

AN 985: Nios[®] V Processor Tutorial



Online Version



Send Feedback

784468

2024.07.24

Contents

1. AN 985: Nios® V Processor Tutorial.....	3
1.1. Overview.....	3
1.1.1. Hardware and Software Requirements.....	3
1.2. Generating the System.....	4
1.2.1. Building Hardware Design in Platform Designer.....	4
1.2.2. Building Software Design with Ashling RiscFree IDE for Intel FPGAs.....	39
1.3. Simulating the Nios V Processor System.....	48
1.3.1. Preparing Hardware Design for Simulation.....	48
1.3.2. Preparing Software Design for Simulation.....	49
1.3.3. Checking Simulation Files.....	49
1.3.4. Running System Simulation.....	50
1.4. Programming the System.....	52
1.4.1. Programming Hardware SOF File.....	52
1.4.2. Downloading the Software ELF File.....	55
1.4.3. Applying External Tool Configuration.....	58
1.4.4. Run Hello World Application.....	59
1.5. Debugging the System.....	60
1.5.1. Software Debugging.....	60
1.6. Document Revision History for AN 985: Nios V Processor Tutorial.....	65



1. AN 985: Nios® V Processor Tutorial

1.1. Overview

This application note is a fundamental guide for you to get started with building a Nios® V processor system and running a simple Hello World program. The target boards are the Intel FPGA development kits. The build software includes both Quartus® Prime software and Ashling* RiscFree* IDE for Intel® FPGAs.

The document provides the following information:

- Introduces you to the basic design flow for the Nios V processor.
- Provides instructions on how to generate a Nios V example design, create a software program, and run the program on an Intel FPGA device.

Related Information

- [Nios V Processor Reference Manual](#)
- [Nios V Processor Intel FPGA IP Release Notes](#)
- [Nios V Processor Software Developer Handbook](#)
- [Nios V Embedded Processor Design Handbook](#)

1.1.1. Hardware and Software Requirements

Hardware Requirements

- Any Altera® FPGA Development Kits
Note: This tutorial uses Agilex™ 7 FPGA F-Series Transceiver-SoC Development Kit (DK-SI-AGF014EA).
- Power adapter
- Intel FPGA Download Cable II

Software Requirements

- Quartus Prime Pro Edition/Quartus Prime Standard Edition Software
- Ashling RiscFree IDE for Intel FPGAs
- Questa* Intel FPGA Edition

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

Related Information

- [Agilex 7 FPGA F-Series Transceiver-SoC Development Kit \(P-Tile and E-Tile\)](#)
- [Quartus Prime Pro Edition User Guide: Getting Started](#)

© Altera Corporation. Altera, the Altera logo, the 'a' logo, and other Altera marks are trademarks of Altera Corporation. Altera and Intel warrant performance of its FPGA and semiconductor products to current specifications in accordance with Altera's or Intel's standard warranty as applicable, but reserves the right to make changes to any products and services at any time without notice. Altera and Intel assume no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera or Intel. Altera and Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

*Other names and brands may be claimed as the property of others.

- [Quartus Prime Standard Edition User Guide: Getting Started](#)
- [Ashling* RiscFree Integrated Development Environment \(IDE\) for Intel FPGAs User Guide](#)
- [Questa*-Intel FPGA Edition Quick-Start: Quartus Prime Pro Edition](#)
- [Questa*-Intel FPGA Edition Quick-Start: Quartus Prime Standard Edition](#)

1.2. Generating the System

The implementation of Intel FPGA devices requires a hardware system developed using the Quartus Prime software. The Nios V processor requires an additional software system that complements the processor hardware system. You can develop a Nios V processor software system that is compatible to your Nios V processor hardware system using Ashling RiscFree IDE for Intel FPGAs.

1.2.1. Building Hardware Design in Platform Designer

Begin designing the system hardware by instantiating the Nios V processor and its peripherals into Platform Designer. After configuring the system assignments and constraints, complete the hardware design by performing a successful compilation.

Table 1. 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 II Intel FPGA IP	Stores data and instructions.
Reset Release Intel FPGA IP	Recommended reset output in SDM-based devices.

You can design the system hardware by using one of the following methods:

- Manual instantiation
- Board-aware flow
- Configurable example design

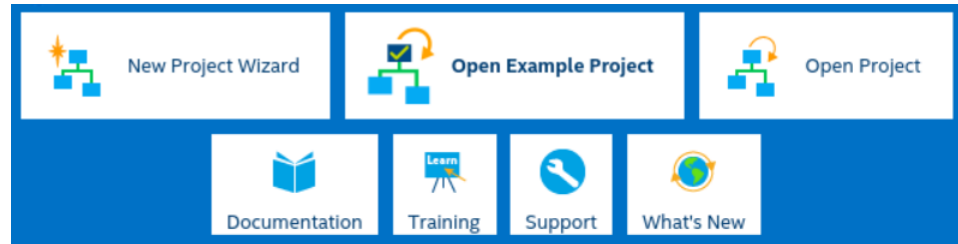
1.2.1.1. Manual Instantiation

1.2.1.1.1. Creating a New Project

Start developing the Nios V processor system by creating a new Quartus Prime software project.

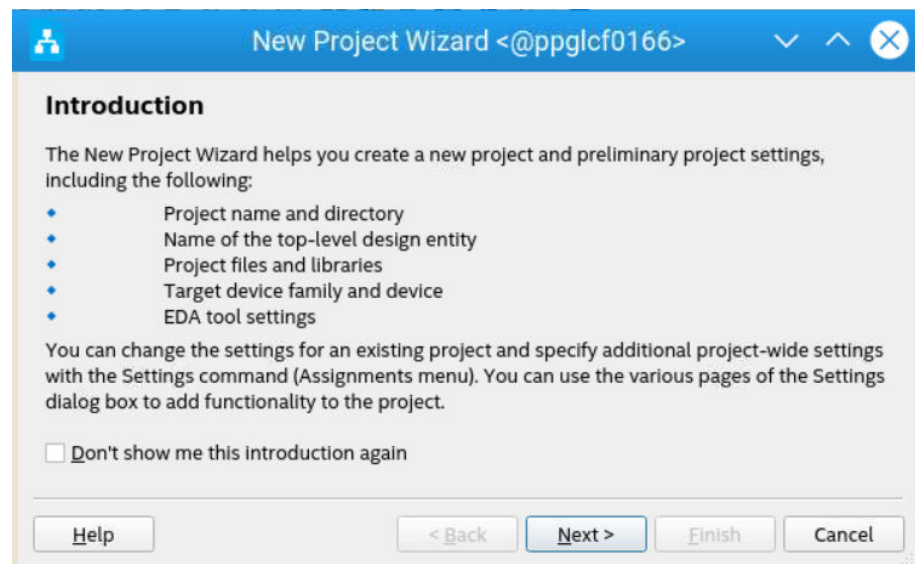
1. Launch **Quartus Prime Pro Edition** software.
2. Click **New Project Wizard** to create a new project.

Figure 1. New Project Wizard



3. Read the **Introduction** and click **Next** to proceed.

Figure 2. New Project Wizard — Introduction



4. In **Family, Device & Board Settings** window,
 - a. Select **Empty project**.

Figure 3. Selecting Project Settings

Family, Device & Board Settings

▼ Project Settings

Select the type of project to create.

☒ **Empty project**
Create new project by specifying project files and libraries, target device family and device, and EDA tool settings.

☐ **Design example**
Create a project from an existing design example. You can choose from design examples installed with the Intel Quartus Prime software, or download design

- b. Enter your working directory, define the name of the project, and enter the top-level design entity as `niosv_top`.

Figure 4. Specifying the Properties of the Project

Specify the properties of the project.

What is the working directory for this project?

/nfs/png/disks/ceg_user_liangyug/users/Projects/NiosVm/first-niosv

What is the name of this project?

niosv_top

What is the name of the top-level design entity for this project? This name is case sensitive and must exactly match the entity name in the design file.

niosv_top

☐ This project uses a Partition Database (.qdb) file for the root partition

Use Existing Project Settings...

- c. Filter **Device Name** as **AGFB014R24B2E2V** which is for Agilex 7 FPGA F-Series Transceiver-SoC Development Kit.

Figure 5. Device Tab

Name	Tile	Core Voltage	ALMs
88 AGFB014R24B2E2V	P-Tile + E-Tile	VID	487200

- d. Click **Next** to proceed.
5. In **Add Files** window, leave it empty and click **Next** to proceed.

Figure 6. Add Files Window

6. In **EDA Tool Settings** windows, select **Questa Intel FPGA**. (This is for simulation in later chapter.)

Figure 7. EDA Tool Settings

Tool Type	Tool Name	Format(s)
Design Entry/Syn...	<None>	<None>
Simulation	Questa Intel FPGA	Verilog HDL
Board-Level	Signal Integrity	<None>

- Click **Next** to review the summary of the new project. Then click **Finish**.

Figure 8. Summary

When you click Finish, the project will be created with the following settings:

Project directory: /nfs/png/disks/ceg_user_liangyug/users/Projects/NiosVm/first-niosv

Project name: niosv_top

Top-level design entity: niosv_top

Number of files added: 0

Number of user libraries added: 0

Device assignments:

Design template: n/a

Family name: Agilex 7 (F-Series/I-Series)

Device: AGFB014R24B2E2V

Board:

EDA tools:

Design entry/synthesis: <None> (<None>)

Simulation: Questa Intel FPGA (Verilog HDL)

Timing analysis: ()

Operating conditions:

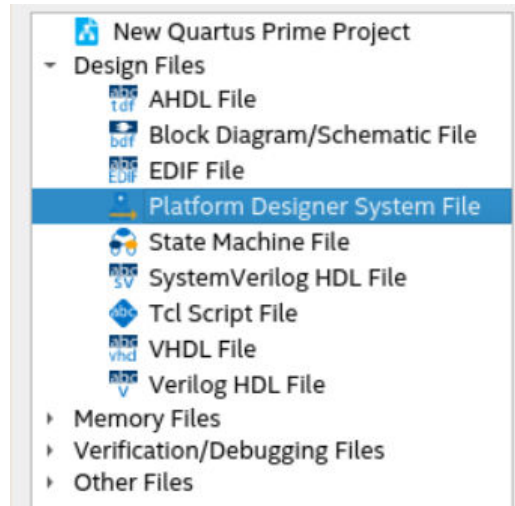
Core voltage: VID2

Junction temperature range: 0-100 °C

1.2.1.1.2. Creating a Platform Designer System

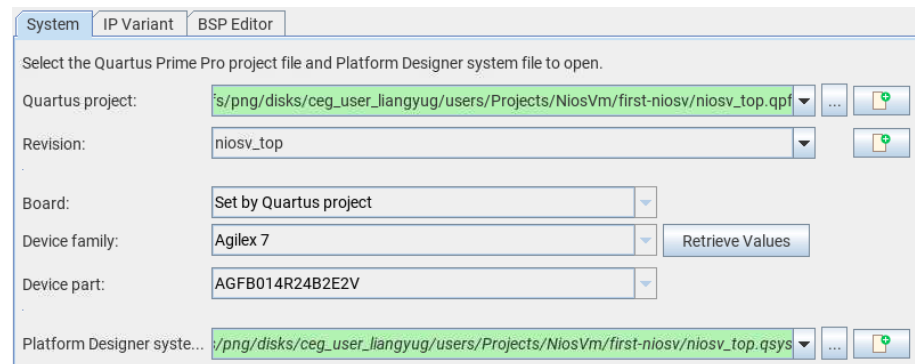
- Click **New**, select **Platform Designer System File**, and click **OK**.

Figure 9. New Window



2. In the **Open System** window, check the project information.
 - a. **Quartus project:** <Working directory>/niosv_top.qpf
 - b. **Revision:** niosv_top
 - c. **Device family:** Agilex 7
 - d. **Device part:** AGFB014R24B2E2V
 - e. **Platform Designer system:** Click **Create new Platform Designer System** icon, and name the QSYS file as niosv_top.

Figure 10. Open System Window



3. Click **Create**.
4. In the Platform Designer system, the software instantiates the Clock Bridge and Reset Bridge Intel FPGA IP by default.
5. In Clock Bridge IP, configure **Explicit clock rate** as 100000000 Hz (100 MHz).
6. In Reset Bridge IP, enable it as an **Active low reset**.

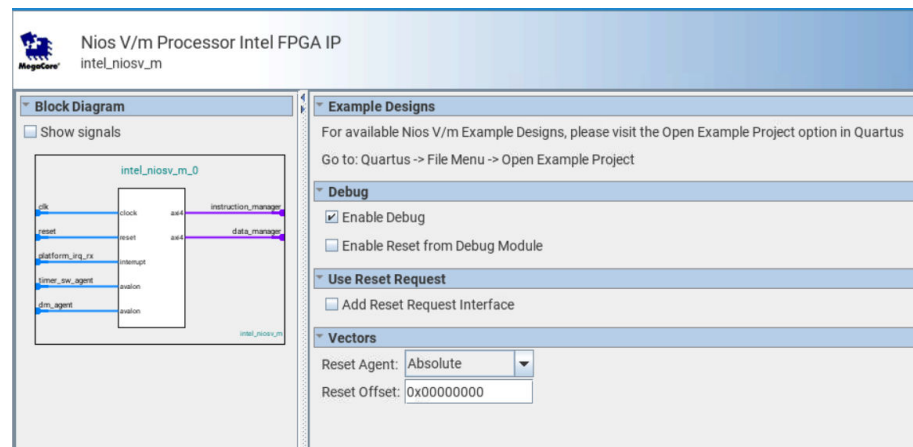
Figure 11. Default Platform Designer System

Conn...	Name	Description	Export	Clock
	clock_in	Clock Bridge Intel FPGA IP		
	in_clk	Clock Input	clk	exported
	out_clk	Clock Output	Double-click to export	clock_in_...
	reset_in	Reset Bridge Intel FPGA IP		
	clk	Clock Input	Double-click to export	clock_in_...
	in_reset	Reset Input	reset	[clk]
	out_reset	Reset Output	Double-click to export	[clk]

Adding Nios V/m Processor Intel FPGA IP

1. Search for **Nios V/m Processor** in the **IP Catalog**.
2. Add the **Nios V/m Processor Intel FPGA IP** under **Processors and Peripherals** ► **Embedded Processors** section. The New IP Variation window appears.
3. Click **Finish** to instantiate the processor. Leave it at the default settings.

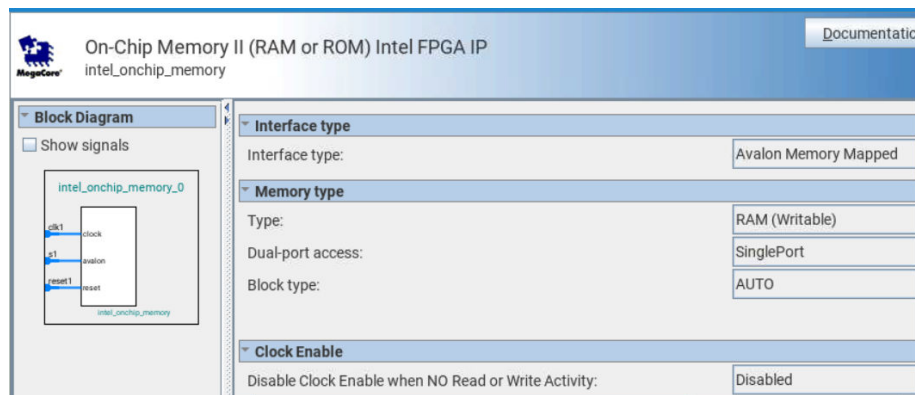
Figure 12. Nios V/m Processor IP Parameter Editor



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

1. Search for **On-Chip Memory** in the **IP Catalog**.
2. Add the **On-Chip Memory II (RAM or ROM) Intel FPGA IP** under **Basic Functions** ► **On Chip Memory**. The New IP Variation window appears.
3. Configure the **Total memory size** as **262144**.
4. Turn on the option **Enable non-default initialization file** and provide the filename `hello.hex`.
5. Leave other settings at default.
6. Click **Finish** to instantiate the peripheral.

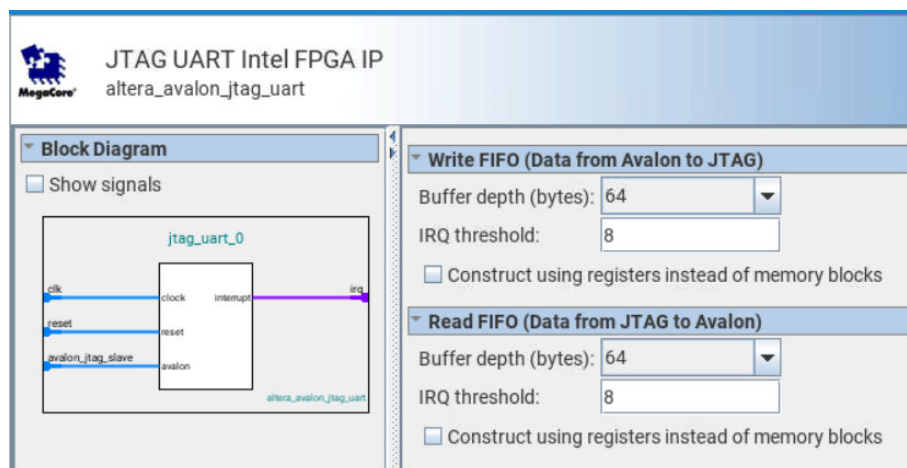
Figure 13. On-Chip Memory II (RAM or ROM) IP Parameter Editor



Adding JTAG UART Intel FPGA IP

1. Search for **JTAG UART** in the **IP Catalog**.
2. Add the **JTAG UART Intel FPGA IP** under **Interface Protocols > Serial** section. The New IP Variation window appears.
3. Click **Finish** to instantiate the peripheral. Leave it at the default settings.

