Building Linux and Simple Application

Version 2.0

March 20, 2014

Corporate HQ & Design Center
380 Stevens Ave. Suite 206
Solana Beach, CA 92075
http://www.macnica-na.com

About Macnica Americas

Macnica Americas is a franchised semiconductor distributor for multiple, high-tech suppliers within North America. Our business model emphasizes unsurpassed technical support and knowledge versus other distribution options at no cost premium. Macnica Americas is the North American based division of Macnica Inc., a $2.4B global leader in semiconductor distribution. We maintain a field support staff as well as centralized design & applications teams.

Optional design services are headquartered in San Diego, CA., USA and offer partial or full turnkey design of FPGAs, power distribution networks, and full PCB design. Our expertise includes all aspects of high speed communications protocols and networking, video broadcast, signal processing, and storage applications. Macnica's specialty is high density, high speed complex FPGA designs utilizing multiple IP cores with fast time to market requirements.

Macnica can help you deliver a winning project with the unique combination of technical support, custom IP, and design services. Setup a meeting today!

http://www.macnica-na.com

License and Terms of Use

Table of Contents

1 Lab Overview

1.1 Introduction and Goals

This lab is designed as a self-paced-learning tool for understanding the fundamentals of the Yocto project building Linux for the Altera SoC. Only relatively small subset of the Yocto project options are explored. It is highly recommended persons attend additional training, such as that offered by Altera directly, for more detailed education on this rather complex flow and device family.

The lab is broken into a series of major sections or milestones representing the common phases of the Yocto flow. Unlike other trainings you may have had, this lab does not explicitly indicate every button to push or value to enter. Instead, your goal is described with the necessary information given. If you are having problems, each section concludes with a series of hints related to the tasks proposed.

1.2 Hardware and Software Requirements

- Linux host OS (This lab uses CentOS 6.4)
- Quartus II v13.1 (web or subscription edition) or stand-alone device programmer
- SoC EDS v13.1 installed on Linux
- Altera's GSRD Yocto Source Package, downloaded but NOT installed/extracted
  - http://releases.rocketboards.org/release/2013.11/gsrd/src/linux-socfpga-gsrd-13.1-src.bsx
- Macnica Helio SoC Evaluation board with 2 micro-USB cables and 1 Ethernet cable
- Win32 Disk Imager for Windows (Optional - writes Linux image/binary files to microSD using Windows host)
- Terminal Program (for Helio board UART interface)

1.3 Assistance

A dedicated e-mail account has been setup to receive support requests for the vWorkshop series. Please identify the course (in this case Yocto Linux) in addition to details on the question.
workshophelp@macnica.com

1.4 Lab Agenda and Milestones

Extract Yocto Source Package archive

The Yocto Source Package is an installer package provided by Altera that contains the Yocto build system, Yocto recipes & layers and all the necessary dependencies to compile the Altera Linux bootloader, kernel and root file system.

Install local set of Yocto recipes

Each user will use a local set of recipes to control each build.

Create local build location

This is a user's local copy of the Linux source.

Build Linux

Utilize the Yocto project concept of recipes and layers to build a complete Linux system, inclusive of the kernel, root file system and bootloader.

Build Preloader and Device Tree Blob

Utilize the SoC EDS tools to generate the necessary preloader and device tree blob to complete the generation of all required images to successfully boot Linux.

Program images to microSD flash

Once all essential Linux files are built, you will program them to the microSD card so that the Helio will boot from power-on.

Create and debug simple Linux application

The pre-defined Linux kernel has the GNU debug system enabled. You will write a simple "Hello World" application, use the Linaro cross-compiler to build the executable, then transfer the executable to the target and step through the application.

Modify U-Boot

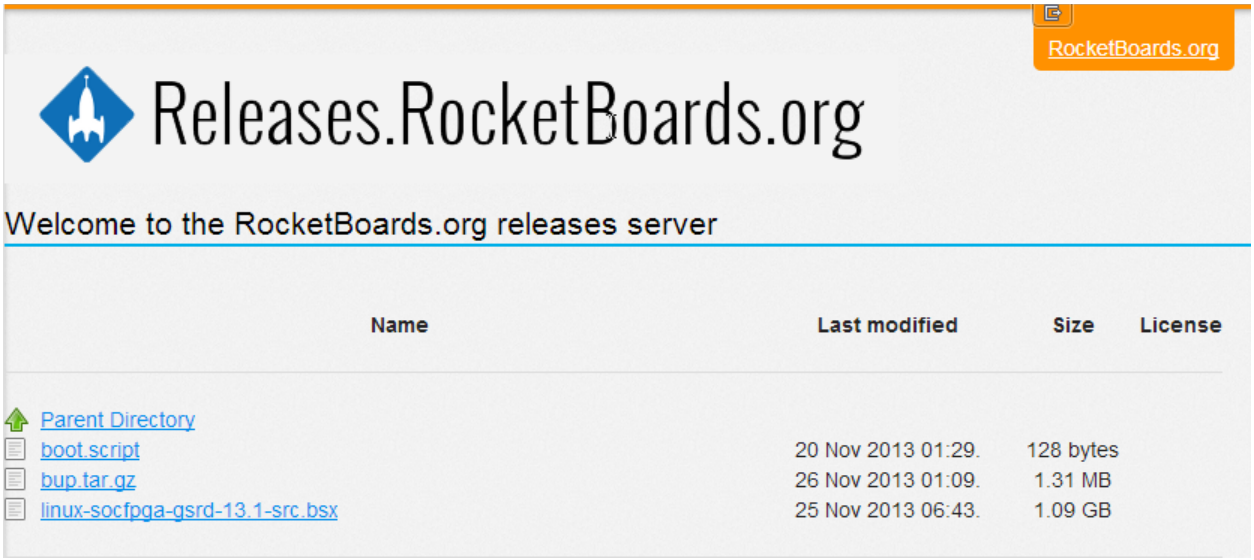
As part of the boot-up process, the bootloader, U-Boot, can be used to access the system at a low-level. Having the ability to modify the bootloader to accommodate custom functionality can prove invaluable.

2 Lab Instructions

2.1 Extract Yocto Source Package archive

2.1.1 Obtain latest Yocto Source Package archive

In order to match the solution project, obtain the latest archive. It can be downloaded from the RocketBoards.org: http://releases.rocketboards.org/release/2013.11/gsrd/src/linux-socfpga-gsrd-13.1-src.bsx

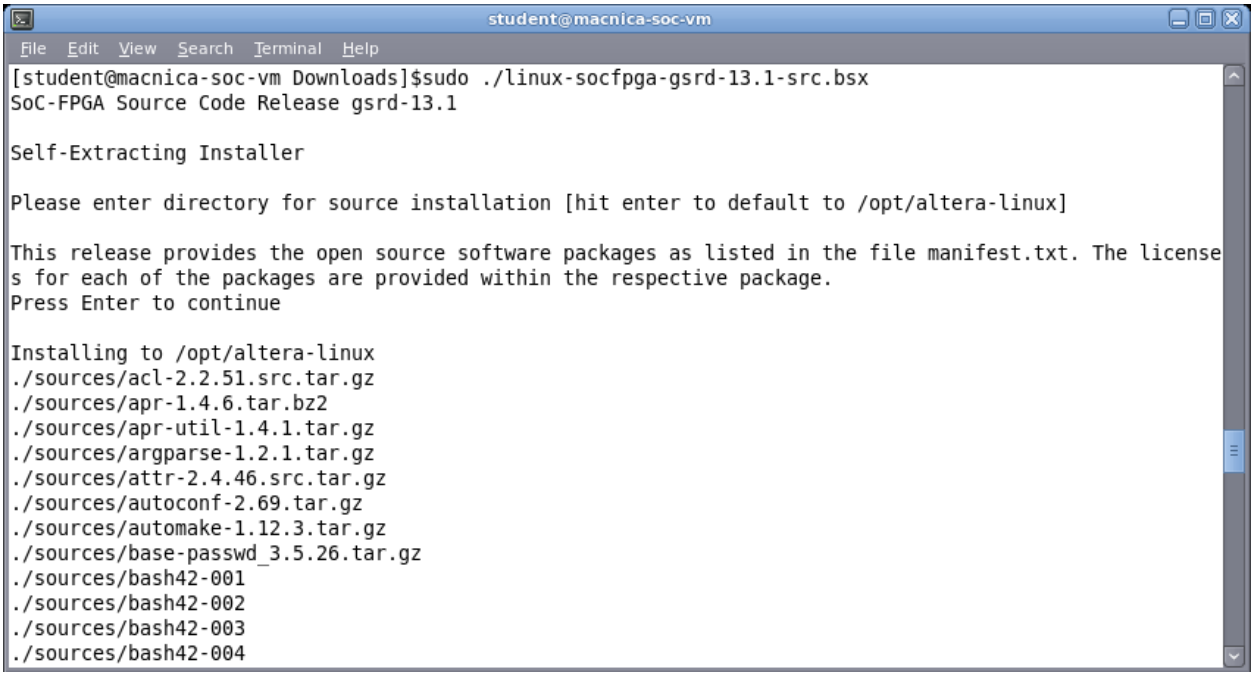


2.1.2 Extract archive

- ☐ The Yocto Source Package may be installed in a publicly accessible location, as this step can be shared by all the users on the system. The default location is /opt/altera-linux.

