AN 730: Nios II Processor Booting Methods in MAX 10 FPGA Devices

AN-730
2017.02.21
1 Nios II Processor Booting Methods in MAX 10 FPGA Devices ................................................. 3
  1.1 Overview .......................................................................................................................... 3
  1.2 Abbreviations................................................................................................................... 3
  1.3 Prerequisite....................................................................................................................... 4
  1.4 MAX 10 FPGA On-chip Flash Overview ......................................................................... 4
    1.4.1 MAX 10 FPGA On-chip Flash ................................................................................. 6
    1.4.2 ERAM Preload Option ............................................................................................. 6
  1.5 Nios II Processor Boot Options and Guidelines................................................................. 7
    1.5.1 Boot Option 1 and Option 2 ..................................................................................... 7
    1.5.2 Boot Option 3: Nios II Processor Application Executes in-place from OCRAM .... 37
    1.5.3 Boot Option 4 and Option 5 .................................................................................... 48
    1.5.4 Summary of Nios II Processor Vector Configurations and BSP Settings ............ 76
  1.6 Appendix A: Booting Elements ....................................................................................... 78
    1.6.1 Nios II Processor Memcpy-based Boot Copier ..................................................... 78
    1.6.2 The alt_load() function ......................................................................................... 79
    1.6.3 Nios II SBT Makefile “mem_init_generate” Target .............................................. 79
    1.6.4 The Convert Programing Files Option ..................................................................... 79
  1.7 Appendix B: Boot Time Performance Analysis ................................................................. 80
    1.7.1 Boot Time Measurement Strategy .......................................................................... 81
    1.7.2 Reducing Nios II Boot Time in MAX 10 FPGA Design ......................................... 82
    1.7.3 Boot Time Performance and Estimation ................................................................. 85
  1.8 Document Revision History ............................................................................................ 89
1 Nios II Processor Booting Methods in MAX 10 FPGA Devices

1.1 Overview

This document describes the various boot or software execution options available with the Nios II processor and MAX 10 FPGAs.

You can configure the Nios® II processor to boot and execute software from different memory locations, including the MAX10 FPGA on-chip RAM and UFM.

MAX 10 FPGA devices contain on-chip flash that is segmented into two parts:

- Configuration Flash Memory (CFM)—store the hardware configuration data for MAX 10 FPGAs.
- User Flash Memory (UFM)—stores the user data or software applications.

1.2 Abbreviations

Table 1. List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>App</td>
<td>Application</td>
</tr>
<tr>
<td>CFM</td>
<td>Configuration Flash Memory</td>
</tr>
<tr>
<td>DDR3</td>
<td>Double Data Rate type-3 synchronous dynamic random-access memory</td>
</tr>
<tr>
<td>EMIF</td>
<td>External Memory Interface</td>
</tr>
<tr>
<td>ERAM</td>
<td>Embedded Random Access Memory</td>
</tr>
<tr>
<td>HEX/.hex</td>
<td>Hexadecimal File¹</td>
</tr>
<tr>
<td>memcpy</td>
<td>Memory copy</td>
</tr>
<tr>
<td>OCGRAM</td>
<td>On-Chip RAM</td>
</tr>
<tr>
<td>POF</td>
<td>Programmer Object File</td>
</tr>
<tr>
<td>QSPI</td>
<td>Quad Serial Parallel Interface</td>
</tr>
<tr>
<td>RAM</td>
<td>Random Access Memory</td>
</tr>
<tr>
<td>SBT</td>
<td>Software Build Tools</td>
</tr>
<tr>
<td>SOF/.sof</td>
<td>SRAM Object File</td>
</tr>
</tbody>
</table>

¹ An ASCII text file with the extension of .hex that stores the initial memory values for a memory block.
1.3 Prerequisite

You are required to have the knowledge of instantiating and developing a Nios II processor based system. Intel recommends you to go through the online tutorials and training materials provided at http://www.altera.com/education/edu-index.html before using this application note.

Related Links

- **Nios II Gen2 Hardware Development Tutorial.**
  A step-by-step procedure to build a Nios II Gen2 processor system.
- **Getting Started with the Graphical User Interface.**
  This document provides the details of the Nios II Software Build Tools using graphical user interface.

1.4 MAX 10 FPGA On-chip Flash Overview

The Altera On-chip Flash IP core supports the following features:

- Read or write accesses to UFM and CFM sectors using the Avalon MM data and control slave interface.
- Supports page erase, sector erase and sector write.
- Simulation model for UFM read / write accesses using various EDA simulation tool.

Table 2. On-chip Flash Regions in MAX 10 FPGA Devices

<table>
<thead>
<tr>
<th>Flash Regions</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration Flash Memory (sectors CFM0-2)</td>
<td>FPGA configuration file storage</td>
</tr>
<tr>
<td>User Flash Memory (sectors UFM0-1)</td>
<td>Nios II processor application and/or user data</td>
</tr>
</tbody>
</table>

MAX 10 FPGA devices support several configuration modes and some of these modes allow CFM1 and CFM2 to be used as an additional UFM region. The following table shows the storage location of the FPGA configuration images based on the MAX 10 FPGA's configuration modes.
### Table 3. Storage Location of FPGA Configuration Images

<table>
<thead>
<tr>
<th>Configuration Mode</th>
<th>CFM2$^2$</th>
<th>CFM1$^2$</th>
<th>CFM0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dual images</td>
<td>Compressed Image 2</td>
<td>Compressed Image 1</td>
<td></td>
</tr>
<tr>
<td>Single uncompressed image</td>
<td>UFM$^3$</td>
<td>Uncompressed image</td>
<td></td>
</tr>
<tr>
<td>Single uncompressed image with Memory Initialization</td>
<td>Uncompressed image (with pre-initialized on-chip memory content)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Single compressed image with Memory Initialization</td>
<td>Compressed image (with pre-initialized on-chip memory content)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Single compressed image with Memory Initialization</td>
<td>UFM$^3$</td>
<td>Compressed Image</td>
<td></td>
</tr>
</tbody>
</table>

You must use the Altera On-chip Flash IP core to access to the flash memory in MAX 10 FPGAs. You can instantiate and connect the Altera On-chip Flash IP to the Nios II processor using the Qsys system design tool in the Quartus II software. The Nios II soft core processor uses the Avalon® Memory-Mapped (Avalon-MM) interface to communicate with the Altera On-chip Flash IP.

### Figure 1. Connection Example for The Altera On-chip Flash IP and the Nios II Processor

<table>
<thead>
<tr>
<th>Use</th>
<th>Connections</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>√</td>
<td>clk_0</td>
<td>Clock Source</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>clk_in</td>
<td>Clock Input</td>
</tr>
<tr>
<td></td>
<td></td>
<td>clk_in_reset</td>
<td>Reset Input</td>
</tr>
<tr>
<td></td>
<td></td>
<td>clk</td>
<td>Clock Output</td>
</tr>
<tr>
<td></td>
<td></td>
<td>clk_reset</td>
<td>Reset Output</td>
</tr>
<tr>
<td></td>
<td></td>
<td>nios2_gen2_0</td>
<td>Nios II Processor</td>
</tr>
<tr>
<td></td>
<td></td>
<td>clk</td>
<td>Clock Input</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reset</td>
<td>Reset Input</td>
</tr>
<tr>
<td></td>
<td></td>
<td>data_master</td>
<td>Avalon Memory Mapped Master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>instruction_master</td>
<td>Avalon Memory Mapped Master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>irq</td>
<td>Interrupt Receiver</td>
</tr>
<tr>
<td></td>
<td></td>
<td>debug_reset_request</td>
<td>Reset Output</td>
</tr>
<tr>
<td></td>
<td></td>
<td>debug_mem_slave</td>
<td>Avalon Memory Mapped Slave</td>
</tr>
<tr>
<td></td>
<td></td>
<td>custom_instruction</td>
<td>Custom Instruction Master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>onchip_flash_0</td>
<td>Altera On-Chip Flash</td>
</tr>
<tr>
<td></td>
<td></td>
<td>clk</td>
<td>Clock Input</td>
</tr>
<tr>
<td></td>
<td></td>
<td>nreset</td>
<td>Reset Input</td>
</tr>
<tr>
<td></td>
<td></td>
<td>data</td>
<td>Avalon Memory Mapped Slave</td>
</tr>
<tr>
<td></td>
<td></td>
<td>csr</td>
<td>Avalon Memory Mapped Slave</td>
</tr>
</tbody>
</table>

#### Related Links
- MAX 10 FPGA Configuration User Guide
- MAX 10 User Flash Memory User Guide

2 Sector is NOT supported in 10M02 device.

3 The CFM sector is configured as virtual UFM.
1.4.1 MAX 10 FPGA On-chip Flash

The Altera On-chip Flash IP core can provide access to five flash sectors:

- UFM0
- UFM1
- CFM0
- CFM1
- CFM2

Important facts about UFM and CFM sectors:

- CFM sectors are intended for configuration (bitstream) data (*.pof) storage.
- You can store user data in the UFM sectors.
- Certain devices do not have a UFM1 sector. You can refer to Table 4 on page 6 for available sectors in each individual MAX 10 FPGA device.
- You can configure CFM2 as a virtual UFM by selecting “Single Uncompressed Image” configuration mode.
- You can configure CFM2 and CFM1 as a virtual UFM by selecting “Single Compressed Image” configuration mode.
- The size of each sector varies with the selected MAX 10 FPGA devices.

Table 4. UFM and CFM Sector Size

This table lists the dimensions of the UFM and CFM arrays.

