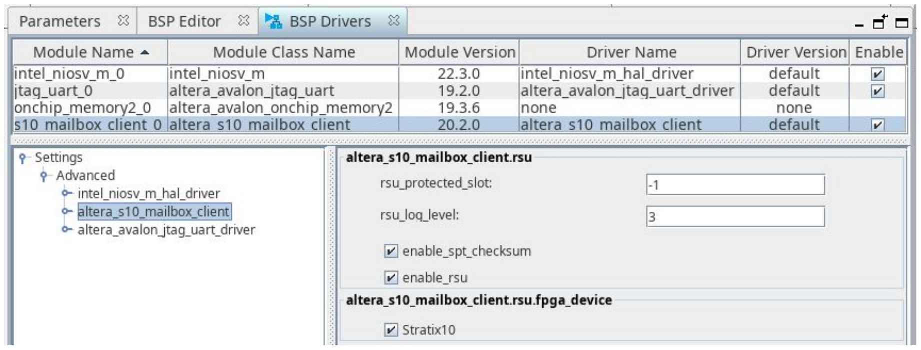
Figure 119. hal.linker Settings



☑ allow_code_at_reset
☑ enable_alt_load
☐ enable_alt_load_copy_rodata
☑ enable_alt_load_copy_rwdata
☐ enable_alt_load_copy_exceptions

3. Navigate to the **BSP Linker Script** tab in the **BSP Editor**.
4. Set all the **Linker Section Name** list to the OCRAM.
5. In the **BSP Drivers**, enable the device driver for Mailbox Client Intel FPGA IP.
6. Go to **Settings ➤ altera_s10_mailbox_client**. You may set **rsu_log_level** as `0` for minimum logging information.
7. Apply **rsu_protected_slot** as `-1` for no slot protection.
8. Enable the following settings:

- **rsu.enable_spt_checksum**
- **rsu.enable_rsu**
- **fpga_device.Stratix10**



Parameters | BSP Editor | BSP Drivers

Module Name	Module Class Name	Module Version	Driver Name	Driver Version	Enable
intel_niosv_m_0	intel_niosv_m	22.3.0	intel_niosv_m_hal_driver	default	✓
jtag_uart_0	altera_avalon_jtag_uart	19.2.0	altera_avalon_jtag_uart_driver	default	✓
onchip_memory2_0	altera_avalon_onchip_memory2	19.3.6	none	none	
s10_mailbox_client_0	altera_s10_mailbox_client	20.2.0	altera_s10_mailbox_client	default	✓

Settings
- Advanced
  - intel_niosv_m_hal_driver
  - altera_s10_mailbox_client
  - altera_avalon_jtag_uart_driver

altera_s10_mailbox_client.rsu

rsu_protected_slot: -1

rsu_log_level: 3

[✓] enable_spt_checksum

[✓] enable_rsu

altera_s10_mailbox_client.rsu.fpga_device

[✓] Stratix10

Figure 120. BSP Drivers Tab

9. Click **Generate BSP**. Make sure the BSP generation is successful.
10. Close the **BSP Editor**.

7.3.2.3.4. Creating Multiple Application Projects

1. Download the example source code using the link below.
2. Navigate to the `software/app` folder and copy the example source codes.
3. Change the default name of Mailbox Client Intel FPGA IP (`MAILBOX_NAME`) based on the `system.h` file, in the following locations:
   a. The example source codes
   b. `bsp/drivers/src/altera_s10_mailbox_client_flash_rsu.c`
   c. `bsp/drivers/src/altera_s10_mailbox_client_rsu.c`
4. Launch the Nios V Command Shell.
5. Execute the command below to generate the user application `CMakeLists.txt`.

```
//For Application Image
niosv-app --app-dir=software/app --bsp-dir=software/bsp \
--srcs=software/app/application.c,zlib/crc32.c \
--incs=zlib

//For Factory Image
niosv-app --app-dir=software/app --bsp-dir=software/bsp \
--srcs=software/app/factory.c,zlib/crc32.c \
--incs=zlib
```

Related Information

Example source code for Nios V Processor LibRSU application

7.3.2.3.5. Building the Application Projects

You can choose to build the application project using Ashling RiscFree IDE for Intel FPGAs, Eclipse Embedded CDT, or through the command line interface (CLI).

If you prefer using CLI, you can build the applications using the following command:

```
cmake -G "Unix Makefiles" -B software/app/build \
-S software/app
make -C software/app/build
```

The user application `.elf` files (`app.elf`) is created in the build folder.

7.3.2.3.6. Generating HEX Files

You must generate a .hex file from the user application .elf files, to memory-initialize the OCRAM in the Nios V processor system.

1. Launch the Nios V Command Shell.
2. For Nios V processor application boot from OCRAM, use the following command line to convert the ELF to HEX for your application.

   ```
   elf2hex software/app/build/app.elf -o app.hex \
       -b <base address of OCRAM> -w <data width of OCRAM> \
       -e <end address of OCRAM>
   ```

3. Recompile the Nios V processor hardware system to memory-initialize the on-chip memory.

7.3.3. Individual Images Generation

Refer to the following steps to generate the four individual images:

1. Repeat the steps in the topics *Hardware Design Flow* and *Software Design Flow* to generate the factory system.
2. Rename the factory image as `factory.sof`.
3. Create three copies of application image and renamed them as
   - `application-0.sof`
   - `application-1.sof`
   - `application-2.sof`

Related Information

- Hardware Design Flow on page 145
- Software Design Flow on page 148

7.3.4. Remote System Update Image Files Generation

To generate the RSU image files in SDM-based devices, you need the Intel Quartus Prime Programming File Generator tool. For generic applications, you can generate the initial image using the `.sof` file. For security application, you need to generate the initial RSU image from the signed or encrypted `.rbf` file.

The next topic describes an example of applying the initial RSU image generation using `.sof` file. For more information on security application, refer to the following collaterals.

Related Information

- Intel Stratix 10 Configuration User Guide: Generating the Initial RSU Image

- Intel Agilex® 7 Configuration User Guide: Generating the Initial RSU Image

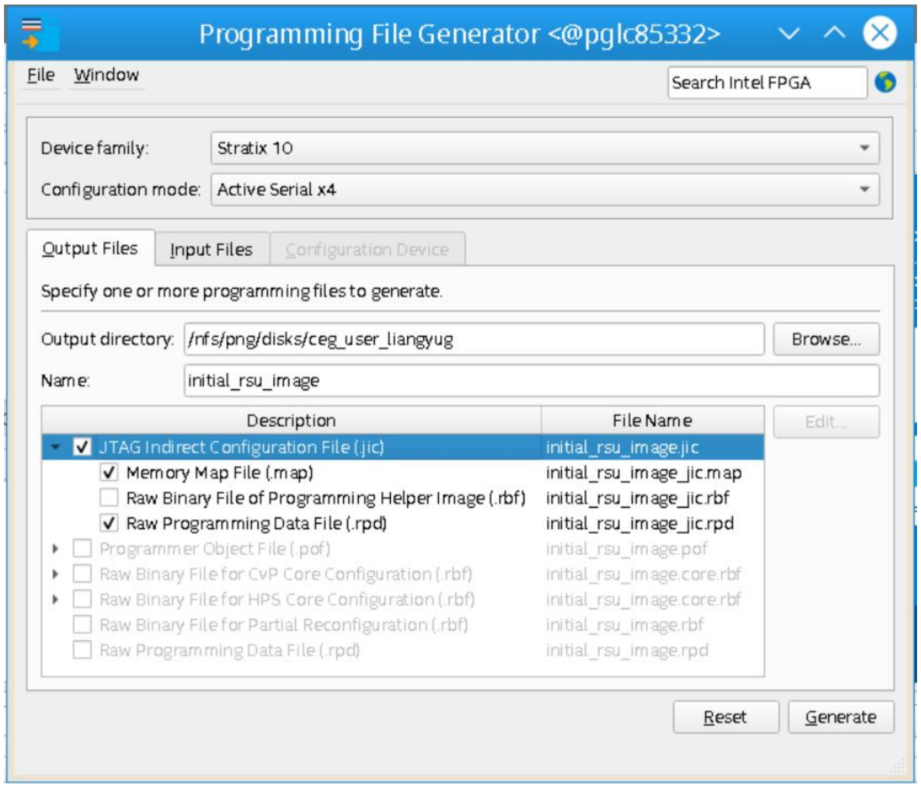
7.3.4.1. Generating Initial RSU Image Using SOF file

1. On the **File** menu, click **Programming File Generator**.
2. Select **Active Serial x4** from the **Configuration mode** drop-down list. The current Intel Quartus Prime software only supports remote system update feature in **Active Serial x4**.
3. On the **Output Files** tab, assign the output directory and file name.
4. Select the output file type as **JTAG Indirect Configuration File (.jic)** with
   a. **Memory Map File (.map)**
   b. **Raw Programming File (.rpd)**

   By default, the `.rpd` file type is little-endian. Set the **Bit swap** to **On** to generate the `.rpd` file in big endian format.

   Note: If you are using a third-party programmer that does not support the little-endian format, set the **Bit swap** to **On**.