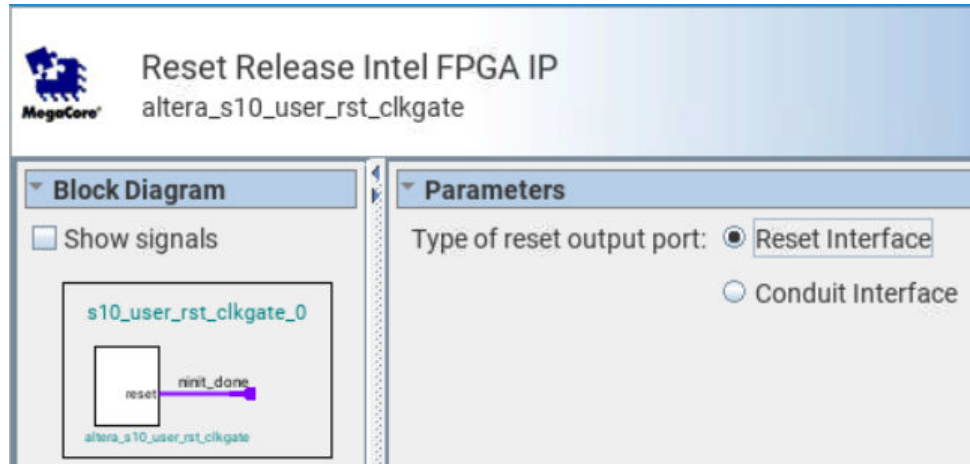
Figure 14. JTAG UART IP Parameter Editor



Adding Reset Release Intel FPGA IP

1. Search for **Reset Release** in the **IP Catalog**.
2. Add the **Reset Release Intel FPGA IP** under **Basic Functions > Configuration and Programming** section. The New IP Variation window appears.
3. Select the type of reset output port as **Reset Interface**.
4. Click **Finish** to instantiate the peripheral.

Figure 15. Reset Release IP Parameter Editor



Connect Interfaces and Signals

Connect the clock bridge, reset bridge, reset release IP, and Nios V processor to the peripherals.

Table 2. Connection between Host and Agent

IP	Host	Peripheral
Clock Bridge IP	out_clk	intel_niosv_m_0.clk
		intel_onchip_memory_0.clk1
		jtag_uart_0.clk
Reset Bridge IP	out_reset	intel_niosv_m_0.reset
		intel_onchip_memory_0.reset1
		jtag_uart_0.reset
Reset Release IP	ninit_done	intel_niosv_m_0.reset
		intel_onchip_memory_0.reset1
		jtag_uart_0.reset
Nios V/m Processor IP	platform_irq_rx	jtag_uart_0.irq
	instruction_manager	intel_onchip_memory_0.s1
	data_manager	intel_onchip_memory_0.s1
		jtag_uart_0.avalon_jtag_slave

Figure 16. Full System Connection

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 FP...
	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
	intel_onchip_memory_0	On-Chip Memory II (RAM or R...
	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_user_rst_clkgate_0	Reset Release Intel FPGA IP
	ninit_done	Reset Output

Clear System Warnings and Errors

1. Navigate to the **System** menu bar.
2. Click **Assign Base Address**.

Figure 17. Example of System Connectivity Issues

top.intel_niosv_m_0.instruction_manager	intel_onchip_memory_0.s1 (0x0..0xffff) overlaps intel_niosv_m_0.dm_agent (0x0..0xffff)
top.intel_niosv_m_0.data_manager	intel_onchip_memory_0.s1 (0x0..0xffff) overlaps intel_niosv_m_0.dm_agent (0x0..0xffff)
top.intel_niosv_m_0.data_manager	jtag_uart_0.avalon_jtag_slave (0x0..0x7) overlaps intel_onchip_memory_0.s1 (0x0..0xffff)

Configuring the Reset Vector of the Nios V Processor

1. Select the Nios V Processor IP and open the IP Parameter Editor.
2. Navigate to the **Vectors** tab.
3. Configure as follows:

- **Reset Agent:** intel_onchip_memory_0.s1
- **Reset Offset:** 0x0

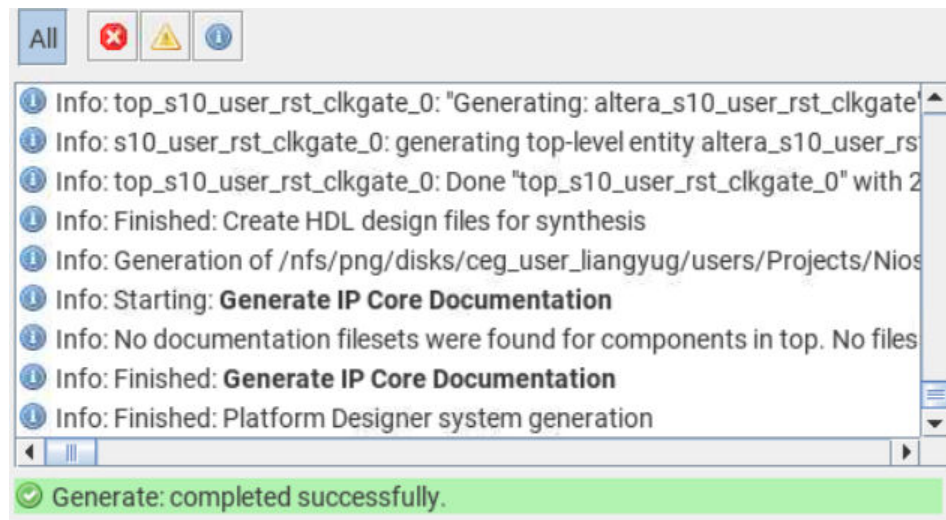
Figure 18. Reset Vector

Vectors	
Reset Agent:	intel_onchip_memory_0.s1
Reset Offset:	0x00000000

Saving and Generating System HDL









1. Save the system.
2. Click **Generate HDL** at the bottom right corner of the Platform Designer.
3. Leave all settings at default, and click **Generate**.
4. Ensure that the Nios V processor hardware system is successfully generated.

Figure 19. Successful Generation



5. The Platform Designer generates a folder named `niosv_top`, which stores the system generation files.
6. Exit the Platform Designer, and return to the project front page.

Figure 20. Generated Project Files








	DNI	File folder	
	ip	File folder	
	niosv_top	File folder	
	qdb	File folder	
	niosv_top.qpf	QPF File	2 KB
	niosv_top.qsf	QSF File	2 KB
	niosv_top.qsys	QSYS File	312 KB
	niosv_top.qsys.legacy	LEGACY File	230 KB

1.2.1.1.3. Configuring Assignment and Constraint

Adding Design Files

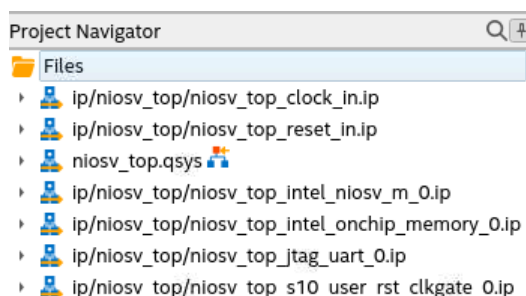
1. In Quartus Prime software, navigate to **Assignments** menu bar and click **Settings**.
2. Navigate to **Files** category. You can find all related IP files and QSYS file added into your project.

Figure 21. Settings – Files Category

File Name	Type
 ip/niosv_top/niosv_top_clock_in.ip	IP File
 ip/niosv_top/niosv_top_reset_in.ip	IP File
 niosv_top.qsys	Platform Designer System File
 ip/niosv_top/niosv_top_intel_niosv_m_0.ip	IP File
 ip/niosv_top/niosv_top_intel_onchip_memory_0.ip	IP File
 ip/niosv_top/niosv_top_jtag_uart_0.ip	IP File
 ip/niosv_top/niosv_top_s10_user_rst_clkgate_0.ip	IP File

3. Earlier during the new project creation, the top-level design entity is named niosv_top. Thus, the niosv_top.qsys file is automatically the top-level design entity.
4. Check the top-level design entity assignment in the **Project Navigator**.

Figure 22. Project Navigator



Adding Synopsys Design Constraint (SDC) File

1. Click **New**, select **Synopsys Design Constraint File** and click **OK**.
2. Add the following constraint:

```
create_clock -name clk -period 10.0 [get_ports clk_clk]
```

3. Save as `niosv_top.sdc`.

Pin Assignments

1. In Quartus Prime software, navigate to **Processing** menu bar and click **Start ► Start Analysis & Elaboration**.
2. Once the analysis is complete, navigate to **Assignments** menu bar and click **Pin Planner**. For this example, there are 2 pins assignments:
 - `clk_clk` assigned to `PIN_U52 (FPGA_SYSTEM_CLK)`
 - `reset_reset_n` assigned to `PIN_G52 (FPGA_SYS_RESETh)`
3. Close **Pin Planner**, and return to the project front page.

Configuration and Power Management Assignments

1. In Quartus Prime software, navigate to **Assignments** menu bar and click **Device ► Device and Pin Options**.
2. Navigate to **Configuration** category.
3. Select **VID mode of operation** as **PMBus Master**.
4. Click **Configuration Pin Options**, and make the following **Configuration pin** assignments,
 - **PWRMGT_SCL** as `SDM_IO0`
 - **PWRMGT_SDA** as `SDM_IO12`
 - **CONF_DONE** as `SDM_IO16`
 - Leave the rest empty.
5. Navigate to **Power Management & VID** category, and make the following settings
 - **Bus speed mode** as `100 kHz`
 - **Slave device type** as `Other`
 - **PMBus device 0 slave address** as `42`
 - **Voltage output format** as `Linear format`
 - **Linear format N** as `-13`
 - **Translated voltage value unit** as `Volts`
 - Turn off **PAGE command**.
6. Click **OK**, and return to the project front page.

1.2.1.1.4. Compiling the Quartus Prime Software Project

1. Navigate to **Processing** menu bar and click **Start Compilation**.
2. Make sure that the Quartus Prime project compiles successfully.

1.2.1.2. Board-Aware Flow

Related Information

AN 988: Using the Board-Aware Flow: in the Intel® Quartus® Prime Pro Edition Software

For more information on creating new Board file and new IP Presets.

1.2.1.2.1. Creating a New Project

Start developing the Nios V processor system by creating a new Quartus Prime software project.

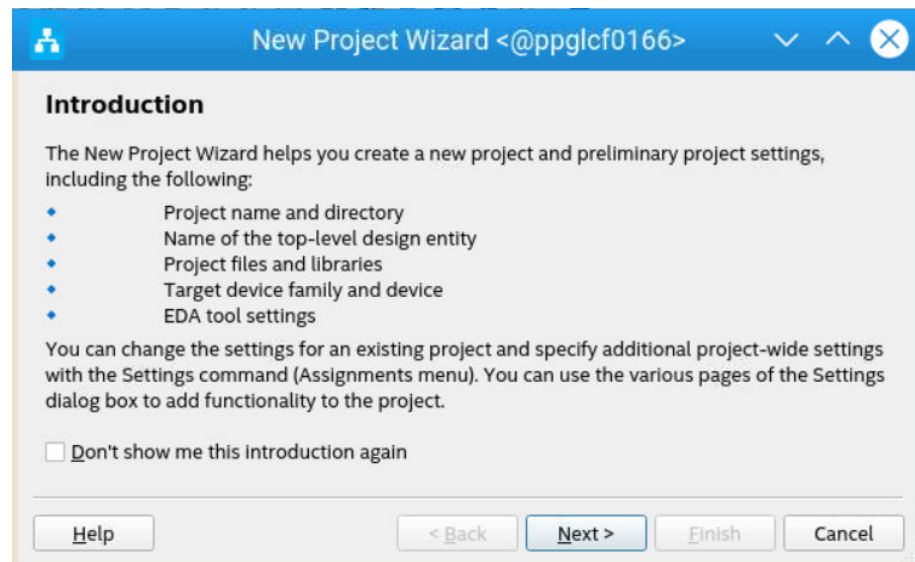
1. Launch **Quartus Prime Pro Edition** software.
2. Click **New Project Wizard** to create a new project.

Figure 23. New Project Wizard



3. Read the **Introduction** and click **Next** to proceed.

Figure 24. New Project Wizard – Introduction



4. In **Directory, Name, Top-Level Entity** window,
 - a. Select **Empty project**.

Figure 25. Selecting Project Settings

Directory, Name, Top-Level Entity

Select the type of project to create.

☒ Empty Project
Create new project by specifying project files and libraries, target device family and device, and EDA tool settings.

☐ Design Example
Create a project from an existing design example. You can choose from design examples installed with the Intel Quartus Prime software, or download design examples from the [Design Store](#).

- b. Enter your working directory, define the name of the project, and enter the top-level design entity as `niosv_top`. Click **Next** to proceed.

Figure 26. Specifying the Properties of the Project

What is the working directory for this project?
board_aware_flow/related_985/design_example/board_aware_flow ...

What is the name of this project?
niosv_top| ...

What is the name of the top-level design entity for this project? This name is case sensitive and must exactly match the entity name in the design file.
niosv_top ...

☐ This project uses a Partition Database (.qdb) file for the root partition
...

Use Existing Project Settings...

- c. Filter **Device Name** as **AGFB014R24B2E2V** which is for Agilex 7 FPGA F-Series Transceiver-SoC Development Kit.

Figure 27. Device Tab

Family, Device & Board Settings

Device **Board**

Select the family and device you want to target for compilation. You can install additional device support with the Install Devices command on the Tools menu.

Device family

Family: Agilex 7 (F-Series/M-Series/I-Series)

Device: All

Target device

☒ Specific device selected in 'Available devices' list

☐ Other: n/a

Filter: Name AGFB014R24B2E2V

	Name	Tile	Core Voltage	ALMs	Total I/Os	GPIOs	HSSI Channels	Memory Bit
789	AGFB014R24B2E2V	1 P-Tile + 1 E-Tile	VID	487200	924	768	32	145612800

Available devices: 1/2080

Help < Back **Next >** Finish Cancel

- d. Click **Next** to proceed.
5. In **Add Files** window, leave it empty and click **Next** to proceed.

Figure 28. Add Files Window

Add Files

Select the design files you want to include in the project. Click Add All to add all design files in the project directory to the project.

Note: you can always add design files to the project later.

File name:

Q <<Filter>> X

File Name	Type	Library	Entity	Design Entry/Synthesis Tool	HDL Version

Specify the path names of any non-default libraries.

Help < Back **Next >** Finish Cancel

6. In **EDA Tool Settings** windows, select **Questa Intel FPGA**. The selection is for simulation in the topic *Simulating the Nios V Processor*.

Figure 29. EDA Tool Settings

Tool Type	Tool Name	Format(s)
Design Entry/Syn...	<None>	<None>
Simulation	Questa Intel FPGA	Verilog HDL
Board-Level	Signal Integrity	<None>

- Click **Next** to review the summary of the new project. Then click **Finish**.

Figure 30. Summary

When you click Finish, the project will be created with the following settings:

Project directory: /niosv/niosvm/board_aware_flow/related_985/design_example/board_aware_flow

Project name: niosv_top

Top-level design entity: niosv_top

Number of files added: 0

Number of user libraries added: 0

Device assignments:

Design template: n/a

Family name: Agilix 7 (F-Series/M-Series/I-Series)

Device: AGFB014R24B2E2V

Board:

Options:

EDA tools:

Design entry/synthesis: <None> (<None>)

Simulation: Questa Intel FPGA (Verilog HDL)

Timing analysis: ()

Operating conditions:

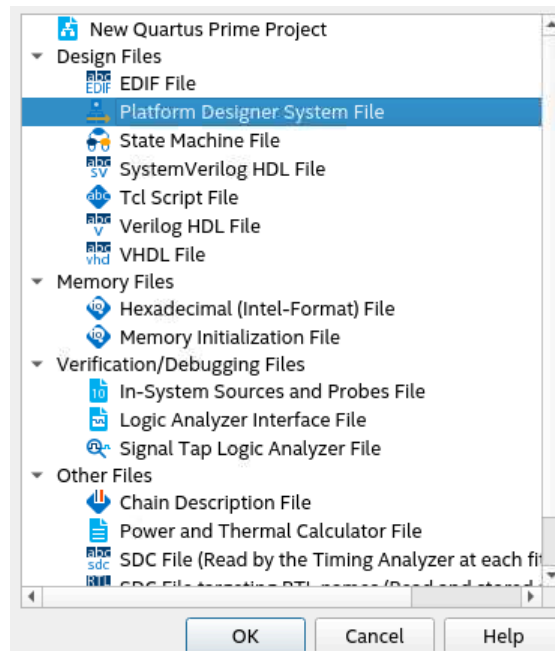
Core voltage: VID2

Junction temperature range: 0-100 °C

1.2.1.2.2. Creating a Platform Designer System

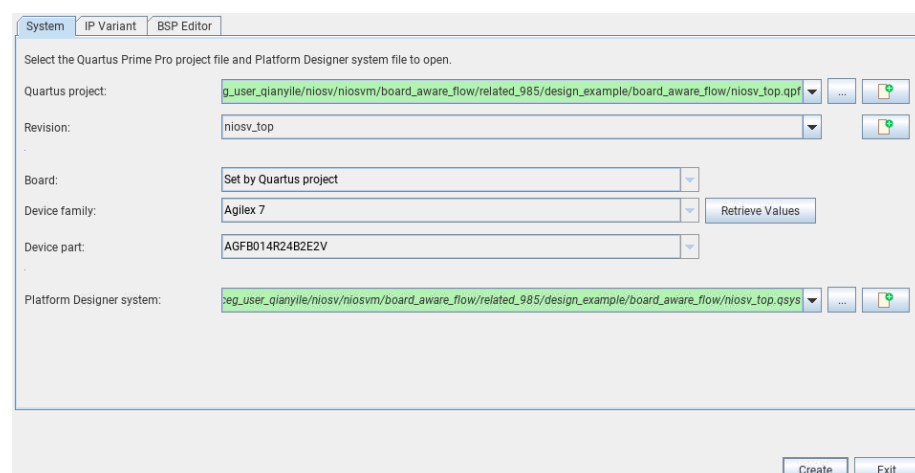
- Click **New**, select **Platform Designer System File**, and click **OK**.