Hints:

- *If you wish to use this location, you will likely need root access in order to access this directory. This is why the command shown below is ran using sudo.*



Additional installed files question

☐ What additional critical files were installed with the Yocto Source Package archive extraction?

_____

Answer:

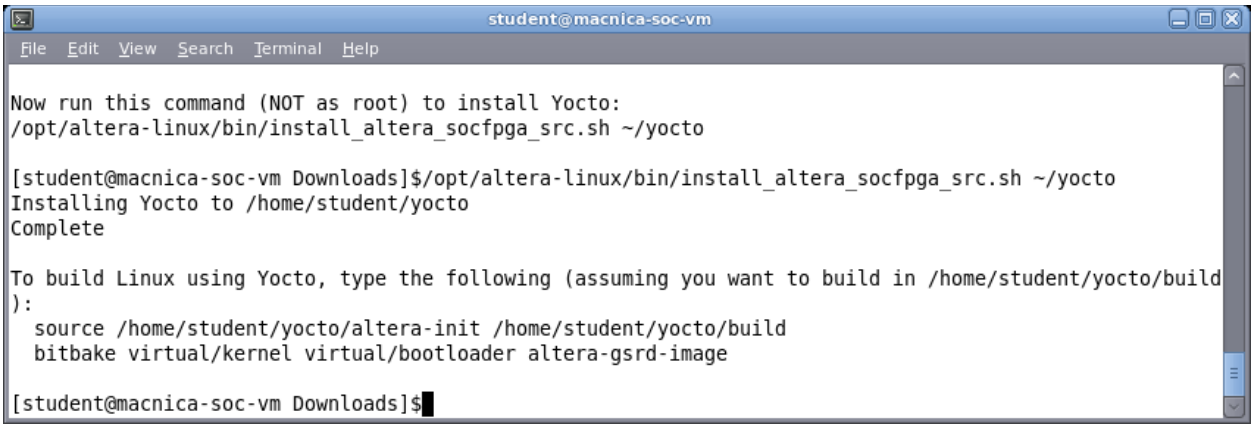
- *The Linaro tool chain.*

2.2 Install local set of Yocto recipes

The next step is to install a local set of Yocto recipes. This could be done in a shared location, but if someone wants or needs to modify them they should have their own version. The default install location for this is within your home directory:

2.2.1 Local recipes

- ☐ Follow the prompts from the archive extraction in the previous step to create the local set of recipes in your home folder ~/yocto.



Meta layers questions

- ☐ 1. What did the 'install_altera_socfpga_src.sh' script do?

  _____

- ☐ 2. What is in the ~/yocto/meta* folders?

  _____

- ☐ 3. What is POKY?

  _____

Meta layer answers:

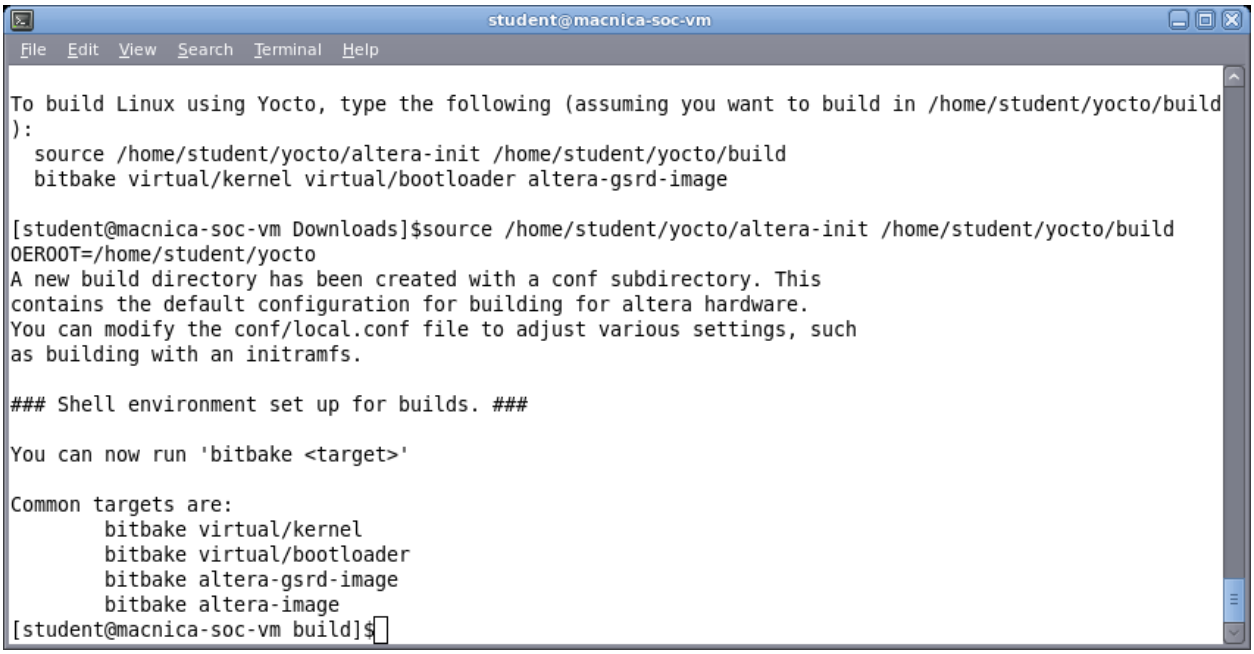
1. **The script makes a local copy of the Poky build layers and recipes from the Yocto Source Package /opt/altera-linux/sources/poky-danny-8.0tar.gz**
2. **The meta layer folders are the collection of recipes that configure the Poky kernel build.**
3. **Poky is an integration of various components to form a complete prepackaged build system and development environment. It features support for building customised embedded device style images. There are reference demo images featuring an X11/Matchbox/GTK themed UI called Sato. The system supports cross-architecture application development using QEMU emulation and a standalone tool chain and SDK with IDE integration.**

2.3 Create local build location

The last step in setting up the build environment is to create a build directory. By keeping this separate from your Yocto source you can erase an entire build without fear of deleting your Yocto sources. Also, you can have several build directories, each with its own configuration, all based on the same Yocto source.

2.3.1 Local build directory

☐ Follow the prompts from the recipe installation in the previous step.



Local build script question

☐ What did the 'altera-init' script do?

_____

Local build script answer:

- **The script serves 2 purposes:**
  - **First, it creates the new build directory using Altera's default configuration. This is the specific configuration that dictates the target hardware, "MACHINE = socfpga_cyclone5" as well as the tool chain to be used, Linaro, and the location of the local source archives, /opt/altera-linux/source.**
  - **Secondly, it sets shell variables that are required for building. Note: If you start a new shell you will need to run script again to set the shell variables again. It will not create another build directory if one exists.**

2.4 Build Linux

In order to boot a functional Linux kernel on the Altera SoC devices, there are five essential items that need to be generated. The first time you build the Yocto Source Package it may take up to several hours depending on your host machine.