<table>
<thead>
<tr>
<th>Device</th>
<th>Pages per Sector</th>
<th>Page Size (Kbit)</th>
<th>Maximum User Flash Memory Size (Kbit)4</th>
<th>Total Configuration Memory Size (Kbit)</th>
<th>OCRAM Size (Kbit)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>UFM1</td>
<td>UFM0</td>
<td>CFM2</td>
<td>CFM1</td>
<td>CFM0</td>
</tr>
<tr>
<td>10M02</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>34</td>
</tr>
<tr>
<td>10M04</td>
<td>0</td>
<td>8</td>
<td>41</td>
<td>29</td>
<td>70</td>
</tr>
<tr>
<td>10M08</td>
<td>8</td>
<td>8</td>
<td>41</td>
<td>29</td>
<td>70</td>
</tr>
<tr>
<td>10M16</td>
<td>4</td>
<td>4</td>
<td>38</td>
<td>28</td>
<td>66</td>
</tr>
<tr>
<td>10M25</td>
<td>4</td>
<td>4</td>
<td>52</td>
<td>40</td>
<td>92</td>
</tr>
<tr>
<td>10M40</td>
<td>4</td>
<td>4</td>
<td>48</td>
<td>36</td>
<td>84</td>
</tr>
<tr>
<td>10M50</td>
<td>4</td>
<td>4</td>
<td>48</td>
<td>36</td>
<td>84</td>
</tr>
</tbody>
</table>

1.4.2 ERAM Preload Option

The FPGA configuration data may contain MAX 10 FPGA On-chip RAM or ERAM initialization data. The ERAM preload occurs during FPGA configuration before the device enters user mode. The ERAM preload option allows initialization data for the On-chip RAM to be stored in the CFM sectors. The initialization data includes the Nios II processor software or any type of application data.

4 The maximum possible value, which is dependent on the configuration mode you select.
All MAX 10 FPGA devices except for the MAX 10 10M02 supports dual FPGA configuration images. The ERAM preload is disabled when you select the dual configuration images mode.

When the ERAM preload feature is set to OFF, features that require initialization of on-chip RAM will not work. The ERAM preload option is set to OFF by default.

Selecting Single Compressed Image with Memory Initialization or Single Uncompressed Image with Memory Initialization configuration mode will enable the ERAM Preload option but will reduce the size of UFM available.

1.5 Nios II Processor Boot Options and Guidelines

There are 5 Nios II processor boot options available.

Table 5. Nios II Processor Boot Options

<table>
<thead>
<tr>
<th>Boot Option</th>
<th>Application Code Storage</th>
<th>Application Runtime Location</th>
<th>Boot Method</th>
<th>Supported Configuration Mode</th>
</tr>
</thead>
</table>
| Option 1: Nios II processor application executes in-place from Altera On-chip Flash (UFM) | UFM                      | UFM (XIP) + OCRAM/ External RAM (for data) | alt_load() function | • Single Uncompressed Image  
• Single Compressed Image  
• Dual Compressed Images |
| Option 2: Nios II processor application copied from UFM to RAM using boot copier | UFM                      | OCRAM/ External RAM          | memcpy-based boot copier | • Single Uncompressed Image  
• Single Compressed Image  
• Dual Compressed Images |
| Option 3: Nios II processor application executes in-place from OCRAM | OCRAM                    | OCRAM                        | No boot copier is required | • Single uncompressed image with Memory Initialization  
• Single compressed image with Memory Initialization |
| Option 4: Nios II processor application executes in-place from QSPI flash | QSPI flash               | QSPI (XIP) + OCRAM/ External RAM (for data) | alt_load() function | • Single Uncompressed Image  
• Single Compressed Image  
• Dual Compressed Images |
| Option 5: Nios II processor application copied from QSPI flash to RAM using boot copier | QSPI flash               | OCRAM/ External RAM          | memcpy-based boot copier | • Single Uncompressed Image  
• Single Compressed Image  
• Dual Compressed Images |

1.5.1 Boot Option 1 and Option 2

**Boot Option 1: Nios II Processor Application Executes in-place from Altera On-chip Flash (UFM)**

This solution is suitable for Nios II processor applications which require limited on-chip memory usage. The alt_load() function operates as a mini boot copier which copies the data sections (.rodata, .rwdata, or .exceptions) from boot memory to RAM based on the BSP settings. The code section (.text), which is a read only section,
remains in the Altera On-chip Flash memory region. This setup minimizes the RAM usage but may limit the code execution performance as access to the flash memory is slower than the on-chip RAM.

The Nios II processor application is programmed into the UFM sector. The Nios II processor’s reset vector points to the UFM sector in order to execute code from the UFM after the system resets.

If you are debugging the application using the source-level debugger, you must use a hardware break-point to debug because the UFM does not support random memory access. Random memory access is required for soft break-point debug.

Figure 2.  Boot Option 1 Block Diagram

Table 6.  RAM and ROM Size Requirement for Boot Option 1

You can manually determine the required RAM size for the Altera On-Chip Flash by referring to the initial part of the .objdump file, created when you build your application.

<table>
<thead>
<tr>
<th>RAM Size Requirement</th>
<th>ROM Size Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equivalent to the dynamic memory space usage during run time which is the sum of the maximum heap and stack size.</td>
<td>Executable code must not exceed the size of the UFM.</td>
</tr>
</tbody>
</table>

Boot Option 2: Nios II Processor Application Copied from UFM to RAM using Boot Copier

Intel recommends this solution for MAX 10 FPGA Nios II processor system designs where there may be multiple iterations of application software development and when high system performance is required. The boot copier is located within the UFM at an offset which is the same address with the reset vector. The Nios II application is located next to the boot copier.

For this boot option, the Nios II processor starts executing the boot copier upon system reset to copy the application from the UFM sector to the OCRAM or external RAM. Once copying is complete, the Nios II processor transfers the program control over to the application.
Figure 3. Boot Option 2 Block Diagram

Table 7. RAM and ROM Size Requirement for Boot Option 2

- You can manually determine the required RAM size for the Altera On-Chip Flash by referring to the initial part of the .objdump file, created when you build your application.
- The OCRAM size is limited, you have to ensure that the size is sufficient for application execution.

<table>
<thead>
<tr>
<th>RAM Size Requirement</th>
<th>ROM Size Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equivalent to the executable code and dynamic memory size required by user program.</td>
<td>Executable code and boot copier must not exceed the size of the UFM.</td>
</tr>
</tbody>
</table>
### Figure 4. Configuration and Booting Flow for Option 1 and Option 2

**Design**
- Create your Nios II Processor based project using Qsys system.

**For Option 1:**
- Add hardware breakpoint option to the Nios II processor.

**For Option 2:**
- Ensure that there is external RAM or onchip RAM in the system design.
- Instantiate the Altera Dual Configuration IP into your Qsys system if you opt for Dual Images configuration mode.

**MAX 10 FPGA Configuration and Compilation**
- Set the same internal configuration mode in Altera On-chip Flash IP in Qsys and Quartus II software.
- Set Nios II processor reset and exception vector based on your boot option.
- Uncheck Initialize memory content option in Altera On-chip Flash IP to include UFM data (.HEX) separately during programming files conversion.
- Optionally, you can check Initialize memory content option to include UFM data (.HEX) into the SOF file.
- Generate your design in Qsys.
- Compile your project in Quartus II software.

**BSP Editor Settings**
- Create Nios II processor HAL BSP.
- Edit Nios II processor BSP settings and Linker Script in BSP Editor.
- Generate BSP project.

**Application & Project Recompilation**
- Develop Nios II processor application code.
- Compile Nios II processor application and generate HEX file using makefile mem_init_generate target.
- Recompile your project in Quartus II software if you check Initialize memory content option in Altera On-Chip Flash IP.

**Programming Files Conversion**
- Generate the .pof file using Convert Programming Files feature in Quartus II software

**Download**
- Program the .pof file into your MAX 10 device.

**Reset & Boot**
- Reset your MAX 10 device using the reset button or power cycle your hardware.

(1) Project re-compilation is needed if Initialize Flash Content option was checked in Altera On-Chip Flash IP. You can ignore this step if Initialize Flash Content option was unchecked in Altera On-Chip Flash IP.

### Related Links
1.5.1.1 Single Uncompressed/Compressed Image Bootable System Guideline

1.5.1.1.1 Qsys Settings

1. In the Nios II Processor parameter editor, set the reset vector memory and exception vector memory based on the boot options below:

<table>
<thead>
<tr>
<th>Boot Option</th>
<th>Reset vector memory:</th>
<th>Exception vector memory:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Option 1a</td>
<td>Altera On-chip Flash</td>
<td>OCRAM/ External RAM</td>
</tr>
<tr>
<td>Option 1b</td>
<td>Altera On-chip Flash</td>
<td>Altera On-chip Flash</td>
</tr>
<tr>
<td>Option 2</td>
<td>Altera On-chip Flash</td>
<td>OCRAM/ External RAM</td>
</tr>
</tbody>
</table>

Figure 5. Nios II Parameter Editor Settings Boot Option 1a and 2

2. In the Altera On-chip Flash IP parameter editor, set the Configuration Mode to Single Uncompressed Image or Single Compressed Image.

5 You can set the exception vector for Boot Option 1 to OCRAM or External RAM (Option 1a) or Altera On-chip Flash (option 1b) according to your design preference.

6 Boot option 1a which sets exception vector memory to OCRAM or External RAM is recommended to make the interrupt processing faster.
Refer to the following table for the options to program UFM data (HEX file) and settings required in Altera On-chip Flash IP.

<table>
<thead>
<tr>
<th>Options to program UFM data</th>
<th>Method</th>
<th>Settings in Altera On-Chip Flash IP</th>
</tr>
</thead>
</table>
| Option 1: Initialize the UFM data in the SOF | Quartus II includes the UFM initialization data in the SOF during compilation. SOF recompilation is needed if there are changes in the UFM data. | a. Check **Initialize flash content**  
b. If default path is used, add `meminit.qip` generated during “make mem_init_generate” into Quartus II project. Refer to Figure 9 on page 13. Make sure the generated HEX naming matches the default naming.  
c. If non-default path is selected, enable the **Enable non-default initialization file** and specify the path of the HEX file.  
*Note:* For more information about Steps 2 and 3, refer to HEX File Generation section. |
| Option 2: Combine UFM data with a compiled SOF during programming files (POF) conversion | UFM data is combined with the compiled SOF during the programming files conversion. SOF recompilation is NOT needed even if there are changes in the UFM data. | Uncheck **Initialize flash content** |

---

7 This is the recommended method for application developer. You are not required to recompile SOF file for application changes during development.
Figure 8. Initialize Flash Contents with Default Initialization File

Figure 9. Adding meminit.qip File in Quartus II

Figure 10. Initialize Flash Content with Non-default Initialization File
4. Ensure that the Altera On-chip Flash csr port is connected to the Nios II processor data_master to enable write and erase operations.

