Intro to Altera SoC Devices for HW Developers Workshop - Generating the Preloader

From Altera Wiki

Contents

- 1 Overview
- 2 Compiling the Preloader
  - 2.1 Generating the Preloader from the command line
- 3 Compiling the Preloader
- 4 Installing the new Preloader
  - 4.1 On a Linux host
    - 4.1.1 Using the "dd" method
- 5 Using the "alt-boot-disk-util" method
- 6 Running the new Preloader
- 7 Restoring the original Preloader

Overview

Anytime an SoC design is generated with Qsys and compiled in Quartus an "hps_isw_handoff" folder will be generated. This folder contains configuration information which is needed by the Preloader to configure clocks, IO, and the SDRAM controller. For this lab the handoff folder was created in the previous section.

<<<<Click here to return to the previous section

To begin, make sure that you have the environment provided by the Embedded Command Shell which is delivered in the SoC EDS tools installation. There are a number of ways that you might get this environment applied to your development host, one way is to simply execute the "embedded_command_shell.sh" shell script that is provided in the SoC EDS tools, like this:

On Linux:

```
[~]$ <path-to-soceds-tools>/embedded/embedded_command_shell.sh
[~]$
```

On Windows:

```
Start->Altera->SoCEDS->Embedded Shell
```

Then from the embedded command shell, change into the directory on the local hard drive of your development host in which you completed the previous lab section.

```
[~]$ cd <path-to-lab-working-directory>/
[WS1-IntroToSoC]$
```

</big>

Before we start the BSP Editor (Preloader Generator) program, we need to collect one piece of information about the development board that we are using for this lab work. Some of the development boards have SDRAM with ECC storage provided and some of the development boards do not. We need to set the option in the BSP Editor based on if we have ECC storage on our board or not. If you know how your board is equipped then you can proceed, if you are not sure, then execute this command in the embedded command shell to find out:

</big>

```
[WS1-IntroToSoC]$ grep "RW_MGR_MEM_DATA_WIDTH" ./hps_isw_handoff/soc_system_hps_0/sequencer_defines.h
#define RW_MGR_MEM_DATA_WIDTH 32
```

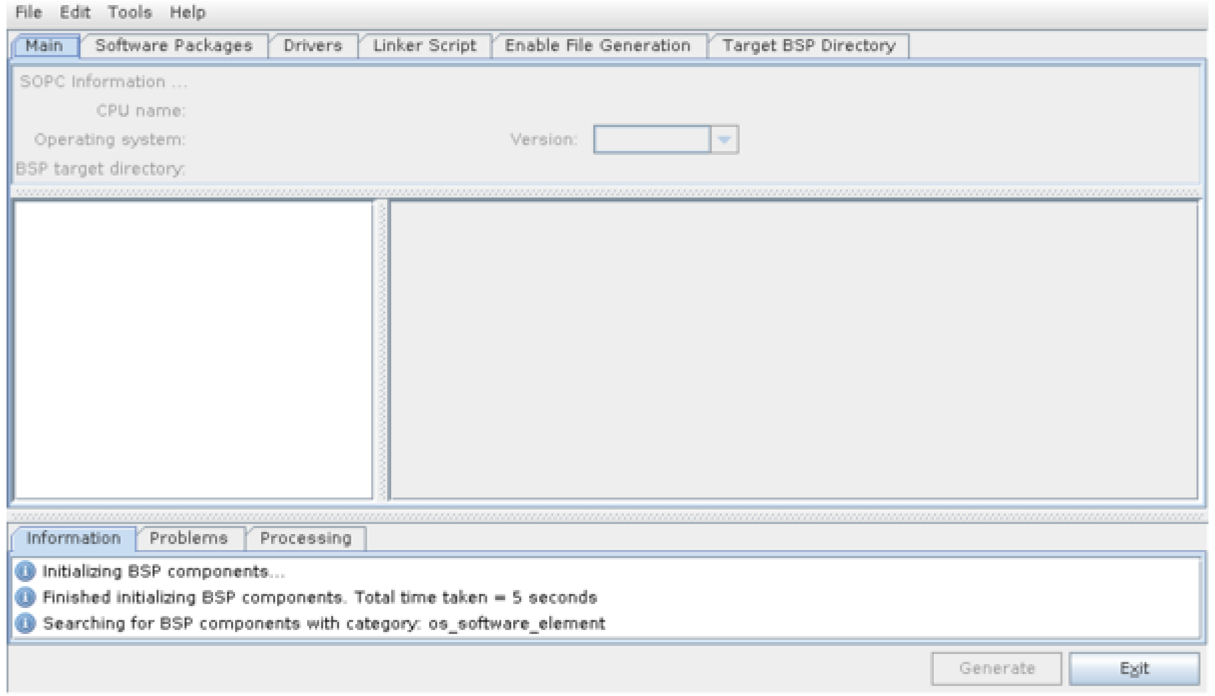
The value of the "RW_MGR_MEM_DATA_WIDTH" macro displayed above is going to equal 8, 16, 24, 32 or 40. If your macro for your board is set to 24 or 40, then your board supports ECC storage, and you will need to enable SDRAM scrubbing in the BSP Editor. If your macro for your board is set to 8, 16, or 32, then your board does not support ECC storage, and you will need to disable SDRAM scrubbing in the BSP Editor.