Figure 31. New Window



2. In the **Open System** window, check the project information.
 - a. **Quartus project:** <Working directory>/niosv_top.qpf
 - b. **Revision:** niosv_top
 - c. **Device family:** Agilex 7
 - d. **Device part:** AGFB014R24B2E2V
 - e. **Platform Designer system:** Click **Create new Platform Designer System** icon, and name the QSYS file as niosv_top.

Figure 32. Open System Window



3. Click **Create**.

4. In the Platform Designer system, the software instantiates the Clock Bridge and Reset Bridge Intel FPGA IP by default.
5. In Clock Bridge IP, configure **Explicit clock rate** as 100000000 Hz (100 MHz).
6. In Reset Bridge IP, enable it as an **Active low reset**.

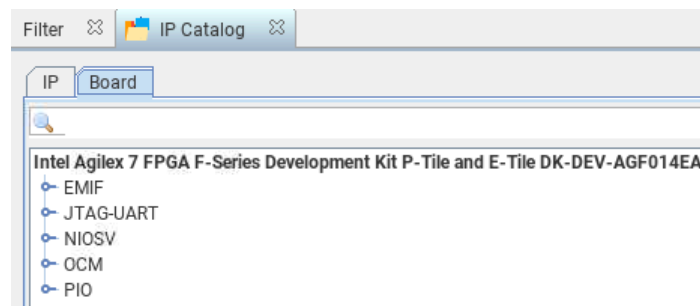
Figure 33. Default Platform Designer System

Conn...	Name	Description	Export	Clock
	clock_in	Clock Bridge Intel FPGA IP		
	in_clk	Clock Input	clk	exported
	out_clk	Clock Output	Double-click to export	clock_in_...
	reset_in	Reset Bridge Intel FPGA IP		
	clk	Clock Input	Double-click to export	clock_in_...
	in_reset	Reset Input	reset	[clk]
	out_reset	Reset Output	Double-click to export	[clk]

Adding Nios V/m Processor Intel FPGA IP

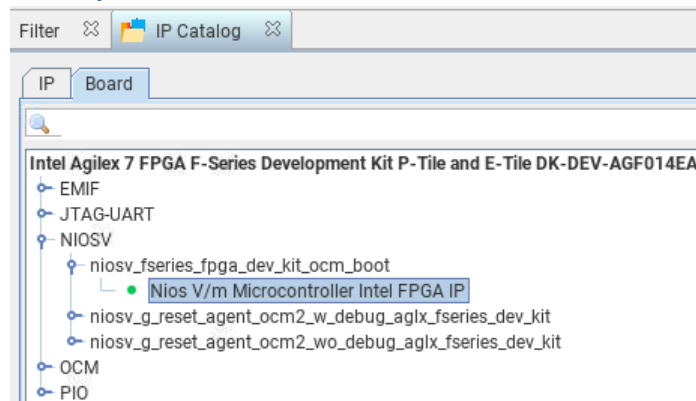
1. Navigate to **IP Catalog** and click **Board**.
2. Expand **NIOSV** and **niosv_fseries_fpga_dev_kit_ocm_boot**.

Figure 34. IP Catalog - Board



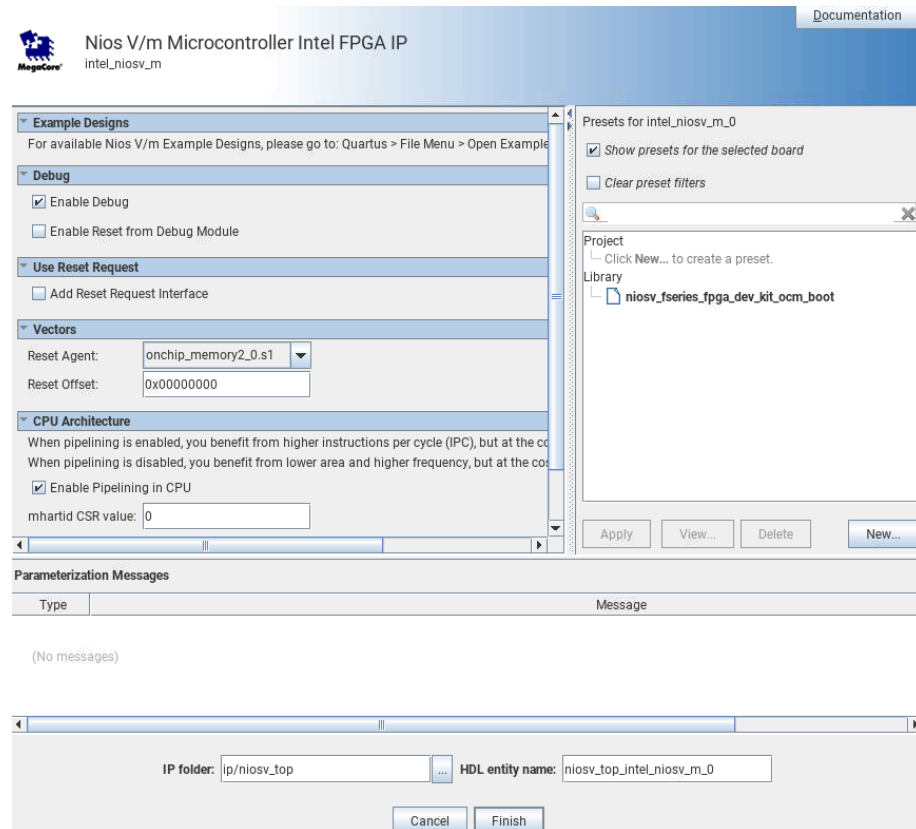
3. Double-click **Nios V/m Microcontroller Intel FPGA IP**. The New IP Variation window appears.

Figure 35. Nios V/m Processor Intel FPGA IP



4. Click **Finish** to instantiate the processor. Leave it at the default settings.

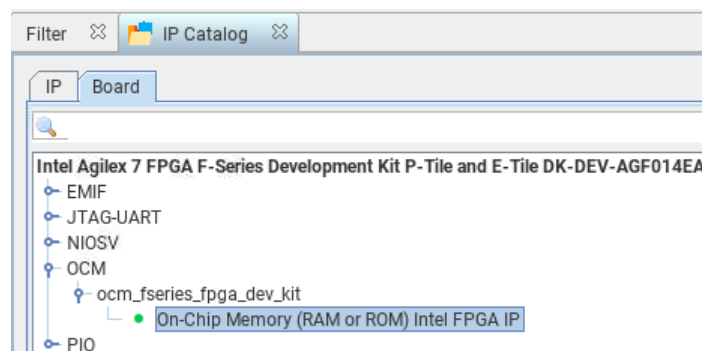
Figure 36. Nios V/m Processor IP Parameter Editor



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

1. In **IP Catalog**, expand **OCM** and **ocm_fseries_fpga_dev_kit**.
2. Double click **On-Chip Memory (RAM or ROM) Intel FPGA IP**. The New IP Variation window appears.

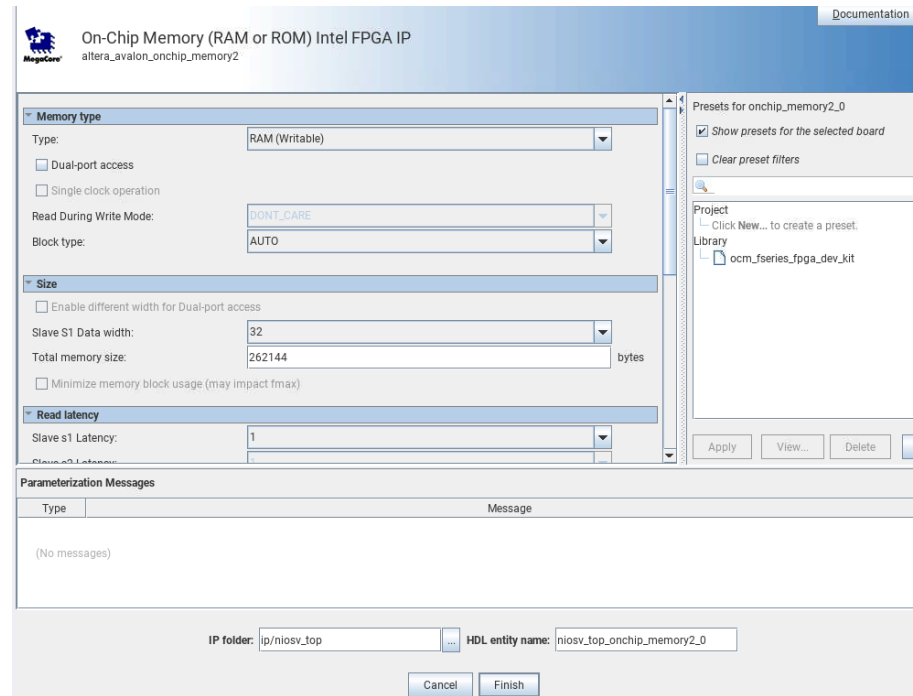
Figure 37. On-Chip Memory (RAM or ROM) Intel FPGA IP



3. Configure the **Total memory size** as **262144**.

4. Navigate to **Memory initialization**, enable **Initialize memory content** and **Enable non-default initialization file**. Provide the filename `hello.hex`.
5. Leave other settings at default.
6. Click **Finish** to instantiate the peripheral.

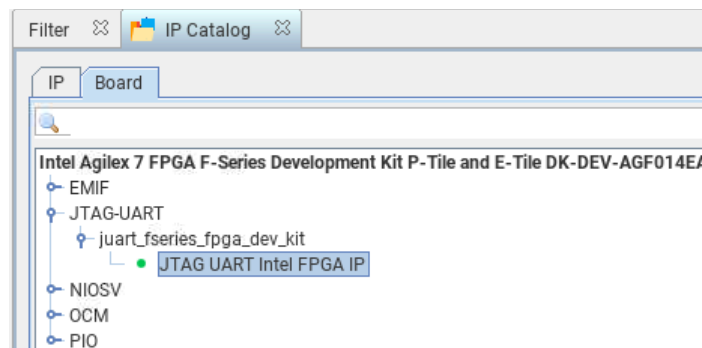
Figure 38. On-Chip Memory (RAM or ROM) IP Parameter Editor



Adding JTAG UART Intel FPGA IP

1. In **IP Catalog**, expand **JTAG-UART** and **juart_fseries_fpga_dev_kit**.
2. Double-click **JTAG UART Intel FPGA IP**. The New IP Variation window appears.

Figure 39. JTAG UART Intel FPGA IP



3. Click **Finish** to instantiate the processor. Leave it at the default settings.

Figure 40. JTAG UART IP Parameter Editor

Documentation

JTAG UART Intel FPGA IP
altera_avalon_jtag_uart

Write FIFO (Data from Avalon to JTAG)

Buffer depth (bytes): 64

IRQ threshold: 8

☐ Construct using registers instead of memory blocks

Read FIFO (Data from JTAG to Avalon)

Buffer depth (bytes): 64

IRQ threshold: 8

☐ Construct using registers instead of memory blocks

Presets for jtag_uart_0

☒ Show presets for the selected board

☐ Clear preset filters

Project

Click New... to create a preset.

Library

juart_fseries_fpga_dev_kit

Apply View... Delete New...

Parameterization Messages

Type	Message
(No messages)	

IP folder: ip/niosv_top HDL entity name: niosv_top_jtag_uart_0

Cancel Finish

Adding System ID Peripheral Intel FPGA IP

1. Search for **System ID Peripheral Intel FPGA IP** in the **IP Catalog**.
2. Add the **System ID Peripheral Intel FPGA IP** under **Basic Functions** > **Simulation; Debug and Verification** > **Debug and Performance** section. The New IP Variation window appears.
3. Click **Finish** to instantiate the processor. Leave it at the default settings.

Figure 41. System ID Peripheral IP Parameter Editor

Documentation

System ID Peripheral Intel FPGA IP
altera_avalon_sysid_qsys

Parameters

32 bit System ID:

Description

Please use hexadecimal numbers only in System ID.

Parameterization Messages

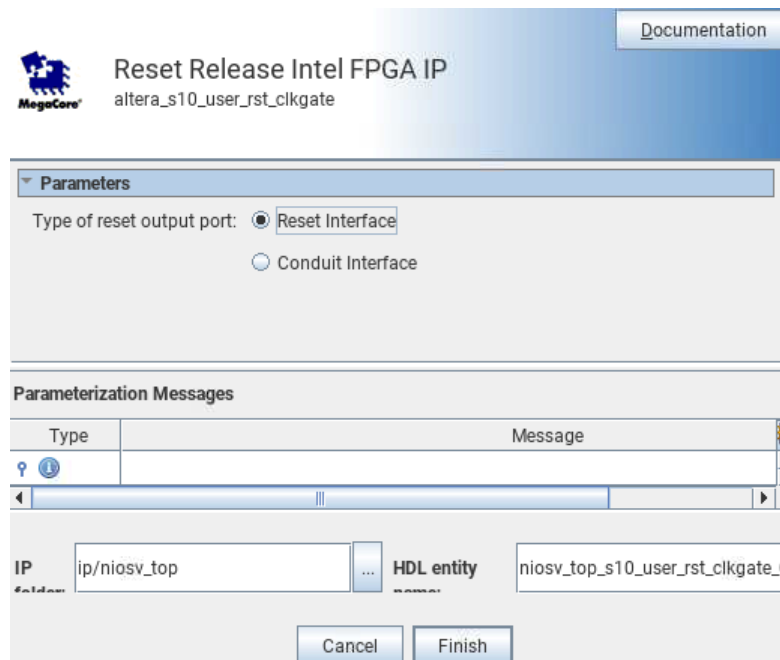
Type	Message
(No messages)	

IP **HDL entity**

Adding Reset Release Intel FPGA IP

1. Search for **Reset Release Intel FPGA IP** in the **IP Catalog**.
2. Add the **Reset Release Intel FPGA IP** under **Basic Functions** ➤ **Configuration and Programming** section. The New IP Variation window appears.
3. Select the type of reset output port as **Reset Interface**.
4. Click **Finish** to instantiate the processor.

Figure 42. Reset Release IP Parameter Editor



Connect Interfaces and Signals

Connect the Clock Bridge IP, Reset Bridge, Reset Release IP and Nios V/m processor to the peripheral.

