Nios® V Processor Quick Start Guide

Updated for Intel® Quartus® Prime Design Suite: **21.3**

IP Version: **21.1.0**

Subscribe

Send Feedback

UG-20345 | 2021.10.04

Latest document on the web: PDF | HTML

Contents

1. About this Document

This document provides an introduction to the Nios® V/m processor and the provided example design. The quick start guide provides the following information:

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

The Nios V/m processor is a soft intellectual property (IP) processor based on the RISC-V specification. You can compile and download a Nios V/m processor system (along with other hardware components) onto an Intel FPGA IP device.

Related Information

- Nios V Processor Reference Manual
- Nios V Release Notes

2. System Requirements

This section provides the hardware and software requirements to build a Nios V/m processor system and software projects.

2.1. Hardware and Software Requirements

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

- Supported Intel FPGA devices:
  - Intel Cyclone® 10 GX
  - Intel Arria® 10
  - Intel Stratix® 10
  - Intel Agilex™
- Intel Quartus® Prime Pro Edition software version 21.3 or later
- Currently supported open-source tools:
  - GNU RISC-V Embedded GCC (8.3.0-2.3)
  - CMake (3.14.10 or later)
  - xPack Windows Build Tools
  - Eclipse* CDT for Embedded C/C++ Developers

Intel Quartus Prime Software does not includes the open-source tools. Please refer to Setting Up Open-Source Tools on page 4 for the installation details.

2.2. Setting Up Open-Source Tools

The following open-source tools allow you to create Nios V processor software projects:

Table 1. Open-source tools

Open-source tool	Description
GNU RISC-V Embedded GCC	A pre-built toolchain to compile programs for RISC-V development.
CMake (for binary distribution)	A system that manages the build process using `CMakeLists.txt`.
xPack Windows Build Tools[1]	A specific package for Microsoft* Windows* that includes GNU Make and BusyBox to perform builds on Microsoft Windows.
Eclipse C/C++ Development Tooling (CDT) for Embedded C/C++ Developers[2]	A development tool that includes Eclipse plug-ins and tools for RISC-V development.

[1] For Windows users only.

The procedure for setting up the open-source tools:

1. Download the respective open-source tools based on your operating system. Refer to the related information for the download links.
2. Extract the downloaded `.zip` or `.tar` file into this directory:

   `<Intel Quartus Prime installation directory>/niosv`
3. Setup the environment variable (PATH) to include the installed tools into the design flow. The directory path may vary due to different tool version.
   a. If you are using Eclipse Embedded CDT, refer to Building the Application Project using Eclipse Embedded CDT on page 14.
   b. If you are using Command-Line Interface (CLI), refer to the table below and proceed with the environment variable (PATH) setup commands base on your operating system.

Table 2. PATH Variable Setup Examples

Open-source tool	PATH variable setup

GNU RISC-V Embedded GCC v8.3.0-2.3	Windows command prompt:
	`set PATH=<Intel Quartus Prime installation directory>/niosv/xpack-riscv-none-embed-gcc-8.3.0-2.3/bin;%PATH%`
	Linux terminal:
	`export PATH=<Intel Quartus Prime installation directory>/niosv/xpack-riscv-none-embed-gcc-8.3.0-2.3/bin:$PATH`

CMake v3.21.1	Windows command prompt:
	`set PATH=<Intel Quartus Prime installation directory>/niosv/cmake-3.21.1-windows-x86_64/bin;%PATH%`
	Linux terminal:
	`export PATH=<Intel Quartus Prime installation directory>/niosv/cmake-3.21.1-linux-x86_64/bin:$PATH`

xPack Windows Build Tools v4.2.1-2	Windows command prompt: `set PATH=<Intel Quartus Prime installation directory>/niosv/xpack-windows-build-tools-4.2.1-2/bin;%PATH%`

Related Information

- GNU RISC-V Embedded GCC
  Provides the GNU RISC-V Embedded GCC package for download. Download version v8.3.0-2.3 which supports the necessary RISC-V Instruction-Set Architecture (ISA).
- CMake packages for binary distributions
  Provides the CMake package for download.
- xPack Windows Build Tools
  Provides the xPack Windows Build Tools for download.

[2] This tool is optional because the design flow can also be implemented in command-line interface (CLI).

3. Nios V/m Processor Example Design

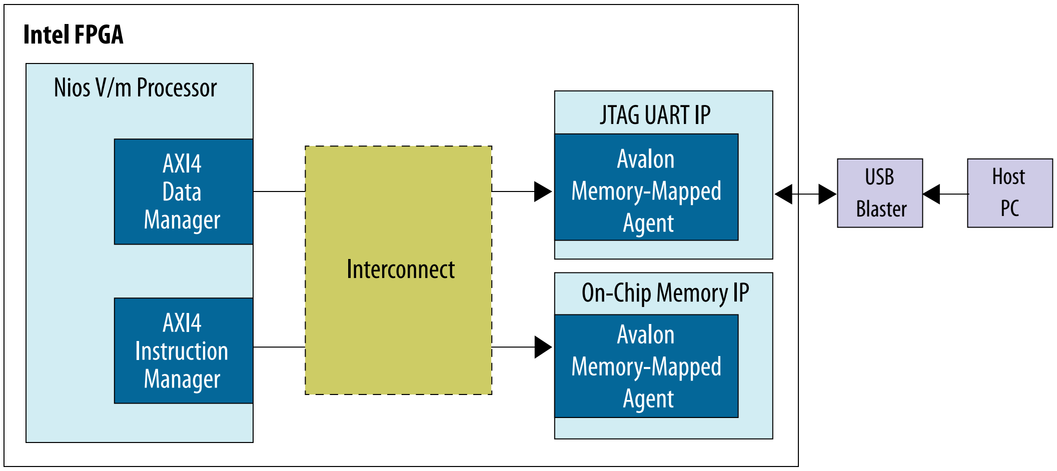
The example design is a simple Nios V/m processor hardware system used to run a “Hello World” application. The Nios V/m processor hardware system provides the following components:

Table 3. Example Design Component Description

Components	Description
Nios V/m Processor Intel FPGA IP	Runs application program by executing data and instruction.
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 instruction.

This section provides the design flow to generate and build a Nios V/m processor example design system. Before building and running an application on Nios V/m processor, you must compile and configure the correct hardware design on the FPGA. The example design provided was configured on the Intel Arria 10 SoC Development Kit.

Figure 1. Nios V/m Example Design Block Diagram



Note: You can modify the provided design to target your desired board by configuring the target device setting in the provided `top.qsf` and `create_qsys.tcl` file, and clock pin setting in the `top.qsf` file. For more information, refer to Generating the Nios V/m Processor Example Design System in Platform Designer on page 8.

Related Information

Intel® Arria® 10 SoC Development Kit

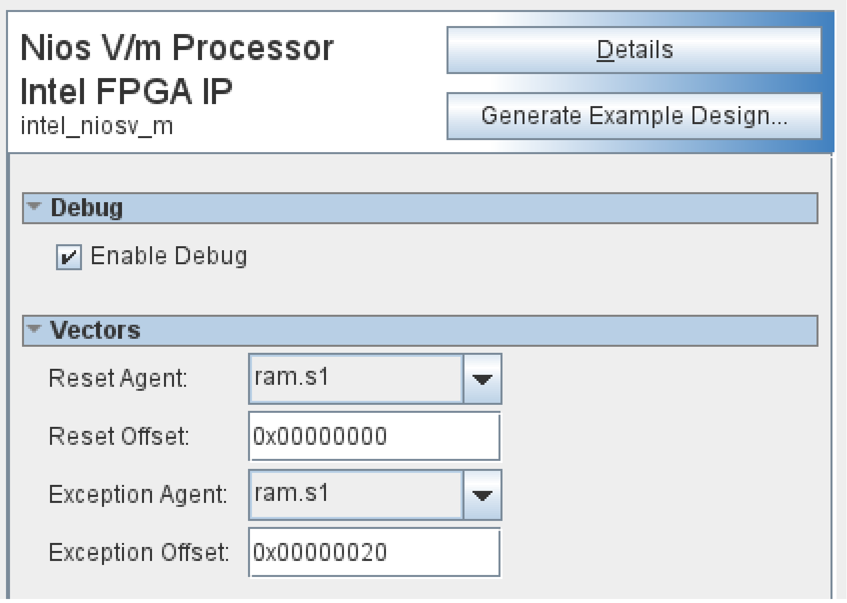
3.1. Generating the Example Design Through Graphical User Interface

3.1.1. Generating the Nios V/m Processor Example Design in Platform Designer

1. In the Intel Quartus Prime software, go to **Tools ➤Platform Designer**.
2. In the Platform Designer, select **IP Variant**.
3. For **Quartus project**, select **None**.
4. In the **IP Variant** dialog box, specify any name for your IP.

   Note: You do not need to save the IP later.
5. Click **Select** in the **Component type**.
   a. The **IP Catalog** opens.
   b. Search for **Nios V/m Processor Intel FPGA IP**.

Figure 2. IP Parameter Editor for Nios V/m Processor Intel FPGA IP



Nios V/m Processor Intel FPGA IP
intel_niosv_m

Debug

☑ Enable Debug

Vectors

Reset Agent: ram.s1

Reset Offset: 0x00000000

Exception Agent: ram.s1

Exception Offset: 0x00000020

   c. Create the IP design.
6. Click **Generate Example Design** and select your project folder.
7. Close the **IP Parameter Editor**. When prompted with **Save changes?**, you do not need to save the IP. Click **Don't Save**.
8. Unzip the example design to your project folder. Refer to Table 4 on page 8 for the example design files and the description.