1. Now start the BSP Editor (Preloader Generator) program from the embedded command shell:

```
[WS1-IntroToSoC]$ bsp-editor &
[WS1-IntroToSoC]$
```

2. You should see the BSP Editor GUI appear like this:

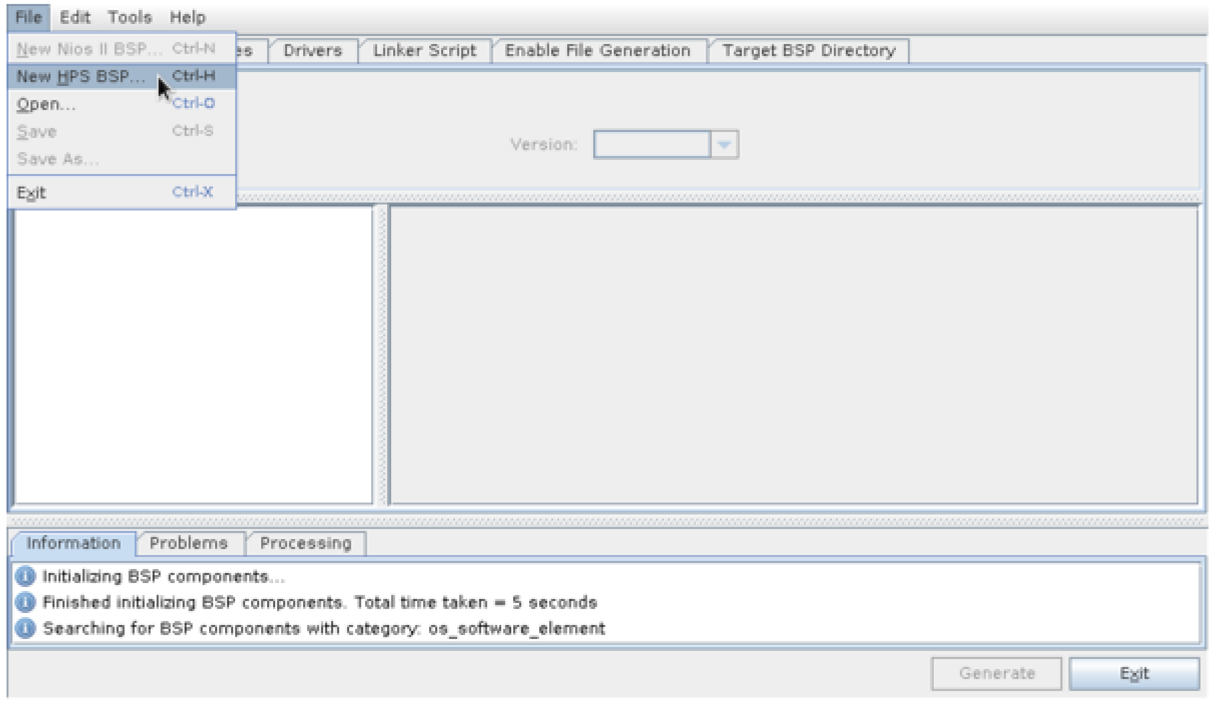
   NOTE: the following images are taken from the !SoC EDS 15.0.1 tools, previous tool releases may look slightly different.



```
Initializing BSP components...
Finished initializing BSP components. Total time taken = 5 seconds
Searching for BSP components with category: os_software_element
```

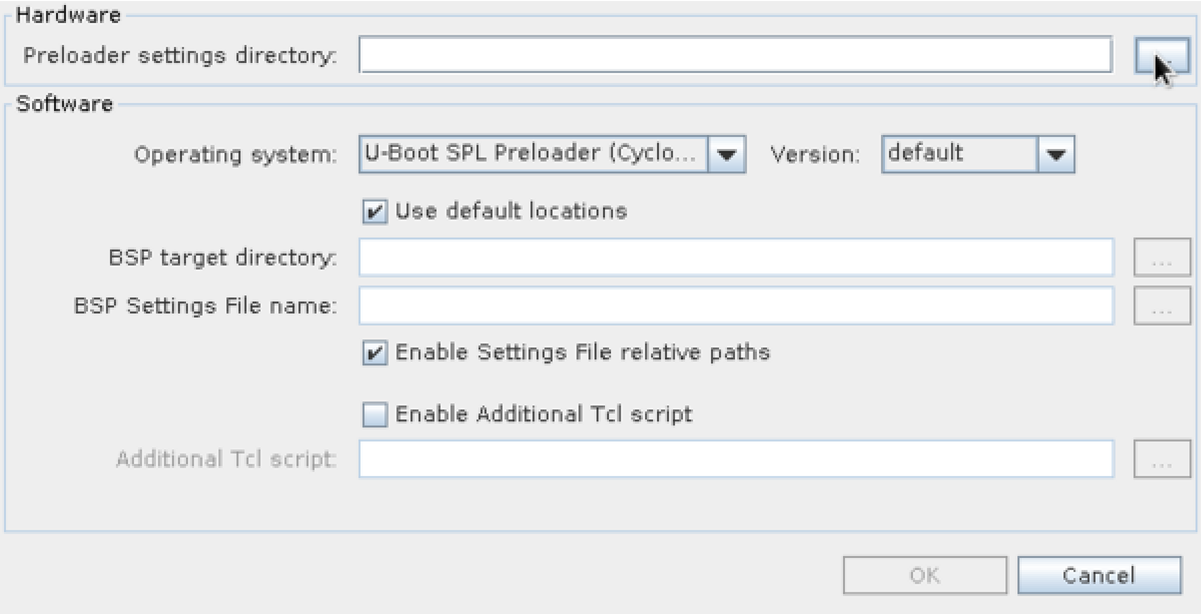
3. In the *BSP Editor* window, select *File->New HPS BSP...* to create a new BSP project for your board's Preloader.

   NOTE: prior to the 15.0 tools release, this menu was just named New BSP



```
Initializing BSP components...
Finished initializing BSP components. Total time taken = 5 seconds
Searching for BSP components with category: os_software_element
```

4. In the *New BSP* window, click the *...* browse button to browse to the handoff files folder:



Preloader settings directory:

Operating system: U-Boot SPL Preloader (Cyclo... Version: default

☑ Use default locations

BSP target directory:

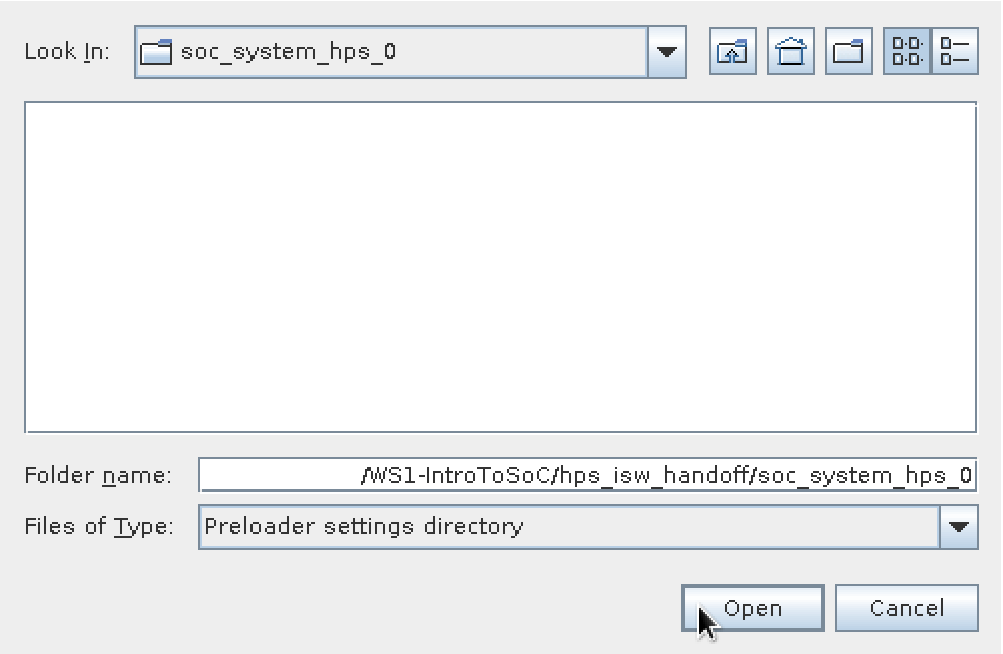
BSP Settings File name:

☑ Enable Settings File relative paths

☐ Enable Additional Tcl script

Additional Tcl script:

5. Select the "<path-to-lab-work>/WS1-IntroToSoC/hps_isw_handoff/soc_system_hps_0" folder and Click *Open*:

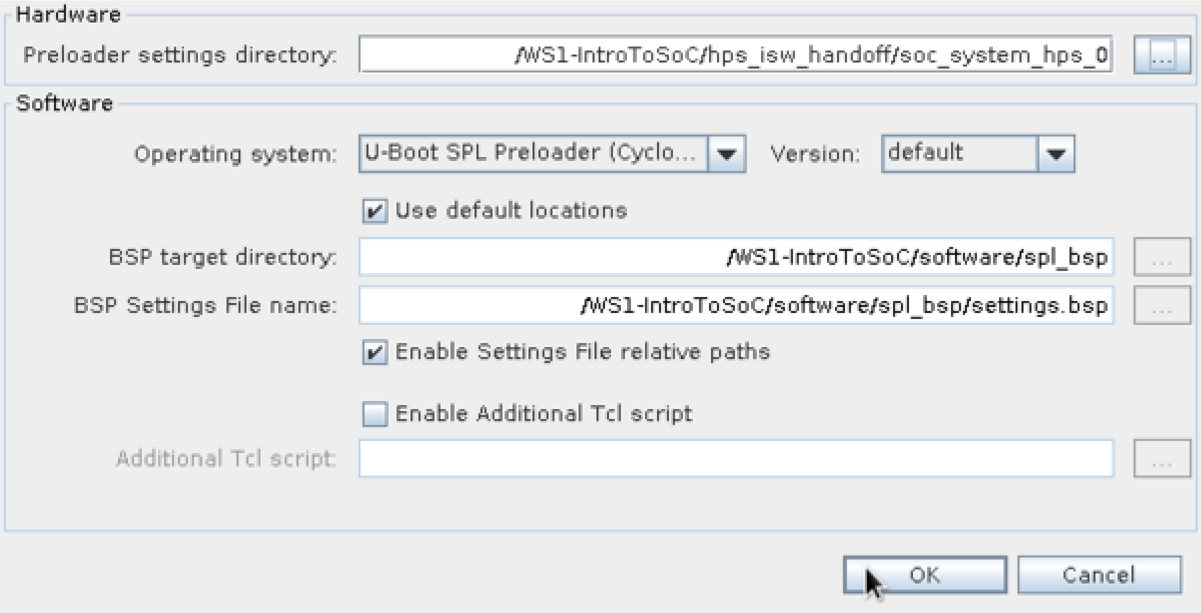


Look In: soc_system_hps_0

Folder name: /WS1-IntroToSoC/hps_isw_handoff/soc_system_hps_0

Files of Type: Preloader settings directory

6. The *New BSP* window will have all the settings populated, based on the handoff folder. Accept the default settings and click *OK*. This will close the window.



Hardware

Preloader settings directory: /WS1-IntroToSoC/hps_isw_handoff/soc_system_hps_0

Software

Operating system: U-Boot SPL Preloader (Cyclo... Version: default

☑ Use default locations

BSP target directory: /WS1-IntroToSoC/software/spl_bsp

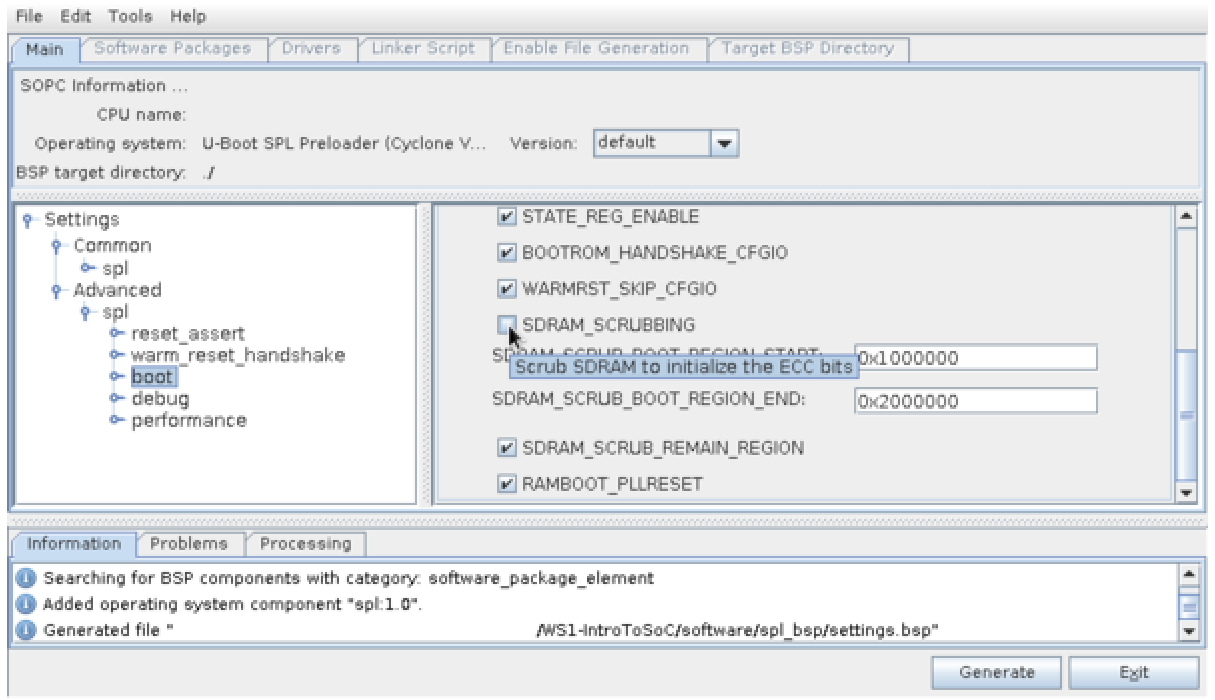
BSP Settings File name: /WS1-IntroToSoC/software/spl_bsp/settings.bsp