File Name	Description
socfpga.dtb	Device Tree Blob file
zImage	Compressed Linux kernel image file
various	Linux root filesystem
preloader-mkpimage.bin	Preloader image
u-boot-socfpga_cyclone5.img	U-boot image

2.4.1 Build bootloader, kernel and root file system

The Linux source package provided by Altera targets the Altera SoC development board which has the capability to program the FPGA from the HPS. However, on the mPression Helio board, the HPS boot-mode pins are hard-wired to boot from the MMC not allowing the HPS to program the FPGA. This is important to realize since the default bootloader source code does not enable the bridges between the HPS and FPGA fabric. We will need to add a simple bootloader command to do this.

Use the Yocto build system 'bitbake' command to:

- ☐ Modify the default U-Boot "bootcmd" to include the "bridge_enable_handoff" command.
- ☐ Build the U-Boot bootloader
- ☐ Build the kernel.
- ☐ Build the root filesystem.

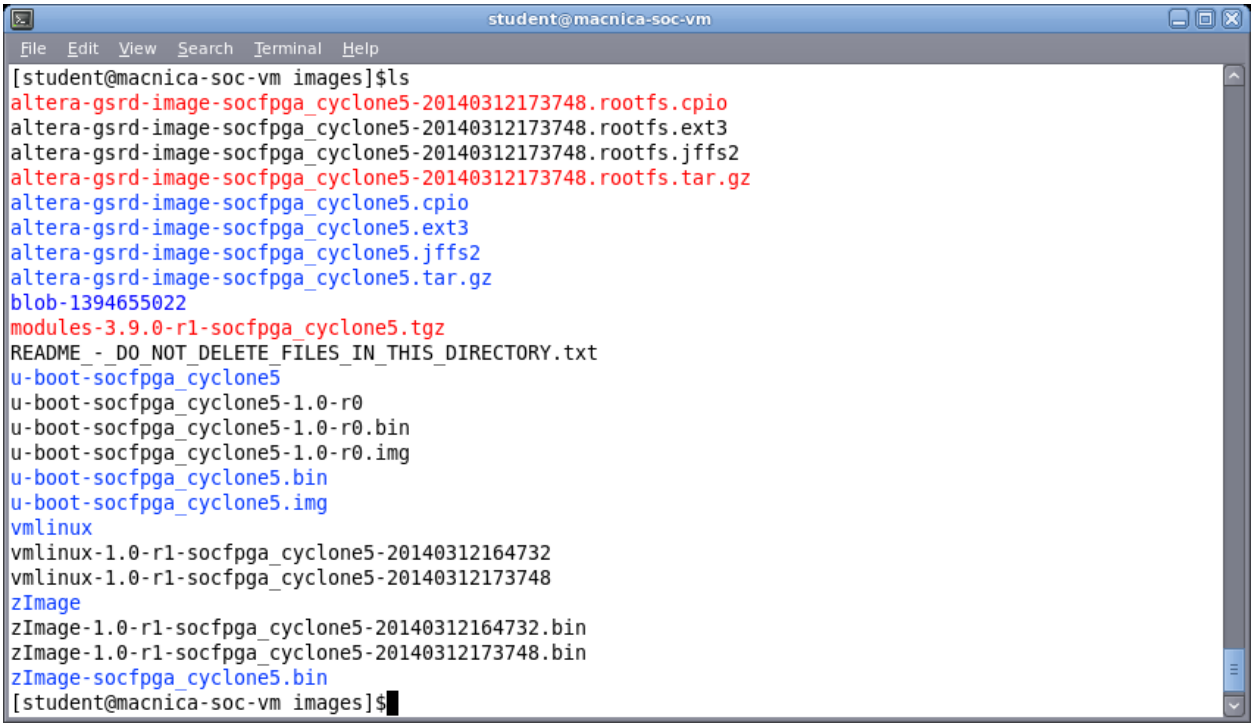
Once finished, all images generated will be placed in `~/yocto/build/tmp/deploy/images`

Hints:

- *Extract the bootloader source from the archives into your local build tree.*

- *'bitbake –c install virtual/bootloader'*

- *The default "bootcmd" is defined in ~ /yocto/build/tmp/work/armv7ahf-vfp-neon-poky-linux-gnueabi/u-boot-altera-dist-1.0-r0/uboot-socfpga/include/configs/socfpga_common.h*
  - *Change line 171 to include the "bridge_enable_handoff" command*
  - *#define CONFIG_BOOTCOMMAND "run bridge_enable_handoff; run callscript; run mmcload; run mmcboot"*
- *Force the bootloader to be compiled and deployed incorporating your changes.*
  - *'bitbake -f -c compile virtual/bootloader && bitbake -f -c deploy virtual/bootloader'*
- *Build kernel*
  - *'bitbake virtual/kernel'*
- *Build root file system.*
  - *'bitbake altera-gsrd-image'*



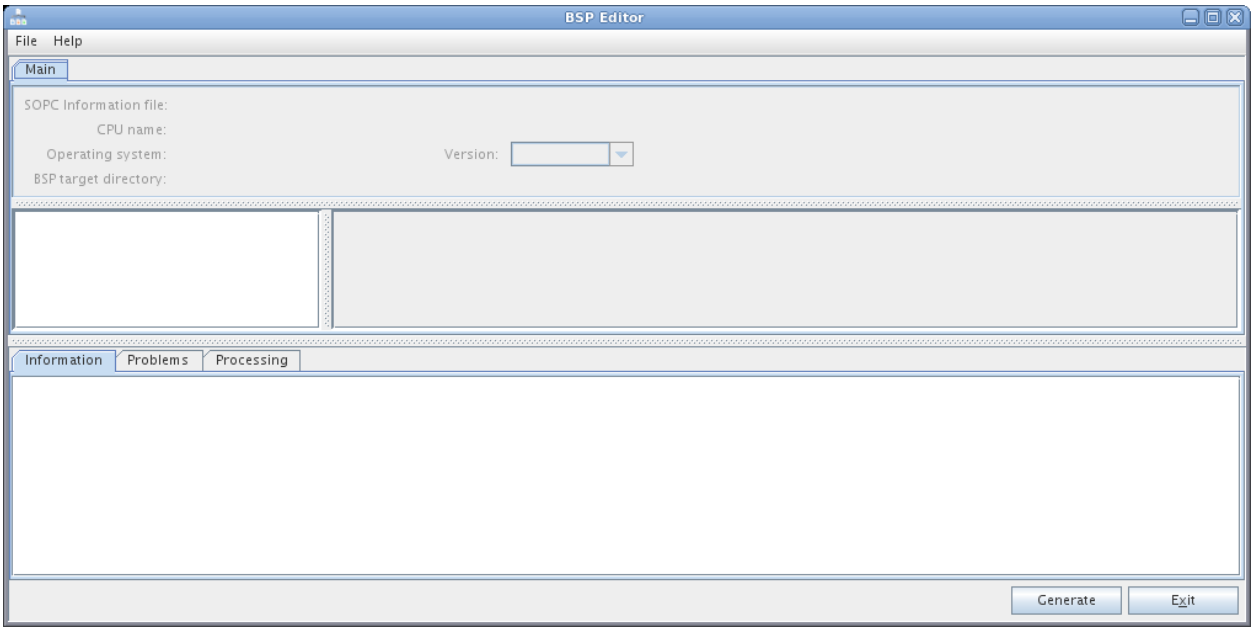
2.4.2 Build preloader and device tree blob

Recall that included in the SoC EDS install are the tools to generate the preloader and device tree blob that are associated with a specific hardware implantation of the SoC device.