**Figure 11. CSR Connection to Nios II data_master**

5. Click **Generate HDL**, the **Generation** dialog box appears.

6. Specify output file generation options, and then click **Generate**.

**Related Links**

- [HEX File Generation on page 19](#)

**1.5.1.1.2 Quartus II Software Settings**

1. In the Quartus II software, click on **Assignment -> Device -> Device and Pin Options -> Configuration**. Set **Configuration mode** to **Single Uncompressed Image** or **Single Compressed Image**.

8 The size of UFM shown will vary according to your device selection.
Figure 12. Configuration Mode Selection in Quartus II Software

![Configuration Mode Selection in Quartus II Software](image)

**Note:** If the configuration mode setting in Quartus II software and Qsys parameter editor is different, the Quartus II project compilation will fail with the following error message.

2. Click **OK** to exit the **Device and Pin Options window**.
3. Click **OK** to exit the **Device** window.
4. Click **Start Compilation** to compile your project and generate the .sof file.

**Related Links**

HEX File Generation on page 19

**1.5.1.1.3 BSP Editor Settings**

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

1. In the Nios II SBT tool, right click on your BSP project in the **Project Explorer** window. Select **Nios II > BSP Editor...** to open the **Nios II BSP Editor**.
2. In Nios II BSP Editor, click on **Advanced** tab under **Settings**.
3. Click on **hal** to expand the list.
4. Click on **linker** to expand the list.
5. Based on the boot option used, do one of the following:

   — For boot option 1a, if exception vector memory is set to OCRAM or External RAM, enable the following:
     - `allow_code_at_reset`
     - `enable_alt_load`
     - `enable_alt_load_copy_rodata`
     - `enable_alt_load_copy_rwdata`
     - `enable_alt_load_copy_exceptions`

   — For boot option 1b, if exception vector memory is set to Altera On-chip Flash, enable the following:
     - `allow_code_at_reset`
     - `enable_alt_load`
     - `enable_alt_load_copy_rodata`
     - `enable_alt_load_copy_rwdata`

   — For boot option 2, leave all the hal.linker settings unchecked.
Figure 14. Advanced.hal.linker Settings for Boot Option 1a

Figure 15. Advanced.hal.linker Settings for Boot Option 1b
6. Click on **Linker Script** tab in the Nios II BSP Editor.

7. Based on the boot option used, do one of the following:

   — For boot option 1a and 1b, set the `.text` item in the **Linker Section Name** to the Altera On-chip Flash in the **Linker Region Name**. Set the rest of the items in the **Linker Section Name** list to the Altera On-chip Memory (OCRAM) or external RAM.

   — For boot option 2, set all of the items in the **Linker Section Name** list to Altera On-chip Memory (OCRAM) or external RAM.
1.5.1.1.4 HEX File Generation

1. In the Nios II SBT tool, right click on your project in the Project Explorer window.
2. Click **Make Targets -> Build...**, the Make Targets dialog box appears. You can also press shift + F9 to trigger the Make Target dialog box.
3. Select **mem_init_generate**.
4. Click **Build** to generate the HEX file.
5. The “mem_init_generate” macro will create two HEX files; `<on_chip_ram.hex>` and `<on_chip_flash.hex>`. The `<on_chip_ram.hex>` will be used for boot option 3 and `<on_chip_flash.hex>` is used for boot option 1 and 2.

**Notes:**
- The `mem_init_generate` target also generates a Quartus II IP file (`meminit.qip`). Quartus II software will refer to the `meminit.qip` for the location of the initialization files.
- All these files can be found under `"<project_folder>/software/<application_name>/mem_init"` folder.

6. Recompile your project in Quartus II software if you check **Initialize memory content** option in Altera On-Chip Flash IP. This is to include the software data (.hex) into the SOF file.

**Related Links**
- Qsys Settings on page 11
- Quartus II Software Settings on page 14

**1.5.1.1.5 Programmer Object File (.pof) Generation**

1. In Quartus II, click on **Convert Programming Files (.pof)** from the **File** tab.
2. Choose **Programmer Object File** as **Programming file type**.
3. Set **Mode** to **Internal Configuration**.
4. Click on **Options/Boot info...**, the MAX 10 Device Options dialog box appears.

5. Based on the **Initialize flash content** settings in the Altera On-chip Flash IP, do one of the following:

   — If **Initialize flash content** is checked, the UFM initialization data was included in the SOF during Quartus II compilation. Select **Page 0** for **UFM source** option. Click **OK** and proceed to next step.

   — If **Initialize flash content** is not checked, choose **Load memory file** for **UFM source**: option. Browse to the generated Altera On-chip Flash HEX file (**on_chip_flash.hex**) in the **File path**: and click **OK**. This will add UFM data separately to the SOF file during the programming file conversion.
Figure 21. Setting Page_0 for UFM Source if Initialize flash content is Checked

![Max 10 Device Options](image)

- Power On Reset scheme: Instant ON
- ✔ Set I/O to weak pull-up prior usermode
- ✔ Auto-reconfigure from secondary image when initial image fails
- ✔ Use secondary image ISP data as default setting when available

Security
- ✔ Verify protect
- ✔ Allow encrypted POF only

Dual Config
- ✔ Enable watchdog

Watch value:

User Flash Memory
- UFM source: Page_0
- File path:

Description:

Specifies device Power On Reset (POR) scheme.
6. In the Convert Programming File dialog box, at the **Input files to convert** section, click **Add File...** and point to the generated Quartus II .sof file.
7. Click **Generate** to create the .pof file.
8. Program the .pof file into your MAX 10 device.

### 1.5.1.2 Dual Compressed Images Bootable System Guideline

#### 1.5.1.2.1 Qsys Settings

1. In the Nios II Processor parameter editor, set the reset vector memory and exception vector memory based on the boot options below:

<table>
<thead>
<tr>
<th>Boot Option</th>
<th>Reset vector memory:</th>
<th>Exception vector memory:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Option 1a(^{10})</td>
<td>Altera On-chip Flash</td>
<td>OCRAM/ External RAM</td>
</tr>
<tr>
<td>Option 1b(^{9})</td>
<td>Altera On-chip Flash</td>
<td>Altera On-chip Flash</td>
</tr>
<tr>
<td>Option 2</td>
<td>Altera On-chip Flash</td>
<td>OCRAM/ External RAM</td>
</tr>
</tbody>
</table>

9. You can set the exception vector for Boot Option 1 to OCRAM or External RAM (Option 1a) or Altera On-chip Flash (option 1b) according to your design preference.

10. Boot option 1a which sets exception vector memory to OCRAM or External RAM is recommended to make the interrupt processing faster.
2. In the Altera On-chip Flash IP parameter editor, set the Configuration Mode to Dual Compressed Images.

3. Refer to the following table for the options to program UFM data (HEX file) and settings required in Altera On-chip Flash IP.

<table>
<thead>
<tr>
<th>Options to program UFM data</th>
<th>Method</th>
<th>Settings in Altera On-Chip Flash IP</th>
</tr>
</thead>
</table>
| Option 1: Initialize the UFM data in the SOF | Quartus II includes the UFM initialization data in the SOF during compilation. SOF recompilation is needed if there are changes in the UFM data. | a. Check Initialize flash content  
b. If default path is used, add meminit.qip generated during "make mem_init_generate" into Quartus II project. Refer to Figure 27 on page 26. Make sure the generated HEX naming matches the default naming.  
c. If non-default path is selected, enable the Enable non-default initialization file and specify the path of the HEX file.  
Note: For more information about Steps 2 and 3, refer to HEX File Generation section. |
| Option 2: Combine UFM data with a compiled SOF during programming files (POF) conversion | UFM data is combined with the compiled SOF during the programming files conversion. SOF recompilation is NOT needed even if there are changes in the UFM data. | Uncheck Initialize flash content |
This is the recommended method for application developer. You are not required to recompile SOF file for application changes during development.
4. Ensure that the Altera On-chip Flash csr port is connected to the Nios II processor data_master to enable write and erase operations.

5. Ensure the Altera Dual Configuration IP is instantiated in Qsys to enable dual images configuration.

6. Click Generate HDL, the Generation dialog box appears.

7. Specify output file generation options, and then click Generate.

Related Links
HEX File Generation on page 32

1.5.1.2.2 Quartus II Software Settings

1. In the Quartus II software, click on Assignment -> Device -> Device and Pin Options -> Configuration. Set Configuration mode to Dual Compressed Images.\(^\text{12}\)

---

\(^{12}\) The size of UFM sector will vary according to your device selection.
Figure 30. **Dual Compressed Images Configuration Mode in Quartus II**

![Configuring Dual Compressed Images in Quartus II](image)

**Note:** If the configuration mode setting in Quartus II software and Qsys parameter editor is different, the Quartus II project compilation will fail with the following error message.

2. Click **OK** to exit the **Device and Pin Options window**.
3. Click **OK** to exit the **Device** window.
4. Click **Start Compilation** to compile your project and generate the .sof file.