☑ Enable Settings File relative paths

☐ Enable Additional Tcl script

Additional Tcl script:

7. In the *BSP Editor* window, we want to accept the defaults settings, however, if your development board supports ECC SDRAM storage you should select the SDRAM scrubbing option shown below, otherwise leave it unselected.

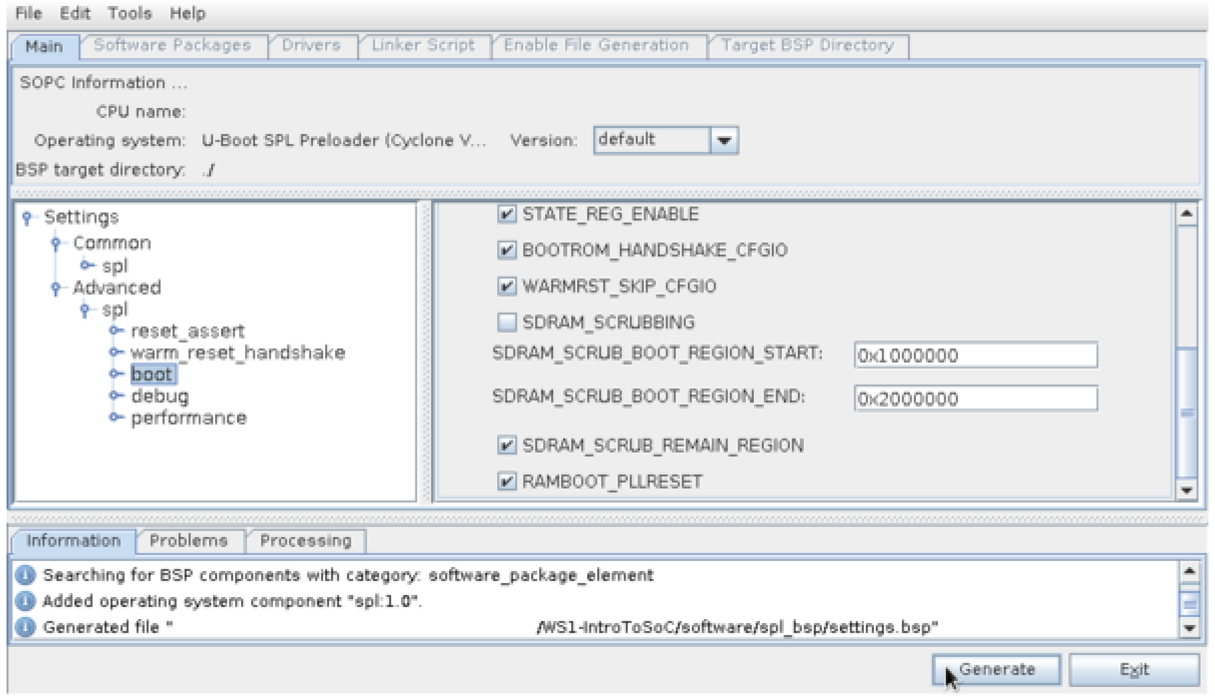


SOPC Information ...
CPU name:
Operating system: U-Boot SPL Preloader (Cyclone V... Version: default
BSP target directory: ./

☑ STATE_REG_ENABLE
☑ BOOTROM_HANDSHAKE_CFGIO
☑ WARMRST_SKIP_CFGIO
□ SDRAM_SCRUBBING
Scrub SDRAM to initialize the ECC bits
SDRAM_SCRUB_BOOT_REGION_START: 0x1000000
SDRAM_SCRUB_BOOT_REGION_END: 0x2000000
☑ SDRAM_SCRUB_REMAIN_REGION
☑ RAMBOOT_PLLRESET

```
Searching for BSP components with category: software_package_element
Added operating system component "spl:1.0".
Generated file "                    /WS1-IntroToSoC/software/spl_bsp/settings.bsp"
```

