snickerdoodle

HARDWARE CONFIGURATION GUIDE

SEPTEMBER 9, 2016

How to Read this Document

This document makes extensive use of links, references and notices in the page margins to detail additional information that can be useful while following the guide.

WARNING A warning notice indicates a potential hazard. If care is not taken to adhere to the safety precautions, damage may be done to snickerdoodle.

Warnings and cautions will be clearly visible in either the body of the text or in the margin and must be paid close attention while following the guided steps.

CAUTION A caution indication denotes a process that requires special attention. If the caution is not exercised and the process not adhered to, failure may result and/or potential damage to snickerdoodle.

Warning, caution and informational notices, such as this one, may also be found in the margin.

Keywords

Keywords and important terms are shown in *italicized* type. Additional important information can be found in the margins of text with superscript notation[1].

Navigation of menus and directories are shown using ***bold italicized*** type. Any hierarchical navigation is shown using an arrow to denote a ***Parent*** → ***child*** relationship.

`Teletype` text is used to highlight inputs, variables and system files within the host environment.

[1] Margin notes, such as this one, reference the body content and highlight technical details or references for further information.

Introduction

The versatility of snickerdoodle is due to it's software definable and reconfigurable hardware interface. The on-board Zynq-7000's programmable logic allows for the hardware interface to be defined by a bitstream that can be generated in a way that is exceedingly similar to software and application development. The development environment, Vivado®, has been architected in such a way that the transition between developing for the programmable logic (PL) and processing subsystems (PS) is seamless.

This guide will get you started developing hardware definitions for snickerdoodle. In this guide, it is assumed that you have an installation of Vivado (this guide uses the 2015.4 release) on a host computer. The process outlined in this guide was created using Ubuntu 14.04 as the host computer operating system. While the process is similar for Windows users, those using Windows operating systems should consult the Windows version of this guide.

Portions of this guide have been excerpted and adapted from the Xilinx® Wiki found at `http://www.wiki.xilinx.com/`

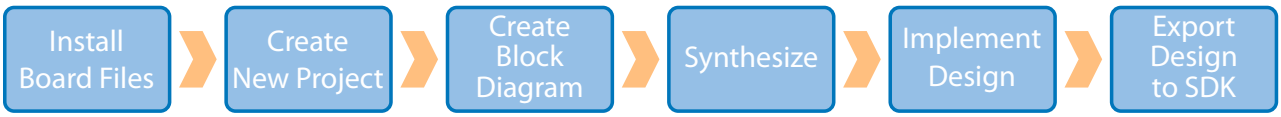


Figure 1: Hardware Configuration Process

Install snickerdoodle Board Files

To begin working with programmable logic on snickerdoodle, a set of board files need to be installed onto the host computer for access by Vivado. The board files are a set of text files that are read by Vivado on startup and provide a set of constraints that define the board hardware.

Download Board Files

The board files can be downloaded directly from GitHub[2] . The files can be downloaded as a `.zip` file from a web browser, downloaded directly using `wget` or cloned using `git clone`.

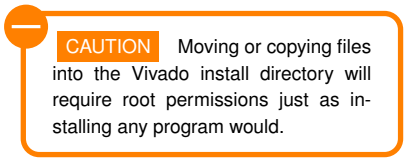
[2] `https://github.com/krtkl/snickerdoodle-board-files`

The archive (`.zip`) can be downloaded directly from the command line by invoking `wget`. The following command will download the board files repository archive:

```
wget https://github.com/krtkl/snickerdoodle-board-files/archive/master.zip
```

Installing Board Files

After downloading the board files archive, the files will need to be installed in the proper location so that they may be read by Vivado. The directory that Vivado uses to load board files when starting is relative to it's install location. Typically, Vivado is installed at `/opt/Xilinx/Vivado/<release>`. The board files are located at `<vivado_install_dir>/data/boards/board_files`. In this example, the board files will be copied to the corresponding directory of a 2015.4 release.



The only files that should be installed are the `snickerdoodle` and `snickerdoodle_black` directories which contain the board files for the corresponding snickerdoodle version. An exclusion file (`EXCLUDE`) is included in the archive to help with the copying process and prevent extraneous files (documentation, README, etc.) from being copied into the board files directory.