**Related Links**

[HEX File Generation on page 32](#)

### 1.5.1.2.3 BSP Editor Settings

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

1. In the Nios II SBT tool, right click on your BSP project in the **Project Explorer** window. Select **Nios II > BSP Editor...** to open the **Nios II BSP Editor**.
2. In Nios II BSP Editor, click on **Advanced** tab under **Settings**.
3. Click on **hal** to expand the list.
4. Click on **linker** to expand the list.
5. Based on the boot option used, do one of the following:

   - For boot option 1a, if exception vector memory is set to OCRAM or External RAM, enable the following:
     - `allow_code_at_reset`
     - `enable_alt_load`
     - `enable_alt_load_copy_rodata`
     - `enable_alt_load_copy_rwdata`
     - `enable_alt_load_copy_exceptions`

   - For boot option 1b, if exception vector memory is set to Altera On-chip Flash, enable the following:
     - `allow_code_at_reset`
     - `enable_alt_load`
     - `enable_alt_load_copy_rodata`
     - `enable_alt_load_copy_rwdata`

   - For boot option 2, leave all the hal.linker settings unchecked.
Figure 32.  Advanced.hal.linker Settings for Boot Option 1a

Figure 33.  Advanced.hal.linker Settings for Boot Option 1b
6. Click on **Linker Script** tab in the Nios II BSP Editor.

7. Based on the boot option used, do one of the following:
   - For boot option 1a and 1b, set the .text item in the **Linker Section Name** to the Altera On-chip Flash in the **Linker Region Name**. Set the rest of the items in the **Linker Section Name** list to the Altera On-chip Memory (OCRAM) or external RAM.
   - For boot option 2, set all of the items in the **Linker Section Name** list to Altera On-chip Memory (OCRAM) or external RAM.
1.5.1.2.4 HEX File Generation

1. In the Nios II SBT tool, right click on your project in the Project Explorer window.
2. Click **Make Targets** -> **Build...**, the Make Targets dialog box appears. You can also press shift + F9 to trigger the Make Target dialog box.
3. Select **mem_init_generate**.
4. Click **Build** to generate the HEX file.
5. The “mem_init_generate” macro will create two HEX files; <on_chip_ram.hex> and <on_chip_flash.hex>. The <on_chip_ram.hex> will be used for boot option 3 and <on_chip_flash.hex> is used for boot option 1 and 2.

Notes:
• The mem_init_generate target also generates a Quartus II IP file (meminit.qip). Quartus II software will refer to the meminit.qip for the location of the initialization files.
• All these files can be found under "<project_folder>/software/<application_name>/mem_init" folder.

6. Recompile your project in Quartus II software if you check Initialize memory content option in Altera On-Chip Flash IP. This is to include the software data (.hex) into the SOF file.

Related Links
• Qsys Settings on page 24
• Quartus II Software Settings on page 27

1.5.1.2.5 Programmer Object File (.pof) Generation
1. In Quartus II, click on Convert Programming Files from the File tab.
2. Choose Programmer Object File as Programming file type.
3. Set Mode to Internal Configuration.
4. Click on Options/Boot info..., the MAX 10 Device Options dialog box appears.

5. Based on the Initialize flash content settings, do one of the following:

   — If Initialize flash content was checked, make sure Page_0 or Page_1 is selected for UFM source: option. Click OK.

     Note: UFM data (.HEX file) can be included in either Page_0 or Page_1 only. The Altera On-chip flash does not support two .HEX files for Dual Compressed images configuration mode.

   — If Initialize flash content was not checked, choose Load memory file for UFM source: option. Browse to the generated Altera On-chip Flash HEX file (on_chip_flash.hex) in the File path: and click OK.
Figure 39. Setting Page_0 or Page_1 for UFM Source If Initialize flash content is Checked
6. In the Convert Programming File dialog box, at the **Input files to convert** section, click **Add File...** and point to the first generated Quartus II .sof file to add the .sof file at page_0.

7. Click on **Add Sof Page** to create additional page for .sof file. This creates SOF data Page_1 automatically. Click **Add File...** and point to the second generated Quartus II .sof file to add the .sof file at page_1.
8. Click **Generate** to create the `.pof` file.
9. Program the `.pof` file into your MAX 10 device.

### 1.5.2 Boot Option 3: Nios II Processor Application Executes in-place from OCRAM

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

This option will not work in any of the following situations:

- When you select a configuration mode that does not support ERAM initialization.
- After a soft reset where the memory contents have been modified by the application and the application code has been corrupted.

**Figure 42. Boot Option 3 Block Diagram**
Table 8. RAM and ROM Size Requirement for Boot Option 3

<table>
<thead>
<tr>
<th>RAM Size Requirement</th>
<th>ROM Size Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equivalent to the executable code and dynamic memory size required by user program.</td>
<td>Not applicable for this boot option.</td>
</tr>
</tbody>
</table>

Figure 43. Configuration and Booting Flow for Option 3

1. **Design**
   - Create your Nios II Processor based project using the Qsys system.
   - Ensure that there is onchip RAM in the system design.

2. **MAX 10 FPGA Configuration and Compilation**
   - Set the same internal configuration mode in the Altera On-chip Flash IP in Qsys and Quartus II software.
   - Set the Nios II processor reset and exception vector based on your boot option.
   - Leave the Initialize memory content option unselected in Altera On-chip Flash IP. Check the Initialize memory content option in the Altera On-chip Memory IP to include software data (.HEX) into the SOF file.
   - Generate your design in Qsys.
   - Compile your project in Quartus II software.

3. **BSP Editor Settings**
   - Create the Nios II processor HAL BSP.
   - Edit the Nios II processor BSP settings and Linker Script in BSP Editor.
   - Generate the BSP project.

4. **Application & Project Recompilation**
   - Develop the Nios II processor application code.
   - Compile the Nios II processor application and generate the Hex file using makefile mem_init_generate target.
   - Recompile your project in Quartus II software to include software data (.HEX) into the SOF file.

5. **Programming Files Conversion**
   - Generate the .pof file using Convert Programming Files feature in Quartus II software

6. **Download**
   - Program the .pof file into your MAX 10 device.

7. **Reset & Boot**
   - Reset your MAX 10 device using the reset button or power cycle your hardware.
1.5.2.1 Single Uncompressed/Compressed Image with Memory Initialization Guideline

1.5.2.1.1 Qsys Settings

1. In the Nios II Processor parameter editor, set both the **Reset vector memory:** and **Exception vector memory:** to Altera On-Chip Memory (OCRAM).

![Figure 44. Nios II Parameter Editor Settings for Boot Option 3](image)

2. In the Altera On-chip Flash IP parameter editor, set the **Configuration Mode** to **Single Uncompressed Image with Memory Initialization** or **Single Compressed Image with Memory Initialization**. Leave the **Initialize flash content** unchecked. This is because the Altera On-chip flash initialization data will not be enabled.

![Figure 45. Configuration Mode with Memory Initialization Selection and Initialize Flash Content Setting](image)

3. In the Altera On-chip Memory (RAM or ROM) IP parameter editor, check **Initialize flash content**. If default path is used, add `meminit.qip` generated during "make mem_init_generate" into Quartus II project. Refer to Figure 47 on page 40. Make sure the generated HEX naming matches the default naming. If non-default path is selected, check **Enable non-default initialization file** and specify the path of the HEX file (`onchip_memory2_0.hex`).

*Note:* The `meminit.qip` stores the location information of the initialization files.
Figure 46. Enable Initialize Memory Content with Default Initialization File in On-Chip Memory Parameter Editor Settings

Figure 47. Adding meminit.qip File into Quartus II
4. Ensure that Altera On-chip Flash csr port is connected to the Nios II processor data_master to enable write and erase operations.

Figure 49. CSR Connection to Nios II data_master

5. Click Generate HDL, the Generation dialog box appears.

6. Specify output file generation options, and then click Generate.

1.5.2.1.2 Quartus II Software Settings

1. In the Quartus II software, click on Assignment -> Device -> Device and Pin Options -> Configuration. Set Configuration mode to Single Uncompressed Image with Memory Initialization or Single Compressed Image with Memory Initialization.\(^\text{13}\)

\(^\text{13}\) The size of UFM sector will vary according to your device selection.
Notes: • If the configuration mode setting in Quartus II software and Qsys parameter editor is different, the Quartus II project compilation will fail with the following error message.

2. Click OK to exit the Device and Pin Options window.
3. Click OK to exit the Device window.
4. Click Start Compilation to compile your project and generate the .sof file.

1.5.2.1.3 BSP Editor Settings

You must edit the BSP editor settings according to the selected Nios II processor boot options.
1. In the Nios II SBT tool, right click on your BSP project in the Project Explorer window. Select Nios II > BSP Editor... to open the Nios II BSP Editor.
2. In Nios II BSP Editor, click on Advanced tab under Settings.
3. Click on hal to expand the list.
4. Click on linker to expand the list.

**Figure 51.** BSP Editor Settings

5. Enable allow_code_at_reset and leave others unchecked to make sure the application starts at address 0x00.
6. Click on **Linker Script** tab in the **Nios II BSP Editor**.
7. Set all the **Linker Section Regions** to **Altera On-chip Memory (OCRAM)**.

**Figure 53. Linker Region Default Settings**

1.5.2.1.4 **HEX File Generation**

1. In the Nios II SBT tool, right click on your project in the Project Explorer window.
2. Click **Make Targets -> Build...**, the Make Targets dialog box appears. You can also press shift + F9 to trigger the Make Target dialog box.
3. Select **mem_init_generate**.
4. Click **Build** to generate the HEX file.
5. The “mem_init_generate” macro will create two HEX files; `<on_chip_ram.hex>` and `<on_chip_flash.hex>`. The `<on_chip_ram.hex>` will be used for boot option 3 and `<on_chip_flash.hex>` is used for boot option 1 and 2.

**Notes:**
- The `mem_init_generate` target also generates a Quartus II IP file (`meminit.qip`). Quartus II software will refer to the `meminit.qip` for the location of the initialization files.
- All these files can be found under `"<project_folder>/software/<application_name>/mem_init"` folder.

6. Recompile your project in Quartus II software to include the software data (.HEX) into the SOF file.

### 1.5.2.1.5 Programmer Object File (.pof) Generation

**Note:** If you are using Quartus II software version 15.0 and above, you can skip all the steps in this section. From Quartus II 15.0 onwards, the Quartus II software will generate both SOF and POF files. Since boot option 3 comprises single image, you may use the generated POF file to program into the MAX 10 FPGA device directly.