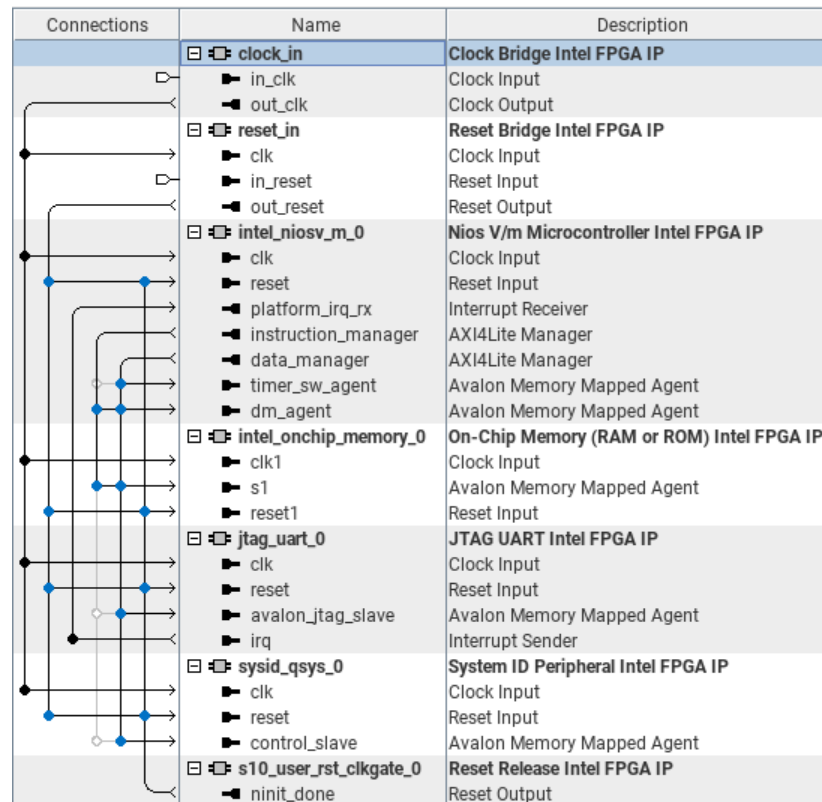
Table 3. Connection between Host and Agent

IP	Host	Peripheral
Clock Bridge IP	out_clk	intel_niosv_m_0.clk
		onchip_memory2_0.clk1
		sysid_qsys_0.clk
		jtag_uart_0.clk
Reset Bridge IP	out_reset	intel_niosv_m_0.reset
		onchip_memory2_0.reset1
		sysid_qsys_0.reset
		jtag_uart_0.reset
Reset Release IP	ninit_done	intel_niosv_m_0.reset
		onchip_memory2_0.reset1
		sysid_qsys_0.reset
		jtag_uart_0.reset
Nios V/m Processor IP	platform_irq_rx	jtag_uart_0.irq
	instruction_manager	onchip_memory2_0.s1

continued...

IP	Host	Peripheral
	data_manager	onchip_memory2_0.s1
		sysid_qsys_0.control_slave
		jtag_uart_0.avalon_jtag_slave

Figure 43. Full System Connection



Clear System Warnings and Errors

1. Navigate to the **System** menu bar.
2. Click **Assign Base Addresses**.






Figure 44. Example of System Connectivity Issues

Type	Path	Message
2 Component Instantiation Warnings		
	niosv_top.reset_in	System Information doesn't match requirements of IP. Double-click to open System Info tab.
	niosv_top.intel_niosv_m_0	Errors found in IP parameterization.

Saving and Generating System HDL

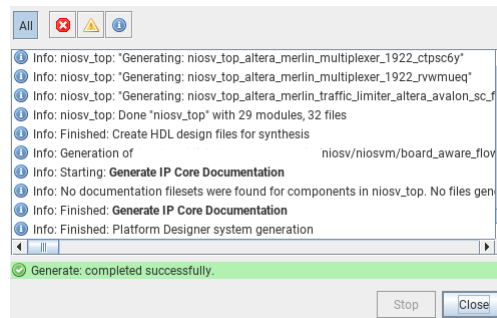
1. Save the system.
2. Click **Generate HDL** at the bottom right corner of the Platform Designer.
3. Leave all settings at default and click **Generate**.
4. Exit the Platform Designer and return to the project front page.

Figure 45. Generated Project Files

 dni	File folder
 ip	File folder
 niosv_top	File folder
 qdb	File folder
 software	File folder

Ensure that the Nios V processor hardware system is successfully generated.

Figure 46. Successful Generation



The Platform Designer generates a folder name `niosv_top`, which stores the system generation files.

1.2.1.2.3. Configuring Assignment and Constraint

Adding Design Files

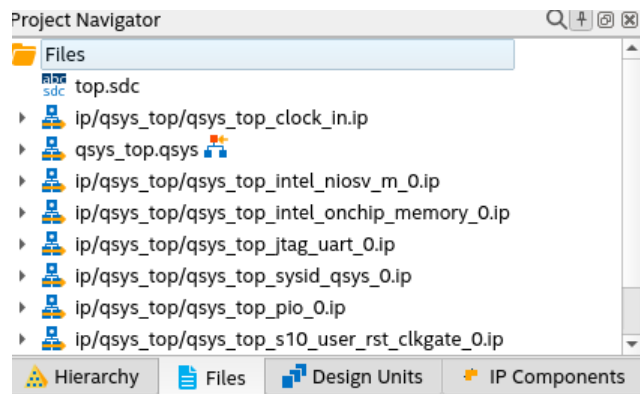
1. In Quartus Prime software, navigate to **Assignments** menu bar and click **Settings**.
2. Navigate to **Files** category. You can find all related IP files and QSYS file added into your project.

Figure 47. Settings – Files Category

File Name	Type
ip/niosv_top/niosv_top_clock_in.ip	IP File
ip/niosv_top/niosv_top_reset_in.ip	IP File
niosv_top.qsys	Platform Designer System File
ip/niosv_top/niosv_top_intel_niosv_m_0.ip	IP File
ip/niosv_top/niosv_top_onchip_memory2_0.ip	IP File
ip/niosv_top/niosv_top_jtag_uart_0.ip	IP File
ip/niosv_top/niosv_top_sysid_qsys_0.ip	IP File
ip/niosv_top/niosv_top_s10_user_rst_clkgate_0.ip	IP File

- Earlier during the project creation, the top-level design entity is named **top**. Thus, the **niosv_top.qsys** file is automatically the top-level design entity.
- Check the top-level design entity assignment in the **Project Navigator**.

Figure 48. Project Navigator



Adding Synopsys Design Constraint (SDC) File

- Click **New**, select **Synopsys Design Constraint File** and click **OK**.
- Add the following constraint:

```
create_clock -name clk -period 10.0 [get_ports clk_clk]
```
- Save as **niosv_top.sdc**.

Pin Assignments

- In Quartus Prime software, navigate to **Processing** menu bar and click **Start ► Start Analysis & Elaboration**.
- Once the analysis is complete, navigate to **Assignments** menu bar and click **Pin Planner**. For this example, there is 2 pins assignment:
 - clk_clk assigned to PIN_U52 (FPGA_SYSTEM_CLK)
 - reset_reset_n assigned to PIN_G52 (FPGA_SYS_RESETn)
- Close **Pin Planner** and return to the project front page.

Configuration and Power Management Assignments

1. In Quartus Prime software, navigate to **Assignments** menu bar and click **Device > Device and Pin Options**.
2. Navigate to **Configuration** category.
3. Select **VID mode of operation** as **PMBus Master**.
4. Click **Configuration Pin Options**, and configure the following **Configuration pin** assignments,
 - **PWRMGT_SCL** as `SDM_IO0`
 - **PWRMGT_SDA** as `SDM_IO12`
 - **CONF_DONE** as `SDM_IO16`
 - Leave the rest empty
5. Navigate to **Power Management & VID** category, and configure the following settings:
 - **Bus speed mode** as `100 kHz`
 - **Slave device type** as `Other`
 - **PMBus device 0 slave address** as `42`
 - **Voltage output format** as `Linear format`
 - **Linear format N** as `-13`
 - **Translated voltage value** unit as `Volts`
 - Turn off **PAGE command**
6. Click **OK** and return to the project front page.

1.2.1.2.4. Compiling the Quartus Prime Software Project

1. Navigate to **Processing** menu bar and click **Start Compilation**.
2. Make sure that the Quartus Prime project compiles successfully.

1.2.1.3. Configurable Example Design

1.2.1.3.1. Creating a New Project

Start developing the Nios V processor system by choosing the example project in Quartus Prime software project.

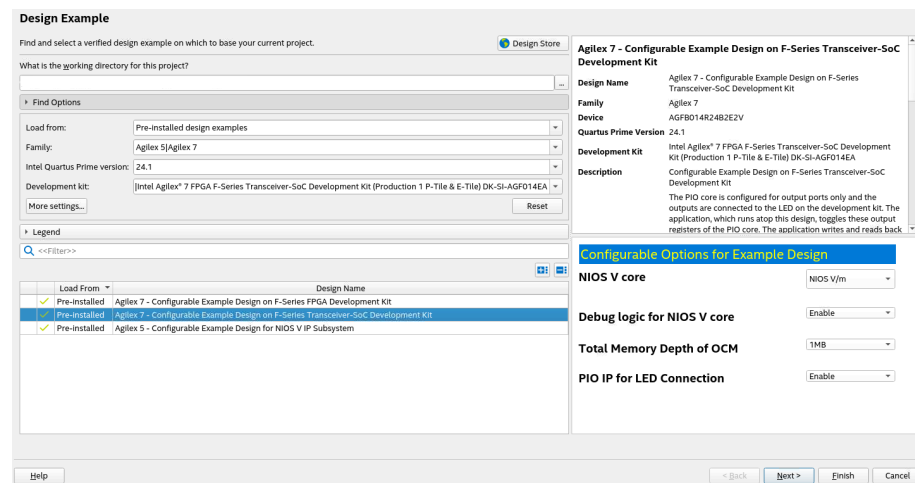
1. Launch **Quartus Prime Pro Edition** software.
2. Click **Open Example Project** to select the configurable example design.

Figure 49. New Project Wizard



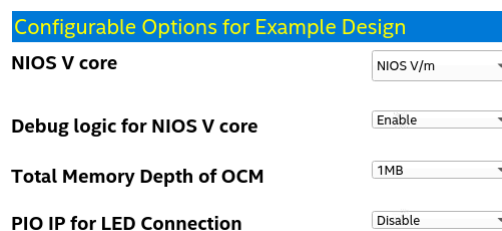
3. Select **Agilex 7 - Configurable Example Design on F-series Transceiver-SoC Development Kit**.

Figure 50. Open Example Project



4. In **Configurable Options for Example Design**, disable **PIO IP for LED Connection**.

Figure 51. Configurable Options for Example Design



5. Click **Next** to review the summary of the new project. Then click **Finish**.

Figure 52. Summary

Summary

When you click Finish, the project will be created with the following settings:

Project directory: /niosv/niosvm/configuration_example_design/relate_985/ced_1

Device assignments:

Design template:	Agilex 7 - Configurable Example Design on F-Series Transceiver-SoC Development Kit
Family name:	Agilex 7
Device:	n/a
Board:	Intel Agilex® 7 FPGA F-Series Transceiver-SoC Development Kit (Production 1 P-Tile & E-Tile) DK-SI-AGF014EA

Options:

proc_core:	NIOS V/m
memory_depth:	1MB
p1o_ip:	Enable
proc_debug:	Enable

EDA tools:

Design entry/synthesis:	()
Simulation:	()
Timing analysis:	()

Operating conditions:

Core voltage:	n/a
Junction temperature range:	n/a

Buttons: Help, < Back, Next >, Finish, Cancel

1.2.1.3.2. Creating a Platform Designer System

1. In **Project Navigator**, click **Files** and double-click on the QSYS file to open the system hardware.

Figure 53. Project Navigator

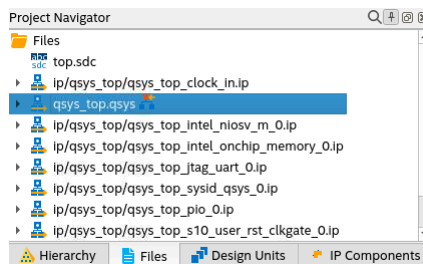


Figure 54. Full System Connection

The full system connection created in configurable example design.

Use	Connections	Name	Description	Export	Clock	Base	End	IRQ
<input checked="" type="checkbox"/>		clock_in	Clock Bridge Intel FPGA IP					
		in_clk	Clock Input	clk	exported			
		out_clk	Clock Output	Double-click to export	clock_in_ou...			
<input checked="" type="checkbox"/>		intel_niosv_m_0	Nios V/m Microcontroller Intel FPGA IP					
		clk	Clock Input	Double-click to export	clock_in_o...			
		reset	Reset Input	Double-click to export	[clk]			
		platform_irq_rx	Interrupt Receiver	Double-click to export	[clk]			IRQ 0
		instruction_manager	AXI4Lite Manager	Double-click to export	[clk]			IRQ 15
		data_manager	AXI4Lite Manager	Double-click to export	[clk]			
		timer_sw_agent	Avalon Memory Mapped Agent	Double-click to export	[clk]	0x0011_0000	0x0011_003f	
		dm_agent	Avalon Memory Mapped Agent	Double-click to export	[clk]	0x0010_0000	0x0010_ffff	
<input checked="" type="checkbox"/>		intel_onchip_memory_0	On-Chip Memory II (RAM or ROM) Intel ...					
		clk1	Clock Input	Double-click to export	clock_in_o...			
		axi_s1	AXI4 Subordinate	Double-click to export	[clk1]	0x0000_0000	0x000f_ffff	
		reset1	Reset Input	Double-click to export	[clk1]			
<input checked="" type="checkbox"/>		jtag_uart_0	JTAG UART Intel FPGA IP					
		clk	Clock Input	Double-click to export	clock_in_o...			
		reset	Reset Input	Double-click to export	[clk]			
		avalon_jtag_slave	Avalon Memory Mapped Agent	Double-click to export	[clk]	0x0011_0058	0x0011_005f	
		irq	Interrupt Sender	Double-click to export	[clk]			
<input checked="" type="checkbox"/>		reset_in	Reset Bridge Intel FPGA IP					
		clk	Clock Input	Double-click to export	clock_in_o...			
		in_reset	Reset Input	Double-click to export	[clk]			
		out_reset	Reset Output	Double-click to export	[clk]			
<input checked="" type="checkbox"/>		sysid_qsys_0	System ID Peripheral Intel FPGA IP					
		clk	Clock Input	Double-click to export	clock_in_o...			
		reset	Reset Input	Double-click to export	[clk]	0x0011_0050	0x0011_0057	
		control_slave	Avalon Memory Mapped Agent	Double-click to export	[clk]			
<input checked="" type="checkbox"/>		s10_user_rst_clkgate_0	Reset Release Intel FPGA IP					
		ninit_done	Reset Output	Double-click to export				

- Click **On-Chip Memory II (RAM or ROM) Intel FPGA IP** in the **System View**.
- In the **Parameter** window, navigate to the **Memory Initialization**, enable **Initial memory content** and **Enable non-default initialization file**. Provide the filename `hello.hex`.

Figure 55. On-Chip Memory II (RAM or ROM) IP Parameter Editor

Parameter: System Info Component Info Domains

System: qsys_top Path: intel_onchip_memory_0

HDL entity name: s_top_intel_onchip_memory_0 IP file: s_top/qsys_top_intel_onchip_memory_0.ip

Any changes here will be written out to disk when the system is saved.

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

intel_onchip_memory

Read latency for S2 = 1.

ROM/RAM Memory Protection

Reset Request: Enabled

ECC

Enable Error Correction Check (ECC)

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: hello.hex

Implement Clock-Enable Circuitry for Use in a Partial Reconfiguration Region

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 hello.hex

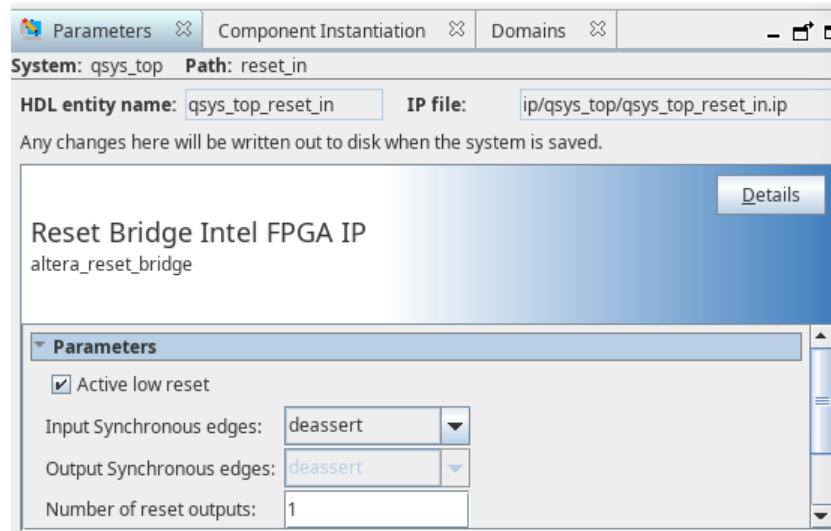
- Click **Reset Bridge Intel FPGA IP** in the **System View**.
- Navigate to **Export** column, double-click on `in_reset` to export the reset pin.

Figure 56. Export Reset Pin

	reset_in	Reset Bridge Intel FPGA IP	
	clk	Clock Input	Double-click to export
	in_reset	Reset Input	reset
	out_reset	Reset Output	Double-click to export

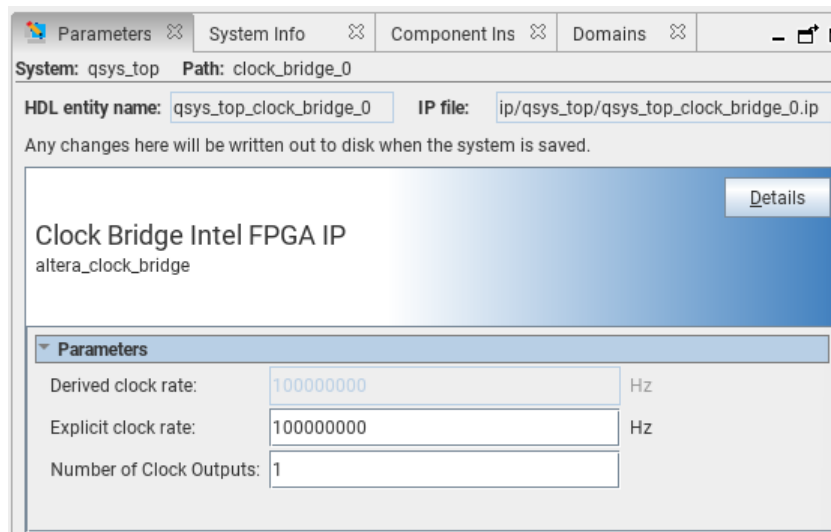
6. In the **Parameter** window, navigate to **Parameters** and enable **Active low reset**.