The following series of commands will extract the `.zip` archive and move the resulting board files into the `board_files` directory:

```
$ wget https://github.com/krtkl/snickerdoodle-board-files/archive/master.zip
$ unzip master.zip
$ cd snickerdoodle-board-files-master
$ rsync -av --exclude-from=EXCLUDE ./* /opt/Xilinx/Vivado/2015.4/data/boards/board_files/
```

Once the board files have been installed, Vivado will need to be restarted to load the files from the data directory.

Creating a New Project

Creating a new project can be done by selecting **`File → New Project...`** from the menubar or by selecting the *Create New Project* icon from the "Quick Start" menu, as shown in Figure 2 .

The first step to creating a new Vivado project is to select the project name and parent directory. By default, Vivado will create a new subdirectory for which to store the project files. It is highly recommended that this setting be left unchanged to keep the project files and directories organized and easily accessible for development (*i. e.,* from the SDK). Figure 3 shows the input of the project name and parent directory when creating a new project.

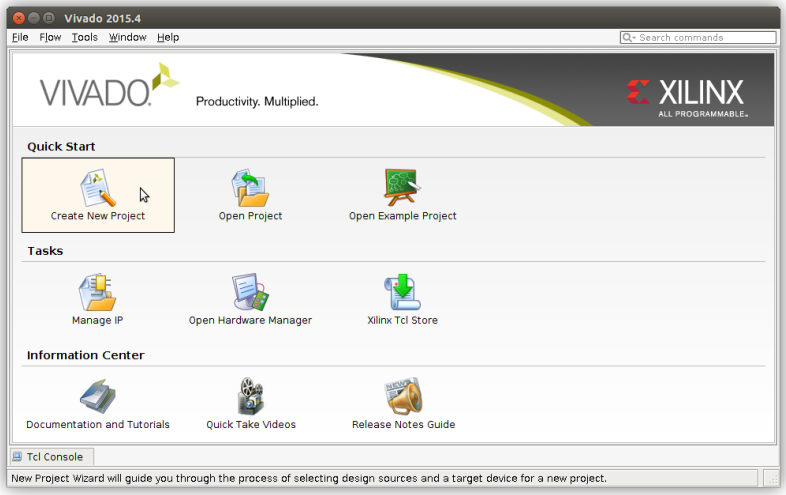


Figure 2: Vivado Startup Create New Project

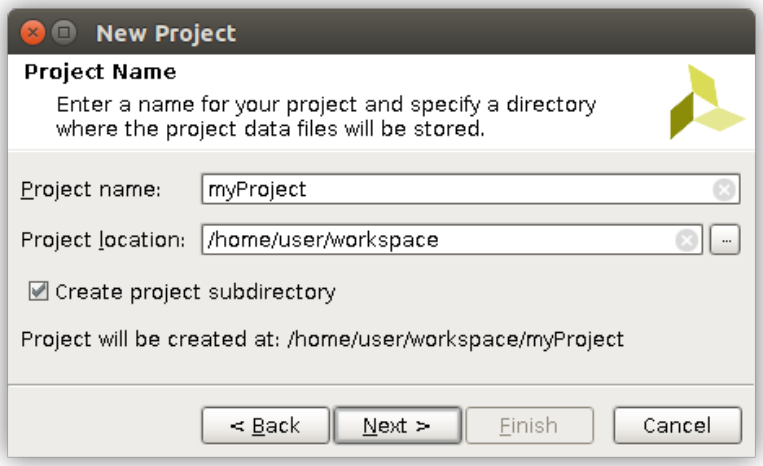


Figure 3: Selecting Project Name and Parent Directory

Project Type and Sources

At this point in the project creation process, IP and design sources can be added to the project. If you would like to include existing sources, you will prompted to include those before choosing a project board. If you choose not to include any existing assets or constraints, the "Do no specify sources..." checkbox, shown in Figure 4 , can be selected to skip this step. Sources can be added to the project at any time after creation as outlined below.