1. In Quartus II, click on **Convert Programming Files** from the **File** tab.
2. Choose **Programmer Object File** as **Programming file type**.
3. Set **Mode** to **Internal Configuration**.
4. Click on **Options/Boot info...**, the MAX 10 Device Options dialog box appears.
5. Make sure Page_0 is set as the **UFM source**: option. Click **OK**.
6. In the Convert Programming File dialog box, at the **Input files to convert** section, click **Add File...** and point to the generated Quartus II .sof file to add the .sof file at **page_0**.
7. Click **Generate** to create the `.pof` file.

8. Program the `.pof` file into your MAX 10 FPGA device.

### 1.5.3 Boot Option 4 and Option 5

**Boot Option 4: Nios II Processor Application Executes in-place from QSPI Flash**

This option is suitable for Nios II processor applications which require code space that is larger than the UFM and/or limited on-chip memory usage. It has a similar concept to boot option 1 where the Nios II processor application execute-in-place from UFM but with more memory space due to use of QSPI flash (depending on QSPI chip selection) instead of the UFM. This is an advantage for supporting larger or multiple software applications.

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

The Nios II processor reset vector points to the QSPI flash to allow code execution after the system resets. If you are debugging the application using the source-level debugger, you must use a hardware break-point because the QSPI cannot support random memory access.
Figure 58. Boot Option 4 Block Diagram

Table 9. RAM and ROM Size Requirement for Boot Option 4

<table>
<thead>
<tr>
<th>RAM Size Requirement</th>
<th>ROM Size Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equivalent to the dynamic memory space usage during run time which is the sum of the maximum heap and stack size.</td>
<td>Executable code must not exceed the size of the QSPI flash.</td>
</tr>
</tbody>
</table>

Option 5: Nios II Processor Application Copied from QSPI Flash to RAM Using Boot Copier

Using a boot copier to copy the Nios II application from QSPI flash to RAM is suitable when multiple iterations of application software development and high system performance are required. It has a similar concept to boot option 2 where Nios II processor application copied from QSPI flash to RAM using boot copier but with larger memory space due to QSPI flash (depending on QSPI chip selection). This is an advantage for supporting software applications that require larger program space and high software performance (compared to XIP).

The Nios II SBT tool automatically adds the Nios II processor memcpy-based boot copier to the system when the executable file (.elf) is converted to the memory initialization file (.hex). The boot copier is located at the base address of the HEX data, followed by the application.

For this boot option, the Nios II processor starts executing the boot copier software upon system reset that copies the application from the QSPI to the on-chip memory or external RAM. Once this process is complete, the Nios II processor transfers the program control over to the application.
Table 10. RAM and ROM Size Requirement for Boot Option 5

<table>
<thead>
<tr>
<th>RAM Size Requirement</th>
<th>ROM Size Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equivalent to the executable code and dynamic memory size required by user program.</td>
<td>Executable code and boot copier must not exceed the size of the QSPI flash.</td>
</tr>
</tbody>
</table>
Figure 60. Configuration and Booting Flow for Option 4 and Option 5

**Design**
- Create your Nios II Processor based project using Qsys system.
- Ensure that there is external RAM or onchip RAM and Generic Quad SPI controller IP in the system design.
- Instantiate the Altera Dual Configuration IP into your Qsys system if you opt for Dual Images configuration mode.

**MAX 10 FPGA Configuration and Compilation**
- Set Nios II processor reset and exception vector based on your boot option.
- Generate your design in Qsys.
- Compile your project in Quartus II software.

**BSP Editor Settings**
- Create Nios II processor HAL BSP based on the .sopcinfo file created from Qsys generation.
- Edit Nios II processor BSP settings and Linker Script in BSP Editor based on your boot option.
- Generate BSP project.

**Application & Project Recompilation**
- Develop Nios II processor application code.
- Compile Nios II processor application and generate .hex file using makefile mem_init_generate target.

**Programming Files Conversion**
- Convert the generated .hex file to .pof file using Convert Programming Files feature in Quartus II software.

**Download**
- Using Quartus II programmer, program the parallel loader .sof into the device to enable parallel loader.
- Once the programming is completed, click on Auto Detect button. Make sure the QSPI flash is detected.
- Program the converted .pof (Nios II application) into the QSPI flash.
- Program the hardware .pof file (from Quartus II) into the MAX10 device.

**Reset & Boot**
- Reset your MAX 10 FPGA device using the reset button or power cycle your hardware.

**Related Links**
1.5.3.1 Single Uncompressed/Compressed Image Bootable System Guideline

1.5.3.1.1 Design Creation

1. Create your Nios II processor project using Quartus II and Qsys.
2. Make sure the Generic Quad SPI Controller IP is added into your Qsys system. Refer to the diagram below for IP connection in Qsys.

Figure 61. Connection for Generic Quad SPI Controller IP

Note: The maximum input clock for Generic Quad SPI Controller IP is 25 MHz. The input clock must not exceed this maximum value.

1.5.3.1.2 Qsys Settings

1. In the Nios II Processor parameter editor, set the reset vector memory and exception vector memory based on the boot options below:

<table>
<thead>
<tr>
<th>Boot Option</th>
<th>Reset vector memory</th>
<th>Exception vector memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Option 4a</td>
<td>QSPI flash</td>
<td>OCRAM/External RAM</td>
</tr>
<tr>
<td>Option 4b</td>
<td>QSPI flash</td>
<td>QSPI Flash</td>
</tr>
<tr>
<td>Option 5</td>
<td>QSPI flash</td>
<td>OCRAM/External RAM</td>
</tr>
</tbody>
</table>

14 You can set the exception vector for Boot Option 4 to OCRAM/External RAM (Option 4a) or QSPI Flash (Option 4b) according to your design preference.

15 Boot option 4a which sets exception vector memory to OCRAM/External RAM is recommended to make the interrupt processing faster.
Figure 62. Nios II Parameter Editor Settings Boot Option 4a and 5

*Note:* `mem_if_ddr3_emif_0.avl` is the external memory (DDR3) used in this example.

![Parameter Editor Settings Boot Option 4a](image)

Figure 63. Nios II Parameter Editor Settings Boot Option 4b

![Parameter Editor Settings Boot Option 4b](image)

2. Open *Altera Generic Quad SPI* controller parameter editor. Change the *Configuration device type* to the QSPI flash selection and make sure the *I/O mode* is set to *QUAD*. 
3. Click **Generate HDL**, the Generation dialog box appears.
4. Specify output file generation options and then click **Generate**.

### 1.5.3.1.3 Quartus II Software Settings

1. In the Quartus II software, click on **Assignment** -> **Device** -> **Device and Pin Options** -> **Configuration**. Set **Configuration mode** to **Single Uncompressed Image** or **Single Compressed Image**.

### Figure 65. Configuration Mode Selection in Quartus II Software

2. Click **OK** to exit the **Device and Pin Options window**.
3. Click **OK** to exit the **Device** window.
4. Click **Start Compilation** to compile your project.
Note: SOF to POF file conversion is not required because there is only single hardware image and Nios II application data will be loaded into the QSPI flash separately. You can use the POF generated during Quartus II project compilation to program into the MAX 10 FPGA.

Related Links
Programming Hardware Design POF File into the MAX10 FPGA on page 62

1.5.3.1.4 BSP Editor Settings

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

1. In the Nios II SBT tool, right click on your BSP project in the Project Explorer window. Select Nios II > BSP Editor... to open the Nios II BSP Editor.
2. In Nios II BSP Editor, click on Advanced tab under Settings.
3. Click on hal to expand the list.
4. Click on linker to expand the list.

Figure 66. BSP Editor Settings

5. Based on the boot option used, do one of the following:
   - For boot option 4a, if exception vector memory is set to OCRAM or External RAM, enable the following:
     - allow_code_at_reset
     - enable_alt_load
     - enable_alt_load_copy_rodata
     - enable_alt_load_copy_rwdata
     - enable_alt_load_copy_exceptions
— For boot option 4b, if exception vector memory is set to QSPI flash, enable the following:

- `allow_code_at_reset`
- `enable_alt_load`
- `enable_alt_load_copy_rodata`
- `enable_alt_load_copy_rwdata`

— For boot option 5, leave all the hal.linker settings unchecked.

Figure 67. Advanced.hal.linker Settings for Boot Option 4a

Figure 68. Advanced.hal.linker Settings for Boot Option 4b
Figure 69. Advanced.hal.linker Settings for Boot Option 5

6. Click on **Linker Script** tab in the Nios II BSP Editor.

7. Based on the boot option used, do one of the following:

   — For boot option 4a and 4b, set the .text item in the **Linker Section Name** to the QSPI flash in the **Linker Region Name**. Set the rest of the items in the **Linker Section Name** list to the Altera On-chip Memory (OCRAM) or external RAM.

   — For boot option 5, set all of the items in the **Linker Section Name** list to Altera On-chip Memory (OCRAM) or external RAM.

Figure 70. Linker Region Settings for Boot Option 4a and 4b
1.5.3.1.5 HEX File Generation

1. In the Nios II SBT tool, right click on your project in the Project Explorer window.
2. Click Make Targets -> Build..., the Make Targets dialog box appears. You can also press shift + F9 to trigger the Make Target dialog box.
3. Select mem_init_generate.
4. Click Build to generate the HEX file.
5. The "mem_init_generate" macro will create two HEX files; <OCRAM_name>.hex and <QSPIFlash_name>.hex. Use <QSPIFlash_name>.hex for boot option 4 and 5 as you are booting from QSPI flash, not from OCRAM.

**Related Links**

Software Programmer Object File (.pof) Generation on page 59

**1.5.3.1.6 Software Programmer Object File (.pof) Generation**

**Important:** The quartus.ini file with PGMIO_SWAP_HEX_BYTE_DATA=ON content is required to byteswap the programming file during the POF generation. Please create the quartus.ini file or use the quartus.ini available in the related information and place it under Quartus II tool directory or project directory before you proceed.