8. Now click the *Generate* button to generate your BSP for your Preloader.

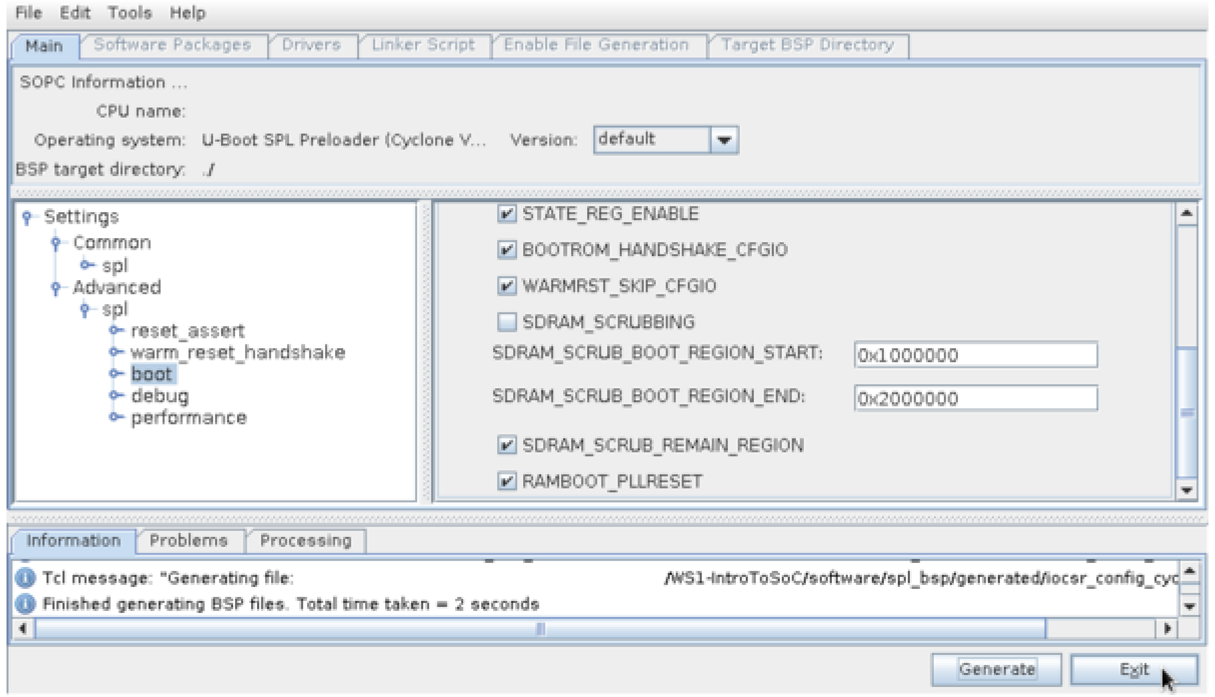


SOPC Information ...
CPU name:
Operating system: U-Boot SPL Preloader (Cyclone V... Version: default
BSP target directory: ./

☑ STATE_REG_ENABLE
☑ BOOTROM_HANDSHAKE_CFGIO
☑ WARMRST_SKIP_CFGIO
□ SDRAM_SCRUBBING
SDRAM_SCRUB_BOOT_REGION_START: 0x1000000
SDRAM_SCRUB_BOOT_REGION_END: 0x2000000
☑ SDRAM_SCRUB_REMAIN_REGION
☑ RAMBOOT_PLLRESET

```
Searching for BSP components with category: software_package_element
Added operating system component "spl:1.0".
Generated file "                    /WS1-IntroToSoC/software/spl_bsp/settings.bsp"
```

9. The message panel on the bottom will indicate the status of the generation. Click *Exit* to close the *BSP Editor*.



SOPC Information ...

CPU name:

Operating system: U-Boot SPL Preloader (Cyclone V... Version: default

BSP target directory: ./

☑ STATE_REG_ENABLE

☑ BOOTROM_HANDSHAKE_CFGIO

☑ WARMRST_SKIP_CFGIO

□ SDRAM_SCRUBBING

SDRAM_SCRUB_BOOT_REGION_START: 0x1000000

SDRAM_SCRUB_BOOT_REGION_END: 0x2000000

☑ SDRAM_SCRUB_REMAIN_REGION

☑ RAMBOOT_PLLRESET

```
Tcl message: "Generating file:          /WS1-IntroToSoC/software/spl_bsp/generated/iocsr_config_cyc
Finished generating BSP files. Total time taken = 2 seconds
```

The following items are generated in the "<path-to-lab-work>/WS1-IntroToSoC/software/spl_bsp/" folder

File	Description
generated	Folder containing source code that was generated based on the information from the handoff folder
settings.bsp	Preloader settings file, that contains the settings from the Preloader Generator
Makefile	Makefile used to build the Preloader
preloader.ds	ARM DS-5 AE that can be used to load the Preloader

Compiling the Preloader

In the Embedded Command Shell go to the generated Preloader folder:

```
[WS1-IntroToSoC]$ cd software/spl_bsp/
[spl_bsp]$ ls
generated  Makefile  preloader.ds  settings.bsp  uboot.ds
[spl_bsp]$
```

Run the "make" command to build the Preloader image:

```
[spl_bsp]$ make
[spl_bsp]$
```

The Makefile (also created by the Preloader Generator) performs the following steps:

```
 1 Extracts the fixed part of the Preloader source code
 1 Builds the Preloader executable using the fixed and the generated parts of the Preloader source code
 1 Converts the executable to binary, then adds the boot ROM required header to it
```

The following files are built in the "<path-to-lab-work>/WS1-IntroToSoC/software/spl_bsp/" folder:

File	Description
uboot-socfpga/spl/u-boot-spl	Preloader ELF file
uboot-socfpga/spl/u-boot-spl.bin	Preloader binary file
preloader-mkpimage.bin	Preloader image with the boot ROM required header

You can see these by listing the directory:

```
[spl_bsp]$ ls
generated  preloader.ds            settings.bsp  uboot-socfpga
Makefile   preloader-mkpimage.bin  uboot.ds
[spl_bsp]$
```

Generating the Preloader from the command line

> NOTE: the following command line Preloader creation description is not required lab work but provided as informational. You can perform the following command line examples below if you wish, but the goal of this lab is to create a new Preloader, which you have done by following the steps above. You may skip this section and resume at the "Using the new Preloader" section below.