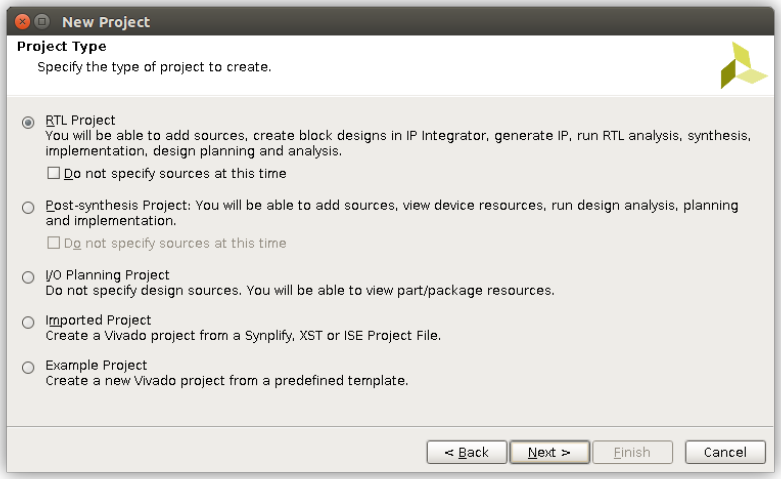


Figure 4: Project Type Selection Dialog

Choosing a Project Board

If the board files have been copied and loaded properly, they will be listed in the table of boards. To view the available boards, choose "Boards" rather than "Parts" at the top of the window as shown in Figure 5 .

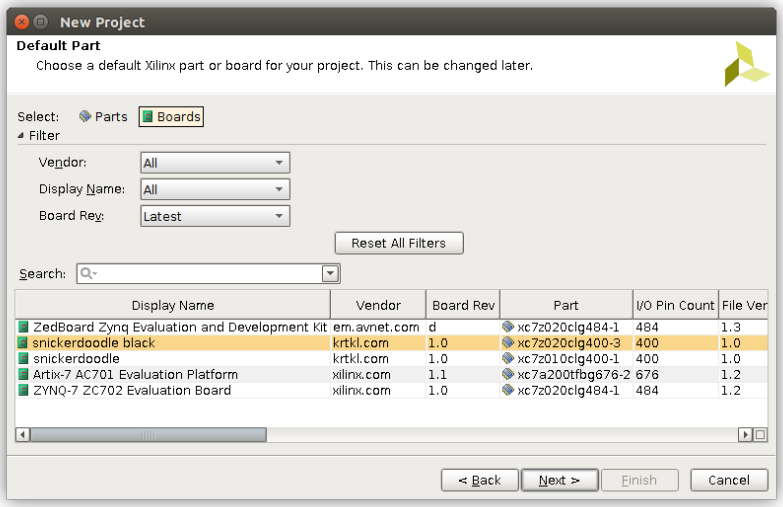


Figure 5: Selecting a Project Board

After selecting a board, the details of the project will be listed in the *New Project Summary* shown in Figure 6 . By clicking the "Finish" button, the project will be created and opened in the Vivado graphical user interface (GUI).