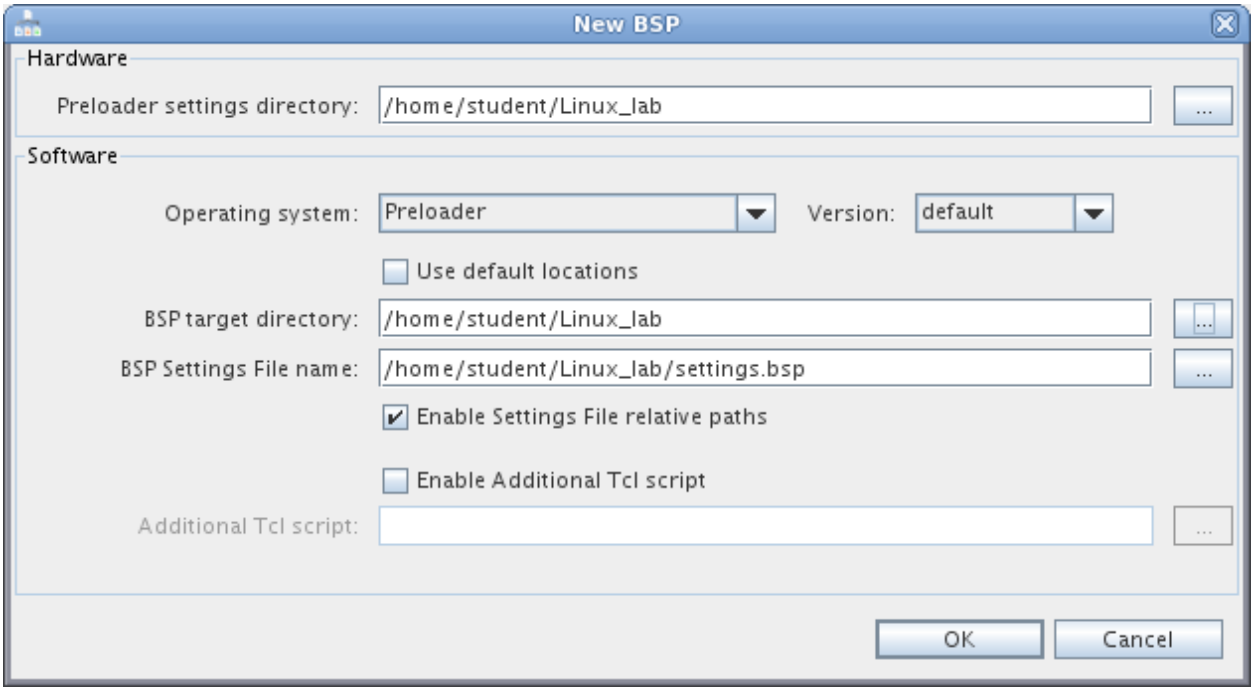
- ☐ Extract the SoC_7_lab.zip files to a unique location on your host machine.
- ☐ Build the preloader buy using the "bsp-editor" GUI to generate the required source tree and Makefile
- ☐ Build the device tree blob by using the device tree generator tools made up of "sopc2dts" and "dtc."

Hints:

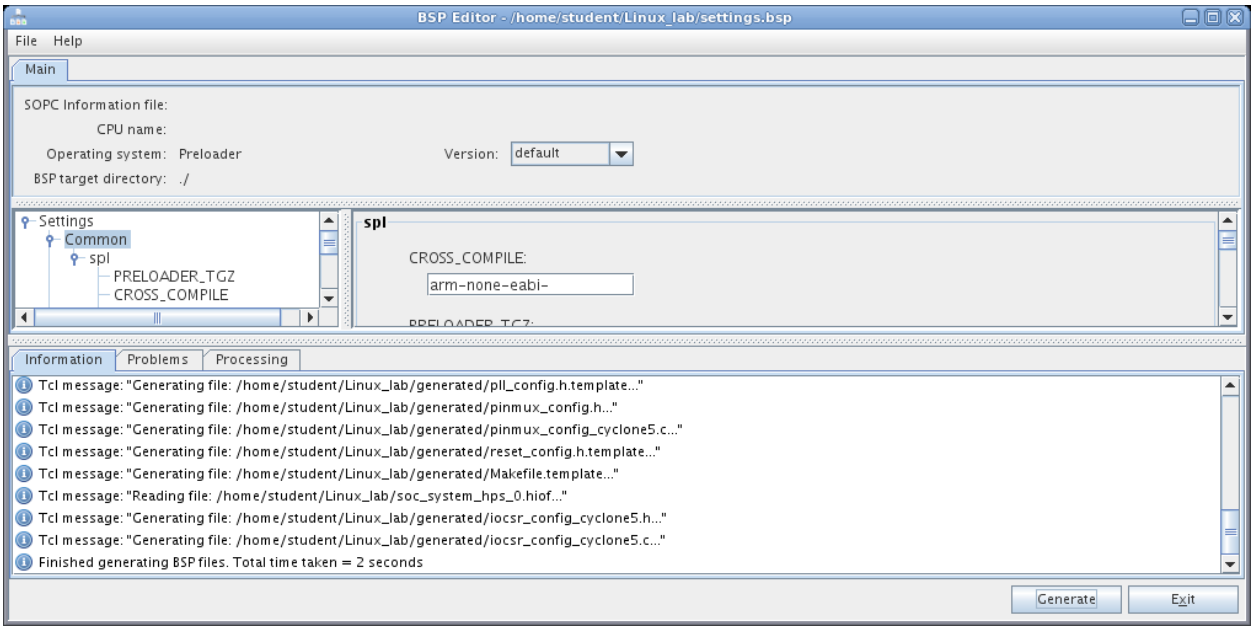
- *Use a decompression tool of your choice to extract the hand-off files in the SoC_7_lab.zip file associated with this lab. In the sample screen shots, the files were extracted to /home/student/Linux_lab.*
    - *Note: These files are generated as part of the Quartus hardware build process.*
- *Start the SoC EDS embedded command shell to set-up appropriate environment*
    - */opt/altera-tools/13.1/embedded/embedded_command_shell.sh*
- *Launch the bsp-editor GUI*
    - *bsp-editor &*



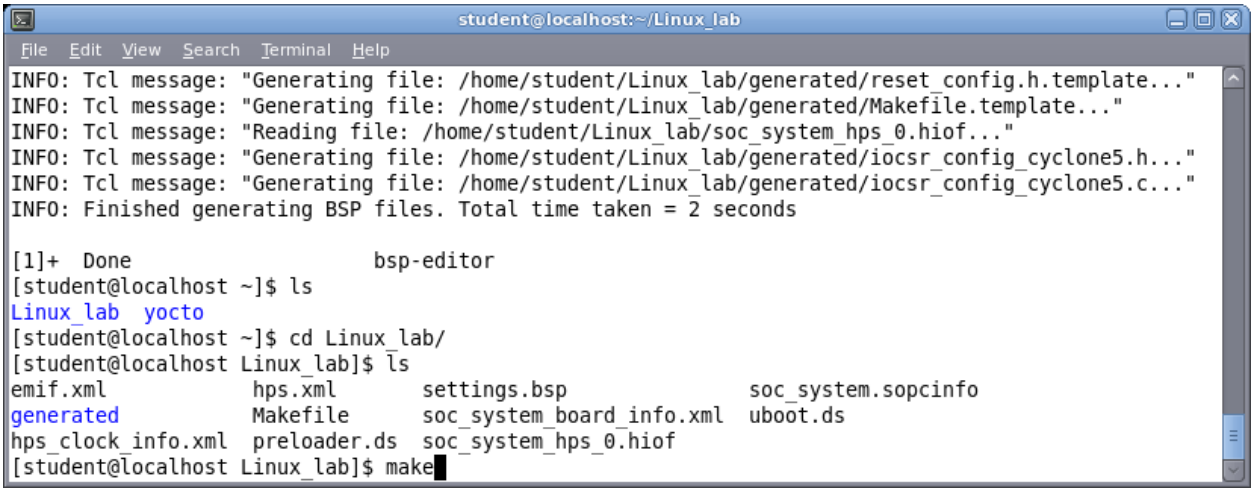
- *Point to the directory where you copied the hand-off files provided.*
- *Change the BSP target directory to point to the lab folder as well.*