1. In Quartus II, click on Convert Programming Files (.pof) from the File tab.
2. Choose Programmer Object File as Programming file type.
4. Set Configuration device to CFI_512Mb.
5. Change the File name to the desired path and name.
6. Remove the SOF Page 0.
7. Click on **Add HEX Data**, choose the HEX file generated in **HEX Generation** section. Select **Absolute Addressing** and click **OK**.
8. Click **Generate** to create the .pof file.

**Figure 73. HEX to POF file Conversion Settings**

**Related Links**
- HEX File Generation on page 58
- POF file Programming into QSPI flash on page 60
- Quartus.ini file

**1.5.3.1.7 POF file Programming into QSPI flash**

1. Programming Parallel Flash Loader into MAX10 Device
   
   *Note:* You need to program the parallel flash loader into the MAX 10 device before programming the QSPI flash.
   
   a. Create Parallel Flash Loader for MAX 10 FPGA in the Quartus II. Assign QSPI pins based on your design. Compile the project to obtain `max10_qpfl.sof` file.
   
   b. Open Quartus II programmer from the Quartus II tool (**Tools** -> **Programmer**).
   
   c. Make sure the **Hardware Setup** is set to your **USB blaster**.
   
   d. Click on **Auto Detect**, select your MAX10 FPGA and select **OK**.
e. Right click on the **MAX 10 FPGA** and select **Edit -> Change File**. Choose the `max_qpfl.sof` file.

f. Check MAX 10 device under **Program/Configure** and click **Start** to start programming.

g. Click on **Auto Detect** after `max10_qpfl.sof` is successfully programmed. Click **Yes** if you are asked to overwrite the existing settings. A new QSPI flash device will be shown on the screen.

**Figure 74. Programming Parallel Flash Loader Programmer Settings**

2. **Programming HEX image into QSPI Flash**
   a. Right click on the **QSPI device** and select **Edit -> Change File**. Choose the generated POF file from Software POF Generation section.
   
   b. Check the HEX file under **Program/Configure** column and click **Start** to start programming.
Figure 75. HEX Image Programming into QSPI Flash

1 Nios II Processor Booting Methods in MAX 10 FPGA Devices

Related Links
Software Programmer Object File (.pof) Generation on page 59

1.5.3.1.8 Programming Hardware Design POF File into the MAX10 FPGA

1. After you successfully programmed HEX data into Quad SPI flash, right click on the MAX 10 FPGA and select Edit -> Change File. Choose the downloaded POF file generated from Quartus II project compilation.

2. Check the MAX10’s CFM0 and UFM under Program/Configure column and click Start to start programming.
1.5.3.2 Dual Compressed Images Bootable System Guideline

1.5.3.2.1 Design Creation

1. Create your Nios II processor project using Quartus II and Qsys.
2. Ensure the Generic Quad SPI Controller IP is added into your Qsys system. Refer to the diagram below for IP connection in Qsys.

3. Ensure Dual Configuration IP is instantiated in your Qsys system to enable dual images configuration.

Note: The maximum input clock for Generic Quad SPI Controller IP is 25 MHz. The input clock must not exceed this maximum value.
1.5.3.2.2 Qsys Settings

1. In the Nios II Processor parameter editor, set the reset vector memory and exception vector memory based on the boot options below:

<table>
<thead>
<tr>
<th>Boot Option</th>
<th>Reset vector memory</th>
<th>Exception vector memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Option 4a</td>
<td>QSPI flash</td>
<td>OCRAM/ External RAM</td>
</tr>
<tr>
<td>Option 4b</td>
<td>QSPI flash</td>
<td>QSPI flash</td>
</tr>
<tr>
<td>Option 5</td>
<td>QSPI flash</td>
<td>OCRAM/ External RAM</td>
</tr>
</tbody>
</table>

Figure 78. Nios II Parameter Editor Settings Boot Option 4a and 5

Note: mem_if_ddr3_emif_0.avl is the external memory (DDR3) used in this example.

16) You can set the exception vector for Boot Option 4 to OCRAM/ External RAM (Option 4a) or QSPI Flash (Option 4b) according to your design preference.

17) Boot option 4a which sets exception vector memory to OCRAM/External RAM is recommended to make the interrupt processing faster.
There are two Nios II application data (HEX file) stored into the QSPI for supporting dual configuration images. Reset vector memory offset has to set correctly for the configuration images to call up the correct HEX data.

Set Reset vector memory offset of the Nios II Processor in first Qsys design to address 0x00000000.

Set Reset vector memory offset of the Nios II Processor in second Qsys design to another address to avoid overlapping. For example: Address 0x02000000 which is half of the QSPI memory size (512Mb).
5. Open **Altera Generic Quad SPI** controller parameter editor. Change the **Configuration device type** to the QSPI flash selection and make sure the **I/O mode** is set to **QUAD**.

**Figure 82. Altera Generic Quad SPI Controller Parameter Settings**

6. Click **Generate HDL**, the Generation dialog box appears.

7. Specify output file generation options and then click **Generate**.

**1.5.3.2.3 Quartus II Software Settings**

1. In the Quartus II software, click on **Assignment -> Device -> Device and Pin Options -> Configuration**. Set **Configuration mode** to **Dual Compressed Images**.
2. Click **OK** to exit the **Device and Pin Options window**.

3. Click **OK** to exit the **Device** window.

4. Click **Start Compilation** to compile your project.

**Related Links**

Programming Hardware Design POF File into the MAX10 FPGA on page 76

### 1.5.3.2.4 BSP Editor Settings

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

1. In the Nios II SBT tool, right click on your BSP project in the **Project Explorer** window. Select **Nios II > BSP Editor...** to open the **Nios II BSP Editor**.

2. In Nios II BSP Editor, click on **Advanced** tab under **Settings**.

3. Click on **hal** to expand the list.

4. Click on **linker** to expand the list.
5. Based on the boot option used, do one of the following:

- For boot option 4a, if exception vector memory is set to OCRAM or External RAM, enable the following:
  
  - allow_code_at_reset
  - enable_alt_load
  - enable_alt_load_copy_rodata
  - enable_alt_load_copy_rwdata
  - enable_alt_load_copy_exceptions

- For boot option 4b, if exception vector memory is set to QSPI flash, enable the following:
  
  - allow_code_at_reset
  - enable_alt_load
  - enable_alt_load_copy_rodata
  - enable_alt_load_copy_rwdata

- For boot option 5, leave all the hal.linker settings unchecked.
Figure 85. Advanced.hal.linker Settings for Boot Option 4a

Figure 86. Advanced.hal.linker Settings for Boot Option 4b
6. Click on **Linker Script** tab in the Nios II BSP Editor.

7. Based on the boot option used, do one of the following:

   — For boot option 4a and 4b, set the `.text` item in the **Linker Section Name** to the QSPI flash in the **Linker Region Name**. Set the rest of the items in the **Linker Section Name** list to the Altera On-chip Memory (OCRAM) or external RAM.

   — For boot option 5, set all of the items in the **Linker Section Name** list to Altera On-chip Memory (OCRAM) or external RAM.

---

**Figure 87. Advanced.hal.linker Settings for Boot Option 5**

**Figure 88. Linker Region Settings for Boot Option 4a and 4b**
1.5.3.2.5 HEX File Generation

*Note:* The following steps apply to both first and second Nios II applications.

1. In the **Nios II SBT tool**, right click on your project in the **Project Explorer** window.
2. Click **Make Targets -> Build...**, the **Make Targets** dialog box appears. You can also press shift + F9 to trigger the Make Target dialog box.
3. Select **mem_init_generate**.
4. Click **Build** to generate the HEX file.
5. The “mem_init_generate” macro will create two HEX files; <OCRAM_name>.hex and <QSPIFlash_name>.hex. Use <QSPIFlash_name>.hex for boot option 4 and 5 as you are booting from QSPI flash, not from OCRAM.

1.5.3.2.6 Hardware Programmer Object File (.pof) Generation

1. In Quartus II, click on Convert Programming Files (.pof) from the File tab.
2. Choose Programmer Object File as Programming file type.
3. Set Mode to Internal Configuration.
4. Change the File name to the desired path and name.
5. Under Input files to convert, select Page_0 and click on Add File button on the right.
6. Browse to the first .sof file and click OK.
7. Click on Add Sof Page to create additional page for .sof file. This creates SOF data Page_1 automatically.
8. Click Add File... and point to the second .sof file to add into Page_1.
9. Click Generate to create the .pof file.
1.5.3.2.7 Software Programmer Object File (.pof) Generation

**Important:** The `quartus.ini` file with `PGMIO_SWAP_HEX_BYTE_DATA=ON` content is required to byteswap the programming file during the POF generation. Please create the `quartus.ini` file or use the `quartus.ini` available in the related information and place it under Quartus II tool directory or project directory before you proceed.