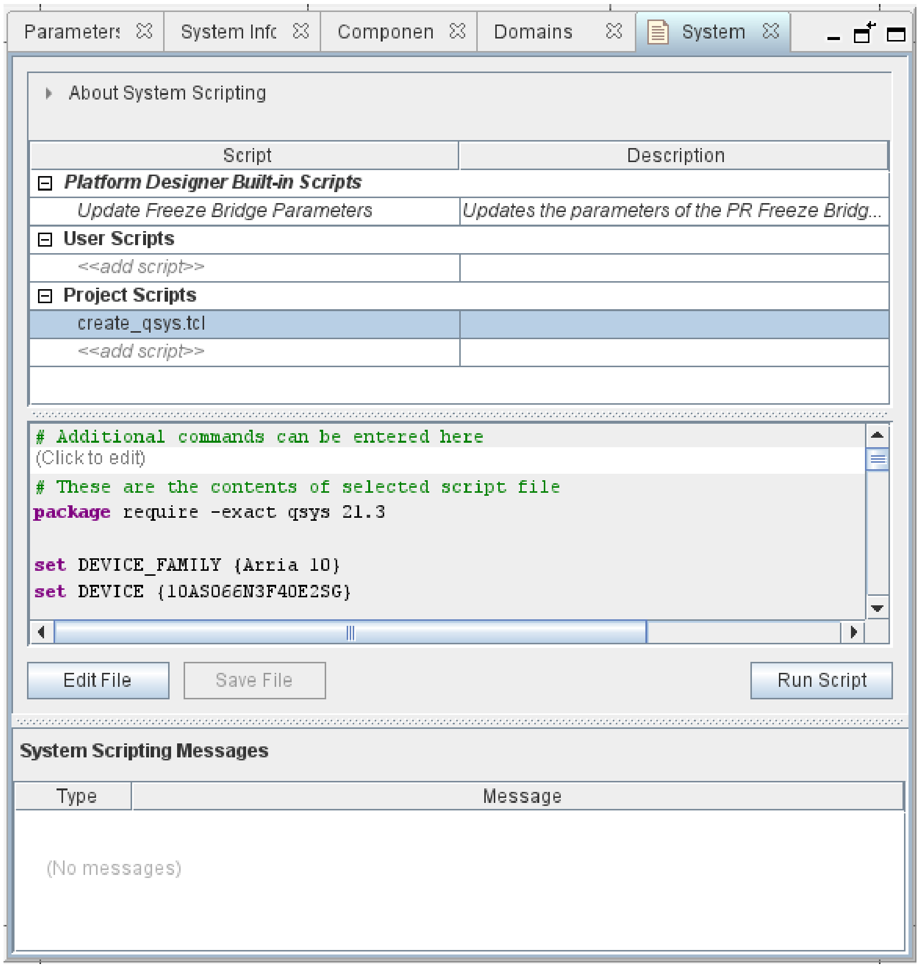
Table 4. Example Design File Description

File	Description
software/app	Folder containing source code for software application.
create_qsys.tcl	TCL script to generate the example design .qsys file.
readme.txt	Description and steps to build the example design.
toggle_issp.tcl	TCL script to reset the design via In-System Sources and Probes (ISSP).
top.qpf	Example design Quartus Project File (.qpf.) file.
top.qsf	Example design Quartus Setting File (.qsf) file.
top.sdc	Example design Synopsys* Design Constraints (.sdc) file.
top.v	Top-level Verilog design.

3.1.2. Generating the Nios V/m Processor Example Design System in Platform Designer

1. Open the project, go to **Tool ➤ Platform Designer**.
2. Create a new Platform Designer system and name it as `sys.qsys`.
3. Save the system.
4. In the Platform Designer, go to **View ➤ System Scripting**. The **System Scripting** window appears.
5. Under the **Project Scripts**, add and run the `create_qsys.tcl`.

Figure 3. System Scripting window



Script	Description
⊟ ***Platform Designer Built-in Scripts***	
Update Freeze Bridge Parameters	*Updates the parameters of the PR Freeze Bridg...*
⊟ **User Scripts**	
<<add script>>	
⊟ **Project Scripts**	
create_qsys.tcl	
<<add script>>	

```
# Additional commands can be entered here
(Click to edit)
# These are the contents of selected script file
package require -exact qsys 21.3

set DEVICE_FAMILY {Arria 10}
set DEVICE {10AS066N3F40E2SG}
```

System Scripting Messages

Type	Message
(No messages)	

6. The generated Platform Designer system consist of a clock bridge, reset bridge, Nios V/m processor, on-chip memory and JTAG UART IP.

7. Change memory size of the OCRAM total memory size to **327680**.

   Note: For more information, refer to KDB link: *Why does "Critical Warning (127003): Can't find Memory Initialization File or Hexadecimal (Intel-Format) File <project_directory>/intel_niosv_m_0_EXAMPLE_DESIGN/onchip_mem.hex -- setting all initial values to 0" happen when compiling Nios V example design that is enabled with Memory initialization file (.hex)*.

8. Click **Generate HDL** to generate the system HDL.

   Note: If you are using other Intel FPGA IP device, update the **FAMILY**, **DEVICE**, and **clock pin** assignments in the `top.qsf` file.

9. Click **Processing ➤ Start Compilation** to perform a full hardware compilation and generate the `.sof` file.

Related Information

KDB Link: Why does "Critical Warning (127003): Can't find Memory Initialization File or Hexadecimal (Intel-Format) File <project_directory>/

intel_niosv_m_0_EXAMPLE_DESIGN/onchip_mem.hex -- setting all initial values to 0" happen when compiling Nios V example design that is enabled with Memory initialization file (.hex).

3.2. Generating the Nios V/m Processor Example Design Using the Command-Line Interface

Alternatively, you can generate the example design system with the following commands:

1. Launch the Nios V Command Shell.

   ```
   <Intel Quartus Prime installation directory>/niosv/bin/niosv-shell
   ```

2. Generate the example design.

   ```
   ip-deploy --component-name=intel_niosv_m --output-name=niosv_m.ip
   qsys-generate niosv_m.ip --example_design
   unzip <Design ZIP file>
   ```

3. Change the OCRAM total memory size by modifying the create_qsys.tcl script line 33 to `set_component_parameter_value memorySize {`**`327680`**`}`
4. Generate the Platform Designer system.