Figure 57. Reset Bridge Intel FPGA IP Parameter Editor



7. Click **Clock Bridge Intel FPGA IP** in the **System View**.
8. In the **Parameter** window, set the **Explicit clock rate** to **100MHz**.

Figure 58. Clock Bridge IP Parameter Editor



9. Connect all reset to ninit_done in **Reset Release Intel FPGA IP**.

Figure 59. Full System Connection

Use	Connections	Name	Description	Export	Clock	Base	End
<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/> clock_in	Clock Bridge Intel FPGA IP				
		▶ in_clk	Clock Input	clk	exported		
		▶ out_clk	Clock Output	Double-click to export	clock_in_ou...		
<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/> intel_niosw_m_0	Nios Vm Microcontroller Intel FPGA IP				
		▶ clk	Clock Input	Double-click to export	clock_in_ou...		
		▶ reset	Reset Input	Double-click to export	[clk]		
		▶ platform_irq_rx	Interrupt Receiver	Double-click to export	[clk]		IRQ 0
		▶ instruction_manager	AXI4Lite Manager	Double-click to export	[clk]		IRQ 1
		▶ data_manager	AXI4Lite Manager	Double-click to export	[clk]		
		▶ timer_sw_agent	Avalon Memory Mapped Agent	Double-click to export	[clk]		
		▶ dm_agent	Avalon Memory Mapped Agent	Double-click to export	[clk]		
<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/> intel_onchip_memory_0	On-Chip Memory II (RAM or ROM) Intel FPG...				
		▶ clk1	Clock Input	Double-click to export	clock_in_ou...		
		▶ axi_s1	AXI4 Subordinate	Double-click to export	[clk1]	0x0011_0000	0x0011_003f
		▶ reset1	Reset Input	Double-click to export	[clk1]	0x0010_0000	0x0010_ffff
<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/> jtag_uart_0	JTAG UART Intel FPGA IP				
		▶ clk	Clock Input	Double-click to export	clock_in_ou...		
		▶ reset	Reset Input	Double-click to export	[clk]		
		▶ avalon_jtag_slave	Avalon Memory Mapped Agent	Double-click to export	[clk]	0x0011_0058	0x0011_005f
		▶ irq	Interrupt Sender	Double-click to export	[clk]		
<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/> sysid_qsys_0	System ID Peripheral Intel FPGA IP				
		▶ clk	Clock Input	Double-click to export	clock_in_ou...		
		▶ reset	Reset Input	Double-click to export	[clk]		
		▶ control_slave	Avalon Memory Mapped Agent	Double-click to export	[clk]	0x0011_0050	0x0011_0057
<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/> s10_user_reset_clkgate_0	Reset Release Intel FPGA IP				
		▶ ninit_done	Reset Output	Double-click to export			
<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/> reset_bridge_0	Reset Bridge Intel FPGA IP				
		▶ clk	Clock Input	Double-click to export	clock_in_ou...		
		▶ in_reset	Reset Input	Double-click to export	[clk]		
		▶ out_reset	Reset Output	Double-click to export	[clk]		

10. Click **Sync System Info** at the bottom right corner of the Platform Designer.

Figure 60. Example of System Connectivity Issues

Type	Path	Message
1 Component Instantiation Warning		
qsys_top.clock_in		System Information doesn't match requirements of IP. Double-click to open System Info tab.

Saving and Generating System HDL

1. Save the system.
2. Click **Generate HDL** at the bottom right corner of the Platform Designer.
3. Leave all settings at default and click **Generate**.

Figure 61. Generation Window

Synthesis
Synthesis files are used to compile the system in a Quartus Prime project.
Create HDL design files for synthesis: Verilog
☐ Create timing and resource estimates for each IP in your system to be used with third-party EDA synthesis tools.
☒ Create block symbol file (.bsf)
☐ IP-XACT
☒ Generate IP Core Documentation

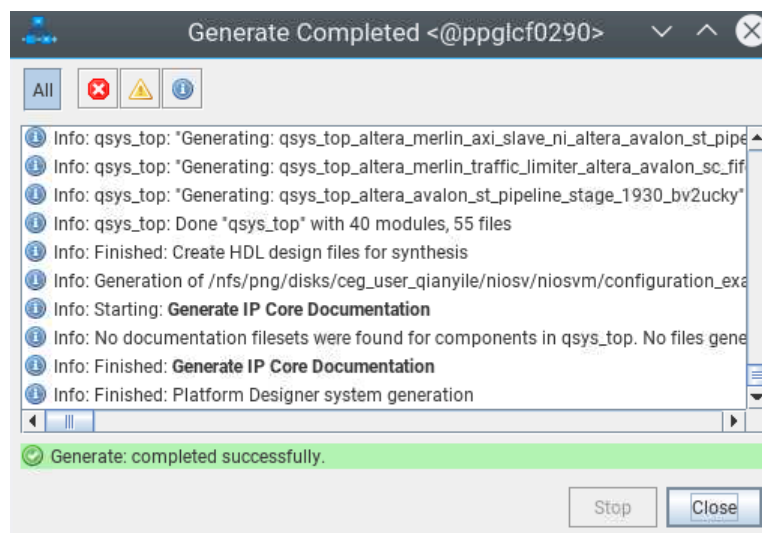
Simulation
The simulation model contains generated HDL files for the simulator, and may include simulation-only features.
Simulation scripts for this component will be generated in a vendor-specific sub-directory in the specified output directory.
Follow the guidance in the generated simulation scripts about how to structure your design's simulation scripts and how to use the *ip-setup-simulation* and *ip-make-simscript* command-line utilities to compile all of the files needed for simulating all of the IP in your design.
Create simulation model: None
Select simulation flow for specific simulators:
ModelSim flow selection: qrun
Select the simulators for which simulation scripts will be generated. If no simulators are selected, simulation scripts will be generated for all simulators.
☐ ModelSim
☐ VCS-MX
☐ VCS
☐ Riviera-PRO
☐ Xcelium

Output Directory
☐ Clear output directories for selected generation targets.

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).

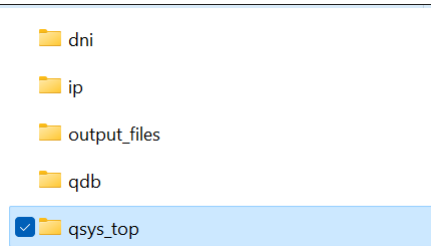
Generate Cancel

Figure 62. Successful Generation



- Exit the Platform Designer and return to the project front page.

Figure 63. Generated Project Files



1.2.1.3.3. Configuring Assignment and Constraint

Adding Design Files

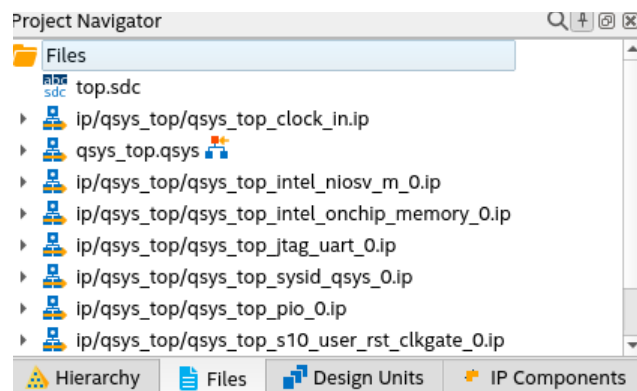
1. In Quartus Prime software, navigate to **Assignments** menu bar and click **Settings**.
2. Navigate to **Files** category. You can find all related IP files and QSYS files added to your project.

Figure 64. Settings – Files Category

File name:		
Q <<Filter>>		
File Name	Type	
top.sdc	SDC File (Read by the Timing Analyzer at each filter stage)	
ip/qsys_top/qsys_top_clock_in.ip	IP File	
qsys_top.qsys	Platform Designer System File	
ip/qsys_top/qsys_top_intel_niosv_m_0.ip	IP File	
ip/qsys_top/qsys_top_intel_onchip_memory_0.ip	IP File	
ip/qsys_top/qsys_top_jtag_uart_0.ip	IP File	
ip/qsys_top/qsys_top_sysid_qsys_0.ip	IP File	
ip/qsys_top/qsys_top_pio_0.ip	IP File	
ip/qsys_top/qsys_top_s10_user_rst_clkgate_0.ip	IP File	
ip/qsys_top/qsys_top_reset_bridge_0.ip	IP File	
ip/qsys_top/qsys_top_reset_in.ip	IP File	

3. During the project creation, the top-level design entity is named `top`. Thus, the `qsys_top.qsys` file is automatically the top-level design entity.
4. Check the top-level design entity assignment in the **Project Navigator**.

Figure 65. Project Navigator



Pin Assignments

1. In Quartus Prime software, navigate to **Processing** menu bar and click **Start ► Start Analysis & Elaboration**.
2. Once the analysis is complete, navigate to **Assignments** menu bar and click **Pin Planner**. For this example, there are 2 pins assignments:
 - `clk_clk` assigned to `PIN_U52` (`FPGA_SYSTEM_CLK`)
 - `reset_reset_n` assigned to `PIN_G52` (`FPGA_SYS_RESETn`)
3. Close **Pin Planner**, and return to the project front page.

Configuration and Power Management Assignments

1. In Quartus Prime software, navigate to **Assignments** menu bar and click **Device ► Device and Pin Options**.
2. Navigate to **Configuration** category.
3. Select **VID mode of operation** as **PMBus Master**.
4. Click **Configuration Pin Options**, and make the following **Configuration pin** assignments,
 - **PWRMGT_SCL** as `SDM_IO0`
 - **PWRMGT_SDA** as `SDM_IO12`
 - **CONF_DONE** as `SDM_IO16`
 - Leave the rest empty.
5. Navigate to **Power Management & VID** category, and make the following settings
 - **Bus speed mode** as `100 kHz`
 - **Slave device type** as `Other`
 - **PMBus device 0 slave address** as `42`
 - **Voltage output format** as `Linear format`
 - **Linear format N** as `-13`
 - **Translated voltage value unit** as `Volts`
 - Turn off **PAGE command**.
6. Click **OK**, and return to the project front page.

1.2.1.3.4. Compiling the Quartus Prime Software Project

1. Navigate to **Processing** menu bar and click **Start Compilation**.
2. Make sure that the Quartus Prime project compiles successfully.

1.2.2. Building Software Design with Ashling RiscFree IDE for Intel FPGAs

After the processor system is ready, you may begin building the software design using Ashling RiscFree IDE for Intel FPGAs software. It consists of the following steps:

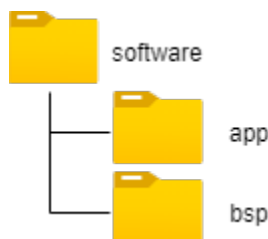
1. Create a board support package (BSP) project.
2. Create a Nios V processor application project with Hello World source code.
3. Import both projects into RiscFree IDE's workspace.
4. Build the Hello World application.

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 66. Software Project Directory Tree

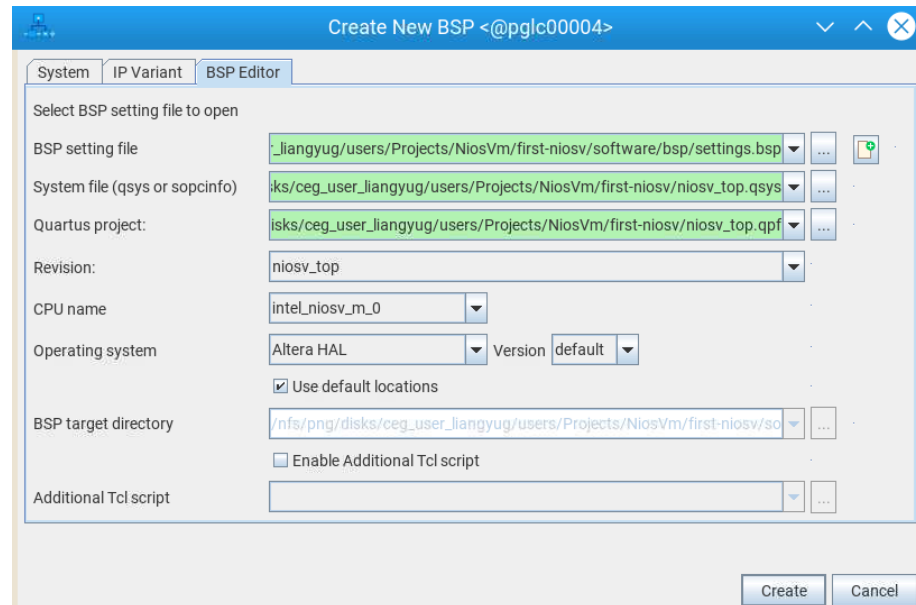


1.2.2.1. Creating a Board Support Package

Board Support Package (BSP) provides a software runtime environment for embedded systems, such as Nios V/m processor systems. Platform Designer includes the BSP Editor tool, to generate and configure BSP contents.

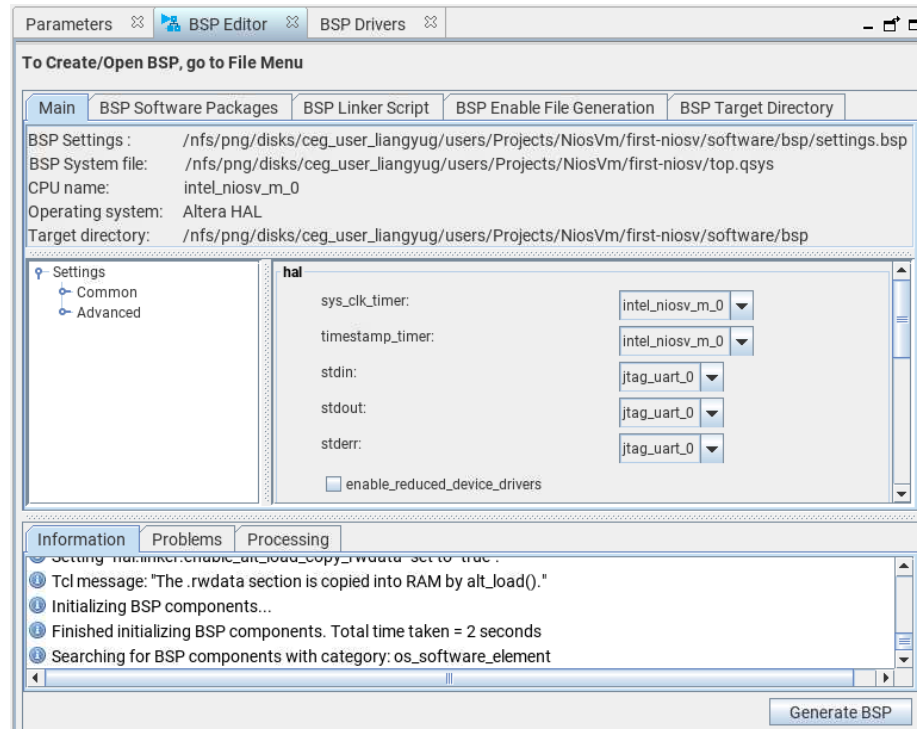
1. In the Quartus Prime software, go to **Tools ► Platform Designer**.
2. In the Platform Designer window, go to **File ► New BSP**.
3. The **Create New BSP** window appears.
4. For **BSP setting file**, create a BSP file (`settings.bsp`) in `<Working directory>/software/bsp/settings.bsp`.
5. For **System file (qsys or sopcinfo)**, select the Nios V/m processor Platform Designer system (`niosv_top.qsys`).
6. For **Quartus project**, select the example design Quartus Project File (`niosv_top.qpf`).
7. For **Revision**, select `niosv_top`.
8. For **CPU name**, select `intel_niosv_m_0`.
9. For **Operating system**, select Altera HAL.
10. Click **Create** to create the BSP file.

Figure 67. Create New BSP window



11. The **BSP Editor** tab appears.
12. You do not need to modify further. This example design uses the default BSP settings.
13. Click **Generate BSP** to generate the BSP file.

Figure 68. BSP Editor Tab



14. The BSP Editor generates the BSP files in <Working directory>/software/bsp folder.

Figure 69. Generated BSP Files

drivers	File folder	
HAL	File folder	
alt_sys_init.c	C File	3 KB
CMakeLists.txt	TXT File	6 KB
linker.h	H File	3 KB
linker.x	X File	14 KB
memory.gdb	GDB File	2 KB
settings.bsp	BSP File	33 KB
summary.html	Chrome HTML Document	44 KB
system.h	H File	10 KB
toolchain.cmake	CMAKE File	2 KB

Alternatively, you can generate the BSP files using CLI.