1. In Quartus II, click on **Convert Programming Files (.pof)** from the **File** tab.
2. Choose **Programmer Object File** as **Programming file type**.
3. Set **Mode** to **1-bit Passive Serial**.
4. Set **Configuration device** to **CFI_512Mb**.
5. Change the **File name** to the desired path and name.
6. Remove the SOF **Page_0**.
7. Click on **Add HEX Data**, choose the HEX file generated in **HEX Generation** section. Select **Absolute Addressing** and click **OK**.
8. Repeat Step 7 to add second HEX file.
9. Click **Generate** to create the .pof file.
Figure 92. **HEX to POF file Conversion Settings**

---

**Related Links**

Quartus.ini file

1.5.3.2.8 POF file Programming into QSPI flash

1. Programming Parallel Flash Loader into MAX10 Device

   *Note:* You need to program the parallel flash loader into the MAX 10 device before programming the QSPI flash.

   a. Create Parallel Flash Loader for MAX 10 FPGA in the Quartus II. Assign QSPI pins based on your design. Compile the project to obtain `max10_qpfl.sof` file.

   b. Open Quartus II programmer from the Quartus II tool (`Tools` -> `Programmer`).

   c. Make sure the **Hardware Setup** is set to **USB blaster**.

   d. Click on **Auto Detect**, select your MAX10 FPGA and select **OK**.

   e. Right click on the **MAX 10 FPGA** and select **Edit** -> **Change File**. Choose the `max_qpfl.sof` file.

   f. Check MAX 10 device under **Program/Configure** and click **Start** to start programming.

   g. Click on **Auto Detect** after `max10_qpfl.sof` is successfully programmed. Click **Yes** if you are asked to overwrite the existing settings. A new QSPI flash device will be shown on the screen.
2. Programming HEX image into QSPI Flash
   a. Right click on the **QSPI device** and select **Edit -> Change File**. Choose the generated POF file from **Software POF Generation** section.
   b. Check the HEX file under **Program/Configure** column and click **Start** to start programming.

**Figure 94. HEX Image Programming into QSPI Flash**
1.5.3.2.9 Programming Hardware Design POF File into the MAX10 FPGA

1. After you successfully programmed HEX data into Quad SPI flash, right click on the MAX 10 FPGA and select Edit -> Change File. Choose the hardware POF file generated from Hardware POF Generation section.

2. Check the MAX10’s CFM0 and UFM under Program/Configure column and click Start to start programming.

Figure 95. POF Image Programming into QSPI Flash

Related Links

- Quartus II Software Settings on page 66
- Hardware Programmer Object File (.pof) Generation on page 72

1.5.4 Summary of Nios II Processor Vector Configurations and BSP Settings

The following table shows a summary of Nios II processor reset and exception vector configurations, and BSP settings.
Table 11. Summary of Nios II Processor Vector Configurations and BSP Settings

<table>
<thead>
<tr>
<th>Boot Option</th>
<th>Reset Vector Configuration</th>
<th>Exception Vector Configuration</th>
<th>BSP Editor Setting: Settings.Advanced.hal.linker</th>
<th>BSP Editor Setting: Linker Script</th>
</tr>
</thead>
</table>
| **Option 1:** Nios II processor application execute-in-place from Altera On-chip Flash (UFM) | Altera On-chip Flash | 1. OCRAM/External RAM, OR 2. Altera On-chip Flash | If the exception vector memory is set to OCRAM/External RAM, enable the following settings in Settings.Advanced.hal.linker:  
  - allow_code_at_reset  
  - enable_alt_load  
  - enable_alt_load_copy_rodata  
  - enable_alt_load_copy_rwlockdata  
  - enable_alt_load_copy_exceptions | • Set .text Linker Section to Altera On-chip Flash  
• Set other Linker Sections (.heap,.rwdata,.rodata,.bss,.stack) to OCRAM/External RAM |
| **Option 2:** Nios II processor application copied from UFM to RAM using boot copier | Altera On-chip Flash | OCRAM/External RAM | Make sure all settings in Settings.Advanced.hal.linker are left unchecked | Make sure all Linker Sections are set to OCRAM/External RAM |
| **Option 3:** Nios II processor application execute-in-place from Altera On-chip Memory (OCRAM) | OCRAM | OCRAM | Enable allow_code_at_reset in Settings.Advanced.hal.linker and other settings are left unchecked. | Make sure all Linker Sections are set to OCRAM |
| **Option 4:** Nios II processor application executes in-place from QSPI flash | QSPI flash | 1. OCRAM/External RAM, or 2. QSPI Flash | If the exception vector memory is set to QSPI Flash, enable the following settings in Settings.Advanced.hal.linker:  
  - allow_code_at_reset  
  - enable_alt_load  
  - enable_alt_load_copy_rodata  
  - enable_alt_load_copy_rwlockdata  
  - enable_alt_load_copy_exceptions | • Set .text Linker Section to QSPI Flash  
• Set other Linker Sections(.heap,.rwdata,.rodata,.bss,.stack) to OCRAM/External RAM |
| **Option 5:** Nios II processor application copied from QSPI flash to RAM using boot copier | QSPI flash | OCRAM/External RAM | Make sure all settings in Settings.Advanced.hal.linker are left unchecked | Make sure all Linker Sections are set to OCRAM/External RAM |
1.6 Appendix A: Booting Elements

All Nios II processor boot options introduced in this application note use the following booting elements to create the necessary boot files:

- The memcpy-based boot copier
- The alt_load function
- The Nios II SBT "make mem_init_generate" target
- The Convert Programming Files feature

1.6.1 Nios II Processor Memcpy-based Boot Copier

The Nios II processor memcpy-based boot copier has the following features:

- Supports EPCQ, CFI, QSPI and Altera On-chip Flash (UFM) flash memories
- Locates software application in the memory
- Unpacks and copies software application image to RAM
- Automatically switches to application code in RAM after copy completes

The memcpy-based boot copier is used to support boot option 2 and 5. The memcpy-based boot copier is automatically appended into the HEX file during memory initialization file generation ("mem_init_generate" target). When you download your .pof file into the FPGA, the boot copier will be placed at the beginning of the UFM or QSPI sector and the software application will be placed at the end of the boot copier.

The function of the memcpy-based boot copier is to copy the software application image to the RAM which is the entry point indicated by the software application (ELF file). Depending on the system design, RAM can be either OCRAM or external RAM. Once the copying is done, the boot copier will pass the system control to the application in RAM.

Figure 96. Memory Map of An Example System Using Default Boot Copier
1.6.2 The alt_load() function

The alt_load() function is a mini-bootcopier included in the HAL code and provides the following capabilities:

- Optionally copies sections from the boot memory to RAM based on BSP settings.
- Able to copy data sections (.rodata, .rwdata, .exceptions) to RAM but not the code sections (.text).

The alt_load() function can be enabled in the BSP Settings as shown in the following table:

<table>
<thead>
<tr>
<th>BSP Settings</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>hal.linker.enable_alt_load</td>
<td>enable alt_load() function</td>
</tr>
<tr>
<td>hal.linker.enable_alt_load_copy_rodata</td>
<td>alt_load() copies .rodata section to RAM</td>
</tr>
<tr>
<td>hal.linker.enable_alt_load_copy_rwdata</td>
<td>alt_load() copies .rwdata section to RAM</td>
</tr>
<tr>
<td>hal.linker.enable_alt_load_copy_exceptions</td>
<td>alt_load() copies .exceptions section to RAM</td>
</tr>
</tbody>
</table>

1.6.3 Nios II SBT Makefile “mem_init_generate” Target

The Nios II SBT application Makefile “mem_init_generate” target is responsible for generating memory initialization files using various file conversion tools. This includes a HEX file for the UFM data, a HEX file for initialization of the on chip RAM in the SOF and a DAT file for initializing the on chip flash model for simulation.

When required, the Nios II SBT tool automatically adds the Nios II processor memcpy-based boot copier to the system when the executable file (.elf) is converted to memory initialization file (.hex). This operation take place whenever the .text section is located in a different memory that the reset vector points to, which indicates a code copy is required. The file conversion happens during execution of “make mem_init_generate” target.

The "make mem_init_generate" target generates different HEX file content based on the specified boot options:

- For boot option 1, 3 and 4 the generated HEX file contains ELF loadable section.
- For boot option 2 and 5, the generated HEX file contains the boot copier and the ELF payload.

The mem_init_generate target also generates a Quartus II IP file (meminit.qip). Quartus II software will refer to the meminit.qip for the location of the initialization files.

1.6.4 The Convert Programming Files Option

You can use the Convert Programming Files option in Quartus II software to convert programming files from one file format to another. This tool is used for combining a .sof and a HEX file into a single .pof file for programming into the Altera On-chip Flash.
1.7 Appendix B: Boot Time Performance Analysis

Intel performed a boot time performance analysis for MAX 10 FPGA based on several use cases. This section provides guidance on how to reduce Nios II boot time in MAX10 design and boot time estimation based on performance analysis. You can use the information in this section as a guideline when designing your custom design.

**Figure 97. Boot Time Performance Analysis Design Block Diagram**

Diagram shows the design was being used to run the MAX 10 FPGA boot time performance analysis. The Nios II processor was configured with special settings for some of the use cases.

Regardless of the boot device, there are only 2 types of boot methods:
1. Nios II processor application execute-in-place.
2. Nios II processor application copied from boot device to RAM using boot copier.

For boot performance analysis, the Nios II processor application sizes varies between 10kB to 64kB.

**Boot Time Performance Analysis Design Example**

The design example was tested on the MAX 10 10M50 Development Kit (Rev B) on using the Quartus II software v15.0 build 145.
The design example boots from the Altera On-chip Flash (UFM). If you want to boot from other boot memory device, you have to change the reset vector and BSP settings according to Table 11 on page 77.

Related Links
Boot Time Performance Analysis Design Example

1.7.1 Boot Time Measurement Strategy

The boot time measurement was done using a boot time performance counter. The boot time counter starts when Nios II comes out of reset and the system will start the boot sequence according to the selected boot options:

1. Nios II processor application execute-in-place:
   - The Alt_load() function copies data sections (.rodata, .rwdata, .exceptions) from boot memory to RAM.
   - The code section (.text) remains in the boot memory.
   - Automatically jump to the user generated application code in the boot memory after copy completed.
   - Run system/driver initialization (alt_main).

2. Nios II processor application copied from boot device to RAM using boot copier:
   - Bootcopyer locates the software application in the boot memory and copies the software application to RAM.
   - Automatically jump to the user generated application code in the RAM after copy completed.
   - Run system/driver initialization (alt_main).

The boot time counter is controlled through software and it will stop once the driver initialization completes.