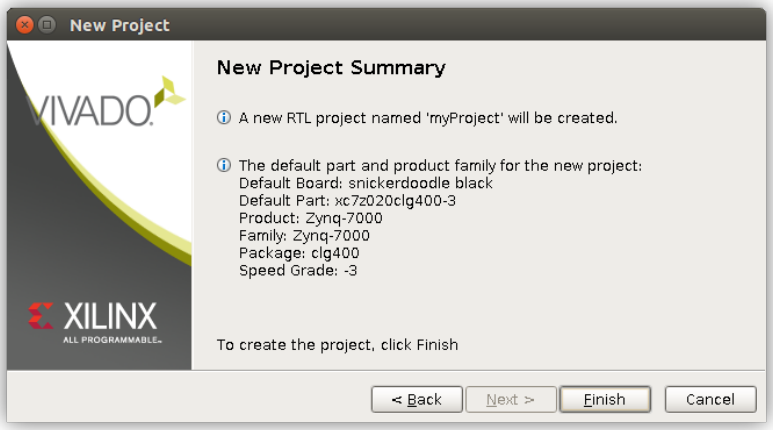


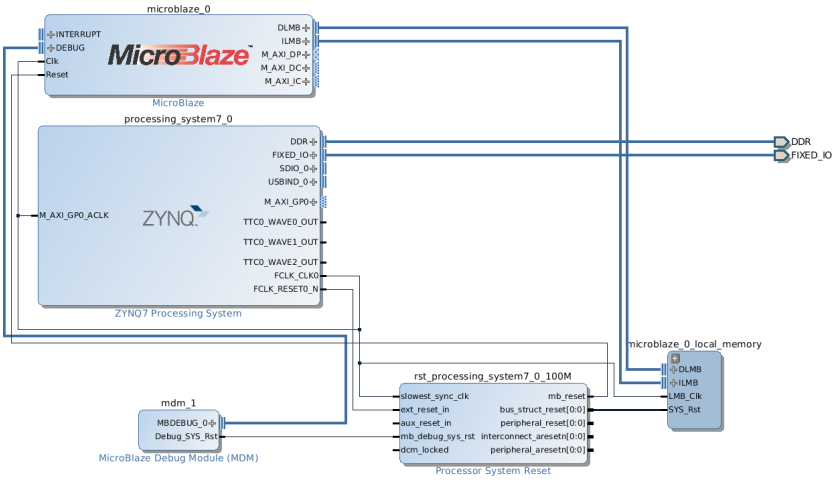
Figure 6: Review and Finish New Project Creation

Working with Vivado

Now that Vivado has been set up with the snickerdoodle board files and a new project has been created, development can begin. The Vivado GUI is similar to many software development environments. Many project elements are contained within panes and editing of those elements can be done through the menubar or from the panes themselves. After creating an empty project, you are free to import existing assets or develop sources from scratch. To get started with development, a block design needs to be created.

Create Block Design

A block design is a visual representation of the hardware configuration. Hardware configurations can be defined by including or creating sources and assigning hardware I/O. Figure 8 shows an example block design with hardware definitions for fixed peripherals such as memory interfaces and a soft processing core (MicroBlaze).



To create a block design for a new project, select ***Flow → Create Block Design*** from the menubar or from the "IP Integrator" section of the *Flow Navigator* pane as shown in Figure 7 .

Adding and integrating sources and IP into a block design can vary greatly depending on the design and application. For additional reading and information on integrating IP can be found at `http://www.xilinx.com/support/documentation/sw_manuals/xilinx2015_4/ug896-vivado-ip.pdf`.

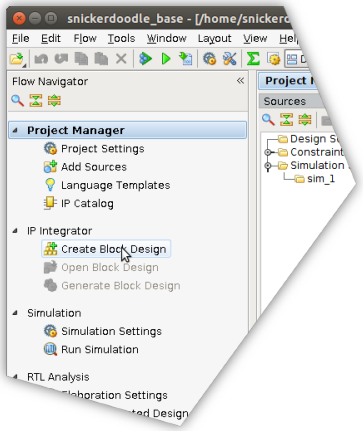


Figure 7: Create Block Design

Figure 8: Example Block Design with Zynq Processing System and Microblaze Soft Core Processor

Add Sources and IP

Adding sources can be done by right clicking inside the *Sources* pane or by selecting ***File*** → ***Add Sources...*** from the menubar. The *Add Sources Wizard* will appear and allow you to select existing sources, IP and constraints to add to the project.