   ```
   qsys-script --script=create_qsys.tcl --quartus-project=top.qpf
   ```

5. Perform the hardware compilation.

   ```
   quartus_sh --flow compile top
   ```

Note: For Linux environment, you must exit the Nios V Command Shell before compiling a project. Do not perform any design compilation within Nios V Command Shell in Linux. For more information, refer to KDB link: *Why does the Quartus project compilation failed in Intel Quartus Prime Pro (Linux version) within Nios V Command Shell?*.

Related Information

KDB Link: Why does the Quartus project compilation failed in Intel Quartus Prime Pro (Linux version) within Nios V Command Shell?

4. Software Design Flow

This section provides the design flow to generate and build a Nios V/m processor software project. Steps on how to generate an Application and Board Support Package (BSP) project using the `niosv-app` and `nios-bsp` utilities are provided. After that, you can choose to build the application project using Eclipse Embedded CDT, or through the command line interface. Intel will provide an IDE for the Nios V processor in a future release of the Intel Quartus Prime software.

Note: You need to setup the PATH variable to include the installed open-source tools.

- If you use Eclipse Embedded CDT, refer to Building the Application Project using Eclipse Embedded CDT on page 14.
- If you use CLI, refer to Setting Up Open-Source Tools on page 4.

4.1. Generating the Board Support Package

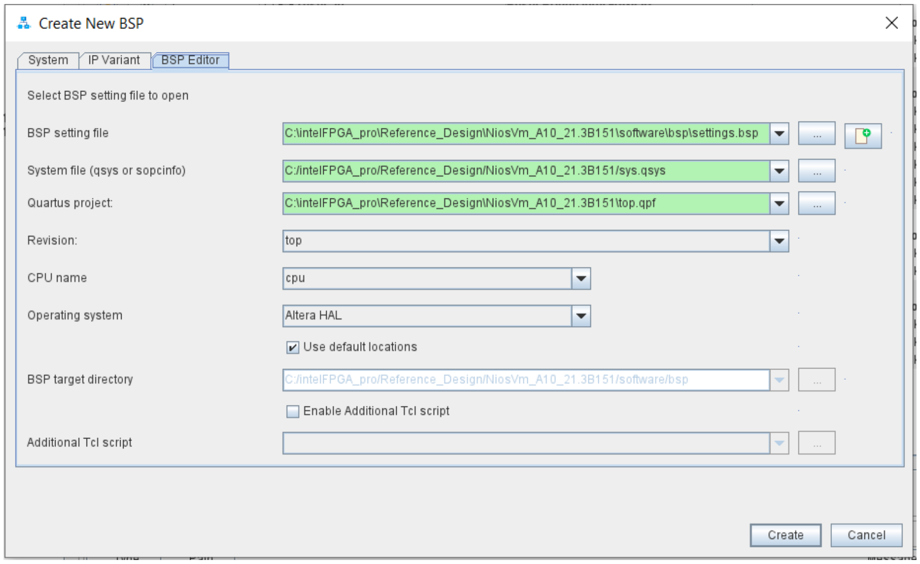
4.1.1. Generating the Board Support Package using the BSP Editor GUI

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/m 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` folder in `software` directory and name the BSP as `settings.bsp`.

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



Select BSP setting file to open

BSP setting file: C:\intelFPGA_pro\Reference_Design\NiosVm_A10_21.3B151\software\bsp\settings.bsp

System file (qsys or sopcinfo): C:/intelFPGA_pro/Reference_Design/NiosVm_A10_21.3B151/sys.qsys

Quartus project: C:\intelFPGA_pro\Reference_Design\NiosVm_A10_21.3B151\top.qpf

Revision: top

CPU name: cpu

Operating system: Altera HAL

☑ Use default locations

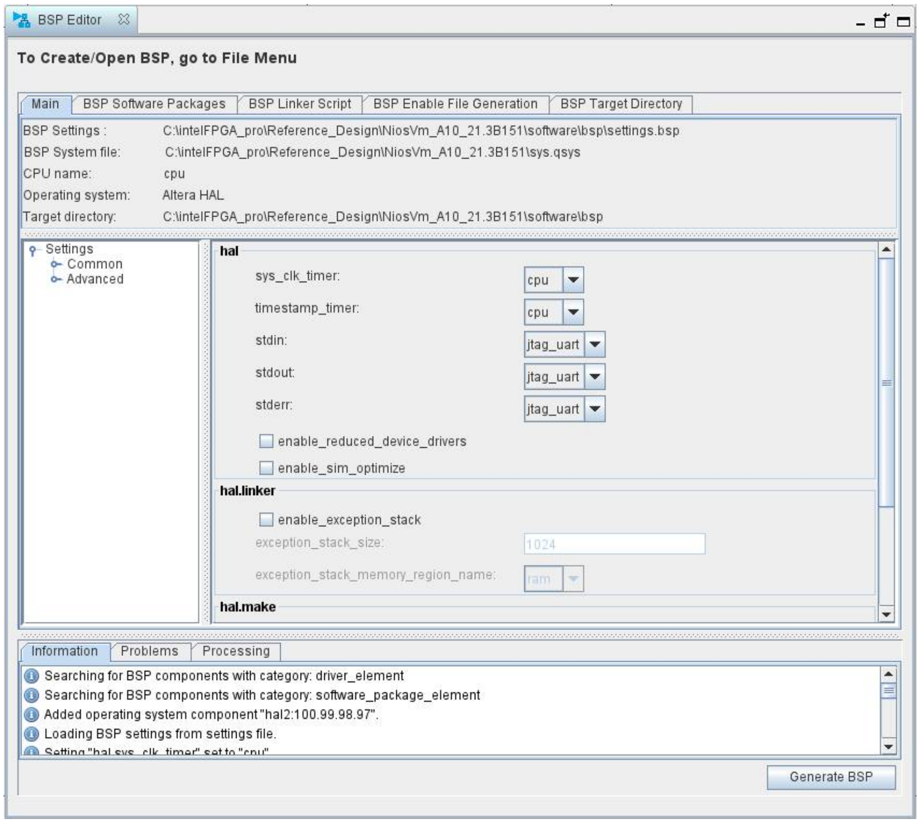
BSP target directory: C:/intelFPGA_pro/Reference_Design/NiosVm_A10_21.3B151/software/bsp