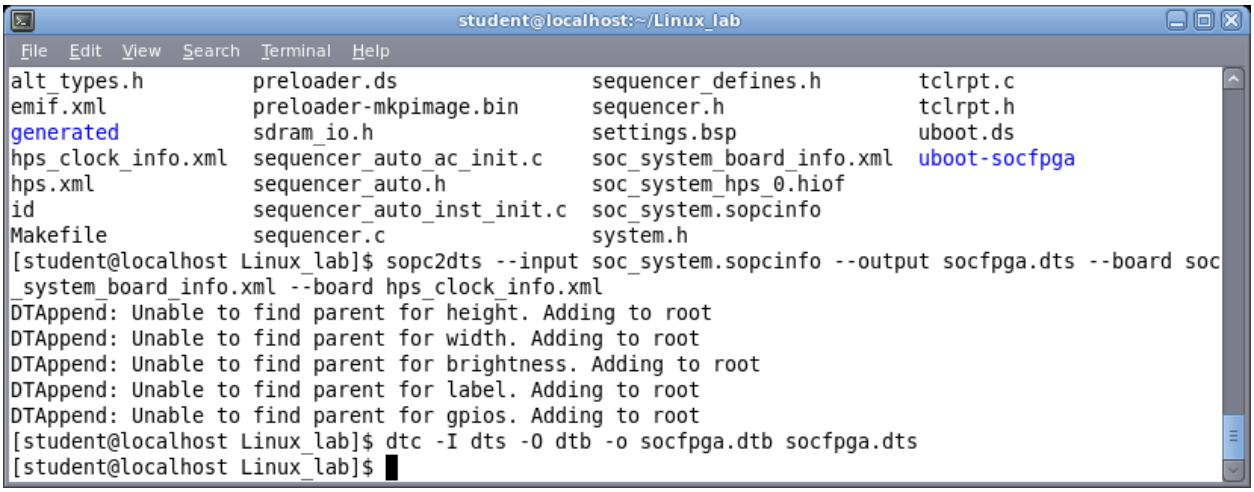
- *Leave all default settings and click Generate to create the Makefile and other BSP settings.*



- *Click Exit and return to the embedded command shell prompt.*
- *Change directory to the location of the Makefile and enter "make" to build the preloader binary.*



- *To build the device tree blob, first generate the device tree source by using the sopc2dts tool in the embedded command shell.*
  - *sopc2dts --input soc_system.sopcinfo*
    --output socfpga.dts
    --board soc_system_board_info.xml
    --board hps_clock_info.xml
- *You can safely ignore the DTAppend messages*
- *Next, use the dtc tool to compile the device tree source.*
  - *dtc -I dts -O dtb -o socfpga.dtb socfpga.dts*



Linux build questions

- ☐ 1. Recall, the preloader is hardware dependant. Where did the source come from?

  _____

- ☐ 2. How would you modify the kernel? What if you want to add/remove features of the kernel?

  _____

- ☐ 3. Is there an easier way to copy all five essential files to the microSD flash card without copying each file independently?

  _____

Linux build answers:

1. **As part of the SoC EDS installation process, a snapshot of the U-Boot source that creates the preloader is installed. The Makefile creates a local copy and builds the preloader from there.**
2. **The widely used 'menuconfig' kernel configuration menu system is available from within the Yocto environment. The command to run is 'bitbake –c menuconfig virtual/kernel'**
3. **Yes. The script included in the Yocto source package 'make_sdimage.sh' will create a single binary image. See next section for details.**

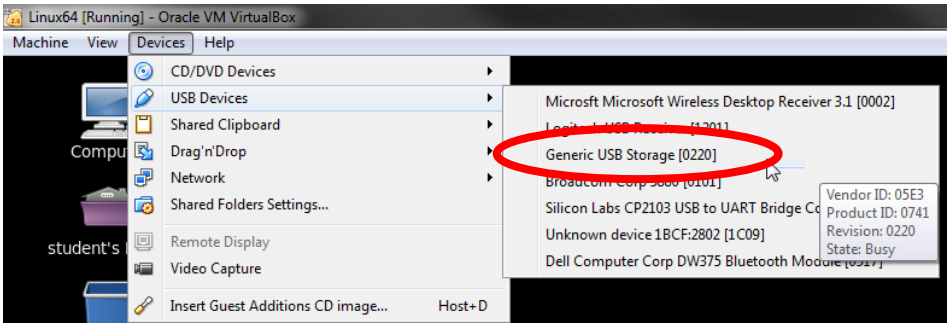
2.5 Program images to microSD flash

To boot Linux on the Helio SoC FPGA development kit, you need to write the images you just built into one of the two Flash devices: SDMMC or QSPI. For this lab, we will use SDMMC due to its easy detachability. For SDMMC boot, all boot images will be located inside microSD/MMC card. A script, make_sdimage.sh is provided that will create a microSD card image, ready to be deployed. You can run the script without parameters to obtain information about the tool:
`/opt/altera-linux/bin/make_sdimage.sh`

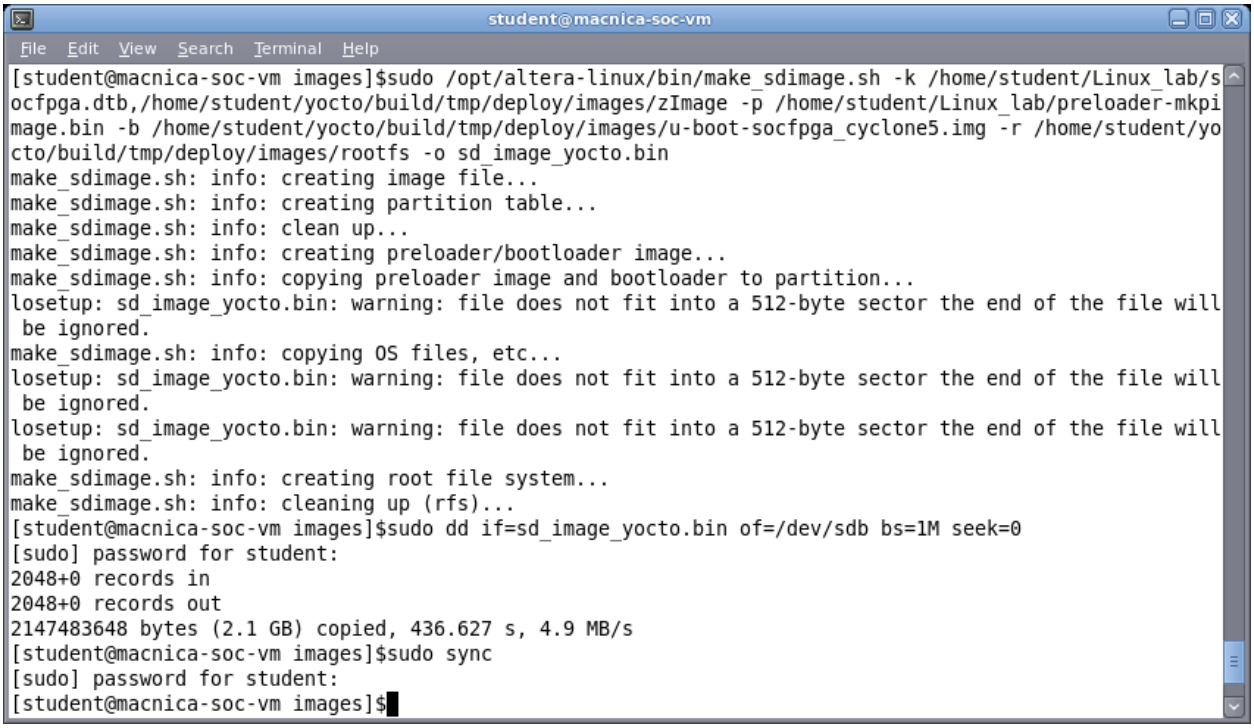
- ☐ Uncompress the root file system, `altera-gsrd-image-socfpga_cyclone5.tar.gz`, into a local directory
- ☐ Invoke the `make_sdimage.sh` script to create the microSD card image.
- ☐ Program the complete image, `sd_image_yocto.bin`, into the microSD card.
- ☐ Boot Linux

Hints:

- *cd ~/yocto/build/tmp/deploy/images*
- *mkdir rootfs*
- *cd rootfs*
- *sudo tar xzf ../altera-image-socfpga_cyclone5.tar.gz*
- *cd ..*
- *sudo /opt/altera-linux/bin/make_sdimage.sh /*
  -k /home/student/Linux_lab/socfpga.dtb,
  */home/student/yocto/build/tmp/deploy/images/zImage *
  *-p /home/student/Linux_lab/preloader-mkpimage.bin *
  *-b /home/student/yocto/build/tmp/deploy/images/u-boot-socfpga_cyclone5.img *
  *-r /home/student/yocto/build/tmp/deploy/images/rootfs *
  -o sd_image_yocto.bin
- *Program microSD card (via Windows or Linux)*
  - *Windows: transfer the sd_image_yocto.bin file to your Windows host, rename it sd_image_yocto.img and use Win32DiskImager utility to program the microSD card. (see vWorkshops_Getting_Started.pdf)*
  - *Linux: Insert microSD USB adapter into host PC. If using a VM, allow the VM to take control of the microSD by selecting "Devices -> USB Devices -> Generic USB Storage"*



- *Use 'sudo blkid' and locate the device that has the "vfat" and "ext3" partitions. This is your microSD card. Typically the microSD is mounted as /dev/sdb. (/dev/sda and /dev/mapper are part of the Linux VM)*
- *sudo dd if=sd_image_yocto.bin of=/dev/sdb bs=1M seek=0*
- *sudo sync*



- *The default serial communications needs to be set-up as*
  - *115600 baud*
  - *8 bit data*
  - *No parity*
  - *1 stop bit*
  - *No flow control*

2.6 Create and debug simple Linux application

As part of the Yocto install, the Linaro tool chain was installed. This was the tool chain that was used to compile the Linux kernel in the previous steps. Now that a complete kernel has been built, programmed to the microSD flash and booted on the target, we can leverage the same tool chain to write and debug user applications that run on the Linux OS running on the target.