Figure 121. Programming File Generator (Output Files)



Device family: Stratix 10
Configuration mode: Active Serial x4

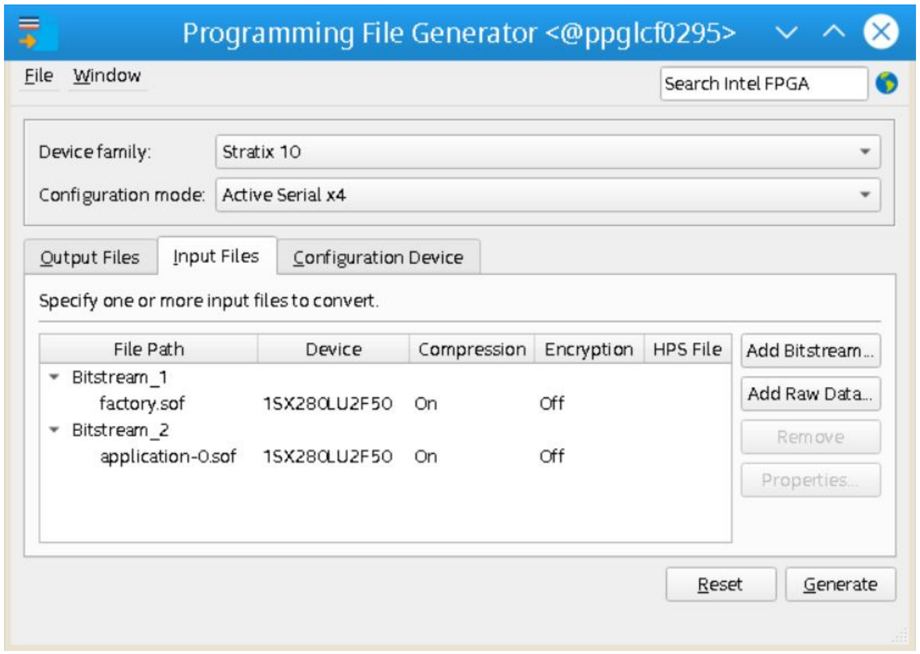
Specify one or more programming files to generate.

Output directory: /nfs/png/disks/ceg_user_liangyug
Name: initial_rsu_image

Description	File Name
☑ JTAG Indirect Configuration File (.jic)	initial_rsu_image.jic
☑ Memory Map File (.map)	initial_rsu_image_jic.map
☐ Raw Binary File of Programming Helper Image (.rbf)	initial_rsu_image_jic.rbf
☑ Raw Programming Data File (.rpd)	initial_rsu_image_jic.rpd
☐ Programmer Object File (.pof)	initial_rsu_image.pof
☐ Raw Binary File for CvP Core Configuration (.rbf)	initial_rsu_image.core.rbf
☐ Raw Binary File for HPS Core Configuration (.rbf)	initial_rsu_image.core.rbf
☐ Raw Binary File for Partial Reconfiguration (.rbf)	initial_rsu_image.rbf
☐ Raw Programming Data File (.rpd)	initial_rsu_image.rpd

5. On the **Input Files** tab, click **Add Bitstream**, select the factory.sof file and click **Open**. Repeat this step for the `application-0.sof`.

Figure 122. Programming File Generator (Input Files)



Device family: Stratix 10

Configuration mode: Active Serial x4

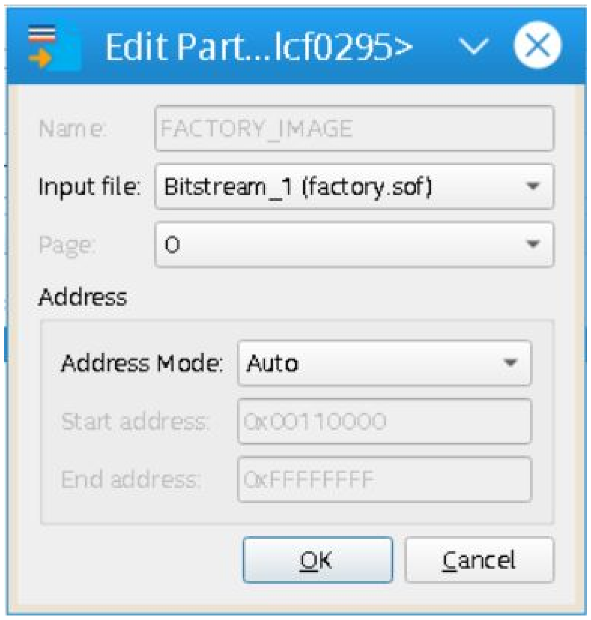
Specify one or more input files to convert.

File Path	Device	Compression	Encryption	HPS File
Bitstream_1				
factory.sof	1SX280LU2F50	On	Off	
Bitstream_2				
application-0.sof	1SX280LU2F50	On	Off	

6. On the **Configuration Device** tab, click **Add Device**, select your flash memory and click **OK**. The Programming File Generator tool automatically populates the flash partitions.
7. Select the FACTORY_IMAGE partition and click **Edit**.
8. In the **Edit Partition** dialog box, select the `factory.sof` file in the **Input Files** drop-down list and click **OK**.

   Note: You must assign **Page 0** to Factory Image. Intel recommends that you let the Intel Quartus Prime software assign the Start address of the FACTORY_IMAGE automatically by retaining the default value for **Address Mode** which is **Auto**.

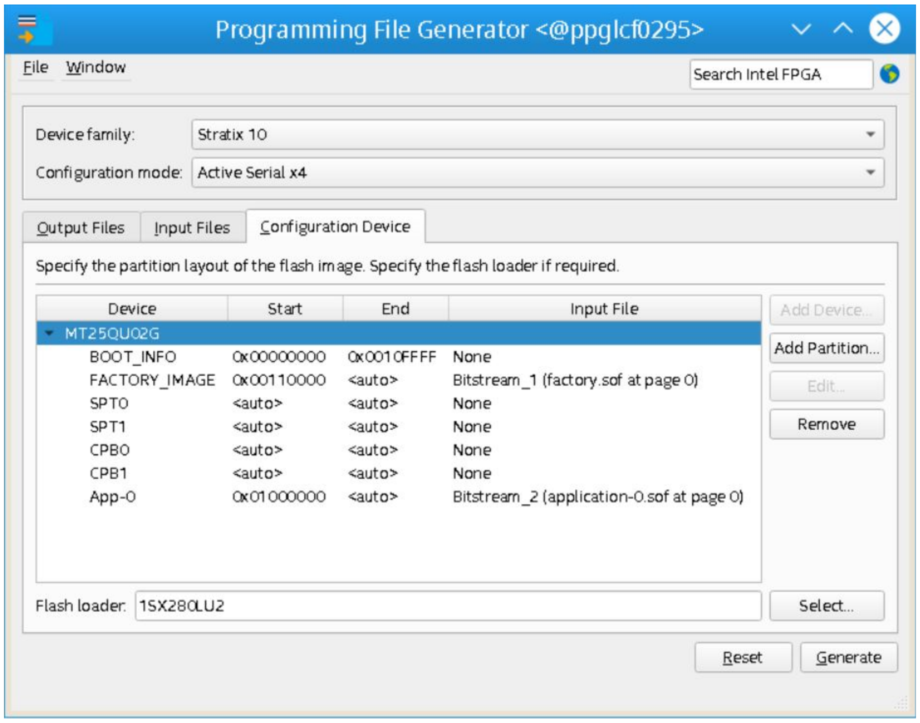
Figure 123. Programming File Generator (Edit Partition)



Name: FACTORY_IMAGE
Input file: Bitstream_1 (factory.sof)
Page: 0
Address
Address Mode: Auto
Start address: 0x00110000
End address: 0xFFFFFFFF

9. Select the flash memory and click **Add Partition**.
10. In the **Add Partition** dialog box, perform the following steps:
    - Define **Name** as **App-0**
    - Select the `application-0.sof` file from the **Input file** drop-down list
    - Assign **Page 1**
    - Assign **Address Mode** as **Start** with starting address at **0x01000000**.
11. If you are generating `.jic` files, click **Select** at the Flash loader, select your device family and device name, and click **OK**.

Figure 124. Programming File Generator (Configuration Device)



Device family: Stratix 10

Configuration mode: Active Serial x4

Specify the partition layout of the flash image. Specify the flash loader if required.

Device	Start	End	Input File
MT25QU02G			
BOOT_INFO	0x00000000	0x0010FFFF	None
FACTORY_IMAGE	0x00110000	<auto>	Bitstream_1 (factory.sof at page 0)
SPT0	<auto>	<auto>	None
SPT1	<auto>	<auto>	None
CPB0	<auto>	<auto>	None
CPB1	<auto>	<auto>	None
App-0	0x01000000	<auto>	Bitstream_2 (application-0.sof at page 0)

Flash loader: 1SX280LU2

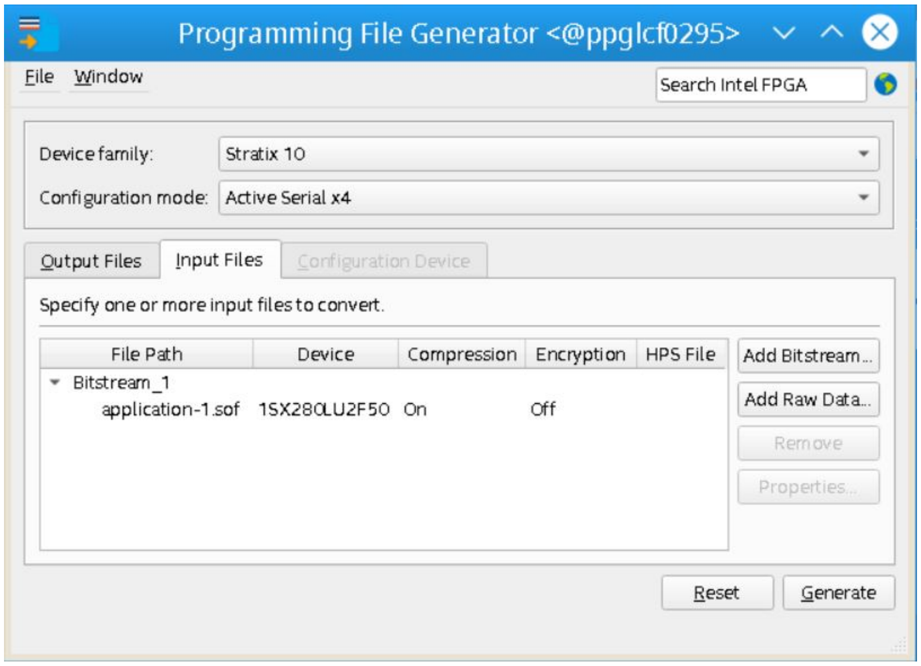
12. Click **Generate** to generate the remote system update programming files. After generating the programming file, proceed to program the flash memory with the initial RSU image.