- a. Search Nios V Command Shell in the **Start Menu** and launch it.
- b. Launch the Nios V Command Shell.

```
$ niosv-shell
```

- c. Execute the command below to generate a BSP folder.

```
$ niosv-bsp -c -p=niosv_top.qpf -s=niosv_top.qsys -t=hal <Working  
directory>/software/bsp/settings.bsp
```

1.2.2.2. Creating an Application Project File

Application Project (app) provides the software application for Nios V/m processor system.

1. In <Working directory>/software/app folder, create a C source code. Name it as hello.c.
2. In hello.c, copy and paste the Hello World application code below

```
#include <stdio.h>
#include <unistd.h>

void loopier() {
    for (int i = 0; i < 1000; ++i) {
        printf("Hello world, this is the Nios V/m cpu checking in %d...\n",
            i);
    }
}

int main() {
    loopier();
    usleep(1000000);
    printf("Bye world!\n");
    fflush(stdout);
    return 0;
}
```



3. Launch the Nios V Command Shell.

```
$ niosv-shell
```

4. Execute the command below to generate an application CMakeLists.txt.

```
$ niosv-app --bsp-dir=software/bsp --app-dir=software/app \  
--srcs=software/app/hello.c --elf-name=hello.elf
```

Figure 70. Generated APP Files

 CMakeLists.txt	TXT File	2 KB
 hello.c	C File	1 KB

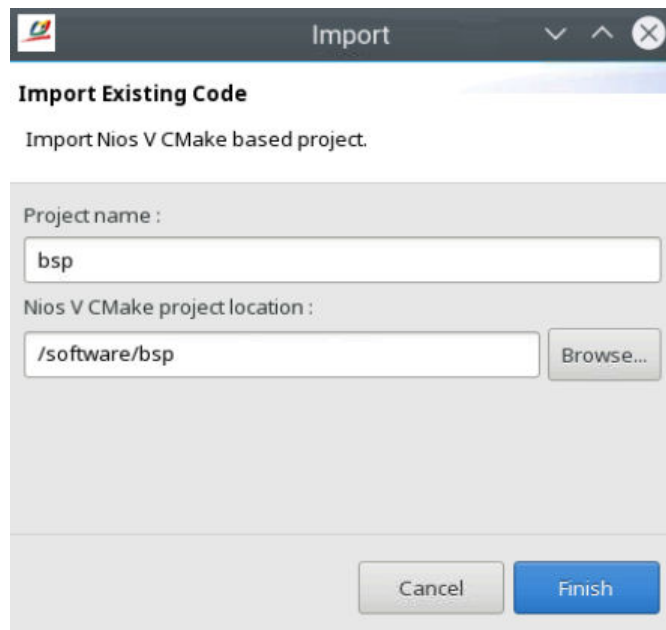
1.2.2.3. Importing Projects into Ashling RiscFree IDE for Intel FPGAs

1.2.2.3.1. Importing Nios V Processor BSP Project

Follow these steps to import the Nios V processor BSP:

1. Search Ashling RiscFree IDE for Intel FPGAs in **Start Menu** and open it.
2. Set the working directory as the **Workspace**.
3. Open the **Import** wizard, click **File ► Import Nios V CMake Project**.
4. In the Import window, browse and select the location of the Nios V processor BSP project.
5. The **Project name** is automatically fill according to the name of BSP project.
6. Click **Finish**. The BSP project is added to the Project Explorer.

Figure 71. Importing BSP Project into Ashling RiscFree IDE for Intel FPGAs



1.2.2.3.2. Importing Application Project

Follow these steps to import the application project:

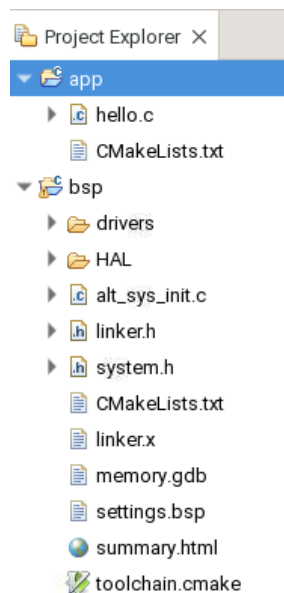
1. Continue with the same Workspace from [Importing Nios V Processor BSP Project](#).
2. Open the **Import** wizard, click **File ► Import Nios V CMake Project**.
3. In the **Import** window, browse and select the location of the Nios V processor APP project.
4. The **Project name** is automatically filled according to the name of the APP project.
5. Click **Finish**. The APP project is added to the **Project Explorer**.
6. Open the **New Project** wizard, click **File ► New ► Project....**

7. In the **New Project** wizard, click **C/C++ > C++ Project** to select the project.
8. In **C++ Project** wizard, turn off **Use default location**. Browse and select the location of the APP project. Enter the project name using the same name as the project folder.
9. Under **Project type**, select **CMake driven > Empty Project**. Under **Toolchains**, select **CMake driven**. Click **Finish**. The APP project is added to the Project Explorer.

1.2.2.4. Building the Hello World Application

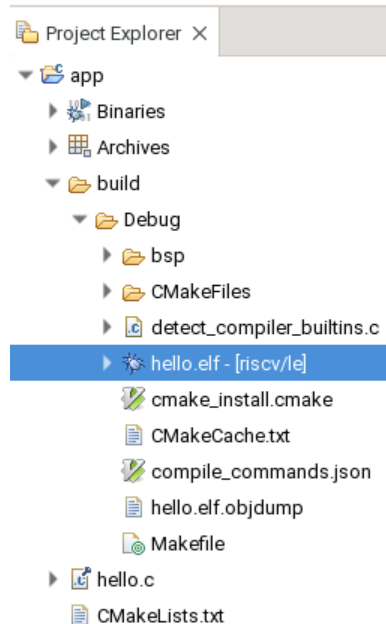
You can browse the BSP and APP project files in Ashling RiscFree IDE for Intel FPGAs **Project Explorer** tab.

Figure 72. Project Explorer Tab (Before Build Project)



1. Go to the **Project** in the menu tab and click **Build Project**.
2. When the build is complete, you can find the console prints as shown in the examples below.
3. In addition to the console prints, you can find the generated ELF file in the APP project folder.

Figure 73. Project Explorer Tab (After Build Project)



Example 1. CMake Console Print

```
cmake -DCMAKE_BUILD_TYPE:String=Debug -DCMAKE_EXPORT_COMPILE_COMMANDS:BOOL=ON -G
"Unix Makefiles" <Working directory>/software/bsp
-- The ASM compiler identification is GNU
-- Found assembler: <Disk drive>/riscfree/toolchain/riscv32-unknown-elf/bin/
riscv32-unknown-elf-gcc
-- The C compiler identification is GNU 12.1.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: <Disk drive>/riscfree/toolchain/riscv32-unknown-
elf/bin/riscv32-unknown-elf-gcc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- The CXX compiler identification is GNU 12.1.0
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: <Disk drive>/riscfree/toolchain/riscv32-
unknown-elf/bin/riscv32-unknown-elf-g++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done
-- Generating done
-- Build files have been written to: <Working directory>/software/bsp/build/Debug
```

Example 2. Make Console Print

```
/usr/bin/gmake -j all
Scanning dependencies of target hal2_bsp
[ 3%] Building ASM object CMakeFiles/hal2_bsp.dir/HAL/src/crt0.S.obj
[ 14%] Building ASM object CMakeFiles/hal2_bsp.dir/HAL/src/machine_trap.S.obj
[ 20%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/
alt_dma_txchan_open.c.obj
[ 21%] Building ASM object CMakeFiles/hal2_bsp.dir/HAL/src/alt_log_macro.S.obj
[ 30%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/alt_find_dev.c.obj
[ 32%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/
alt_dev_llist_insert.c.obj
[ 32%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/alt_busy_sleep.c.obj
[ 45%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/alt_icache_flush.c.obj
```

```
[ 48%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/alt_irq_handler.c.obj
[ 53%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/alt_ioctl.c.obj
[ 58%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/alt_printf.c.obj
[ 98%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/alt_alarm_start.c.obj
[ 98%] Building C object CMakeFiles/hal2_bsp.dir/drivers/src/
altera_avalon_jtag_uart_write.c.obj
[ 98%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/
alt_dcache_flush_no_writeback.c.obj
[ 98%] Building ASM object CMakeFiles/hal2_bsp.dir/HAL/src/alt_mcount.S.obj
[ 98%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/alt_close.c.obj
[ 98%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/alt_env_lock.c.obj
[ 98%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/alt_do_dtors.c.obj
[ 98%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/alt_dev.c.obj
[ 98%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/alt_errno.c.obj
[ 98%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/
alt_dma_rxchan_open.c.obj
[ 98%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/alt_exit.c.obj
[ 98%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/alt_doctors.c.obj
[ 25%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/
alt_dcache_flush_all.c.obj
[ 98%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/alt_execve.c.obj
[ 26%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/alt_dcache_flush.c.obj
[ 98%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/alt_fcntl.c.obj
[ 98%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/alt_envIRON.c.obj
[ 98%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/alt_find_file.c.obj
[ 98%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/alt_fd_lock.c.obj
[ 98%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/alt_fd_unlock.c.obj
[ 98%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/alt_fs_reg.c.obj
[ 98%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/alt_getchar.c.obj
[ 98%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/alt_gettod.c.obj
[ 38%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/alt_fork.c.obj
[ 98%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/alt_flash_dev.c.obj
[ 98%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/alt_iic.c.obj
[ 98%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/
alt_instruction_exception_register.c.obj
[ 98%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/alt_gmon.c.obj
[ 98%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/alt_fstat.c.obj
[ 98%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/alt_get_fd.c.obj
[ 98%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/alt_getpid.c.obj
[ 98%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/alt_io_redirect.c.obj
[ 98%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/
alt_iic_isr_register.c.obj
[ 98%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/
alt_icache_flush_all.c.obj
[ 98%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/alt_kill.c.obj
[ 98%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/alt_link.c.obj
[ 98%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/alt_putchar.c.obj
[ 98%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/alt_main.c.obj
[ 98%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/alt_open.c.obj
[ 98%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/alt_remap_uncached.c.obj
[ 98%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/alt_remap_cached.c.obj
[ 98%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/alt_log_printf.c.obj
[ 98%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/alt_isatty.c.obj
[ 98%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/alt_sbrk.c.obj
[ 98%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/alt_read.c.obj
[ 98%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/alt_release_fd.c.obj
[ 98%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/alt_putstr.c.obj
[ 98%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/alt_rename.c.obj
[ 98%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/alt_malloc_lock.c.obj
[ 98%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/alt_load.c.obj
[ 98%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/alt_settod.c.obj
[ 98%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/alt_stat.c.obj
[ 98%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/alt_putcharbuf.c.obj
[ 77%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/alt_tick.c.obj
[ 98%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/alt_usleep.c.obj
[ 98%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/alt_times.c.obj
[ 98%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/alt_uncached_free.c.obj
[ 98%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/alt_write.c.obj
[ 98%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/alt_wait.c.obj
[ 98%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/alt_lseek.c.obj
[ 98%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/alt_tls.c.obj
```

```
[ 98%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/
intel_fpga_api_niosv.c.obj
[ 98%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/alt_unlink.c.obj
[ 98%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/
intel_fpga_platform_api_niosv.c.obj
[ 98%] Building C object CMakeFiles/hal2_bsp.dir/alt_sys_init.c.obj
[ 98%] Building C object CMakeFiles/hal2_bsp.dir/drivers/src/
altera_avalon_jtag_uart_fd.c.obj
[ 98%] Building C object CMakeFiles/hal2_bsp.dir/drivers/src/
altera_avalon_jtag_uart_ioctl.c.obj
[ 98%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/mtimer.c.obj
[ 98%] Building C object CMakeFiles/hal2_bsp.dir/drivers/src/
altera_avalon_jtag_uart_init.c.obj
[ 98%] Building C object CMakeFiles/hal2_bsp.dir/drivers/src/
altera_avalon_jtag_uart_read.c.obj
[ 98%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/intel_niosv_irq.c.obj
[ 98%] Building C object CMakeFiles/hal2_bsp.dir/HAL/src/
alt_uncached_malloc.c.obj
[100%] Linking C static library libhal2_bsp.a
[100%] Built target hal2_bsp
```

Alternatively, you can build the application project using CLI.

1. Launch the Nios V Command Shell.

```
$ niosv-shell
```

2. Execute the command below to build the application.

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

1.3. Simulating the Nios V Processor System

Before experimenting with the designs in actual hardware, Altera recommends you to simulate the whole processor system and evaluate its intended functions. The benefits of simulation is not only for resource utilization but also to protect the actual hardware from unexpected damage due to mishandling within the hardware or software design.

1.3.1. Preparing Hardware Design for Simulation

Note: Before continuing the simulation, you must successfully build the hardware SOF file (with the memory-initialization feature enabled).

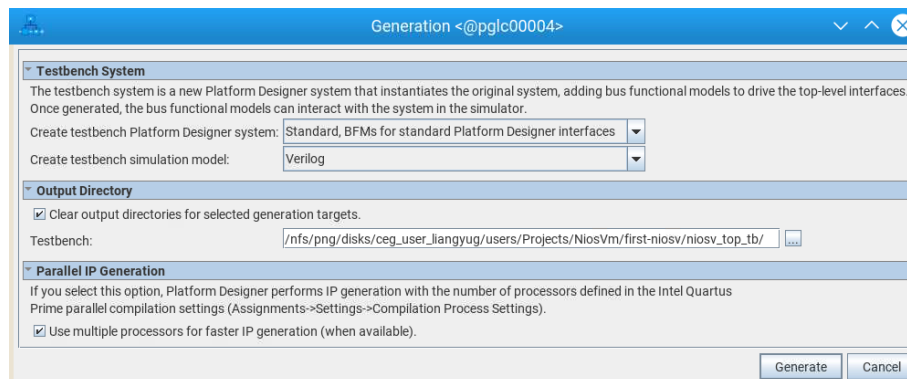
To generate hardware simulation files, perform the following steps:

1. Launch the Quartus Prime software and open the **Platform Designer** from the **Tools** menu.
2. Open the `niosv_top.qsys` file.
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, BFM's for standard Platform Designer interfaces*.
 - b. Create testbench simulation model—*Verilog*
 - c. Turn on *Use multiple processors for faster IP generation (when available)*.
5. Click **Generate**, and **Save**, if prompted.

Intel IP cores and Platform Designer systems generate simulation setup scripts. Modify these scripts to set up supported simulators. You can find the script, `msim_setup.tcl` in the following path:

```
<Working directory>/niosv_top_tb/sim/mentor
```

Figure 74. Testbench Generation



1.3.2. Preparing Software Design for Simulation

Note: Before continuing the simulation, you must successfully build the software ELF file.

To generate software memory initialization file, perform the following command:

```
elf2hex <Working directory>/software/app/build/Default/hello.elf \
-b 0x0 -w 32 -e 0x3ffff \
<Working directory>/software/app/build/Default/hello.hex
```

The command converts the software ELF into a HEX memory initialization file for the On-Chip RAM. The RAM has a data width of 32 bits, which starts from base address of 0x0 and ends at 0x3FFFF.

1.3.3. Checking 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.

Table 4. Simulation Files Generated

File	Description
<Working directory>/ niosv_top_tb/*	Platform Designer generates a testbench system when you enable the Create testbench Platform Designer system option.
<Working directory>/ niosv_top_tb/niosv_top_tb/sim/ mentor/msim_setup.tcl	Sets up a Questa simulation environment and creates alias commands to compile the required device libraries and system design files in the correct order and loads the top-level design for simulation.
<Working directory>/ software/app/build/Default/ hello.hex	Memory Initialization Files (.hex) is required to initialize memory components in your system.

1.3.4. Running System Simulation

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
<code>dev_com</code>	Compile device library files.
<code>com</code>	Compiles the design files in correct order.
<code>elab</code>	Elaborates the top-level design.
<code>elab_debug</code>	Elaborates the top-level design with the <code>novopt</code> option.
<code>ld</code>	Compiles all the design files and elaborates the top-level design.
<code>ld_debug</code>	Compiles all the design files and elaborates the top-level design with the <code>vopt</code> option. <i>Note:</i> The <code>vopt</code> option is to run optimization before elaborating the top-level design in the simulator.

You can run the simulation in the Questa simulator by performing the following steps,

1. Launch the Nios V Command Shell.
2. Open the **Questa for Intel FPGA** simulator using the command `vsim`.
3. In the Questa **Transcript** window, change your working directory to the `mentor` folder.

```
cd <Working directory>/niosv_top_tb/niosv_top_tb/sim/mentor
```

4. Copy the memory initialization file into the `mentor` folder.

```
file copy -force \
```

```
<Working directory>/software/app/build/Default/hello.hex ./
```

5. Run the `msim_setup.tcl`.

```
do msim_setup.tcl
```

6. Compiles all the design files and elaborates the top-level design with **vopt** option.

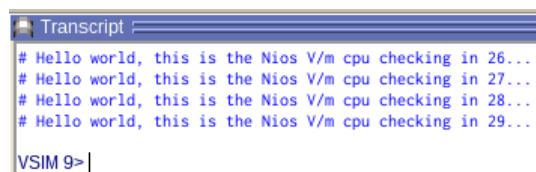
```
ld_debug
```

7. Run the simulation for more than 5 milliseconds.

```
run 10ms
```

At the end of the simulation, you can find the following message prints in the Questa **Transcript** window: Hello world, this is the Nios V/m cpu checking in <loop number>.