Figure 98. Boot Time Measurement

```c
void main ( void )
{
  IOWR (BOOT_TIME_COUNTER_BASE, 0, 0);
}
```
1.7.2 Reducing Nios II Boot Time in MAX 10 FPGA Design

There are a few elements that you can consider to improve the Nios II boot time in MAX10 designs. Based on the Nios II Boot Time analysis that Intel has performed, the following subsections can be referred to for general guidance.

1.7.2.1 Boot Methods

1. Nios II application boot time for execute-in-place is faster than using a boot copy/run from RAM based boot (due to the time taken to copy the code to RAM being much longer than the time gained by the higher performance when running from RAM instead of ROM).

2. The boot copier method is recommended for systems that require higher performance. Although this configuration takes longer to boot, it delivers higher application performance compared to an execute-in-place configuration.

1.7.2.2 Boot Device Performance

1. For Nios II application execute-in-place, different boot devices will have different boot times based on their individual memory performance.

2. The following shows the performance for the supported boot device in MAX 10 FPGAs, when Nios II applications execute-in-place:

   Figure 99. Supported Boot Device Performance

<table>
<thead>
<tr>
<th>Fastest</th>
<th>Slowest</th>
</tr>
</thead>
<tbody>
<tr>
<td>On-chip Memory</td>
<td>On-chip Flash</td>
</tr>
<tr>
<td>OCRAM</td>
<td>UFM</td>
</tr>
<tr>
<td>QSPI Flash</td>
<td></td>
</tr>
</tbody>
</table>

1.7.2.3 Peripheral Initialization

Nios II systems initialize all HAL peripherals before main() by default, therefore the boot time has a dependency on the peripherals selected. Peripherals that are slow to initialize or have external dependencies, will increase the boot time and potentially make it less deterministic. If this occurs, you need to calibrate the external memory such as DDR3 for it to work properly.

DDR3 is an example of a peripheral where the initialization time is significant in comparison to the boot time. The calibration time is long particularly when compared to boot times for execute-in-place boot configurations. The calibration time will significantly impact Nios II application that execute-in-place.

To avoid this, in execute-in-place boot configurations, remove the external memory from the Nios II linker region if it is not in use. If size is not an issue, you can choose to use OCRAM. If you are confident working with Nios II software, another option is to remove the DDR3 initialization routine from the boot code and initialize the memory later—once the application code has started running.
1.7.2.4 Nios II Processor Caches

1. Enable Nios II processor caches. Caches improve the boot time because the data and instruction caches reduce memory bandwidth limitations during the boot sequence.

2. You can add caches to your Nios II hardware configuration for both execute-in-place or bootcopier boot methods to improve boot time.

Figure 100. Nios II Design Block Diagram with Processor Caches Enabled

1.7.2.5 System Speed

Using Nios II processor with higher clock speed improves the boot time. For example, you can increase the Nios II processor speed from 75 Mhz to 125 MHz.
1.7.2.6 MAX 10 FPGA Nios II Flash Accelerator

Enable the Nios II flash accelerator when system resets from the MAX 10 FPGA on-chip flash to improve the boot time.
1.7.3 Boot Time Performance and Estimation

1.7.3.1 Boot Time Performance for Design that Boots from On Chip Flash with Flash Accelerator

The following bar graphs show the boot time performance for designs that boot from on-chip flash (application code stored in on-chip flash) with and without the flash accelerator (FA) unit. Three scenarios have been evaluated:

1. Design with EMIF that runs from external memory DDR3 (using boot copier).
2. Design without EMIF that runs from on-chip memory (using boot copier)
3. Design without EMIF that runs from on-chip flash (execute in-place)

Note: None of the designs include an instruction or data cache.
Figure 103. Summary of Boot Times for Designs that Boot from On-Chip Flash with or without Flash Accelerator

- **External Memory**
  - Without FA (9kB App): 2.68 M
  - With FA (9kB App): 2.6 M
  - Without FA (32kB App): 4.66 M
  - With FA (32kB App): 2.70 M

- **On-Chip Memory**
  - Without FA (9kB App): 0.75 M
  - With FA (9kB App): 0.17 M
  - Without FA (32kB App): 25.8 k
  - With FA (32kB App): 5.5 k

- **On-Chip Flash**
  - Without FA (9kB App): 0.63 M
  - With FA (9kB App): 25.8 k

The boot time was reduced by ~20% and ~44% for 9 kB and 32 kB application sizes respectively when the Nios II FA is enabled. Larger application sizes will result in longer boot times because the code has to be copied from the on-chip flash to the external memory during the boot process. Overall, the boot time for the scenario where the boot code is running from the external memory is the slowest. This is due to the long time taken by external memory calibration during system boot up and the time taken to copy the application code to external memory.
The boot time was reduced by ~77% and ~76% for 9 kB and 32 kB application sizes respectively when the Nios II FA is enabled. Running the boot code from the on-chip memory is faster than running from external memory because the on-chip memory does not need to go through memory calibration during system boot up. Additionally, access time to the on-chip memory are faster than those for external DDR3 memory.

Figure 105. Boot Time for Design without EMIF that runs from On-Chip Memory

Figure 106. Boot Time for Design without EMIF that runs from On-Chip Flash (Execute In-Place)
The boot time was reduced by ~78% and ~77% for 9 kB and 32 kB application sizes respectively when the Nios II FA is enabled. Running the boot code from the on-chip flash (execute in-place) is faster than running from external memory and on-chip memory. This is because the code executes in-place directly from the on-chip flash and does not need to be copied into the external memory or on-chip memory for execution which saves a lot of time.

1.7.3.2 Boot Time Estimation

The tables below show estimates of boot time for different MAX 10 FPGA boot configurations. The boot time shown will help you to gauge the boot configuration required for your system design.

Notes:
- Instruction or data cache and flash accelerator are not enabled in the design.
- Boot time values are based on design with 75 MHz Nios II processor speed.

Table 13. Boot Time Estimation on Execute-in-place test cases

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Boot From</th>
<th>Boot Time Counter (Approximate Nios II Clock cycles)</th>
<th>Boot Time (millisecond)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design without External Memory</td>
<td>On Chip Flash</td>
<td>41,000</td>
<td>0.55</td>
</tr>
<tr>
<td></td>
<td>On Chip Memory</td>
<td>18,000</td>
<td>0.24</td>
</tr>
<tr>
<td></td>
<td>QSPI Flash</td>
<td>410,000</td>
<td>5.5</td>
</tr>
<tr>
<td>Design with External Memory</td>
<td>On Chip Flash</td>
<td>2,000,000</td>
<td>27</td>
</tr>
<tr>
<td></td>
<td>On Chip Memory</td>
<td>2,000,000</td>
<td>27</td>
</tr>
<tr>
<td></td>
<td>QSPI Flash</td>
<td>2,300,000</td>
<td>31</td>
</tr>
</tbody>
</table>

Table 14. Boot Time Estimation on Boot Copier test cases

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Boot from</th>
<th>Run From</th>
<th>Boot Time Counter (Clock cycle)</th>
<th>Boot Time (millisecond)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design without External Memory</td>
<td>On Chip Flash</td>
<td>On Chip Memory</td>
<td>2,800,000</td>
<td>37</td>
</tr>
<tr>
<td></td>
<td>Quad SPI Flash</td>
<td>On Chip Memory</td>
<td>28,000,000</td>
<td>370</td>
</tr>
<tr>
<td>Design with External Memory</td>
<td>On Chip Flash</td>
<td>External Memory</td>
<td>4,700,000</td>
<td>63</td>
</tr>
<tr>
<td></td>
<td>On Chip Memory</td>
<td>2,800,000</td>
<td>37</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Quad SPI Flash</td>
<td>External Memory</td>
<td>30,000,000</td>
<td>400</td>
</tr>
<tr>
<td></td>
<td></td>
<td>On Chip Memory</td>
<td>30,000,000</td>
<td>400</td>
</tr>
</tbody>
</table>

18 Boot time counter is applicable to all software application (.elf) sizes.

19 Boot time counter is for every 32kB .elf size.
# 1.8 Document Revision History

Table 15. Document Revision History

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>February 2017</td>
<td>2017.02.21</td>
<td>Rebranded as Intel.</td>
</tr>
<tr>
<td>May 2016</td>
<td>2016.05.24</td>
<td>• Updated Configuration and Booting Flow figures for all boot options.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Updated BSP Editor Settings for Boot option 3.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Updated Summary of Nios II Processor Vector Configurations and BSP Settings table.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Updated Programming Hardware Design POF File into the MAX10 FPGA step for Boot option 4 and 5 dual compressed images.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Added note in Programmer Object File (.pof) Generation for Boot option 3.</td>
</tr>
<tr>
<td>November 2015</td>
<td>2015.11.19</td>
<td>• Added Boot Time Performance for Design that Boots from On Chip Flash with Flash Accelerator subsection.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Restructure boot options and flow by combining boot options and guidelines.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Added Boot option 4 and boot option 5 supporting QSPI.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Added Boot Time Performance Analysis design example information and link.</td>
</tr>
<tr>
<td>June 2015</td>
<td>2015.06.15</td>
<td>• Added MAX 10 FPGA Nios II Design Boot Time Estimation and Guidance section containing boot time performance analysis.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Added The alt_load() function table.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Added ROM Size Requirement to RAM and ROM Size Requirement For Each Boot Option table.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Added block diagrams for all boot options in Nios II Processor Booting Options Using On-chip Flash.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Added OCRAM size in UFM and CFM Array Size table.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Added Boot Option: 3 Nios II processor application execute in-place from Altera On-chip Memory.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Updated ‘Configuration and Booting Flow’ to include Boot Option: 3.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Updated Steps to Build a Bootable System to Guidelines to Build a Bootable System.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Updated Single Uncompressed/Compressed Image Bootable System Guideline to support Single Compressed Image Mode.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Added the third guideline; ‘Single Uncompressed/Compressed Image with Memory Initialization Bootable System Guideline’.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Editorial changes.</td>
</tr>
<tr>
<td>January 2015</td>
<td>2015.01.23</td>
<td>Initial release.</td>
</tr>
</tbody>
</table>