7.3.4.2. Generating an Application Update Image

1. On the **File** menu, click **Programming File Generator**.
2. Select **Active Serial x4** from the **Configuration mode** drop-down list. The current Intel Quartus Prime software only supports remote system update feature in **Active Serial x4**.
3. On the **Output Files** tab, assign the output directory and file name.
4. Select the output file type as **Raw Programming File (.rpd)**.
5. By default, the `.rpd` file type is little-endian. Set the **Bit swap** to **On**.

   Note: If you are using a third-party programmer that does not support the little-endian format, set the **Bit swap** to **On** to generate the .rpd file in big endian format.
6. On the **Input Files** tab, click **Add Bitstream**. Then, select application update image `.sof` file (`application-1.sof`) file and click **Open**.

Figure 125. Programming File Generator (Input Files)



Device family: Stratix 10
Configuration mode: Active Serial x4

Specify one or more input files to convert.

File Path	Device	Compression	Encryption	HPS File
Bitstream_1				
application-1.sof	1SX280LU2F50	On	Off	

7. Click **Generate** to generate the remote system update programming files. You can now add or update the application image into the initial RSU image.

Example 6. Command to generate application image

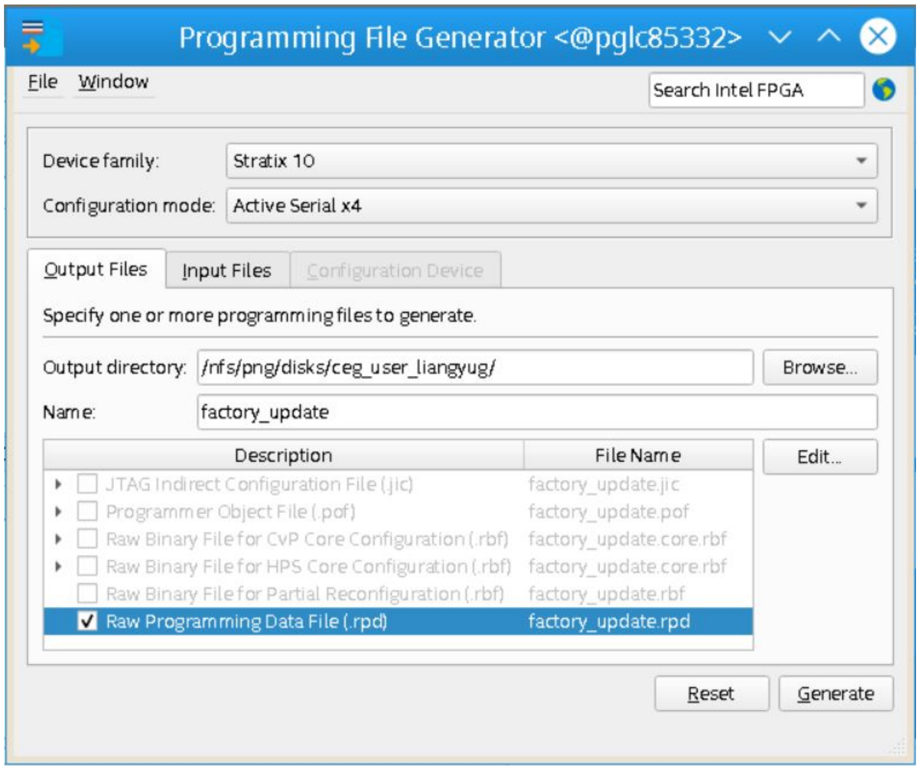
```
quartus_pfg -c application-1.sof app_image.rpd -o mode=ASX4 -o bitswap=ON
```

7.3.4.3. Generating a Factory Update Flash Image

1. On the **File** menu, click **Programming File Generator**.
2. Select **Active Serial x4** from the **Configuration mode** drop-down list. The current Intel Quartus Prime software only supports remote system update feature in **Active Serial x4**.
3. On the **Output Files** tab, assign the output directory and file name.
4. Select the output file type as **Raw Programming File (.rpd)**.
5. By default, the .rpd file type is little-endian. Set the **Bit swap** to **On**.

   Note: If you are using a third-party programmer that does not support the little-endian format, set the **Bit swap** to **On** to generate the .rpd file in big endian format.

Figure 126. Programming File Generator (Output Files)



Device family: Stratix 10

Configuration mode: Active Serial x4

Specify one or more programming files to generate.

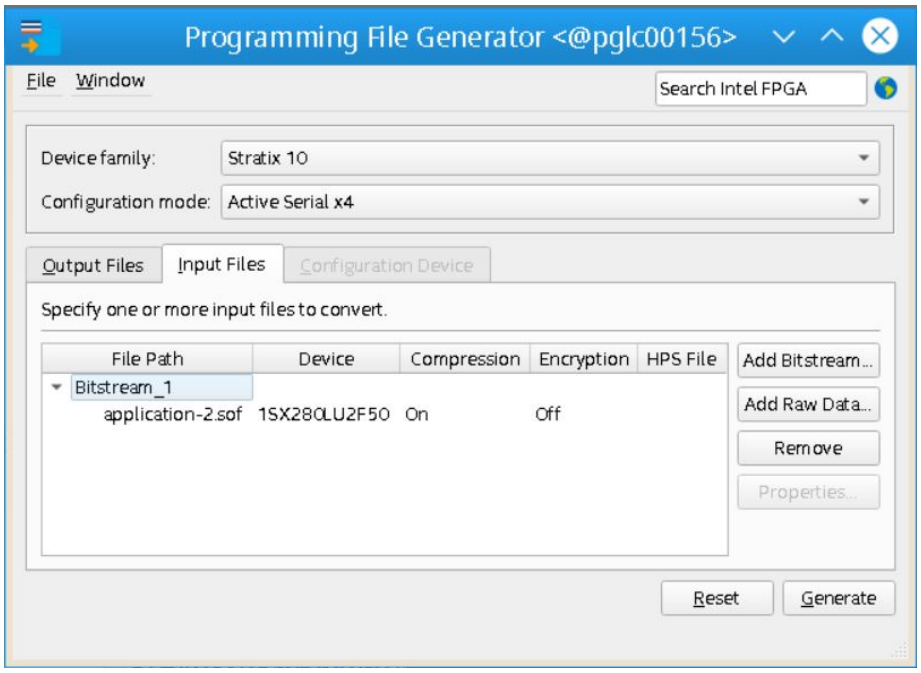
Output directory: /nfs/png/disks/ceg_user_liangyug/

Name: factory_update

Description	File Name
☐ JTAG Indirect Configuration File (.jic)	factory_update.jic
☐ Programmer Object File (.pof)	factory_update.pof
☐ Raw Binary File for CvP Core Configuration (.rbf)	factory_update.core.rbf
☐ Raw Binary File for HPS Core Configuration (.rbf)	factory_update.core.rbf
☐ Raw Binary File for Partial Reconfiguration (.rbf)	factory_update.rbf
✓ Raw Programming Data File (.rpd)	factory_update.rpd

6. On the **Input Files** tab, click **Add Bitstream**. Change the **Files of type** to SRAM Object File (`*.sof`). Then, select factory update image `.sof` file (`application-2.sof`) and click **Open**.

Figure 127. Programming File Generator (Input Files)



Programming File Generator <@pglc00156>

Device family: Stratix 10

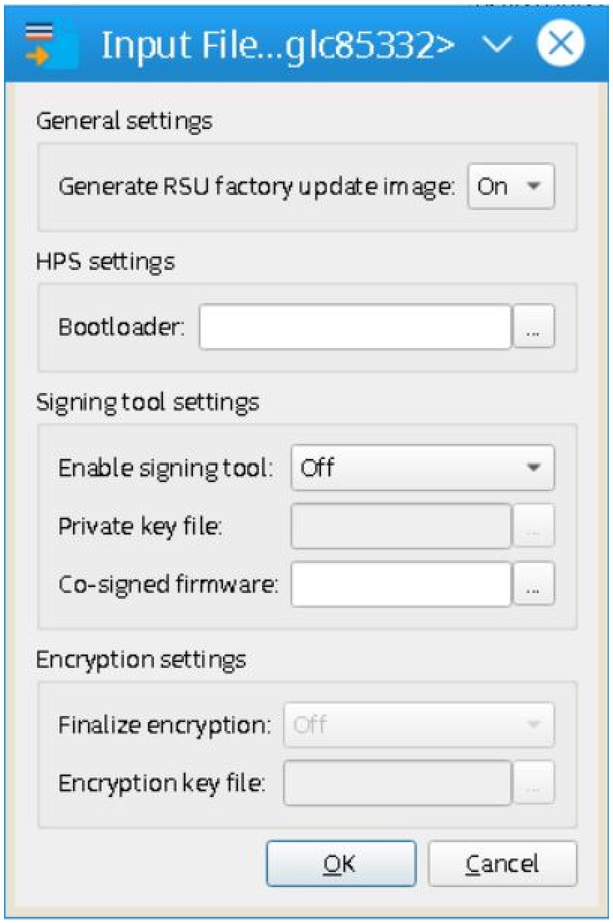
Configuration mode: Active Serial x4

Specify one or more input files to convert.