Figure 75. Questa Transcript Window Message



You can observe the simulation results from the waveform viewer:

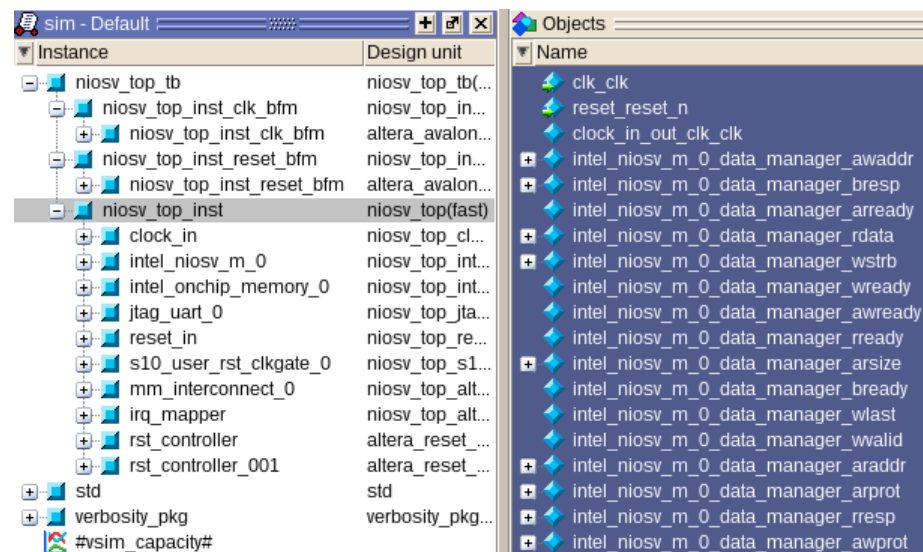
1. In Questa for Intel FPGA simulator, navigate to the **Instance** window.
2. Unroll niosv_top_tb.
3. Select niosv_top_inst, and the simulator populates a list of signals in the **Objects** window.
4. Right-click any of the signals, and click **Add Waves** to add them into the **Wave** window.
5. Restart and run the simulation for more than 5 milliseconds.

restart

run 10ms

6. Wait for the simulator to complete the given time.

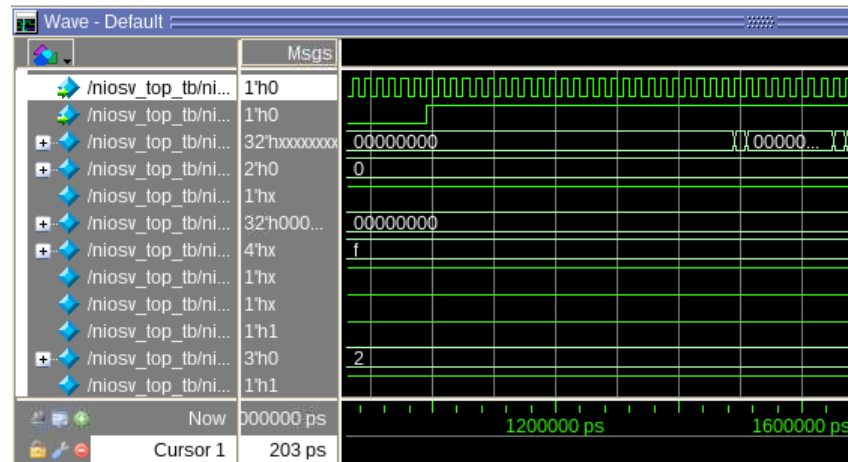
Figure 76. Questa Instance and Objects Windows



The following figure shows the simulated wave result.

- 1st signal: Clock Input
- 2nd signal: Reset Input
- Other signals: Any one or more signals within the niosv_top_inst

Figure 77. Waveform Viewer



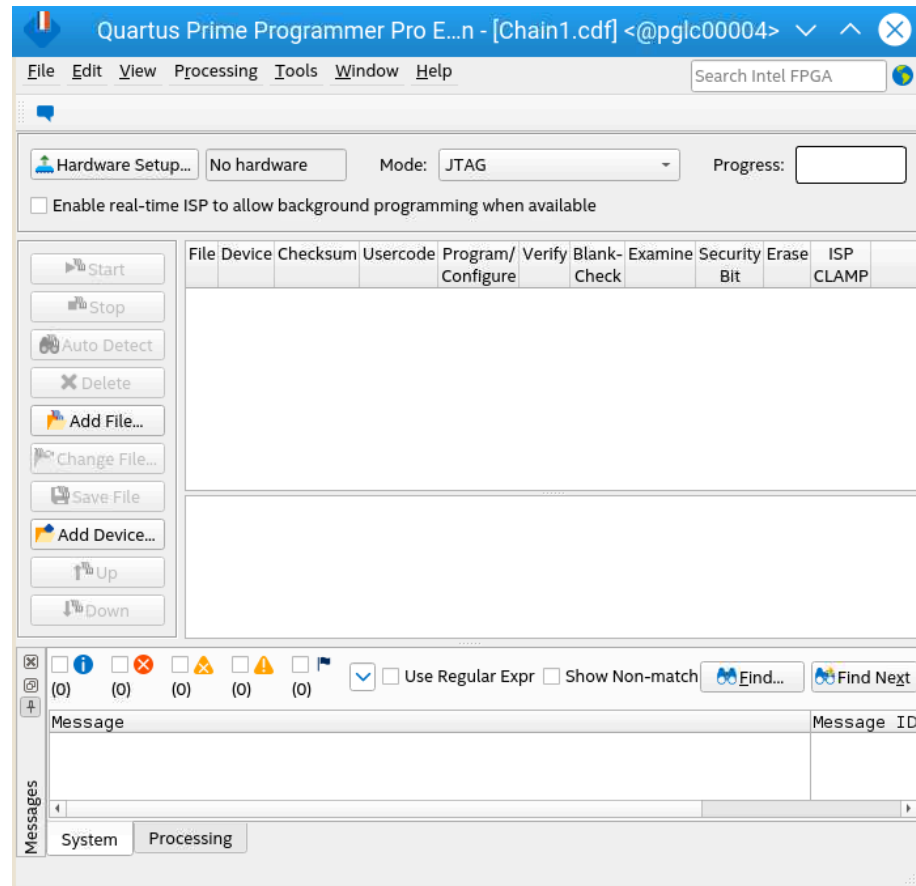
1.4. Programming the System

Use the Quartus Prime Programmer tool and the Ashling RiscFree IDE for Intel FPGAs to program the Nios V processor-based system into the FPGA and to run your application.

1.4.1. Programming Hardware SOF File

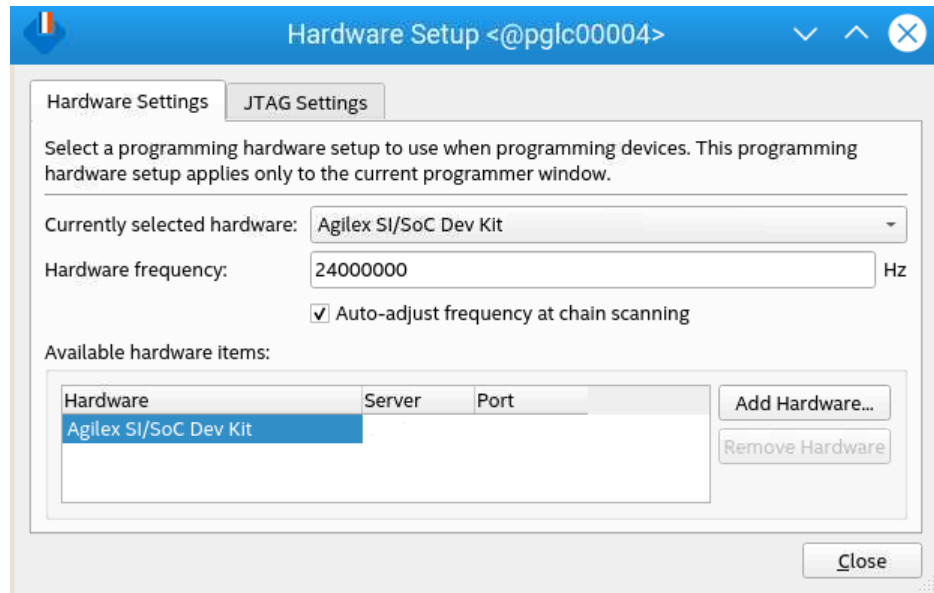
1. Connect the Agilex 7 FPGA F-Series Transceiver-SoC Development Kit to the host PC using the Intel FPGA Download Cable II.
2. Open the Quartus Prime Programmer.

Figure 78. Quartus Prime Programmer



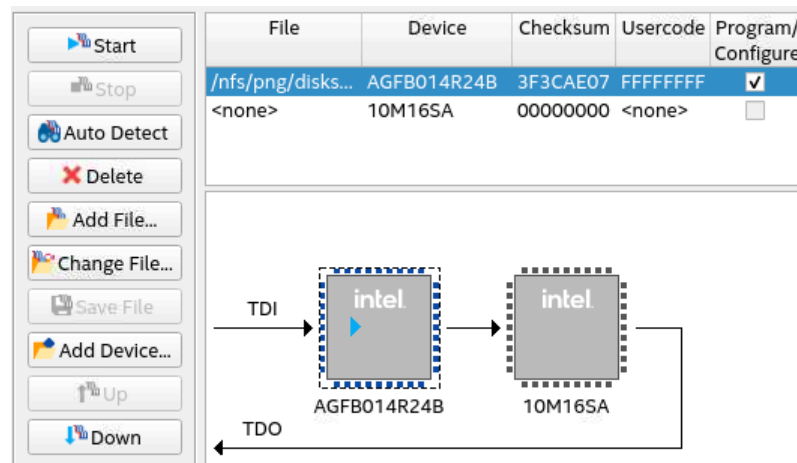
3. Click **Hardware Setup**.
4. Check the availability of the development kit in **Available hardware items**.
 - a. If available, select the development kit in **Currently selected hardware**.
 - b. If not available, check the cable connection, and the JtagServer driver installation.

Figure 79. Hardware Setup



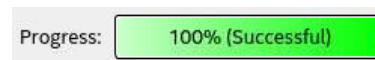
5. Click **Auto Detect** and select the appropriate device OPN.
6. Select the Agilex device, click **Change File** and select niosv_top.sof file.
7. Once the SOF file is ready, check **Program/Configure** and click **Start**.

Figure 80. List of Devices with JTAG Chain



8. Wait until the **Progress** bar reaches 100% (Successful).

Figure 81. Progress Bar



9. You have successfully configured the development kit with the processor hardware system.
Alternatively, you can program the hardware SOF file through CLI.

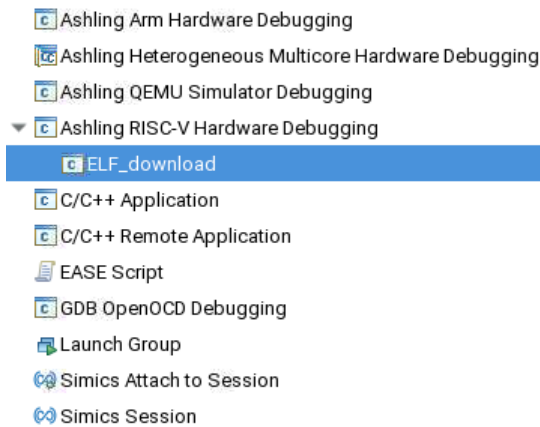
Execute the following command to program the SOF file.

```
$ quartus_pgm -c 1 -m JTAG -o p;<Working directory>/output_files/  
niosv_top.sof@1
```

1.4.2. Downloading the Software ELF File

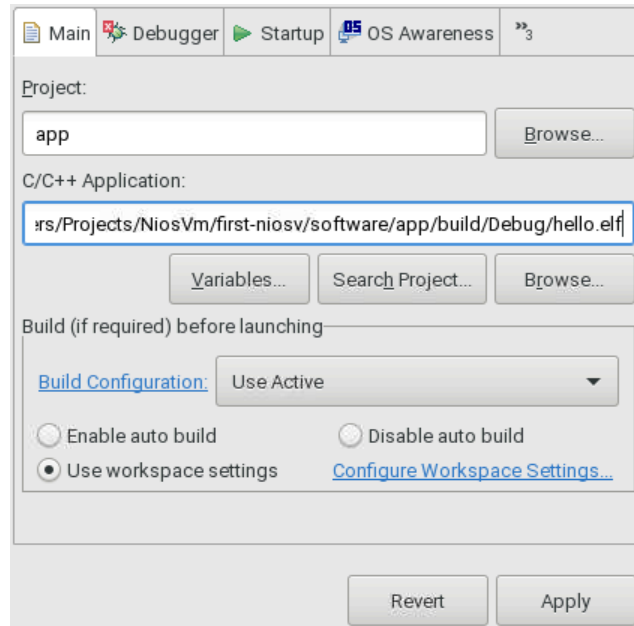
1. Ensure that the development kit is successfully configured with the processor system.
2. Launch the Ashling RiscFree IDE for Intel FPGAs.
3. Navigate to **Run ► Run Configurations**.
4. In **Run Configuration** window, double click **Ashling RISC-V Hardware Debugging** and name it as **ELF_download**.

Figure 82. Run Configuration



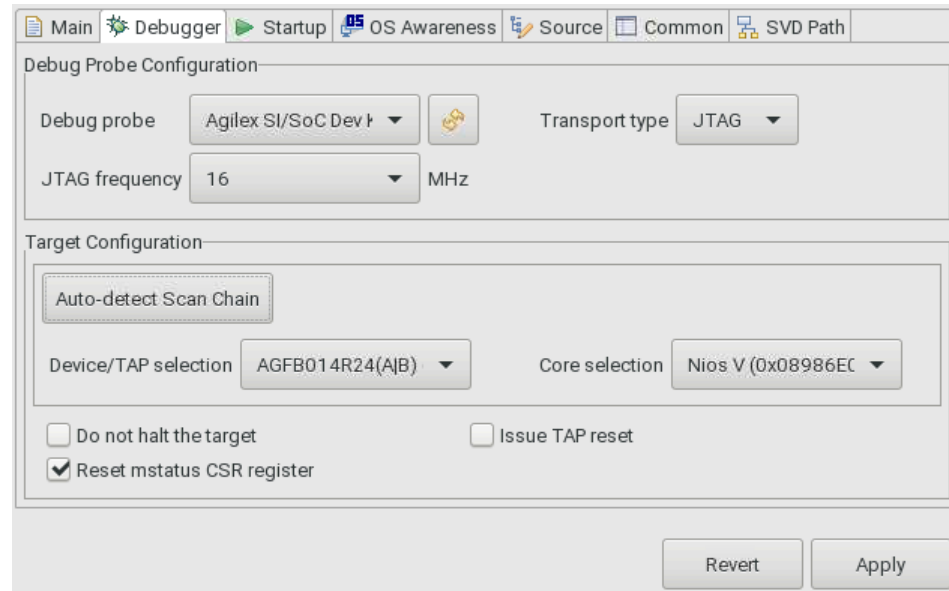
5. In the **Main** tab, make the following settings:
 - a. **Project:** app
 - b. **C/C++ Application:** <Working directory>/software/app/build/Debug/hello.elf

Figure 83. Main Tab



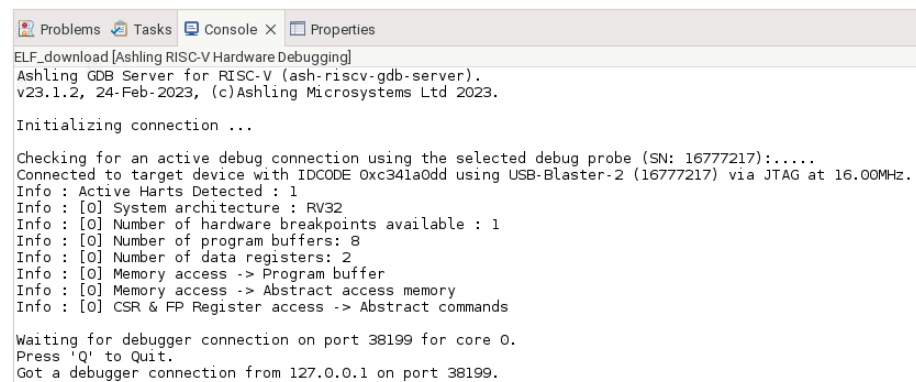
6. In the **Debugger** tab, make the following settings:
 - a. **Debug Probe Configuration:**
 - i. **Debug probe:** Agilex development kit
 - ii. **Transport type:** JTAG
 - iii. **JTAG frequency:** 16 MHz
 - b. **Target Configuration:** Click **Auto-detect Scan Chain** to list all possible cores. Select the appropriate **Device/TAP** and **Nios V Processor Core**.

Figure 84. Debugger Tab



- Click **Apply** and **Run**. Ashling RiscFree IDE for Intel FPGAs prints the following message in its **Console**.

Figure 85. ELF_download Message (Console tab)



Alternatively, you can download the software ELF file using CLI.