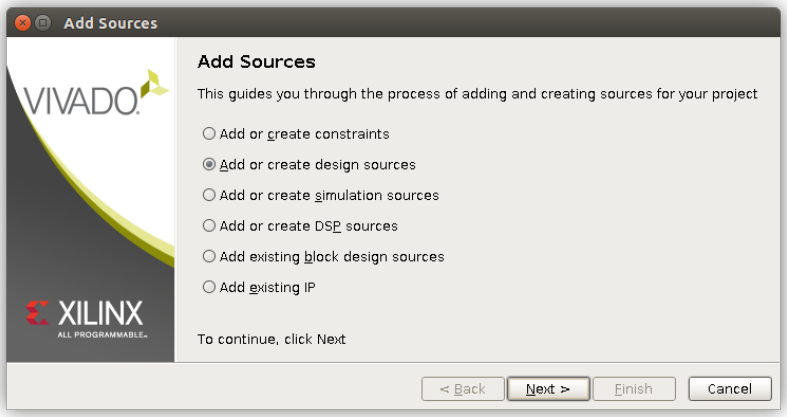


Figure 9: Adding New Sources to the Project with the Add Sources Wizard

Create HDL Wrapper

Before generating a bitstream for the project, an HDL wrapper must be generated for the design. To do this, right click on the design (in this case "base_design") from within the *Sources* pane and select "Create HDL Wrapper...".

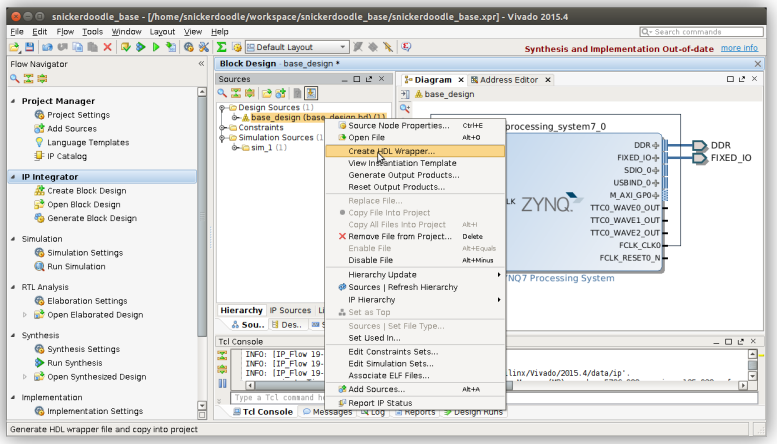


Figure 10: Creating an HDL Wrapper for the Design

Generating Bitstreams

Bitstreams can be generated by selecting ***Flow → Generate Bitstream*** from the menubar or from within the "Program and Debug" section of the *Flow Navigator* pane as shown in Figure 11 . The bitstream contains all the information necessary to define the programmable logic and the associated hardware peripherals/interfaces.

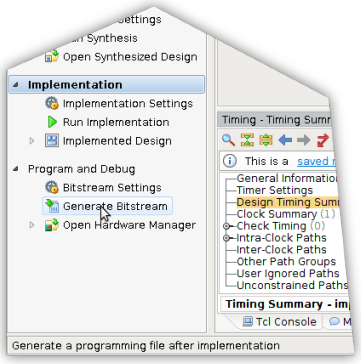


Figure 11: Generate Bitstream from Vivado Flow Navigator

Exporting Design with SDK

Export the Hardware Platform

Designs that are implemented using Vivado can be exported and opened in the SDK directly from Vivado. Before a design can be opened in the SDK, the hardware profile must be exported which can be done by selecting ***File → Export → Export Hardware...*** from the menubar as shown in Figure 12 .