File Path	Device	Compression	Encryption	HPS File
Bitstream_1				
application-2.sof	1SX280LU2F50	On	Off	

7. Select the `application-2.sof` and then click **Properties**. Turn on **Generate RSU factory update image**.

Figure 128. Generate RSU Factory Update Image



General settings

Generate RSU factory update image: On

HPS settings

Bootloader:

Signing tool settings

Enable signing tool: Off

Private key file:

Co-signed firmware:

Encryption settings

Finalize encryption: Off

Encryption key file:

8. Click **Generate** to generate the RSU programming files. You can now update the decision firmware, decision firmware data, and the factory image into the initial RSU image.

Example 7. Command to generate factory update image

```
quartus_pfg -c application-2.sof factory_update.rpd -o mode=ASX4 -o bitswap=ON -
o rsu_upgrade=ON
```

7.3.5. QSPI Flash Programming

7.3.5.1. Programming the Initial RSU Image

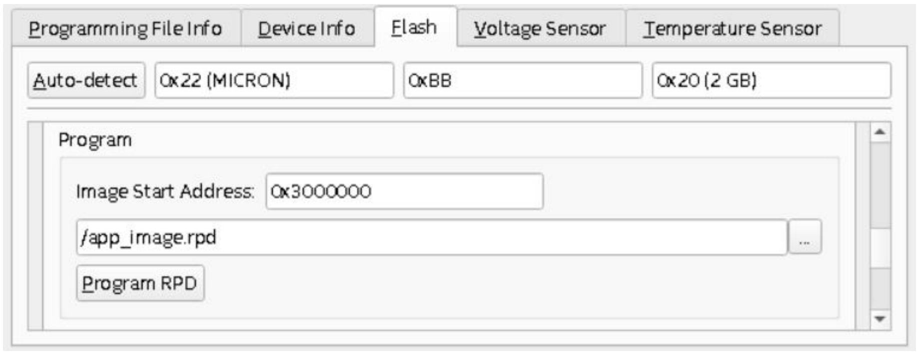
1. Ensure that the Intel FPGA device's Active Serial (AS) pin is routed to the QSPI flash. This routing allows the flash loader to load into the QSPI flash and configure the board correctly.
2. Ensure the MSEL pin setting on the board is configured for AS programming.
3. Open the **Intel Quartus Prime Programmer** and make sure JTAG is detected under the **Hardware Setup**.
4. Select **Auto Detect** and choose the FPGA device according to your board.

5. Right-click the selected Intel FPGA device and select **Edit ➤ Change File**. Next, select the initial RSU image JIC file.
6. Select the **Program/ Configure** check boxes for FPGA and QSPI devices.
7. Click **Start** to start programming.

7.3.5.2. Programming the Update Images

1. Ensure that the Intel FPGA device's Active Serial (AS) pin is routed to the QSPI flash. This routing allows the flash loader to load into the QSPI flash and configure the board correctly.
2. Ensure the MSEL pin setting on the board is configured for AS programming.
3. Open the **Intel Quartus Prime Configuration Debugger** and make sure JTAG is detected under the **Hardware Setup**.
4. Click **Load Device** and select the Intel FPGA device.
5. Navigate to the **Flash** tab.
6. Click **Auto-detect** to auto-detect the QSPI Flash that is attached to the device.
7. Navigate to the **Program** function. Assign **Image Start Address** and **RPD file path**.
   - For `app_image.rpd`, the **Image Start Address** is 0x3000000.
   - For `factory_update.rpd`, the **Image Start Address** is 0x3800000.
8. Click **Program RPD** to begin.

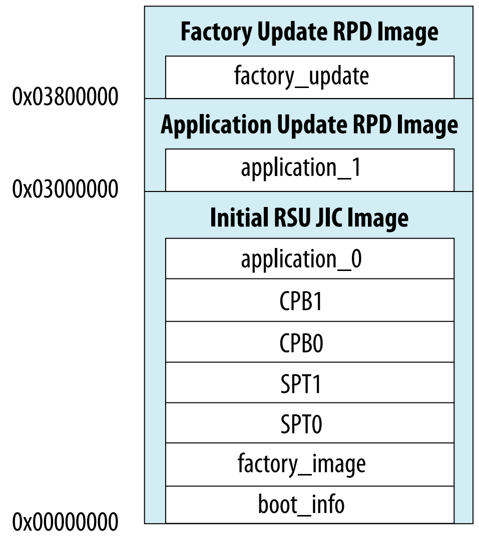
Figure 129. Configuration Debugger - Flash



0x22 (MICRON) | 0xBB | 0x20 (2 GB)

Image Start Address: 0x3000000
/app_image.rpd

Figure 130. Quad SPI Flash Address Map



Example 8. Memory Map File of Initial RSU JIC Image

```
BLOCK                         START ADDRESS   END ADDRESS

BOOT_INFO                     0x00000000      0x0010FFFF
FACTORY_IMAGE                 0x00110000      0x0084FFFF (0x0080CFFF)
SPT0                          0x00850000      0x00857FFF
SPT1                          0x00858000      0x0085FFFF
CPB0                          0x00860000      0x00867FFF
CPB1                          0x00868000      0x0086FFFF
App-0                         0x01000000      0x01432FFF


Configuration device: 1SX280LU2
Configuration mode: Active Serial x4
```

Related Information

AN 955: Programmer's Configuration Debugger Tool

7.3.6. Operating the RSU Client API

The RSU Client API performs the following operations:

- Trigger Intel FPGA device reconfiguration with selected image
- Update the application image
- Update the factory image

To display the Nios V processor application messages, the example design utilizes the JTAG UART Intel FPGA IP. You can begin the display message by using the following command:

```
juart-terminal
```

The JTAG UART terminal displays the RSU message logs, followed by the RSU Menu. While the factory image provides the full list of operations, the application images can support status log acquisition and reconfiguration operations only. The RSU Menu offers the following options:

1. Acquire RSU status log
2. Acquire Decision Firmware Status Log
3. Trigger reconfiguration with Factory Image
4. Trigger reconfiguration with Application Images
    a. Application-0 Image
    b. Application-1 Image
5. Add Application-1 Image
6. Update the Factory Image
7. Erase Decision Firmware

Note: For application image, the Menu options are only Options 1 to 4.

7.3.6.1. Trigger Reconfiguration Menu with Selected Image

The Trigger reconfiguration menu, Option 3 and 4 performs device reconfiguration with the factory and application images respectively. The following table shows the RSU status log after the device reconfiguration is successful.

Table 44. Trigger Device Reconfiguration

Options	RSU Status Log

Trigger reconfiguration with Factory Image	`Current Image : 0x00110000`
	`Last Fail Image : 0x00000000`
	`State : 0x00000000`
	`Version : 0x00000202`
	`Error location : 0x00000000`
	`Error details : 0x00000000`
	`Retry counter : 0x00000000`
	`Running factory image: yes`

Trigger reconfiguration with Application-0 Image	`Current Image : 0x01000000`
	`Last Fail Image : 0x00000000`
	`State : 0x00000000`
	`Version : 0x00000202`
	`Error location : 0x00000000`
	`Error details : 0x00000000`
	`Retry counter : 0x00000000`
	`Running factory image: no`

Trigger reconfiguration with Application-1 Image
(After performing Option 5.)
	`Current Image : 0x01800000`
	`Last Fail Image : 0x00000000`
	`State : 0x00000000`
	`Version : 0x00000202`
	`Error location : 0x00000000`
	`Error details : 0x00000000`
	`Retry counter : 0x00000000`
	`Running factory image: no`

7.3.6.2. Updating an Application Image

The Add Application-1 Image, Option 5 performs RSU Image update by adding Application-1 image into RSU slot 1, called **App-1**. It reads the Application-1 RPD image from the QSPI flash, starting from address 0x3000000. After the RPD image is read successfully, the software proceeds to add and verify the configuration bitstream into **App-1** slot. Once the verification completes, you can proceed to trigger reconfiguration with Application-1 Image in the table *Trigger Device Reconfiguration*.

Example 9. Application Image Update Log

```
Reading the Application-1 image based on RPD memory map....
Read Successfully.
Slot App-1 created at 0x1800000 with size =  0x460000 bytes.
Slot 1 is erased.
      NAME: App-1
    OFFSET: 0x0000000001800000
      SIZE: 0x00460000
  PRIORITY: [disabled]
Slot 1 was programmed with size=4587520.
Slot 1 was verified with size=4587520.
Add and Verify the image successfully in Slot 1

Please proceed with Option 4 - Trigger reconfiguration to Application Images.
And select Application-1 Image.
```

7.3.6.3. Updating the Factory Image

The Update the Factory Image menu, Option 6 performs RSU Image update by updating to a new factory image and decision firmware version. Before initiating Option 6, you are recommended to run Option 7 to erase the current decision firmware to show that a new decision firmware is updated along with the factory image.

The operation begin by reading the Factory Update RPD image from the QSPI flash, starting from address 0x3800000. After the RPD image is read successfully, the software proceeds to add and verify the configuration bitstream into a temporary **FactoryUpdate** slot. Once it is completed, power cycle the device.

Note: After the factory image is updated completely, the device automatically reconfigure to the application image with the highest priority.

Table 45. Decision Firmware Status Log

Erase Decision Firmware		Power Cycle	
Before	After	Before	After
DCMF0: OK **DCMF1: OK** **DCMF2: OK** **DCMF3: OK**	DCMF0: OK **DCMF1: Corrupted** **DCMF2: Corrupted** **DCMF3: Corrupted**	DCMF0: OK **DCMF1: Corrupted** **DCMF2: Corrupted** **DCMF3: Corrupted**	DCMF0: OK **DCMF1: OK** **DCMF2: OK** **DCMF3: OK**

Example 10. Factory Image Update Log

```
Reading the Factory Update image based on RPD memory map....
Read Successfully.
Slot FactoryUpdate created at 0x2000000 with size =  0x460000 bytes.
Slot 2 is erased.
      NAME: FactoryUpdate
    OFFSET: 0x0000000002000000
      SIZE: 0x00460000
  PRIORITY: [disabled]
Slot 2 was programmed with size=4587520.
Slot 2 was verified with size=4587520.
Add and Verify the image successfully in Slot 2
```

```
Please power cycle the device to update the factory image.
```

8. Nios V Processor — Using Custom Instruction

8.1. Introduction

The Nios V/g processor supports custom instruction feature. This feature allows you to connect the processor to a custom processing engine (custom logic blocks). You can develop the processing engine to support the following functions:

- Enable unimplemented instruction in the Nios V Processor Instruction Set Architecture (ISA).
- Perform hardware acceleration on software algorithms.