2.6.1 Build "Hello World" Linux application

- ☐ In your host Linux machine, write a simple "Hello World" application.
- ☐ Set up Linux host environment for Linaro cross-compiler.
- ☐ Build the application with the Linaro cross-compiler.
- ☐ Connect the target board to the same network as your host machine.
- ☐ Set up Ethernet interface on the target (if the DHCP server in your network did not automatically configured an IP address.)
- ☐ Transfer the application executable ELF to the running Linux OS on the target.
- ☐ Run application on the target.

Hints:

- ```
  #include <stdio.h>
  int main()
  {
          printf("Hello World!\n");
          return 0;
  }
  ```
- Set path the Linaro tool chain
  - *export PATH=/opt/altera-linux/linaro/gcc-linaro-arm-linux-gnueabihf-4.7-2012.11-20121123_linux/bin:$PATH*
- Compile application
  - *arm-linux-gnueabihf-gcc –Wall –g –O0 helloworld.c –o helloworld*
- Log into the target via the serial link and set a password for root
  - *passwd root*
- Set IP address of target
  - *ifconfig eth0 192.168.1.100 (IP address needs to be in the same domain as host)*
- From the host secure copy the executable to target
  - *scp helloworld root@192.168.1.100:/home/root/helloworld*
- On the target you can now execute application
  - *./helloworld*
  ```

2.6.2 Debug "Hello World" Linux application

Part of the default kernel build includes the GNU gdb debug server. You can now launch the application just built using the debug server and connect to the debugger from the host.

- ☐ Start the GDB debug server on the target using the above compiled application.
- ☐ Connect to target from host to debug the application.
- ☐ Set a breakpoint at main().
- ☐ Examine registers, step application from host, etc.

Hints:

- Start gdbserver on the target
  - */usr/bin/gdbserver <host ip>:1234 ./helloworld &*
- Start debugger on host
  - *arm-linux-gnueabihf-gdb ./helloworld*
- Connect to target running gdbserver
  - *target remote192.168.1.100:1234*
- Set breakpoint at main()
  - *b main*
- Examine registers & stack
  - *info all-registers*
  - *info stack*
- Start application and run to first breakpoint
  - *cont*
- Step application
  - *n*
- *End debug session*
  - *disconnect*

2.7 Modify U-Boot (Requires Quartus II Programmer)

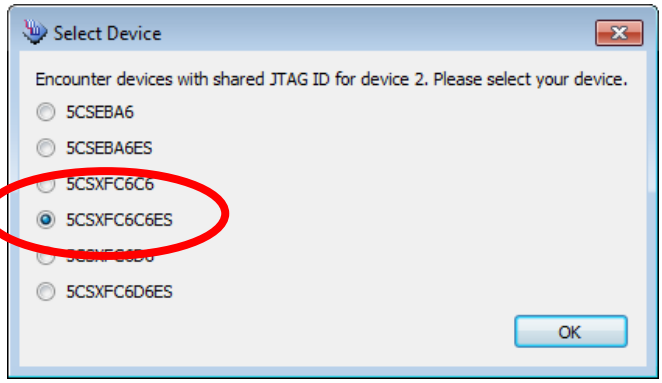
A fundamental piece to booting Linux is the bootloader, typically U-Boot. Sometimes it is necessary to access the embedded system BEFORE the kernel has been loaded for verification of settings, IP control, or any number of low-level reasons. In the following example, you will be modifying the default U-Boot menu system to add in the capability to directly access control registers in the FPGA based LED driver IP.

It is necessary to have the FPGA programmed with the example design for the Helio board which includes the FPGA LED IP.

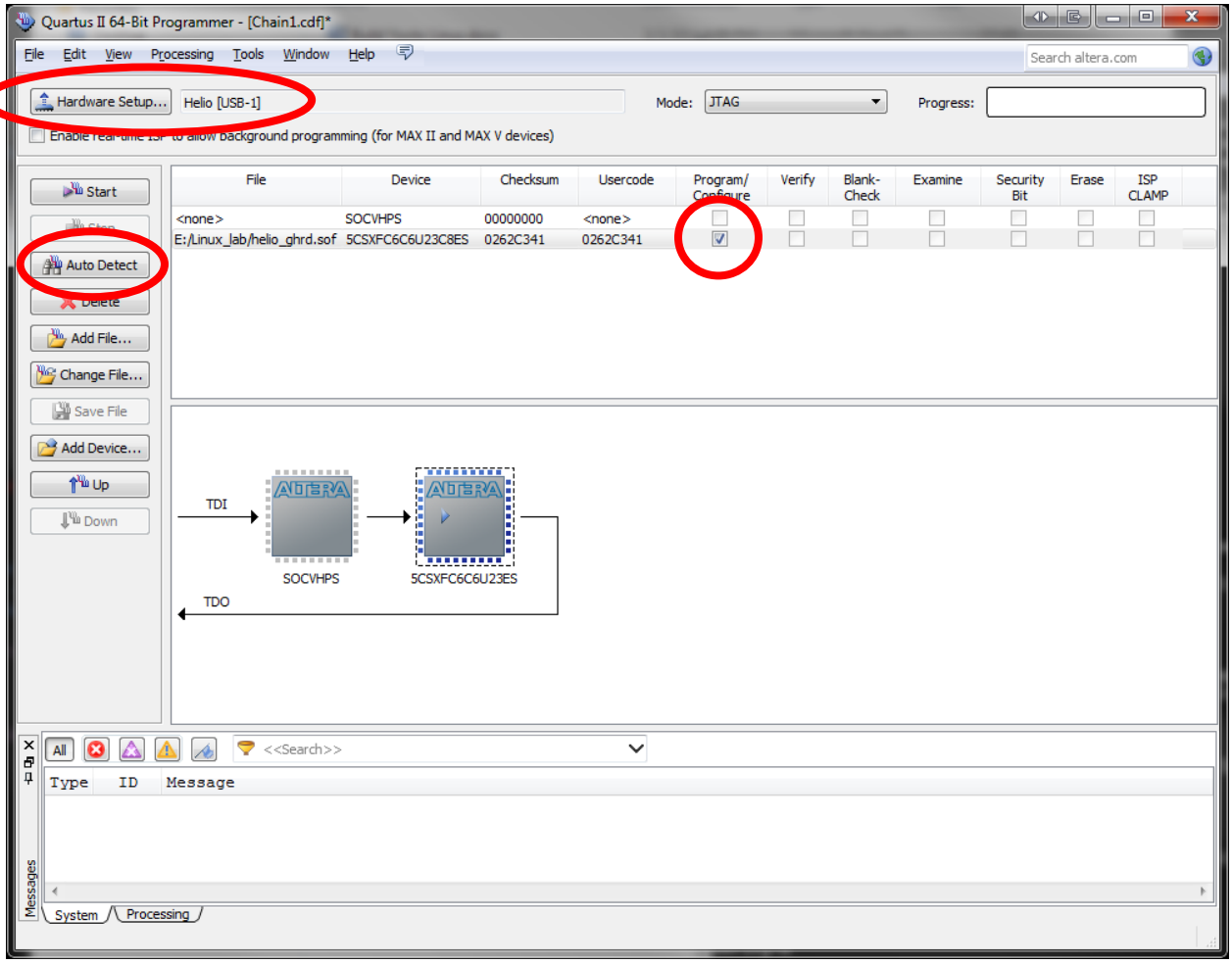
2.7.1 Program the FPGA

The helio_ghrd.sof file used to program the SoC FPGA can be found in the resources folder included with this lab manual.

- ☐ Connect a USB cable to the **BLASTER USB** port on the Helio board.
- ☐ Launch the Quartus II Programmer: **~\quartus\bin\quartus_pgmw** (or equivalent on Windows)
- ☐ **Auto Detect** the chain and select the **5CSXFC6C6ES** device.



- ☐ Right click on the FPGA device and chose **Change File** and select the **helio_ghrd.sof** file.
- ☐ Check the box to **Program/Configure** the FPGA device.
- ☐ Ensure the **Cyclone V SoC Base Board [USB-x]** connection is visible in the **Hardware Setup** dialog box. If not, click the button and select the cable. (Drivers must be previously installed. See the Altera web site for instructions.)



- ☐ Click **Start** to initiate the programming of the FPGA.
- ☐ Upon successful programming, observe the 4 LEDs illuminated near the power input jack.
    - Note: If you power cycle the board, you will need to configure the FPGA again.

2.7.2 Modify U-Boot Source

The details of the code modifications that are going to be made are beyond the scope of this lab, however, for the purpose of understanding the functionality of the code below, the following details are presented:

The control register to turn the LEDs on/off is located at 0xFF21_0040. Writing 0x0 illuminates the LEDs and writing 0xFFFF turns the LEDs off.

Recall that running the '*install_altera_socfpga_src.sh*' script created a local set of layers and recipes then running the '*bitbake virtual/bootloader*' command used those recipes and created a local copy of the source code for the bootloader build. The C source file that contains the U-Boot menu, cmd_mem.c, is located here:

~yocto/build/tmp/work/armv7ahf-vfp-neon-poky-linux-gnueabi/u-boot-altera-dist-1.0-r0/uboot-socfpga/common/cmd_mem.c

Modify the cmd_mem.c file as follows:

- ☐ At line 200, after the "do_mem_mw" function add:

```
int do_led_on (cmd_tbl_t * cmdtp, int flag, int argc, char * const argv[])
{
	*((ulong *)0xFF210040) = 0x0;
}

int do_led_off (cmd_tbl_t * cmdtp, int flag, int argc, char * const argv[])
{
	*((ulong *)0xFF210040) = 0xFFFF;
}
```

- ☐ At the very end of the file add:

```
U_BOOT_CMD(
	ledon,  1,	1,	do_led_on,
	"turn fpga leds on",
	""
);

U_BOOT_CMD(
	ledoff, 1,	1,	do_led_off,
	"turn fpga leds off",
	""
);
```

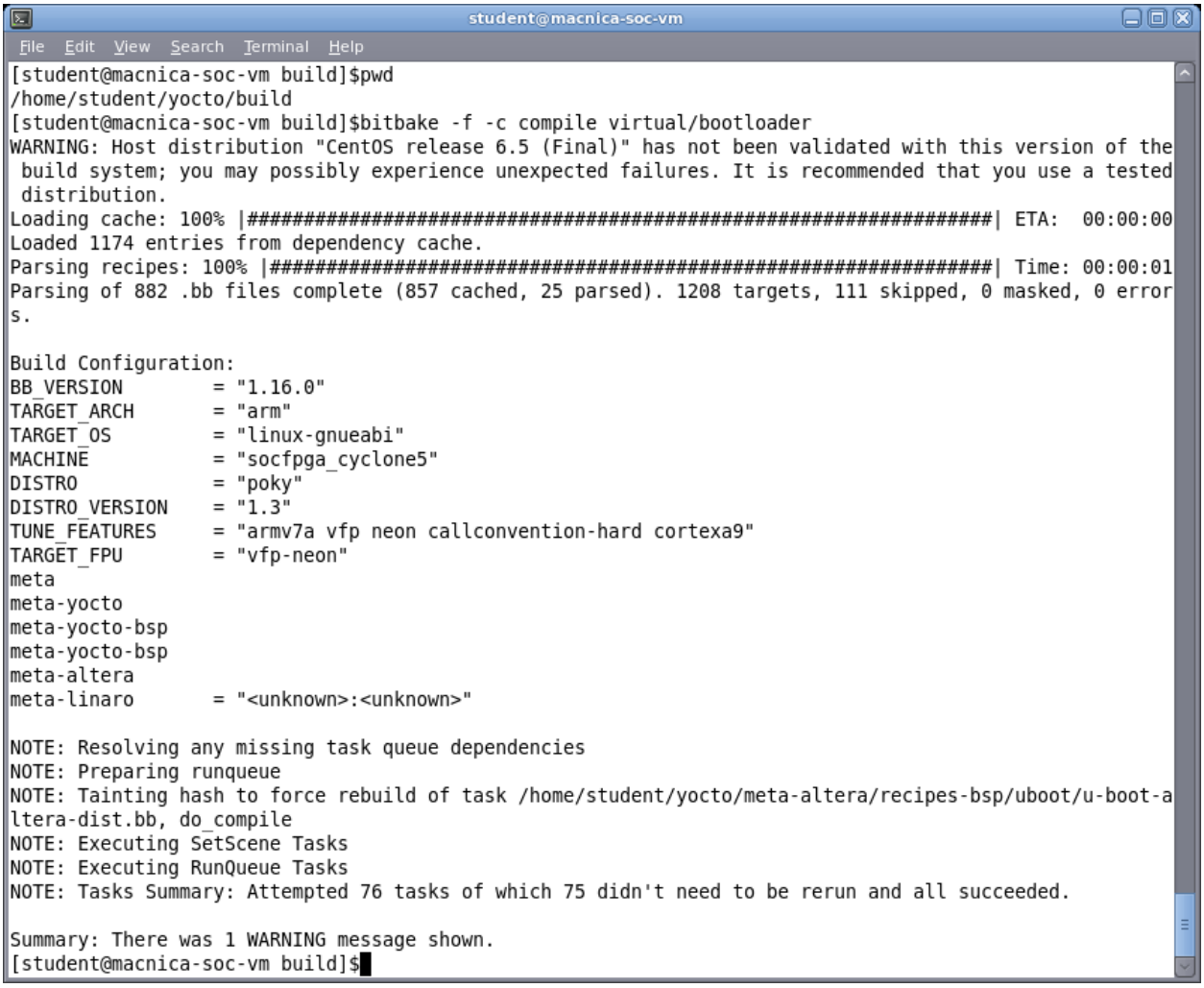
- ☐ Return to the root of the current build and "force" bitbake to recompile the U-Boot target and redeploy the output. The resulting updated U-Boot image will be **u-boot-socfpga_cyclone5-1.0-r0.img**

- ☐ Copy the updated bootloader image to the microSD card above the four preloader images in the 0xA2 partition.

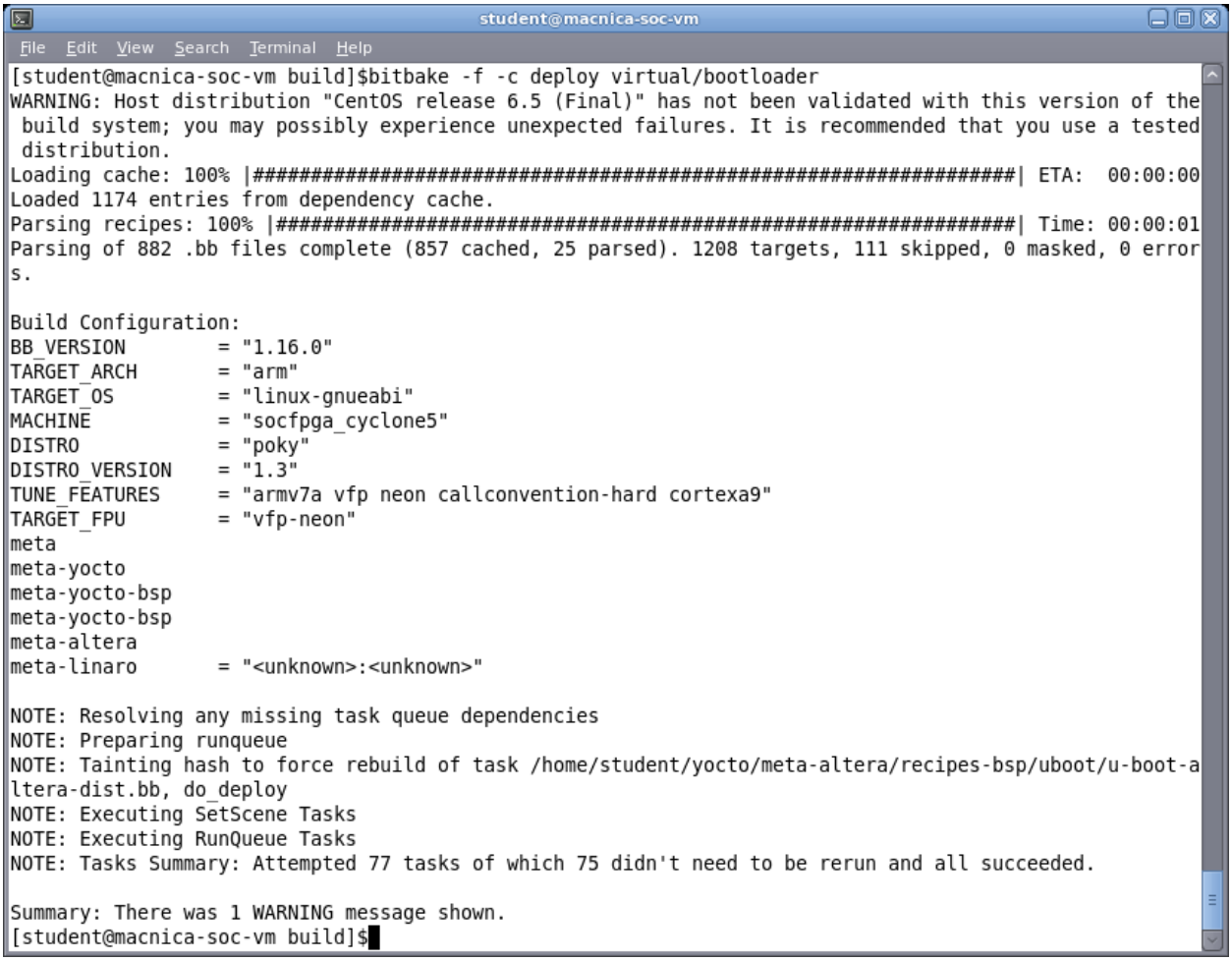
- ☐ Observe the updated U-Boot bootloader command options that were just added. Toggle the LEDs.

Hints:

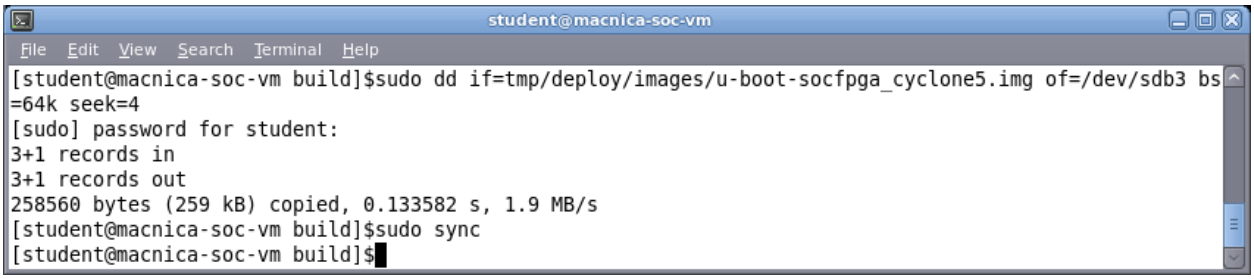
- *Change directory to the root of the current build*
  - *cd ~/yocto/build*
- *Force the recompile of the source for U-Boot*
  - *bitbake -f -c compile virtual/bootloader*



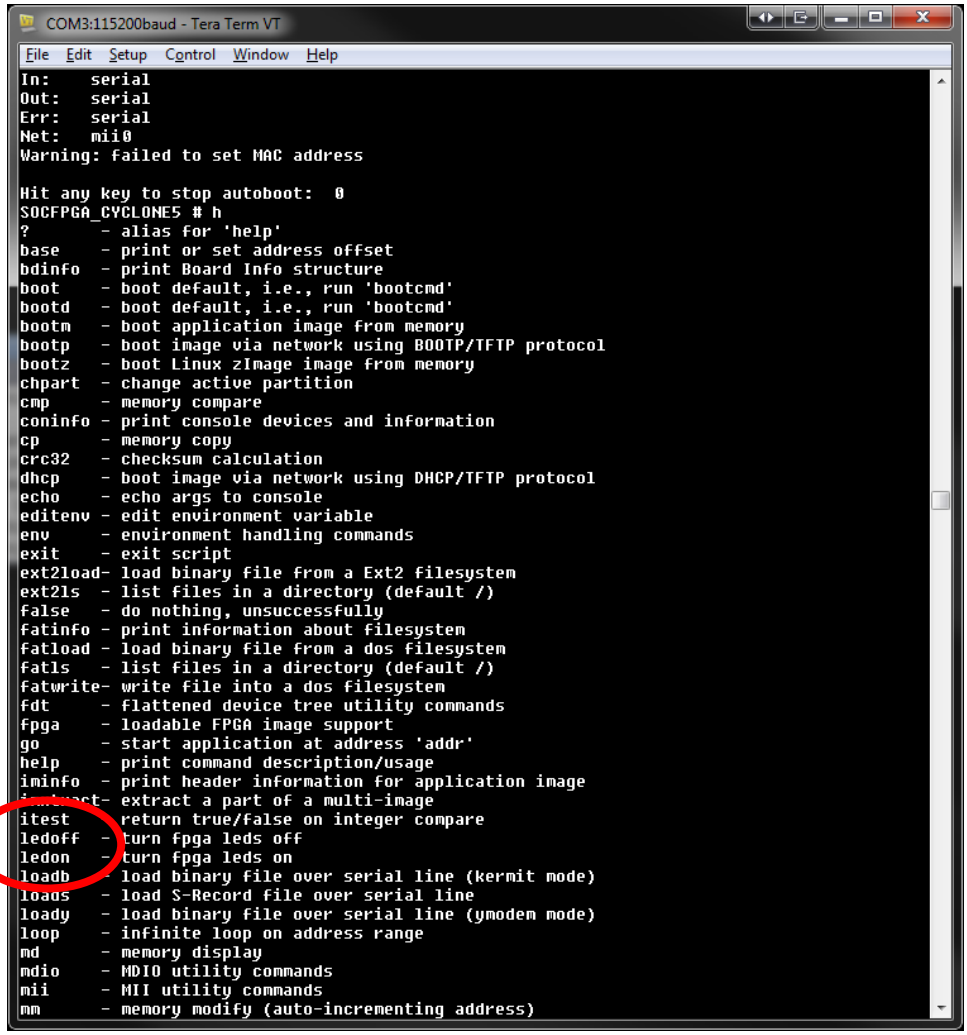
- *Force the redeployment of the updated U-Boot bootloader*