If you would prefer to use a command line flow instead of the BSP Editor GUI to create your Preloader, you can do the following. It's easiest to run thru the BSP Editor GUI flow once, in order to create a "settings.bsp" file from the current tools release that contains the proper options and settings required to generate a proper Preloader, much like we described above. Once you have created your Preloader BSP and you have a valid "settings.bsp" file you can extract the settings from that database by running the "bsp-query-settings" command like this:

```
[spl_bsp]$ bsp-query-settings --settings settings.bsp --get-all --show-names
spl.CROSS_COMPILE arm-altera-eabi-
spl.PRELOADER_TGZ $(SOCEDS_DEST_ROOT)/host_tools/altera/preloader/uboot-socfpga.tar.gz
spl.boot.BOOTROM_HANDSHAKE_CFGIO 1
spl.boot.BOOT_FROM_NAND 0
spl.boot.BOOT_FROM_QSPI 0
spl.boot.BOOT_FROM_RAM 0
spl.boot.BOOT_FROM_SDMMC 1
spl.boot.CHECKSUM_NEXT_IMAGE 1
spl.boot.EXE_ON_FPGA 0
spl.boot.FAT_BOOT_PARTITION 1
spl.boot.FAT_LOAD_PAYLOAD_NAME u-boot.img
spl.boot.FAT_SUPPORT 0
spl.boot.FPGA_DATA_BASE 0xffff0000
spl.boot.FPGA_DATA_MAX_SIZE 0x10000
spl.boot.FPGA_MAX_SIZE 0x10000
spl.boot.NAND_NEXT_BOOT_IMAGE 0xc0000
spl.boot.QSPI_NEXT_BOOT_IMAGE 0x60000
spl.boot.RAMBOOT_PLLRESET 1
spl.boot.SDMMC_NEXT_BOOT_IMAGE 0x40000
spl.boot.SDRAM_SCRUBBING 0
spl.boot.SDRAM_SCRUB_BOOT_REGION_END 0x2000000
spl.boot.SDRAM_SCRUB_BOOT_REGION_START 0x1000000
spl.boot.SDRAM_SCRUB_REMAIN_REGION 1
spl.boot.STATE_REG_ENABLE 1
spl.boot.WARMRST_SKIP_CFGIO 1
spl.boot.WATCHDOG_ENABLE 1
spl.debug.DEBUG_MEMORY_ADDR 0xfffffd00
spl.debug.DEBUG_MEMORY_SIZE 0x200
spl.debug.DEBUG_MEMORY_WRITE 0
spl.debug.HARDWARE_DIAGNOSTIC 0
spl.debug.SEMIHOSTING 0
spl.debug.SKIP_SDRAM 0
spl.performance.SERIAL_SUPPORT 1
spl.reset_assert.DMA 0
spl.reset_assert.GPIO0 0
spl.reset_assert.GPIO1 0
spl.reset_assert.GPIO2 0
spl.reset_assert.L4WD1 0
spl.reset_assert.OSC1TIMER1 0
spl.reset_assert.SDR 0
spl.reset_assert.SPTIMER0 0
spl.reset_assert.SPTIMER1 0
spl.warm_reset_handshake.ETR 1
spl.warm_reset_handshake.FPGA 1
spl.warm_reset_handshake.SDRAM 0
```

If you pass the output of that command through "sed", you can format the output in an easily reusable way that you can build a BSP creation command from it.

```
[spl_bsp]$ bsp-query-settings --settings settings.bsp --get-all --show-names | sed -e "s/^\(.*\)\s\(.*\)$/--set \1 \"\2\" \\\/"
--set spl.CROSS_COMPILE "arm-altera-eabi-" \
--set spl.PRELOADER_TGZ "$(SOCEDS_DEST_ROOT)/host_tools/altera/preloader/uboot-socfpga.tar.gz" \
--set spl.boot.BOOTROM_HANDSHAKE_CFGIO "1" \
--set spl.boot.BOOT_FROM_NAND "0" \
--set spl.boot.BOOT_FROM_QSPI "0" \
--set spl.boot.BOOT_FROM_RAM "0" \
--set spl.boot.BOOT_FROM_SDMMC "1" \
--set spl.boot.CHECKSUM_NEXT_IMAGE "1" \
--set spl.boot.EXE_ON_FPGA "0" \
--set spl.boot.FAT_BOOT_PARTITION "1" \
--set spl.boot.FAT_LOAD_PAYLOAD_NAME "u-boot.img" \
--set spl.boot.FAT_SUPPORT "0" \
--set spl.boot.FPGA_DATA_BASE "0xffff0000" \
--set spl.boot.FPGA_DATA_MAX_SIZE "0x10000" \
--set spl.boot.FPGA_MAX_SIZE "0x10000" \
--set spl.boot.NAND_NEXT_BOOT_IMAGE "0xc0000" \
--set spl.boot.QSPI_NEXT_BOOT_IMAGE "0x60000" \
--set spl.boot.RAMBOOT_PLLRESET "1" \
--set spl.boot.SDMMC_NEXT_BOOT_IMAGE "0x40000" \
--set spl.boot.SDRAM_SCRUBBING "0" \
--set spl.boot.SDRAM_SCRUB_BOOT_REGION_END "0x2000000" \
--set spl.boot.SDRAM_SCRUB_BOOT_REGION_START "0x1000000" \
--set spl.boot.SDRAM_SCRUB_REMAIN_REGION "1" \
--set spl.boot.STATE_REG_ENABLE "1" \
--set spl.boot.WARMRST_SKIP_CFGIO "1" \
--set spl.boot.WATCHDOG_ENABLE "1" \
--set spl.debug.DEBUG_MEMORY_ADDR "0xfffffd00" \
--set spl.debug.DEBUG_MEMORY_SIZE "0x200" \
--set spl.debug.DEBUG_MEMORY_WRITE "0" \
--set spl.debug.HARDWARE_DIAGNOSTIC "0" \
--set spl.debug.SEMIHOSTING "0" \
--set spl.debug.SKIP_SDRAM "0" \
--set spl.performance.SERIAL_SUPPORT "1" \
--set spl.reset_assert.DMA "0" \
--set spl.reset_assert.GPIO0 "0" \
--set spl.reset_assert.GPIO1 "0" \
--set spl.reset_assert.GPIO2 "0" \
--set spl.reset_assert.L4WD1 "0" \
--set spl.reset_assert.OSC1TIMER1 "0" \
--set spl.reset_assert.SDR "0" \
--set spl.reset_assert.SPTIMER0 "0" \
--set spl.reset_assert.SPTIMER1 "0" \
--set spl.warm_reset_handshake.ETR "1" \
--set spl.warm_reset_handshake.FPGA "1" \
--set spl.warm_reset_handshake.SDRAM "0" \
```

If we back out of the "software/spl_bsp" directory to where we began the Preloader generation flow originally and delete the previously generated directory like this:

```
[spl_bsp]$ cd ../..
[WS1-IntroToSoC]$ rm -rf software/spl_bsp/
[WS1-IntroToSoC]$ ls software/
BareMetal_fft  Linux_fft_app
```