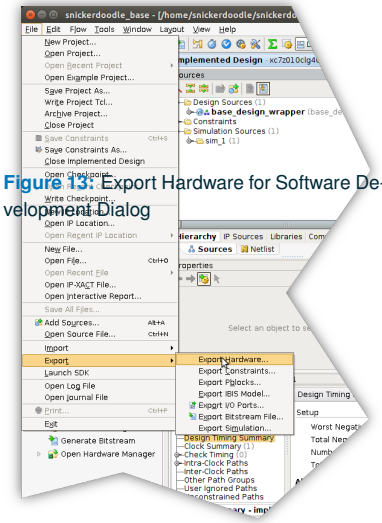


Figure 12: Export Hardware Configuration from Vivado

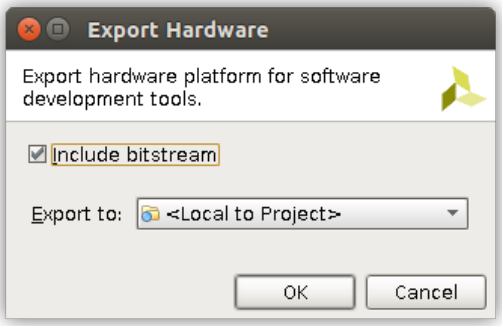


Figure 13: Export Hardware for Software Development Dialog

Figure 13 shows the options for hardware export. Select the "Include bitstream" checkbox to include the bitstream with the hardware definition files for use with the SDK. If you would like to use the hardware platform in a workspace other than the hardware project directory, change the selection of the "Export to:" input. This will be the workspace for the SDK.

> *If you choose an export location other than `<Local to Project>`, the SDK will need to be launched separately from Vivado to access the exported workspace location*

Launching the SDK

The SDK can be launched directly from Vivado to use the exported hardware profile for software development. To launch the SDK from Vivado, select ***File →***

Launch SDK from the menubar, as shown in Figure 14 . If the hardware platform was exported to a directory within the Vivado project (by selecting <Local to Project> as export location), the hardware platform will be immediately available within the SDK workspace. If the hardware was exported to a different directory, you will need to change the workspace directory after the SDK has launched.

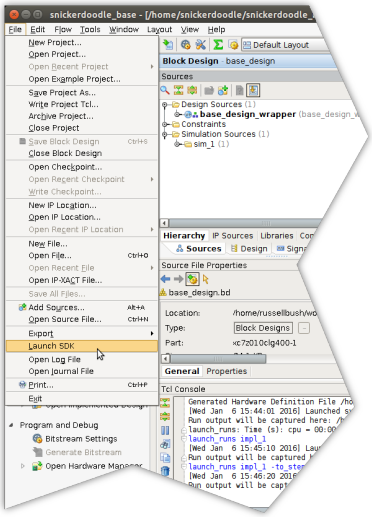


Figure 14: Launch SDK from Vivado

Building Bitstream into `BOOT.bin`

Bitstreams can be built into the boot image and specified to be loaded when the boot image starts the boot process. This method can be useful for bitstreams that need to be loaded *before* the operating system is booted. There are two methods for building the boot image (`BOOT.bin`): a GUI based method executed from the SDK environment and a command line method from which the GUI process is derived. The required components (termed *partitions*), for creating a boot image are as follows:

`fsbl.elf` - First stage boot loader (FSBL). Responsible for loading the bitstream (if one exists), loading into memory and handing off the boot process to the second stage bootloader.

`u-boot.elf` - Second stage boot loader. Responsible for loading Linux system components (devicetree, uImage, file system)

`system.bit` - The bitstream to be built into the boot image.

`devicetree.dtb` - The Linux devicetree to be loaded by FSBL or U-Boot.

`uImage.bin` - Linux kernel image to be loaded by FSBL or U-Boot.

Building Boot Images from SDK

The SDK provides a graphical way to select the components/partitions for the boot image and generate a *boot image file* (`.bif`). The graphical front-end provides the necessary interface for selecting existing boot partitions (typically pre-compiled and/or provided by Vivado project) and will generate the `.bif` file along with the boot image. Figure 15 shows the graphical interface for creating Zynq boot images from the SDK. To access the interface and begin generating images, select ***Xilinx Tools → Create Boot Image*** from the menubar in the SDK.

Prebuilt `.bif` files can be used by the SDK interface by selecting "Import from exisiting BIF file", shown at the top of the window in Figure 15 . Boot partitions are listed in the "Boot image partitions" table within the interface. To add boot partitions from the interface, selecting "Add" will open an "Add partition" dialog (shown in Figure 16) which will allow you to select and specify the partition file.