☐ Enable Additional Tcl script

Additional Tcl script:

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

Figure 5. BSP Editor



To Create/Open BSP, go to File Menu

BSP Settings : C:\intelFPGA_pro\Reference_Design\NiosVm_A10_21.3B151\software\bsp\settings.bsp
BSP System file: C:\intelFPGA_pro\Reference_Design\NiosVm_A10_21.3B151\sys.qsys
CPU name: cpu
Operating system: Altera HAL
Target directory: C:\intelFPGA_pro\Reference_Design\NiosVm_A10_21.3B151\software\bsp

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
exception_stack_size: 1024
exception_stack_memory_region_name: ram

hal.make

```
Searching for BSP components with category: driver_element
Searching for BSP components with category: software_package_element
Added operating system component "hal2:100.99.98.97".
Loading BSP settings from settings file.
Setting "hal.sys_clk_timer" set to "cpu"
```

Note: In the **BSP Editor**, the default selection for **sys_clk_timer** and **timestamp_timer** are configured to **cpu** to use the Nios V/m 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.

4.1.2. 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. Execute the following CLI command to generate the BSP file. Select the **type** as **hal** or **ucosii**.

```
niosv-bsp -c --quartus-project=top.qpf --qsys=sys.qsys --type=<hal or
ucosii> software/bsp/settings.bsp
```

4.2. Generating the Application Project File

You can find the Altera HAL application source file `hello.c` in the `software/app` folder with the generated design example.

If you selected Micrium MicroC/OS-II(ucosii) during BSP generation, please replace the application source file `hello.c` with `hello_ucosii.c` in the `<Intel Quartus Prime installation directory>/niosv/examples/software/hello_ucosii` folder.

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
--elf-name=hello.elf
```

4.3. Building the Application Project

4.3.1. Building the Application Project using Eclipse Embedded CDT

The Eclipse Embedded CDT setup consists of:

1. Tool set up
2. Project setting configuration
3. Software build flow

Refer to the related information for instructions to build your application project. After you complete the build, the application project file generates in the `<Project directory>/software/app/build/Debug` folder.

Related Information

Nios V processor tool setup for Eclipse Embedded CDT and OpenOCD
For more guidance about Eclipse Embedded CDT, refer to RocketBoards.org

4.3.2. Building the Application Project using the Command-Line Interface

You can also build the “Hello World” application using the CLI command:

1. Setup the PATH variable.
2. Enter the CLI Command:

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

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

This step creates the "Hello World" application in the form of a `.elf` file inside the `<Project directory>/software/app/build` folder.

5. Programming, Memory Initialization, Simulation, and Debug

To program Nios V/m processor into the FPGA and to run your application, you will need the following tools:

Table 5. Purpose and Tools

Purpose	Tool
Device programming	Intel Quartus Programmer
Simulation	Questa*-Intel FPGA Edition
Debug	Open On-Chip Debugger (OpenOCD)

5.1. Programming Nios V/m into the FPGA Device

1. To create the Nios V/m 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;top.sof@1
   ```

   Linux:

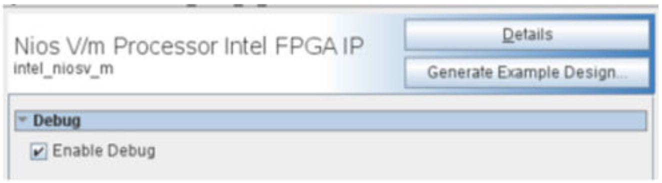
   ```
   quartus_pgm -c 1 -m JTAG -o p\;top.sof@1
   ```

   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 <elf file>
   ```

   Note: Set the **Enable Debug** option to use `niosv-download` command.


Nios V/m Processor Intel FPGA IP
intel_niosv_m

Debug
☑ Enable Debug

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

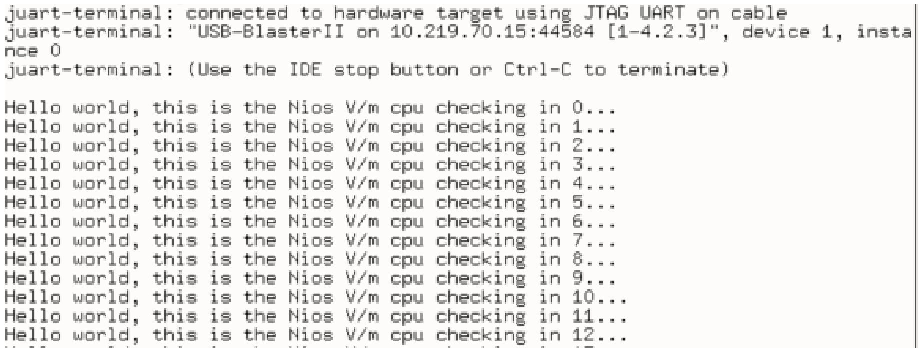
   ```
   quartus_stp -t toggle_issp.tcl
   ```

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