Intel recommends you to understand custom instruction feature in Nios V processor by reading *AN 977: Nios V Processor Custom Instruction* before proceeding with the example design.

Related Information

AN 977: Nios V Processor Custom Instruction
For more information about custom instructions that allow you to customize the Nios® V processor to meet the needs of a particular application.

8.2. Unimplemented Instruction Example Design

You can refer to *Custom Instruction Design on Nios V/g Processor* in the Intel FPGA Design Store as a reference.

Related Information

Intel Agilex® 7 FPGA - Custom Instruction Design on Nios® V/g Processor

8.2.1. Hardware and Software Requirements

You need the following hardware and software in order to apply a custom instruction on a Nios V/g processor.

- Intel Quartus Prime Pro Edition software version 23.1 or later
- Ashling RiscFree for Intel FPGAs software version 23.1 or later

  Note: Intel recommends you install the same software version for all softwares.
- One of the supported Intel FPGA devices
  — The example design implemented on Intel Agilex® 7 F-Series FPGA development kit (DK-DEV-AGF014EA).
- Intel FPGA Download Cable II

You must connect your development board to a host PC on the USB/JTAG ports.

Related Information

Intel Agilex 7 FPGA F-Series Development Kits

8.2.2. Overview

You can download the Intel Agilex 7 FPGA - Custom Instruction Design on Nios V/g Processor in the Intel FPGA Design Store. The example designs are based on the Intel Agilex 7 F-Series FPGA Development Kit. Using the scripts, the hardware and software design are generated, and programmed as SRAM Object Files (`.sof`) and Executable and Linking Format (`.elf`) into the device. The example design connects two similar processing engines to a Nios V processor system. The processing engines contain custom bit manipulation operations. These operations are not natively supported in the Nios V processor ISA.

8.2.3. Acquiring the Example Design File

To generate the example design, perform the following steps:

1. Go to Intel FPGA Design Store.
2. Search for *Intel Agilex® 7 FPGA - Custom Instruction Design on Nios® V/g Processor*.
3. Click on the link at the title.
4. Accept the *Software License Agreement*.
5. Download the package according to the Intel Quartus Prime software version of your host PC.
6. Refer to the `readme.txt` for the how-to guide.

Table 46. Example Design File Description

File	Description
`custom_logic/`	Contains the custom logic processing engines, which holds eight different operations.
`hw/`	Contains file necessary to run the hardware project.
`ready_to_test/`	Contains pre-built hardware and software binaries to run the design on the target hardware. For this package, the target hardware is Intel Arria 10 SoC development kit. This design is targeted on Intel Agilex 7 F-Series FPGA Development Kit DK-DEV-AGF014EA.
`scripts/`	Consists of scripts to build the design.
`sw/`	Contains software application files.
`readme.txt`	Contains description and steps to apply the pre-built binaries or rebuild the binaries from scratch.

Related Information

Intel FPGA Design Store

8.2.4. Hardware Design Files

The Intel Agilex 7 FPGA - Custom Instruction Design on Nios V/g Processor is developed using the Platform Designer. You can generate the hardware files using the `build_sof.py` Python script.

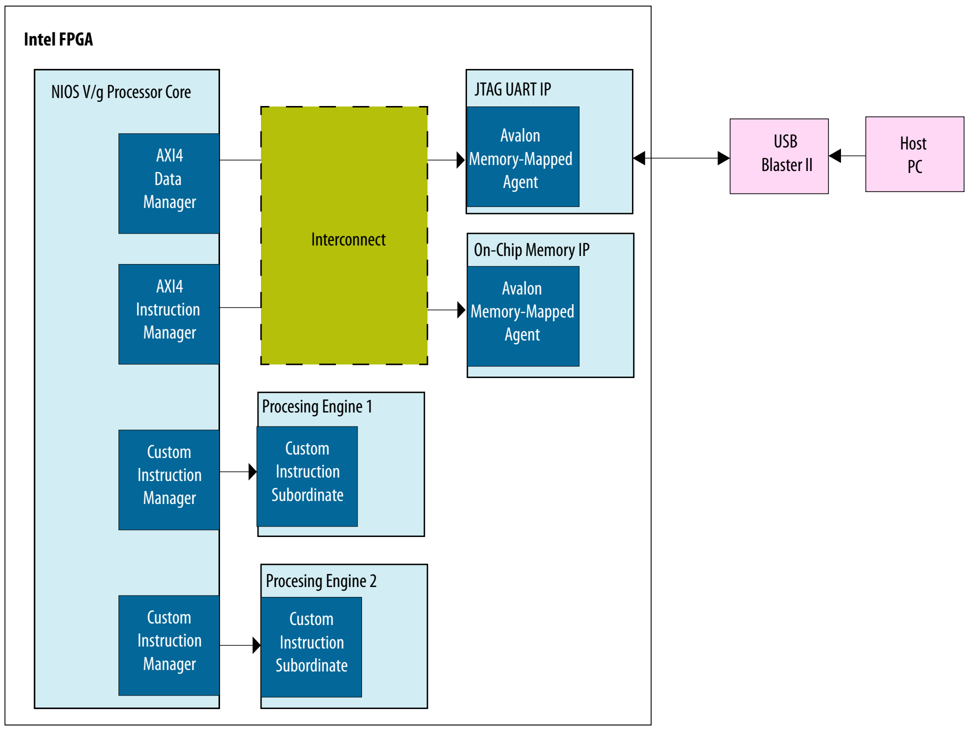
The example design consists of:

- Nios V Processor Intel FPGA IP
- On-Chip Memory II Intel FPGA IP
- JTAG UART Intel FPGA IP
- Processing Engine 1 (PE1) – Declares `funct3` as user-defined intermediate (3'bxxx). All custom operations share a single software C-macro. You can select them using `funct3` input argument.
- Processing Engine 2 (PE2) – Defines `funct3` as extension index (3'b000 to 3'b111). Each operations have its own C-macros. You can call their respective C-macros.

The processing engine comprises of the following operations, which are selected based on the 3-bits `funct3` field.

- Operation 0: 1's complement of `Data0`
- Operation 1: 2's complement of `Data0`
- Operation 2: Multiply `Data0` with `Data1`
- Operation 3: Bit reversal of `Data0`
- Operation 4: Byte reversal of `Data0`
- Operation 5: Word reversal of `Data0`
- Operation 6: Lower word merge of `Data0` and `Data1`
- Operation 7: Higher word merge of `Data0` and `Data1`

Figure 131. Example Design Block Diagram



8.2.5. Software Design Files

You can find the application file(`custom_instr_app.c`) in the example design zip file. The software file is available in the `sw/app` folder. The source code begins the application by interfacing with PE1, followed by PE2. Within each processing engine, the source code calls all operations, and display the result through the JTAG UART IP into the host PC. The source code provides the same inputs (data0 and data1) into the processing engines. Thus, both PE1 and PE2 return the same responses.

Related Information

Overview on page 167
For more information about the example design.

8.2.6. Development Flow

8.2.6.1. Hardware Development Flow

You can create the example designs hardware system using the `build_sof.py` Python script. The scripts are stored in the `scripts` folder. You can refer to the readme file (`readme.txt`) to develop the example designs using the provided scripts, or develop the design manually using the Platform Designer and Nios V processor tools.

After launching the Nios V Command Shell, run the script using the following command :

```
$ quartus_py scripts/build_sof.py
```

8.2.6.2. Software Development Flow

Creating the example design software image for the custom instruction example design consist of the following general steps:

1. Creating a board support package (BSP) project with `niosv-bsp`.
2. Creating a Nios V processor application project with the provided software design files with `niosv-app`.
3. Building the application project with CMake and Make.

After launching the Nios V Command Shell, run the following commands.

```
$ niosv-bsp -c --quartus-project=hw/<Project Name>.qpf \
--qsys=hw/<System Name>.qsys --type=hal sw/bsp/settings.bsp
$ niosv-app --bsp-dir=sw/bsp --app-dir=sw/app \
--srcs=sw/app/custom_instr_app.c
$ cmake -S ./sw/app -G "Unix Makefiles" -B sw/app/build
$ make -C sw/app/build
```

8.2.6.3. Device Programming

To program Nios V processor based system into the FPGA and to run your application, use Intel Quartus Prime Programmer tool.

1. To create the Nios V processor inside the FPGA device, program the `.sof` file onto the board with the following command.

Table 47. Command

Operating System	Command
Windows	`quartus_pgm -c 1 -m JTAG -o p;<SOF File>@1`
Linux	`quartus_pgm -c 1 -m JTAG -o p\;<SOF File>@1`

Note:
- -c 1 is referring to cable number connected to the Host Computer.
- @1 is referring to device index on the JTAG Chain and may differ for your board.