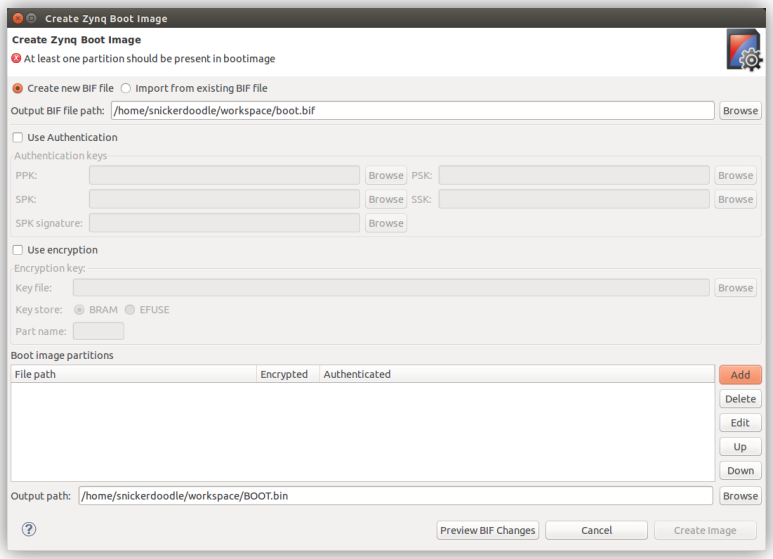


Figure 15: Interface to Create Boot Image from Xilinx SDK

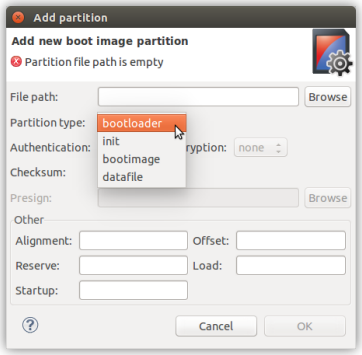


Figure 16: Create New Boot Image Parition

Building Boot Images with bootgen

The GUI interface for building boot images from within the SDK is simply a wrapper for the command line executable that serves the same purpose. `bootgen` can be provided with a `.bif` file to specify the boot image partitions. Below is an example `.bif` file:

> **CAUTION** When building boot images with `bootgen` the boot partition files and `.bif` file should be isolated in a working directory from which `bootgen` is executed.

```
image : {
    [bootloader]fsbl.elf
    u-boot.elf
    bitstream.bit
    [load=0x2ff0000]devicetree.dtb
    [load=0x3000000]uImage.bin
}
```

After copying the boot partition file and the `.bif` file to a local working directory, the `bootgen` utility can be executed from that directory to output the boot image. Invoking the following command will create a boot image named `BOOT.bin` using the boot image file `boot.bif`:

```
bootgen -image boot.bif -o i BOOT.bin
```

Loading Bitstream from Linux

Bitstreams can be loaded to the FPGA from a booted Linux system. This allows bitstreams to be loaded on a system without needing to regenerate a boot image or reboot the system. This can be useful for testing and development.

Preparing Bitstream for Loading (bit-reversing)

The bitstreams must be bit-reversed before they can be loaded from Linux. The `-split` argument can be supplied to the `bootgen` utility to tell it to create a bit-reversed copy of the bitstream that can be loaded from Linux.

```
bootgen -image boot.bif -split bin -o i BOOT.bin
```

Using `-split bin` will append a `.bin` suffix to the input files specified in the boot image file (`boot.bif`). In the case of the `.bif` example above, this will

output a new, bit-reversed bitstream named `bitstream.bit.bin`.

Writing Bitstream to Device

Now that the bitstream has been bit-reversed, it can be written to the FPGA using the `xdevcfg` driver. To load the bitstream, write the contents of the bitstream file to the `xdevcfg` device using the following command:

```
cat bitstream.bit.bin > /dev/xdevcfg
```

> **CAUTION** Attempting to write bitstreams that have not been reformatted to bit-reversed will not successfully load to `xdevcfg` and the `prog_done` flag will fail to be asserted.

Writing the bitstream to the device can take some time. The `prog_done` flag can be read to check whether the bitstream has finished loading:

```
cat /sys/devices/amba.0/f8007000.ps7-dev-cfg/prog_done
1
```

This process allows bitstreams to be loaded, and thus the FPGA to be reconfigured, without needing to alter the *BOOT* partition of the SD card (*i.e.,* `BOOT.bin`) or reboot the system.