```
juart-terminal
```

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

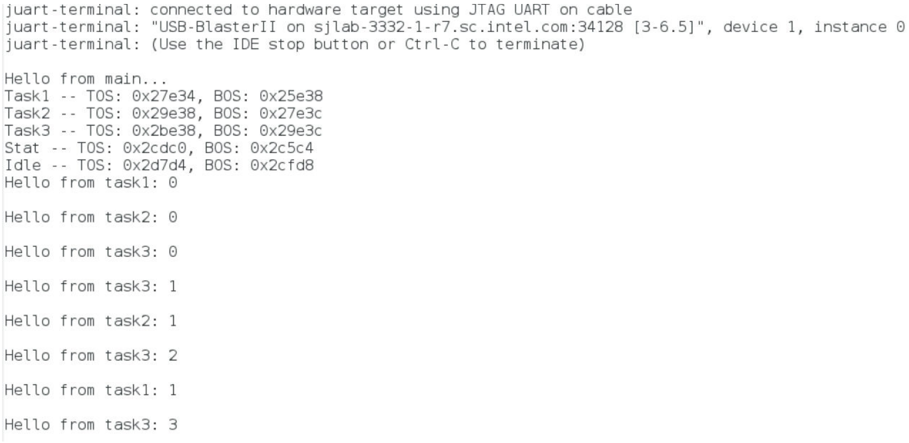
Figure 6. **Output of the Hello World application using hello.c**



```
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

5.2. Memory Initialization

Memory initialization offers another method to download the application image into FPGA device. With memory initialization, the on-chip memory is initialized during FPGA configuration with the data from a Nios V processor application image.

The memory initialization is required before you start the simulation in Chapter Simulation on page 18. In this example design, the memory initialization feature is enabled.

Perform the following steps to enable **Memory initialization**:

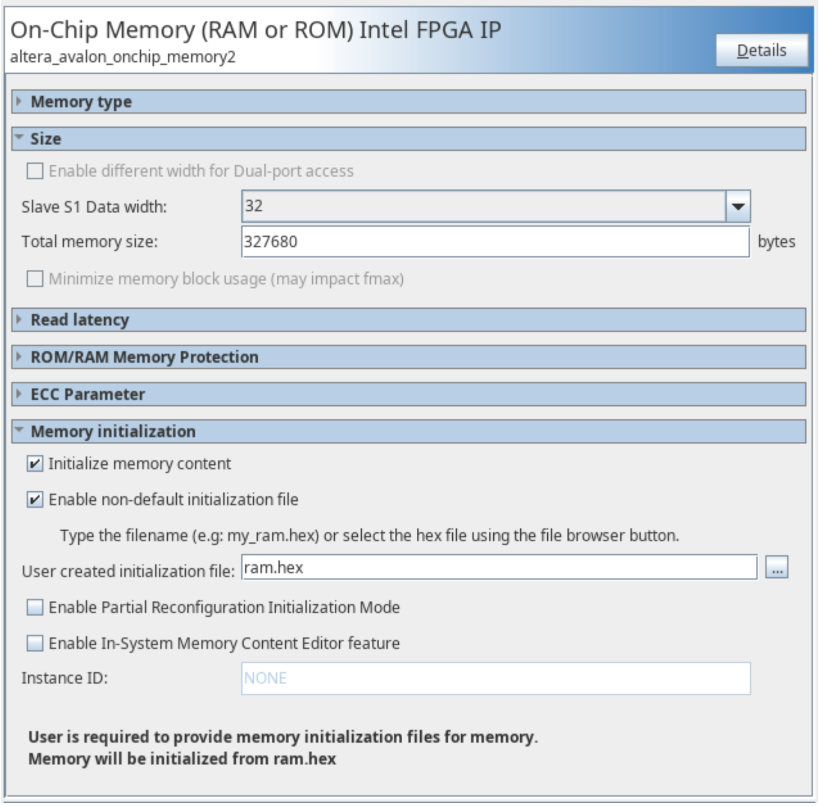
1. Generate the application image .hex file using the elf2hex command.

```
elf2hex <elf input file> -b <On-chip memory start address> -w <On-chip
memory data width in bits> -e < On-chip memory end address> <hex output
file>
```

```
e.g.      elf2hex software/app/build/hello.elf  -b 0x0 -w 32 -e 0x4FFFF -o
ram.hex
```

2. In the Platform Designer, go to the **IP Parameter Editor** for On-Chip Memory (RAM or ROM) Intel FPGA IP.
   a. In the **IP Parameter Editor**, select the following setting in **Memory Initialization**:
      i. Enable **Initialize memory content**.
      ii. Enable **Enable non-default initialization file**
      iii. At **User created initialization file**, browse to the created ram.hex file.