2. Download the `.elf` using the `niosv-download` command.

```
niosv-download -g <elf file>
```

3. Use the JTAG UART terminal to print the stdout and stderr of the Nios V processor system.

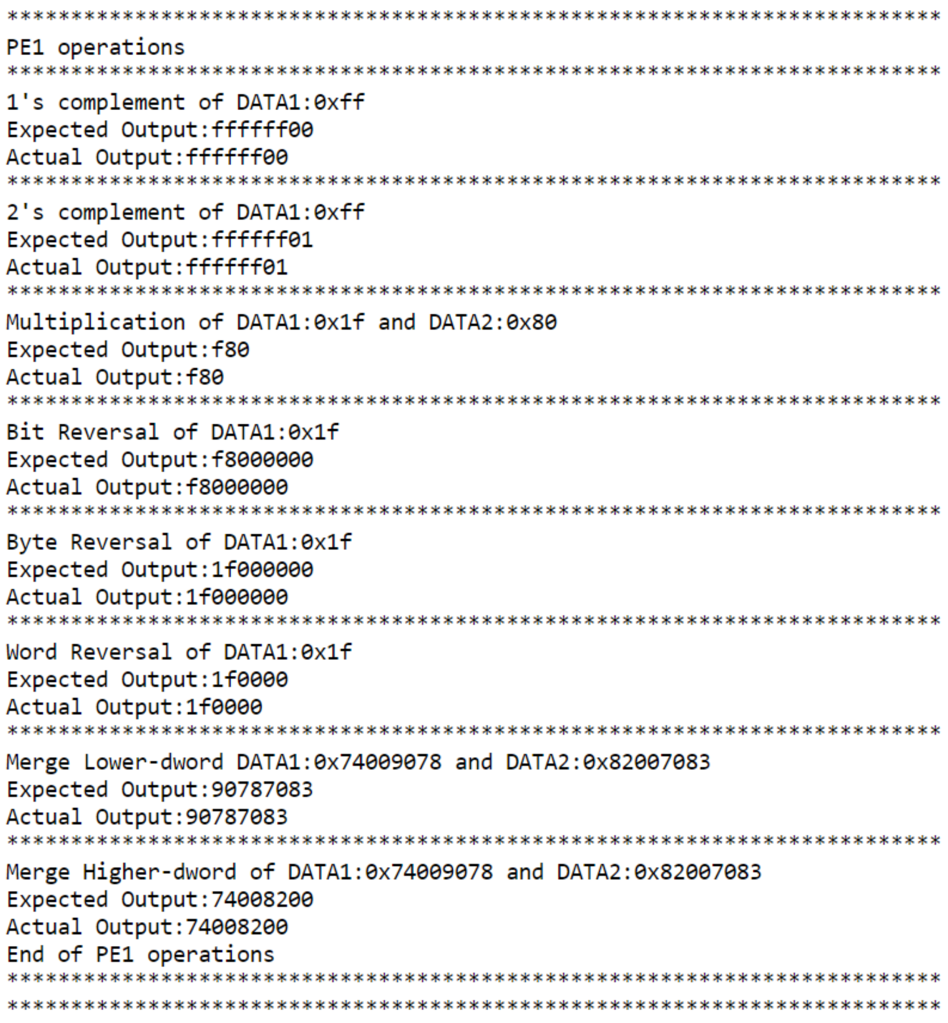
```
juart-terminal
```

8.2.7. Operating the Example Design

To display the application messages, the example design utilizes the JTAG UART Intel FPGA IP. You can begin the display message by using the following command:

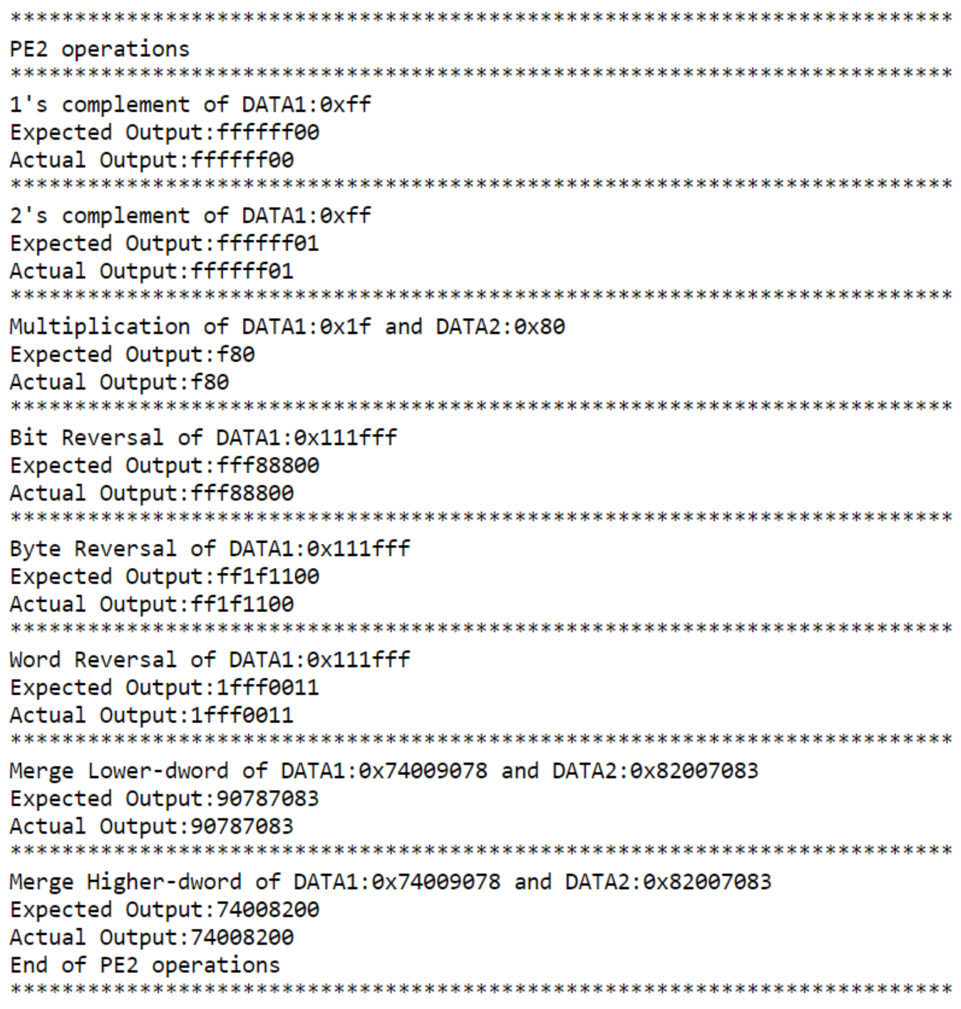
```
juart-terminal
```

Figure 132. Output Result from PE1



```
*************************************************************************
PE1 operations
*************************************************************************
1's complement of DATA1:0xff
Expected Output:ffffff00
Actual Output:ffffff00
*************************************************************************
2's complement of DATA1:0xff
Expected Output:ffffff01
Actual Output:ffffff01
*************************************************************************
Multiplication of DATA1:0x1f and DATA2:0x80
Expected Output:f80
Actual Output:f80
*************************************************************************
Bit Reversal of DATA1:0x1f
Expected Output:f8000000
Actual Output:f8000000
*************************************************************************
Byte Reversal of DATA1:0x1f
Expected Output:1f000000
Actual Output:1f000000
*************************************************************************
Word Reversal of DATA1:0x1f
Expected Output:1f0000
Actual Output:1f0000
*************************************************************************
Merge Lower-dword DATA1:0x74009078 and DATA2:0x82007083
Expected Output:90787083
Actual Output:90787083
*************************************************************************
Merge Higher-dword of DATA1:0x74009078 and DATA2:0x82007083
Expected Output:74008200
Actual Output:74008200
End of PE1 operations
*************************************************************************
*************************************************************************
```

Figure 133. Output Result from PE2



```
**************************************************************************
PE2 operations
**************************************************************************
1's complement of DATA1:0xff
Expected Output:ffffff00
Actual Output:ffffff00
**************************************************************************
2's complement of DATA1:0xff
Expected Output:ffffff01
Actual Output:ffffff01
**************************************************************************
Multiplication of DATA1:0x1f and DATA2:0x80
Expected Output:f80
Actual Output:f80
**************************************************************************
Bit Reversal of DATA1:0x111fff
Expected Output:fff88800
Actual Output:fff88800
**************************************************************************
Byte Reversal of DATA1:0x111fff
Expected Output:ff1f1100
Actual Output:ff1f1100
**************************************************************************
Word Reversal of DATA1:0x111fff
Expected Output:1fff0011
Actual Output:1fff0011
**************************************************************************
Merge Lower-dword of DATA1:0x74009078 and DATA2:0x82007083
Expected Output:90787083
Actual Output:90787083
**************************************************************************
Merge Higher-dword of DATA1:0x74009078 and DATA2:0x82007083
Expected Output:74008200
Actual Output:74008200
End of PE2 operations
**************************************************************************
```

8.3. Hardware Acceleration Example Design

You can refer to *CRC Custom Instruction Design on Nios V/g processor* in *Intel FPGA Design Store* as a reference.

Related Information

Intel Agilex 7 FPGA - CRC Custom Instruction Design on Nios® V/g processor

8.3.1. Hardware and Software Requirements

You need the following hardware and software to apply a custom instruction on a Nios V/g processor.

- Intel Quartus Prime Pro Edition software version 23.1 or later.
- Ashling RiscFree for Intel FPGAs software version 23.1 or later.

  Note: Intel recommends you install the same software version for all softwares.
- One of the supported Intel FPGA devices:
  - The example design implemented on Intel Agilex 7 F-Series FPGA development kit (DK-DEV-AGF014EA).
  - Intel FPGA Download Cable II

  You must connect your development board to a host PC on the USB/JTAG ports.

Related Information

Intel Agilex 7 FPGA F-Series Development Kits

8.3.2. Overview

You can download the *CRC Custom Instruction Design on Nios V/g processor* in the Intel FPGA Design Store. The example designs are based on the Intel Agilex 7 F-Series FPGA Development Kit. Use the scripts to generate and programme the hardware and software design as SRAM Object Files (`.sof`) and Executable and Linking Format (`.elf`) into the device.