Then with just a few edits an a couple additions to the output we created above, we can create a properly formatted "bsp-create'settings" command to create a new Preloader BSP, like this:%BR% NOTE: we have added the "--bsp-dir", "--preloader-settings-dir", and "--settings" arguments and we changed the "$(SOCEDS_DEST_ROOT)" macro notation to "${SOCEDS_DEST_ROOT}" which is compatible with our bash shell. Other than this, all of the output from our "bsp-query-settings" output is reusable.

```
[WS1-IntroToSoC]$ bsp-create-settings \
--bsp-dir "./software/spl_bsp" \
--preloader-settings-dir "./hps_isw_handoff/soc_system_hps_0" \
--settings "./software/spl_bsp/settings.bsp" \
--type spl \
--set spl.CROSS_COMPILE "arm-altera-eabi-" \
--set spl.PRELOADER_TGZ "${SOCEDS_DEST_ROOT}/host_tools/altera/preloader/uboot-socfpga.tar.gz" \
--set spl.boot.BOOTROM_HANDSHAKE_CFGIO "1" \
--set spl.boot.BOOT_FROM_NAND "0" \
--set spl.boot.BOOT_FROM_QSPI "0" \
--set spl.boot.BOOT_FROM_RAM "0" \
--set spl.boot.BOOT_FROM_SDMMC "1" \
--set spl.boot.CHECKSUM_NEXT_IMAGE "1" \
--set spl.boot.EXE_ON_FPGA "0" \
--set spl.boot.FAT_BOOT_PARTITION "1" \
--set spl.boot.FAT_LOAD_PAYLOAD_NAME "u-boot.img" \
--set spl.boot.FAT_SUPPORT "0" \
--set spl.boot.FPGA_DATA_BASE "0xffff0000" \
--set spl.boot.FPGA_DATA_MAX_SIZE "0x10000" \
--set spl.boot.FPGA_MAX_SIZE "0x10000" \
--set spl.boot.NAND_NEXT_BOOT_IMAGE "0xc0000" \
--set spl.boot.QSPI_NEXT_BOOT_IMAGE "0x60000" \
--set spl.boot.RAMBOOT_PLLRESET "1" \
--set spl.boot.SDMMC_NEXT_BOOT_IMAGE "0x40000" \
--set spl.boot.SDRAM_SCRUBBING "0" \
--set spl.boot.SDRAM_SCRUB_BOOT_REGION_END "0x2000000" \
--set spl.boot.SDRAM_SCRUB_BOOT_REGION_START "0x1000000" \
--set spl.boot.SDRAM_SCRUB_REMAIN_REGION "1" \
--set spl.boot.STATE_REG_ENABLE "1" \
--set spl.boot.WARMRST_SKIP_CFGIO "1" \
--set spl.boot.WATCHDOG_ENABLE "1" \
--set spl.debug.DEBUG_MEMORY_ADDR "0xfffffd00" \
--set spl.debug.DEBUG_MEMORY_SIZE "0x200" \
--set spl.debug.DEBUG_MEMORY_WRITE "0" \
--set spl.debug.HARDWARE_DIAGNOSTIC "0" \
--set spl.debug.SEMIHOSTING "0" \
--set spl.debug.SKIP_SDRAM "0" \
--set spl.performance.SERIAL_SUPPORT "1" \
--set spl.reset_assert.DMA "0" \
--set spl.reset_assert.GPIO0 "0" \
--set spl.reset_assert.GPIO1 "0" \
--set spl.reset_assert.GPIO2 "0" \
--set spl.reset_assert.L4WD1 "0" \
--set spl.reset_assert.OSC1TIMER1 "0" \
--set spl.reset_assert.SDR "0" \
--set spl.reset_assert.SPTIMER0 "0" \
--set spl.reset_assert.SPTIMER1 "0" \
--set spl.warm_reset_handshake.ETR "1" \
--set spl.warm_reset_handshake.FPGA "1" \
--set spl.warm_reset_handshake.SDRAM "0"
```

Now that we have a new "settings.bsp" file created, we can generate the Preloader BSP contents with the "bsp-generate-files" command, like this:

```
[WS1-IntroToSoC]$ bsp-generate-files \
--bsp-dir "./software/spl_bsp" \
--settings "./software/spl_bsp/settings.bsp"
```

Compiling the Preloader

And then we can compile the Preloader just as we did in the BSP Editor flow:

```
[WS1-IntroToSoC]$ cd software/spl_bsp/
[spl_bsp]$ make
[spl_bsp]$ ls
generated  preloader.ds            settings.bsp  uboot-socfpga
Makefile   preloader-mkpimage.bin  uboot.ds
```

Installing the new Preloader

On a Linux host

Using the "dd" method

Insert your SD card into your development host machine using whatever SD card adapter you used to program the SD card image initially.

We need to know what block device our SD card is referenced with on our host, we can determine this by looking in the "/sys/block" directory. In our case we have used a USB SD card adapter and this is the only USB block device that we have attached to our system, so if we run the following command we can easily isolate the device we are interested in:

```
[spl_bsp]$ ls -l /sys/block/ | grep "usb"
lrwxrwxrwx 1 root root 0 Jun 20 10:30 sdb -> ../devices/pci0000:00/0000:00:1a.0/usb3/3-1/3-1.2/3-1.2:1.0/host9/target9:0:0/9:0:0:0/block/sdb
[spl_bsp]$
```

From the above command we see that the device appears to "sdb". We can also verify that in the "/proc/parittions" file, like this:

```
[spl_bsp]$ cat /proc/partitions
major minor  #blocks  name

...cut...
   8       16    3887104 sdb
   8       17     262144 sdb1
   8       18     260096 sdb2
   8       19       1024 sdb3
[spl_bsp]$
```

If you are not using a USB SD card adapter but have an internal SD card reader in your development host, then you may need to search for block devices installed on something other than USB. A common SD card reader device type is "mmc", so if you perform the above commands but search for "mmc" instead of "usb" then your output may look like this instead.

```
[spl_bsp]$ ls -l /sys/block | grep "mmc"
lrwxrwxrwx 1 root root 0 Jun 23 00:48 mmcblk0 -> ../devices/pci0000:00/0000:00:1c.0/0000:02:00.0/mmc_host/mmc0/mmc0:0007/block/mmcblk0
[spl_bsp]$ cat /proc/partitions
major minor  #blocks  name

...cut...
 179        0    3887104 mmcblk0
 179        1     262144 mmcblk0p1
 179        2     260096 mmcblk0p2
 179        3       1024 mmcblk0p3
[spl_bsp]$
```

For the purposes of the rest of this description, we will stick with the USB block device example and use "/dev/sdb" as our block device in the following commands.