Figure 8. IP Parameter Editor for On-Chip Memory (RAM or ROM) Intel FPGA IP



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

Memory type

Size

□ Enable different width for Dual-port access

Slave S1 Data width: 32

Total memory size: 327680 bytes

□ Minimize memory block usage (may impact fmax)

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

3. Go to **Processing ➤ Start Compilation** or run quartus_sh --flow compile top to perform a full hardware compilation and generate the .sof file with memory initialization.

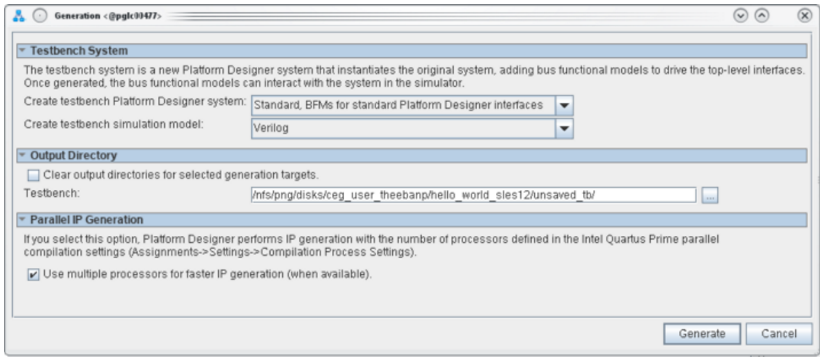
Note: Memory initialization features will enable on-chip memory loaded with Nios V/m processor instructions (`.hex` memory initialization file) upon downloading the hardware `.sof` file. Therefore, the step to download `.elf` file using `niosv-download` (Programming Nios V/m into the FPGA Device on page 15) is not required in this use case.

Proceed to program the device using the newly generated `.sof` file following the steps in Programming Nios V/m into the FPGA Device on page 15. After programming, reset the FPGA device, and the application should start running. Use the JTAG UART terminal to print the stdout and stderr of the Nios V/m processor system.

5.3. Simulation

1. To generate the simulation files, follow these steps:
   a. In **Platform Designer**, navigate to **Generate ➤ Generate Testbench System**.
   b. Input the generation settings as listed below.
      - For **Create testbench Platform Designer System**, select **Standard, BFMs for standard Platform Designer interfaces**.
      - For **Create testbench simulation model**, select **Verilog**.
      - Select **Use multiple processors for faster IP generation (when available)**.

Figure 9. Testbench Generation



Generation <@pglc00477>

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: Standard, BFMs for standard Platform Designer interfaces

Create testbench simulation model: Verilog

Output Directory

☐ Clear output directories for selected generation targets.

Testbench: /nfs/png/disks/ceg_user_theebanp/hello_world_sles12/unsaved_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).

2. Click **Generate**.
3. Generate the memory initialization file (`.hex`) using the `.elf` file. Please refer to Memory Initialization on page 16.

   Note: The `.hex` file name must match the name of the initialization file in the On-Chip Memory IP.
4. Open command prompt on Windows, or terminal on Linux and launch the Questa Simulator using the command `vsim`.
5. In the simulator console, navigate to Mentor folder within generated testbench system folder:

```
cd <Project directory>/sys_tb/sys_tb/sim/mentor
```

6. Execute the following steps within the simulator:

a. Copy the memory initialization file generated into the current path (Mentor folder)

```
file copy -force <Project directory>/ram.hex ./
```

b. Then, execute the commands below:

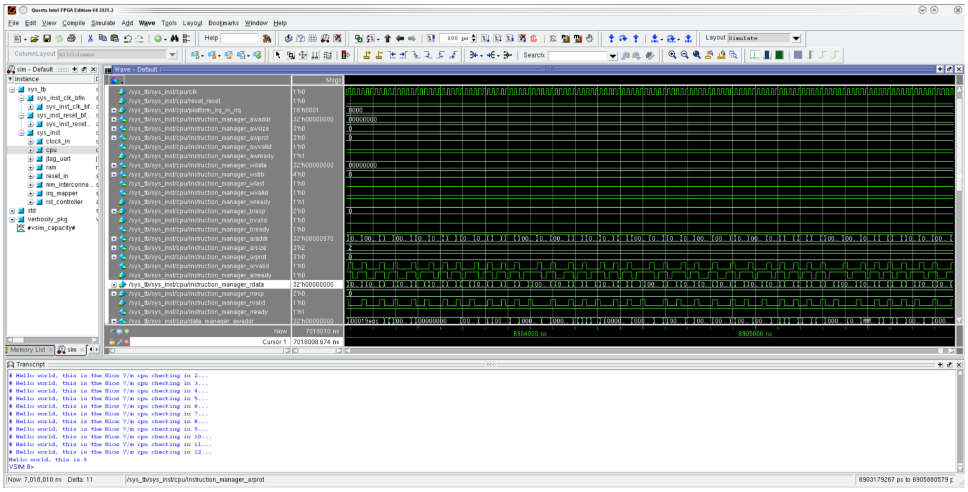
```
do msim_setup.tcl
```

```
ld_debug
```

c. Add signals of interest and run the simulation for 10 milliseconds. You can extend the period for longer simulation results.