The example design connects a custom logic CRC processing engine to a Nios V processor system. In the Nios V software application, the processor feeds the same checksum data into three CRC decoders (custom logic CRC processing engine, CRC software algorithm, and optimized CRC software algorithm). All three CRC decoders return the same CRC results, and the latency is compared among themselves.

8.3.3. Acquiring the Example Design File

To generate the example design, perform the following steps:

1. Go to Intel® FPGA Design Store.
2. Search for *CRC Custom Instruction Design on Nios® V/g processor*.
3. Click on the link at the title.
4. Accept the *Software License Agreement*.
5. Download the package according to the Intel Quartus Prime software version of your host PC.
6. Refer to the `readme.txt` for the how-to guide.

Table 48. Example Design File Description

File	Description
`custom_logic/`	Contains the custom logic CRC processing engine.
`hw/`	Contains file necessary to run the hardware project.

continued...

File	Description
`ready_to_test/`	Contains pre-built hardware and software binaries to run the design on the target hardware. For this package, the target hardware is Intel Agilex® 7 F-Series FPGA Development Kit DK-DEV-AGF014EA.
`scripts/`	Consists of scripts to build the design.
`sw/`	Contains software application files.
`readme.txt`	Contains description and steps to apply the pre-built binaries or rebuild the binaries from scratch.

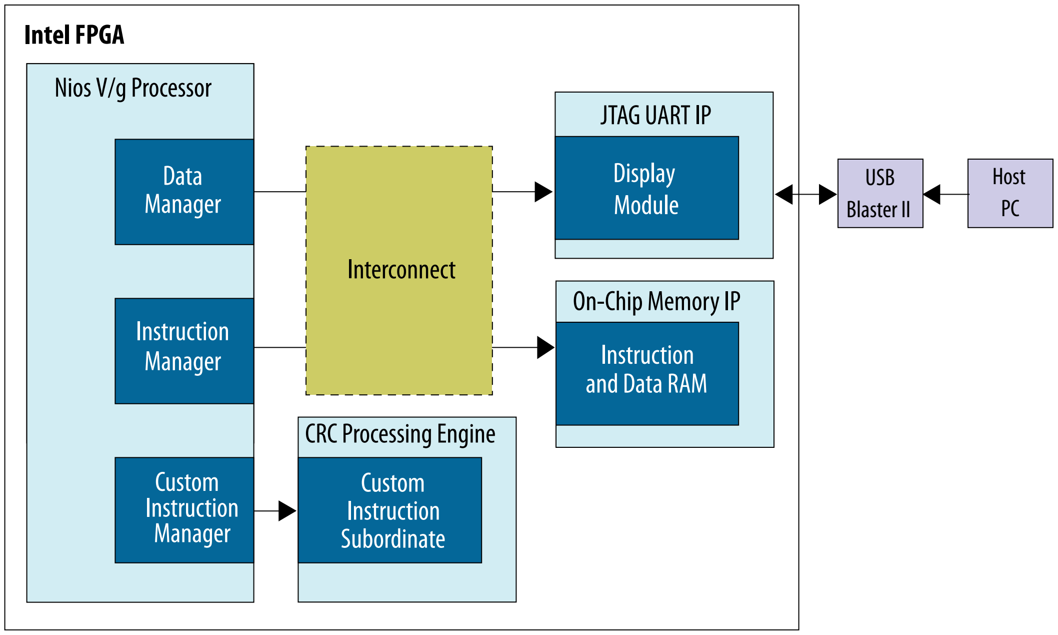
8.3.4. Hardware Design Files

The *CRC Custom Instruction Design on Nios® V/g processor* is developed using the Platform Designer. You can generate the hardware files using the `build_sof.py` Python script.

The example design consists of:

- Nios V Processor Intel FPGA IP
- On-Chip Memory II Intel FPGA IP
- JTAG UART Intel FPGA IP
- CRC Processing Engine

Figure 134. Example Design Block Diagram



8.3.5. Software Design Files

You can find the following application files in the example design zip file. These software files are available in the `sw/app_crc/srcs` folder.

Table 49. Software Design Files

File	Description
`ci_crc.c`	Defines a macro to access the CRC processing engine.
`ci_crc.h`	Contains the function prototype for the macro.
`crc.c`	Defines macros for both software CRC and optimized software CRC algorithms.
`crc.h`	Contains the function prototypes for the software CRC application.
`crc_main.c`	Compute the checksum value using all CRC decoder.

The source code begins the application by computing the checksum value using all three CRC decoder and validating the CRC results. Once the CRC results are matched, the application reports the processing performance of the CRC decoder respectively.

8.3.6. Development Flow

8.3.6.1. Hardware Development Flow

You can create the example designs hardware system using the `build_sof.py` Python script. The scripts are stored in the `scripts` folder. You can refer to the readme file (`readme.txt`) to develop the example designs using the provided scripts, or develop the design manually using the Platform Designer and Nios V processor tools.

After launching the Nios V Command Shell, run the script using the following command :

```
$ quartus_py scripts/build_sof.py
```

8.3.6.2. Software Development Flow

Creating the example design software image for the custom instruction example design consist of the following general steps:

1. Creating a board support package (BSP) project with `niosv-bsp`.
2. Creating a Nios V processor application project with the provided software design files with `niosv-app`.
3. Building the application project with CMake and Make.

After launching the Nios V Command Shell, run the following commands.

```
$ niosv-bsp -c --quartus-project=hw/<Project Name>.qpf \
--qsys=hw/<System Name>.qsys --type=hal sw/bsp/settings.bsp
$ niosv-app --bsp-dir=sw/bsp_crc --app-dir=sw/app_crc \
--srcs=sw/app_crc/srcs/
$ cmake -S ./sw/app_crc -G "Unix Makefiles" -B sw/app_crc/build
$ make -C sw/app_crc/build
```

8.3.6.3. Device Programming

To program Nios V processor based system into the FPGA and to run your application, use Intel Quartus Prime Programmer tool.

1. To create the Nios V processor inside the FPGA device, program the `.sof` file onto the board with the following command.

Table 50. Command

Operating System	Command
Windows	`quartus_pgm -c 1 -m JTAG -o p;<SOF File>@1`
Linux	`quartus_pgm -c 1 -m JTAG -o p\;<SOF File>@1`

Note:
- -c 1 is referring to cable number connected to the Host Computer.
- @1 is referring to device index on the JTAG Chain and may differ for your board.

2. Download the `.elf` using the `niosv-download` command.

```
niosv-download -g <elf file>
```

3. Use the JTAG UART terminal to print the stdout and stderr of the Nios V processor system.

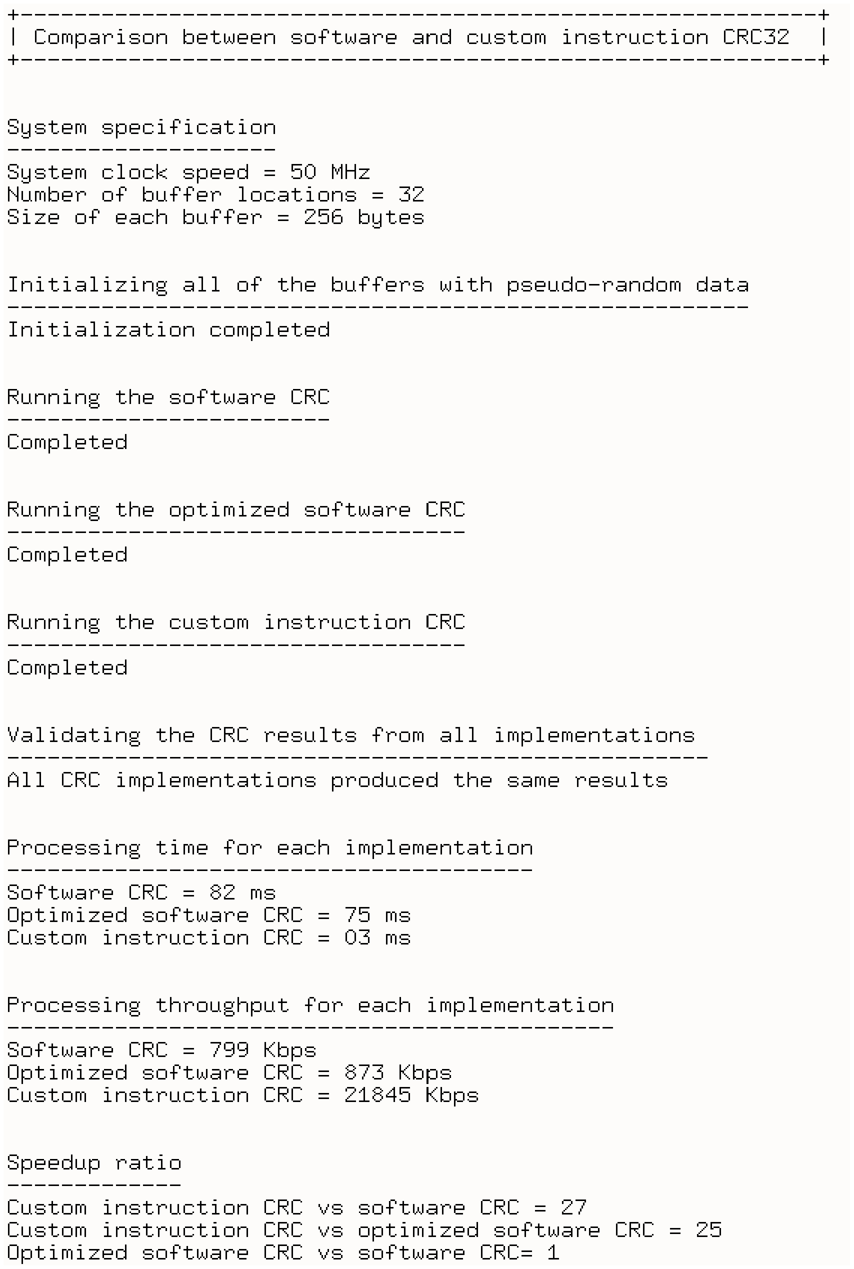
```
juart-terminal
```

8.3.7. Operating the Example Design

To display the application messages, the example design utilizes the JTAG UART Intel FPGA IP. You can begin the display message by using the following command:

```
juart-terminal
```