- Launch the Nios V Command Shell.

```
$ niosv-shell
```

- Execute the command below to download the ELF file.

```
$ niosv-download <Working directory>/software/app/debug/hello.elf -g -r
```

1.4.3. Applying External Tool Configuration

External tool configuration is a generic feature where you can configure the Ashling RiscFree IDE for Intel FPGAs to include external tools. An example of such tools is the `juart-terminal` executable, which is used to display the Hello World message through the JTAG UART IP.

To perform external tool configuration for `juart-terminal`, follow these steps:

1. Go to **Run > External Tools > External Tools Configurations**.
2. Double click **Program** to open a **New_configuration** window.
3. Rename the configuration as Nios V JTAG UART Output.
4. In **Location**, click **Browse File system**.
5. Browse and select the `juart-terminal` file in the following paths:

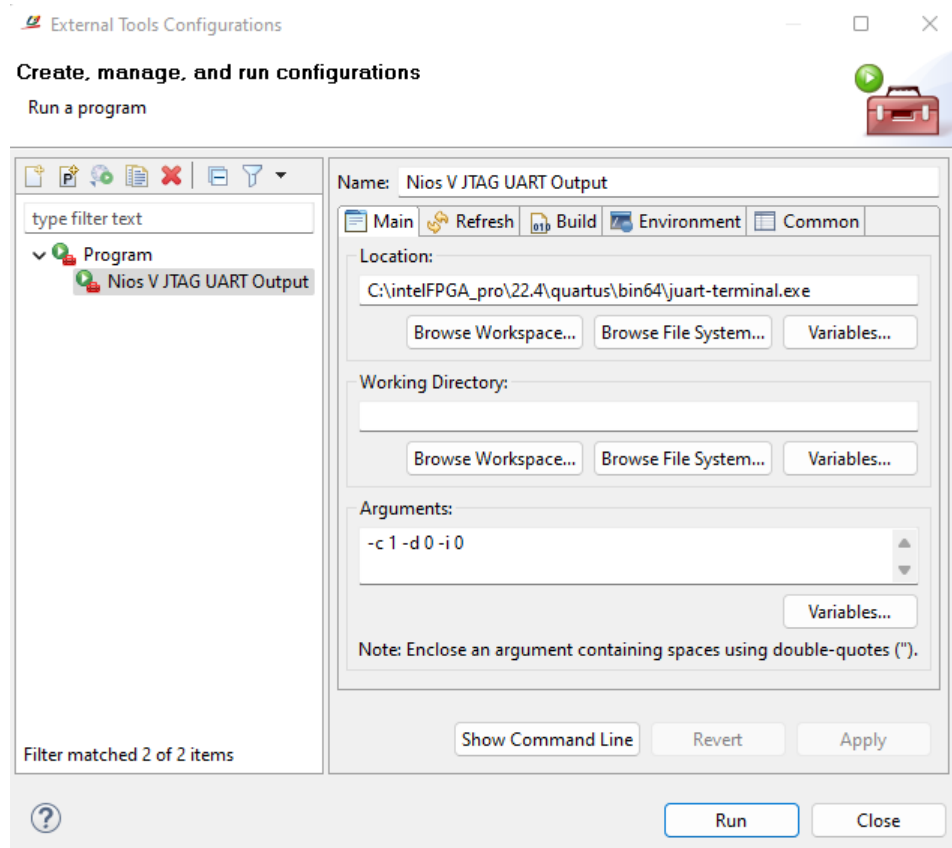
Table 5. Path

Operating System	Path
Windows*	<Intel Quartus Prime installation directory>/quartus/bin64/juart-terminal.exe
Linux*	<Intel Quartus Prime installation directory>/quartus/linux64/juart-terminal

6. Set the **Arguments** as `"-c 1 -d 1 -i 0"`. This configures the JTAG UART connection is towards the JTAG UART IP at cable 1, device 1, and instance 0.
7. Click **Apply**.

Note: JTAG device chain index varies according to your setup. Run `jtagconfig --debug` to obtain the latest device chain index numbering.

Figure 86. External Tools Configuration

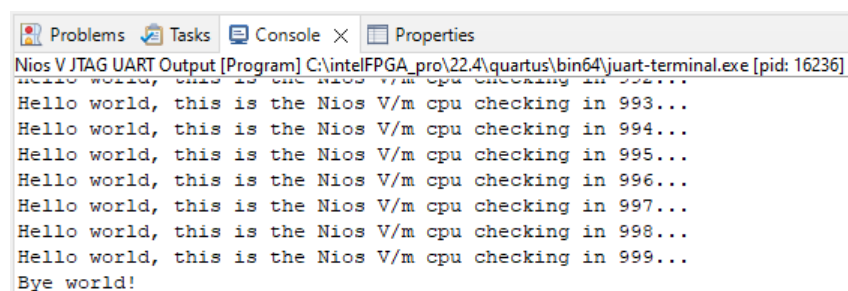


1.4.4. Run Hello World Application

With the **External Tool Configuration (Nios V JTAG UART Output)** defined successfully, you may proceed to use it to display the Hello World application.

1. Go to **Run > External Tools > External Tools Configurations**.
2. Select **Nios V JTAG UART Output**.
3. Click **Run**.
4. The Hello World message is printed on the **Console** tab.

Figure 87. Hello World Message (Console tab)



Alternatively, you can display the Hello World application using CLI.

1. Launch the Nios V Command Shell.

```
$ niosv-shell
```

2. Execute the command below to display the Hello World application.

```
$ juart-terminal -c 1 -d 0 -i 0
```

1.5. Debugging the System

Occasionally, the generated system might fail to run or display unexpected behaviors. You can debug the hardware and software system to pinpoint the root cause and ultimately return the system to normal. Optionally, you can also apply the debug tools to view a processor system's inner workings and running mechanisms while on-the-fly.

1.5.1. Software Debugging

1.5.1.1. Setting Initial Breakpoint

Before debugging any system, you need to have at least one breakpoint to suspend the execution inside the program.

1. Open the Ashling RiscFree IDE for Intel FPGAs.
2. Open the Nios V processor software project (hello.c).
3. Set a breakpoint at `main()` by double-clicking on the left-margin of the line containing `main()`.
4. Ensure that a blue circle have appeared beside `main()`.

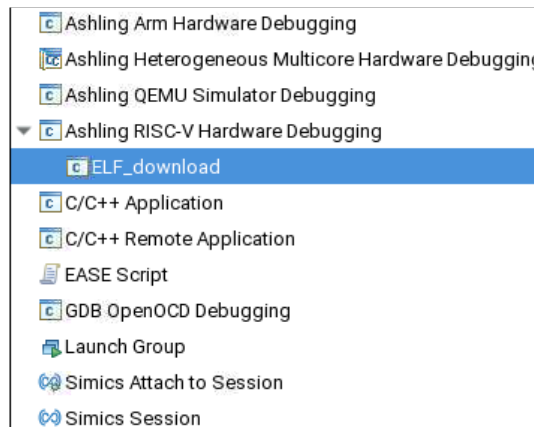
Figure 88. Initial Breakpoint at `main()`



1.5.1.2. Starting the Debugger

1. In the **Run** menu tab, select **Debug Configurations**.
2. Under **Ashling RISC-V Hardware Debugging**, `ELF_download` is found (Create in 4.2 Download Software ELF File).
3. Select `ELF_download` to start the Debugger using the same settings.

Figure 89. Debug Configuration



4. In the **Main** tab, check the following settings
 - a. **Project:** app
 - b. **C/C++ Application:** <Working directory>/software/app/build/Debug/hello.elf
5. In the **Debugger** tab, check the following settings,
 - a. **Debug Probe Configuration,**
 - i. **Debug probe:** Intel Agilex development kit
 - ii. **Transport type:** JTAG
 - iii. **JTAG frequency:** 16 MHz
 - b. **Target Configuration:** Click **Auto-detect Scan Chain** to list all possible cores. Select the appropriate **Device/TAP** and **Nios V Processor Core**.

Figure 90. Main Tab

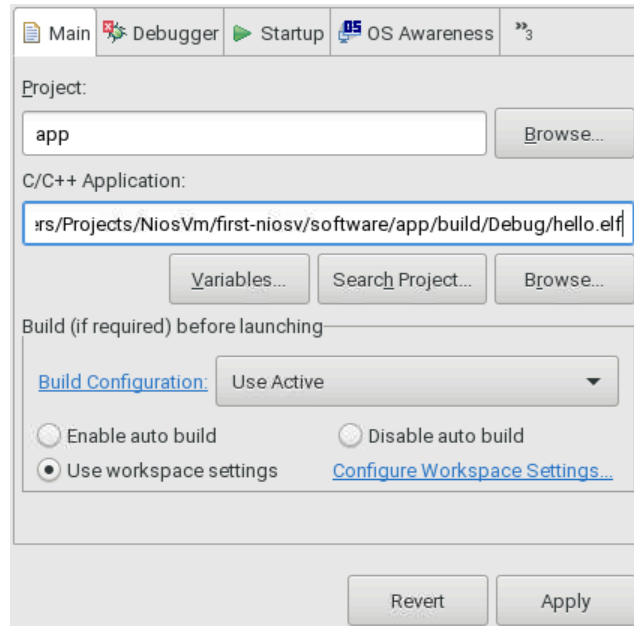
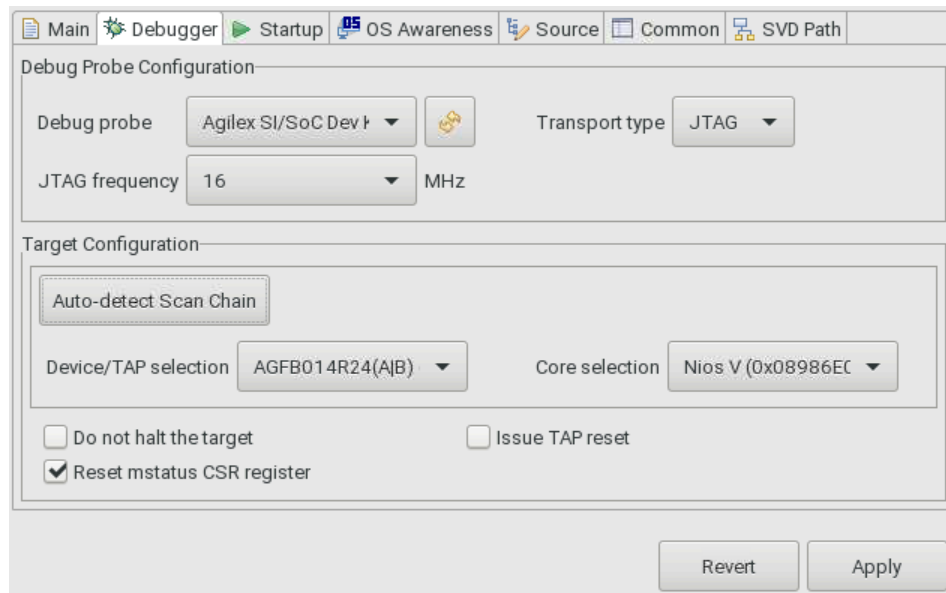


Figure 91. Debugger Tab



6. Click **Apply** and **Debug**.
7. The Ashling RiscFree IDE for Intel FPGAs switches to the **Debug Perspective**.
8. The program begins execution, and suspends at the initial breakpoint, `main()`.

Figure 92. Suspended at Initial Breakpoint

```
int main() {
    loop();
    usleep(1000000);
    printf("Bye world!\n");
    fflush(stdout);
    return 0;
}
```

1.5.1.3. View Global Variables

Ashling RiscFree IDE for Intel FPGAs has dedicated global variables.

To view the global variables in the application (.elf), follow these steps:

1. Go to **Window ► Show View ► Other... ► Debug ► Global Variables View**.
2. Right-click in the **Global Variables View** tab, and select the **Add Global Variables** icon.
3. Select the required variables to view.

Note: You can change the value of a variable. Right-click the variable and select **Change Value**.

Figure 93. Global Variables View

Memory Global Variables View X	
Name	Type
> jtag_uart_0	altera_avalon_jtag_uart_dev

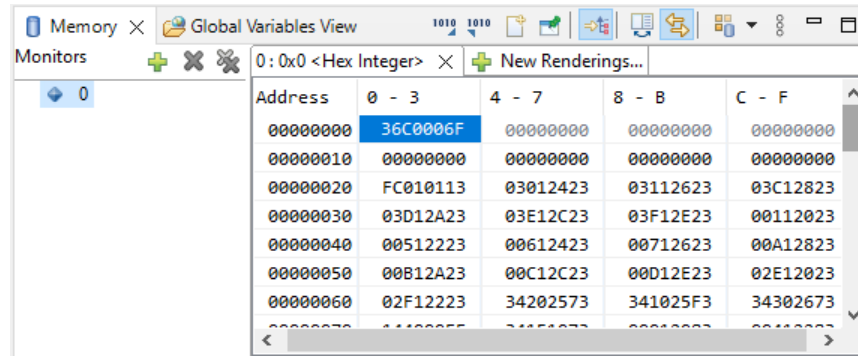
1.5.1.4. View Memory

Ashling RiscFree IDE for Intel FPGAs supports memory browser. You can view the content of On-Chip Memory (RAM) or other memory devices. This example targets the start address of On-Chip Memory (RAM), which the Hello World application begins.

To launch the memory browser, follow these steps:

1. Go to **Window ► Show View ► Memory Browser**.
2. Select **Add Memory Monitor**.
3. Provide the memory address 0x0, and click **OK**.

Figure 94. Memory Browser at Address 0x0



- Go to <Working directory>/software/app/build/Debug folder.
- Open the hello.elf.objdump file.
- Search for Disassembly of section .entry.
- The disassembly shows that the information at starting address 0 is 0x36c0006f, which is exactly the same as in the **Memory Browser**.
- You can continue to verify section .exceptions.

Figure 95. Disassembly of Hello World Application

```
Disassembly of section .entry:

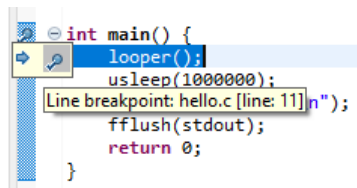
00000000 <_reset>:
    * Jump to the _start entry point in the .text section if reset code
    * is allowed or if optimizing for RTL simulation.
    */

    /* Jump to the _start entry point in the .text section. */
    tail _start
0: 36c0006f          j    36c <_start>
...
```

1.5.1.5. Setting a Second Breakpoint

- Open the Nios V processor software project (hello.c).
- Set a second breakpoint at `looper()` by double-clicking on the left-margin of the line containing `looper()`.
- Ensure that a blue circle has appeared beside `looper()`.

Figure 96. Second Breakpoint at `looper()`



1.5.1.6. Control Debugger

Using the two breakpoints, you can control the debugging process using the Debug bar.

Figure 97. Execution Control (Debug bar)

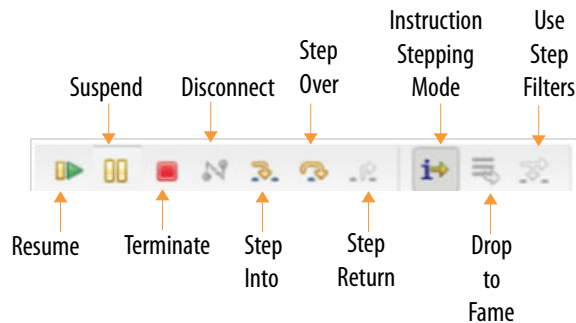


Table 6. Description of Execution Control

Function	Description
Resume	Continues the debugging process until the next breakpoint, or until the end of the debugging process.
Suspend	Halts the running code while debugging.
Terminate	Terminates the debug session. You need to relaunch the debug session after terminated.
Disconnect	Disconnects the debug tool connected to the RiscFree* IDE. Note: You are required to terminate the debug session before disconnecting the debug tool.
Step Into	Steps into the next method call (entering it) at the currently executing line of code.
Step Over	Steps over the next method call (without entering it) at the currently executing line of code. The method is still executed.
Step Return (Step Out)	Returns from a method which has been stepped into (exiting it). The remainder of the code that was skipped by returning is still executed.
Instruction Stepping Mode	Switches to instruction stepping mode.
Drop to Frame	Pops the current stack frame and puts control back out to the calling method, resetting any local variables.
Use Step Filters	Changes whether step filters should be used in the current Debug View.

1.6. Document Revision History for AN 985: Nios V Processor Tutorial

Document Version	Changes
2024.07.24	<ul style="list-style-type: none"> Updated <i>Hardware and Software Requirements</i>. Added the following new sub-topics for <i>Building Hardware Design in Platform Designer</i>: <ul style="list-style-type: none"> Board-Aware Flow Configurable Example Design
2024.05.15	Updated the following topics:
continued...	

Document Version	Changes
	<ul style="list-style-type: none"> • <i>Building Hardware Design in Platform Designer</i> • <i>Adding Nios V/m Processor Intel FPGA IP</i> • <i>Connect Interfaces and Signals</i> • <i>Adding Nios V/m Processor Intel FPGA IP</i> • <i>Connect Interfaces and Signals</i> • <i>Creating an Application Project File</i> • <i>Importing Nios V Processor BSP Project</i> • <i>Importing Application Project</i>
2023.08.18	Initial release.