Following figure is the example output of the certain selected signals:

Figure 10. Signal Output in Questa (Intel FPGA Edition)



```
/sys_tb/sys_inst/cpu/clk                              1'h0
/sys_tb/sys_inst/cpu/reset_reset                      1'h0
/sys_tb/sys_inst/cpu/platform_irq_rx_irq              16'h0001
/sys_tb/sys_inst/cpu/instruction_manager_awaddr       32'h00000000
/sys_tb/sys_inst/cpu/instruction_manager_awsize       3'h0
/sys_tb/sys_inst/cpu/instruction_manager_awprot       3'h0
/sys_tb/sys_inst/cpu/instruction_manager_awvalid      1'h0
/sys_tb/sys_inst/cpu/instruction_manager_awready      1'h1
/sys_tb/sys_inst/cpu/instruction_manager_wdata        32'h00000000
/sys_tb/sys_inst/cpu/instruction_manager_wstrb        4'h0
/sys_tb/sys_inst/cpu/instruction_manager_wlast        1'h0
/sys_tb/sys_inst/cpu/instruction_manager_wvalid       1'h0
/sys_tb/sys_inst/cpu/instruction_manager_wready       1'h1
/sys_tb/sys_inst/cpu/instruction_manager_bresp        2'h0
/sys_tb/sys_inst/cpu/instruction_manager_bvalid       1'h0
/sys_tb/sys_inst/cpu/instruction_manager_bready       1'h0
/sys_tb/sys_inst/cpu/instruction_manager_araddr       32'h00000970
/sys_tb/sys_inst/cpu/instruction_manager_arsize       3'h2
/sys_tb/sys_inst/cpu/instruction_manager_arprot       3'h0
/sys_tb/sys_inst/cpu/instruction_manager_arvalid      1'h0
/sys_tb/sys_inst/cpu/instruction_manager_arready      1'h0
/sys_tb/sys_inst/cpu/instruction_manager_rdata        32'h00000000
/sys_tb/sys_inst/cpu/instruction_manager_rresp        2'h0
/sys_tb/sys_inst/cpu/instruction_manager_rvalid       1'h0
/sys_tb/sys_inst/cpu/instruction_manager_rready       1'h1
/sys_tb/sys_inst/cpu/data_manager_awaddr              32'h00000000
                                              Now     7018010 ns
                                         Cursor 1     7018008.674 ns
```

```
# 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...
Hello world, this is t
VSIM 8>
```

```
Now: 7,018,010 ns   Delta: 11     /sys_tb/sys_inst/cpu/instruction_manager_arprot     6903179267 ps to 6905880579 p
```

5.4. Debug

Refer to the related information for more information about how to use OpenOCD to debug Nios V/m processor application.

Related Information

- OpenOCD Debug Configuration
  Provides more information about generating an OpenOCD configuration file.
- Debugging
  Provides more information about debugging.

6. Nios V/m Processor Reference

6.1. Utility and Script Summary

You can use the following command-line utilities and scripts when building Nios V/m processor design. You can run these tools from the command prompt in Windows or terminal in Linux.

To view the help documentation, type the following command:

```
<name of tool> --help
```

Table 6. Command-line tools

Command-line Tools	Description
niosv-shell	To open the Command Shell
niosv-bsp[3]	To create or update a BSP settings file and create the BSP files.
niosv-app[3]	To generate and configure an application project.
cmake	To generate a project build-system.
make	To generate the project .elf file.
niosv-download[3]	To download the ELF file to a Nios V/m processor.
juart-terminal[3]	To monitor stdout and stderr, and to provide input to a Nios V/m processor subsystem through stdin. This tool only applies to the JTAG UART IP when it is connected to the Nios V/m processor.
elf2hex[3]	To translate the .elf file to .hex format for memory initialization.
openocd[4]	To execute OpenOCD.
openocd-cfg-gen	To generate the OpenOCD configuration file

6.2. CMakeLists.txt

The Nios V/m design flow generates CMakeList.txt files and pass the files to CMake to generate build files for Nios V/m C/C++ software projects.

There are two kinds of CMakeLists.txt:

[3] This tool can only run within Nios V Command Shell.

[4] Intel recommends that you use OpenOCD debugging in Eclipse Embedded CDT. Refer to Debug on page 19 for the OpenOCD setup. CLI is also an option to start the OpenOCD debugging.

- BSP `CMakeLists.txt` – A `CMakeLists.txt`, generated from niosv-bsp, is a CMake recipe that describes how to generate build files for the BSP project. The `CMakeLists.txt` lists the files and directories targeted by the build, and the included `toolchain.cmake` file lists compiler configurations for the build.
- Application or user library `CMakeLists.txt` - A `CMakeLists.txt`, generated from niosv-app that describes how to generate build files to build an application or user library with user-provided source files and links with the specific BSP.

Intel recommends you to use the BSP Editor in Platform Designer to manage and modify the BSP `CMakeLists.txt`. As for the application or user library, you can use the generated `CMakeLists.txt` or write on your own `CMakeLists.txt`.

6.3. Hardware Abstraction Layer (HAL) driver

The HAL serves as a device driver package, providing a consistent interface to the peripherals in your system. Like Nios II processor system, the HAL drivers are also supported in the Nios V/m processor system.

Notable difference between Nios V/m processor and Nios II processor is Nios V/m supports 16 general purpose interrupts whereas Nios II processor supports 32 general purpose interrupts. To account for this, HAL interrupt APIs using interrupt numbers can accept values up to 16 only.

7. Document Revision History for Nios V Processor Quick Start Guide

Document Version	Intel Quartus Prime Version	IP Version	Changes
2021.10.04	21.3	21.1.1	Initial release.