Now that we know which block device we want to program, we need to determine which partition we need to program. The partition that stores the preloader is the 0xA2 partition on the SD card, we can determine which partition that is by running "fdisK" like this:

```
[spl_bsp]$ sudo fdisk -l /dev/sdb

Disk /dev/sdb: 3.7 GiB, 3980394496 bytes, 7774208 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x06bfaed7

Device     Boot  Start     End Sectors  Size Id Type
/dev/sdb1         4096  528383  524288  256M  b W95 FAT32
/dev/sdb2       528384 1048575  520192  254M 83 Linux
/dev/sdb3         2048    4095    2048    1M a2 unknown

Partition table entries are not in disk order.
[spl_bsp]$
```

From the above command we see that the 0xA2 partition is associated with "/dev/sdb3", so that is the partition that we will "dd" the new preloader into.%BR%

- %RED%CAUTION: be absolutely certain that you have the correct device, as "dd" to the wrong device can destroy your host system.%ENDCOLOR%* %BR%

NOTE: replace the XXX below with your block partition reference.

```
[spl_bsp]$ sudo dd if=preloader-mkpimage.bin of=/dev/XXX
512+0 records in
512+0 records out
262144 bytes (262 kB) copied, 0.238997 s, 1.1 MB/s
[spl_bsp]$
```

After we have programmed the new preloader onto our SD card we can run "sync" to ensure that all the IO is complete and then we can remove the SD card from our development host machine.

```
[spl_bsp]$ sync
[spl_bsp]$
```

Using the "alt-boot-disk-util" method

If you are uncomfortable using the "dd" command to program the new Preloader as shown above, or if you would like a safer mechanism to accomplish this, Altera provides a utility called "alt-boot-disk-util" in the !SoC EDS tools installation that attempts to provide this functionality for you. So if you follow the procedure above to determine the block device that your SD card is represented with on your development host, you can use "alt-boot-disk-util" to program your new Preloader. "alt-boot-disk-util" will inspect the device that you pass to it to ensure that it appears to be the block device with a 0xA2 partition on it and then program the new Preloader image into the proper location. You invoke it like this:

```
[spl_bsp]$ sudo <path-to-soceds-installation>/host_tools/altera/diskutils/alt-boot-disk-util -p preloader-mkpimage.bin -a write /dev/sdb
Altera Boot Disk Utility
```

```
Copyright (C) 1991-2014 Altera Corporation

Altera Boot Disk Utility was successful.
[spl_bsp]$
```

---+++ On a Windows host.

---+++ Using the "alt-boot-disk-util" method.

Insert your SD card into your development host machine using whatever SD card adapter you used to program the SD card image initially.

Determine the drive letter that the Windows environment represents the SD card image with on your host. "alt-boot-disk-util" will inspect the device that you pass to it to ensure that it appears to be the block device with a 0xA2 partition on it and then program the new Preloader image into the proper location. You invoke it like this:

```
[spl_bsp]$ alt-boot-disk-util -p preloader-mkpimage.bin -a write -d <drive-letter>
Altera Boot Disk Utility
Copyright (C) 1991-2014 Altera Corporation

Altera Boot Disk Utility was successful.
[spl_bsp]$
```

Running the new Preloader

To run the new preloader on your development target.

```
 1 Plug the SD card you just programmed with the new Preloader into the development target board.
 2 Connect the development target board to your development host machine with the USB serial cable.
 3 Start an appropriate terminal emulation program on your development host. See the "[[Documentation.SoCSWWorkshopSeriesTargetOrientation#USB_serial_console_interface.][USB serial console interface]]" page for advice on this.
 4 Power on your development target board.
```

What you should see in your terminal emulator program once you power up your develpment target board is the new Preloader that you just built and programmed onto the SD card recycling about every second. Why is this happening? The default settings for the Preloader are configured for the Preloader to load the next software stage from the 0xA2 partition at offset 0x40000, but the workshop image for the SD card does not use this boot flow, so there is nothing valid programmed into the 0xA2 partition at that offset. So when the Preloader loads the data from that offset and attempts to validate it, it fails, and the Preloader just falls into an infinite loop with nothing more that it can do. Eventually the system watchdog timer expires and produces a warm reset event to the HPS system and the whole process repeats.

Restoring the original Preloader

Now that we have proven that you can break your target by installing a preloader that cannot escape, let's restore the SD card to it's previous state.

```
 1 Power off and remove the SD card from your development target. %BR% %BR%
 2 Insert the SD card back into your development host. %BR% %BR%
 3 Mount the FAT partition from the SD card. %BR% %BR%
 4 Locate the original Preloader image "preloader-mkpimage.bin" in your board specific directory. (Here we use the DE0_NANO_SOC as an example)%BR%  600px %BR% %BR%
```

5 Now program the original Preloader back onto your SD card using one of the methods that was described above when proramming the new Preloader to the SD card. (Something like this)%BR%

```
[spl_bsp]$ alt-boot-disk-util -p <path-to-original-Preloader>/preloader-mkpimage.bin -a write /dev/sdb
Altera Boot Disk Utility
Copyright (C) 1991-2014 Altera Corporation
```

```
Altera Boot Disk Utility was successful.
[spl_bsp]$
```

%BR%

```
 1 After reprogramming the original Preloader, remove the SD card from your development host machine, plug it back into your development target and verify that your target boots all the way into linux again.
```

Retrieved from "http://www.alterawiki.com/wiki/index.php?title=Intro_to_Altera_SoC_Devices_for_HW_Developers_Workshop_-_Generating_the_Preloader&oldid=48697"
Category: Pages with broken file links

- This page was last modified on 9 December 2015, at 14:34.