  - *bitbake -f -c deploy virtual/bootloader*



- *Insert the microSD card into the host PC.*
- *Copy the updated bootloader image to the correct partition of the microSD card:*
  sudo dd if=u-boot-socfpga_cyclone5.img of=/dev/sdb3 bs=64k seek=4



- *Reinsert the microSD card into the Helio board; reset the HPS by pressing the cold reset (SW5). Interrupt the bootloader before the 5 second timeout to gain access to the U-Boot menu system.*
- *Enter 'h' to see a list of available commands.*



- *Simply enter 'ledon' and 'ledoff' to toggle the LEDs.*

Bootloader questions

☐ 1. If your CPU reset, why? Why did your first attempt to turn the LEDs off cause a "data abort" and reset the CPU?

_____

☐ 2. How can you keep this from happening?

_____

Linux build answers:

1. **When the SoC powers-up, the bridges between the HPS and the FPGA fabric are disabled. The attempt to communicate over the bridge causes the data failure.**
2. **After interrupting the bootloader, you can use the built-in environment command "run bridge_enable_handoff" to turn the bridges on.**

3 Notes

Document Revision History

Revision	Date	Comments
0.1	May 8, 2013	Initial Draft
0.2	May 23, 2013	Internal Review
1.0	May 27, 2013	Initial Release
2.0	March 20, 2014	