Figure 135. Output Result from CRC Decoders



```
+-----------------------------------------------------------+
| Comparison between software and custom instruction CRC32  |
+-----------------------------------------------------------+


System specification
--------------------
System clock speed = 50 MHz
Number of buffer locations = 32
Size of each buffer = 256 bytes


Initializing all of the buffers with pseudo-random data
-------------------------------------------------------
Initialization completed


Running the software CRC
------------------------
Completed


Running the optimized software CRC
----------------------------------
Completed


Running the custom instruction CRC
----------------------------------
Completed


Validating the CRC results from all implementations
----------------------------------------------------
All CRC implementations produced the same results


Processing time for each implementation
---------------------------------------
Software CRC = 82 ms
Optimized software CRC = 75 ms
Custom instruction CRC = 03 ms


Processing throughput for each implementation
---------------------------------------------
Software CRC = 799 Kbps
Optimized software CRC = 873 Kbps
Custom instruction CRC = 21845 Kbps


Speedup ratio
-------------
Custom instruction CRC vs software CRC = 27
Custom instruction CRC vs optimized software CRC = 25
Optimized software CRC vs software CRC= 1
```

9. Nios V Embedded Processor Design Handbook Archives

For the latest and previous versions of this user guide, refer to Nios® V Embedded Processor Design Handbook. If an IP or software version is not listed, the user guide for the previous IP or software version applies.

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.

Send Feedback

10. Document Revision History for the Nios V Embedded Processor Design Handbook

Document Version	Intel Quartus Prime Version	Changes
2023.12.04	23.4	• Updated the note to refer to *AN 980: Nios V Processor Intel Quartus Prime Software Support* throughout the document. • Updated the titles and figures in *Instantiating Nios V Processor Intel FPGA IP* for the following Nios V processor cores: — Nios V/c Compact Microcontroller Processor Intel FPGA IP — Nios V/m Microcontroller Intel FPGA IP — Nios V/g General Purpose Processor Intel FPGA IP • Updated the table: *CPU Architecture* to remove atomic extensions. • Added the command for Intel Quartus Prime Standard Edition version in the following topics: — Table: *GUI Tools and Command-line Tools Tasks Summary*. — Topic: *Generating the Board Support Package* in *Creating Nios V Processor Software*. • Edited the title for *Example Design on Unimplemented Instruction (Custom Instruction Design on Nios® V/g processor)* to *Unimplemented Instruction Example Design*. • Added topic *Hardware Acceleration Example Designs*.
2023.10.02	23.3	• Added new topics based on new addition Nios V/c processor: — *Nios V Processor Licensing* — *Instantiating Nios V Processor IP Core* — *Instantiating Nios V/c Processor Intel FPGA IP* — *Instantiating Nios V/m Processor Intel FPGA IP* — *Instantiating Nios V/g Processor Intel FPGA IP* — *Debugging Nios V/c Processor* — *Steps to Debug Nios V/c Processor* • Updated *Nios V Processor Configuration and Booting Solutions* with TCM related in the following topics: — *Nios V Processor Booting Methods* — Added *Nios V Processor Application Execute-In-Place from TCM* — Added *Nios V Processor Booting from Tightly Coupled Memory* (TCM) — *Summary of Nios V Processor Vector Configuration and BSP Settings* • Updated the mention of Nios V/m to Nios V in related topics with the release of Nios V/g and Nios V/c processors.

continued...

Document Version	Intel Quartus Prime Version	Changes
2023.09.01	23.2	• Updated *Software Design Flow* in *Processor Application Executes-In-Place from Configuration QSPI Flash*: — Updated figures: • *Linker Region Settings When Exceptions is set to OCRAM / External RAM*. • *Linker Region Settings When Exceptions is set to QSPI Flash* . — Added steps to disable gsfi driver. • Added new section: *Nios V Processor RSU Quick Start Guide in SDM-based Devices*. • Updated figure *Nios V Processor System Design Flow* in the topic *Embedded System Design*.
2023.05.26	23.1	• Added links to *AN 980: Nios V Processor Intel Quartus Prime Software Support*. • Added a new section: *Nios V Processor — Using Custom Instruction*.
2023.04.10	23.1	• Added new topics: — *Caches and Peripheral Regions Tab* — *Custom Instruction Tab* • Added table *GSFI Bootloader for Nios V Processor Core* in the topic *GSFI Bootloader*. • Added a new step in the topic *Generating HEX File* from the section *Processor Application Copied from Configuration QSPI Flash to RAM Using Boot Copier (GSFI Bootloader)*. • Updated product family name to "Intel Agilex 7".

Document Version	Intel Quartus Prime Version	IP Version	Changes
2023.02.14	22.4	22.4.0	• Edited topic *Intel Quartus Prime Software Support*. • Edited topic *Nios V/m Processor Example Design*. • Added a note in the following topics to refer to the topic *Intel Quartus Prime Software Support* — *Generating the Board Support Package using the BSP Editor GUI* — *Nios V Board Support Package Editor* — *Software Design Flow* — *Creating a BSP project* • Updated the following topics to align with the design store migration steps: — *Generating the Application Project File* — *GSFI Bootloader Example Design* — *SDM Bootloader Example Design* — *MicroC/TCP-IP Example Designs: Overview* — *Acquiring the Example Design Files* — *Creating an Application Project* — *Device Programming* — *Optional Configuration* • Removed the following topics: — *Generating the Example Design Through Graphical User Interface* — *Generating the Nios V/m Processor Example Design Using the Command Line Interface* — *Generate Nios V processor example design from Platform Designer* — *HEX File Generation*
2022.10.31	22.1std	1.0.0	• Updated references from *Intel Quartus Prime Pro Edition* to Intel Quartus Prime to indicate support for both Pro and Standard Edition. • Added new topic: *Intel Quartus Prime Software Support*.
2022.10.25	22.3	22.3.0	• Added new section: *Nios V Processor — Remote System Update*.
2022.09.26	22.3	22.3.0	• Updated *Configure Nios V Processor Parameters* — Edited *Debug Tab* — Added *Use Reset Request Tab* — Edited *Vectors Tab*. Removed *Exception Agent* and *Exception Offset* • Updated the following figures: — *Nios V/m Processor IP instance in Platform Designer* — *Example connection of Nios V processor with other peripherals in Platform Designer* — *hal.linker Settings for QSPI Flash* — *Connections for Nios V Processor Project* — *hal.linker Settings* — *Linker Region Settings* — *hal.make Settings* — *BSP Driver tab* • Added *Enable Reset from Debug Module* to the following figures: — *Parameter Editor Settings* — *Nios V Parameter Editor Settings*

continued...

Document Version	Intel Quartus Prime Version	IP Version	Changes
			• Removed the mention of *exception vector*, *exception RAM*, *exception agent*, and *.exception* in the following topics: — *Defining System Component Design* — *Nios V Processor Design, Configuration and Boot Flow (Control Block-based Device)* — *Reset Agent Settings for Nios V Processor Execute-In-Place Method* — *Reset Agent Settings for Nios V Processor Boot-copier Method* — *Nios V Processor Design, Configuration and Boot Flow (SDM-based Devices)* — *Nios V Processor Application Copied from Configuration QSPI Flash to RAM Using Boot Copier (SDM Bootloader)* — Table: *Description of Memory Organization* — *Design, Configuration and Booting Flow* in *Nios V Processor Application Executes in-place from OCRAM* — Table: *Summary of Nios V Processor Vector Configurations and BSP Settings* • Edited *Configuring BSP Editor and Generating the BSP Project* in *Nios V Processor Design, Configuration and Boot Flow (Control Block-based Device)*. • Added Table: *Settings for BSP Editor* in *Software Design Flow (SDM Bootloader Project)*.
2022.08.12	22.2	21.3.0	• Edited the steps in *Programming Nios V/m into the FPGA Device*. • Edited Table: *Debug Tab Parameter* to add the description for *dbg_reset*. • Edited topic *On-Chip Memory Configuration - RAM or ROM* topic. Added a link to *Nios V Processor Application Execute-In-Place from OCRAM*.

continued...

Document Version	Intel Quartus Prime Version	IP Version	Changes
			• Changed the topic title from *Clocks and Resets* to *Clocks and Resets Best Practices*. • Added the following new topics: — *Reset Request Interface* — *Typical Use Cases* — *Assigning a Default Agent* • Added a note about configuring the RISC-V toolchain prefix in the topic *Eclipse CDT for Embedded C/C++ Developer*.
2022.06.21	22.2	21.3.0	• Added the support for RiscFree IDE for Intel FPGAs. • Removed the following topics: — *Setting Up Open-Source Tools* — *Building the Application Project using Eclipse Embedded CDT* — *Building the Application Project using the Command-Line Interface* — *Creating a Software Project using Platform Designer & Eclipse Embedded CDT* — *Creating a Software Project Using Command Line* • Edited the Figure : *Software Design Flow* to include RiscFree IDE for Intel FPGAs • Added the following topics: — *Nios V Software Development Flow* — *Board Support Package Project* — *Application Project* — *Intel FPGA Embedded Development Tools* — *Nios V Board Support Package Editor* — *RiscFree* IDE for Intel FPGA* — *Eclipse* CDT for Embedded C/C++ Developer* — *Nios V Utilities Tools* — *File Format Conversion Tools* — *Other Utilities Tools* — *Generating the Board Support Package* — *Generating the Application Project File* — *Building the Application Project*
2022.04.04	22.1	21.2.0	Initial release.