Embedded Linux Getting Started Guide

Document Revision History

Date	Version	Description
March 8, 2013	1.0	Released

Contents

Overview

Introduction

This document walks through the basic software flow to have a “Hello World” Linux application running on the Cortex-A9 processors in the Cyclone V SoC FPGA development kit. This guide focuses purely on getting a basic Linux application running and has no interaction with programmable logic (FPGA) portion of SoC FPGA. Hence, to simplify things, we use:

- No FPGA design
- Pre-built bootloader/Linux software images for the Cyclone V SoC FPGA development kit

For topics such as hardware flow for custom logic and preloader customization, please see other documentations such as “Golden System Reference Design User Guide”.

Release Contents and Location

Altera provides Linux BSP support for the Cyclone V SoC FPGA Development Kit, and provides the following:

- Linux kernel 3.7
- Preloader
- u-boot version 2012.10
- Yocto version 'Danny'
- The packages for the root file system.
- The tool chain (Linaro-GCC, v4.7)

Yocto is used to build the sources of the kernel, the u-boot and the root file system. There are many source code packages available under the Yocto project. Should you enable a package that is not provided with our BSP, it will be downloaded (SVN, GIT, etc). If you are behind a proxy, you need to make sure the network configuration of your Linux host is ready.

The Linux BSP release is composed of three packages: documentation, sources and binaries.

Package Name	Contents
linux-socfpga-13.02-src.bsx	Source code (self extracting)
linux-socfpga-13.02-bin.tar.gz	Binaries
Embedded Linux Getting Started Guide (this doc)	Documentation
Linux BSP User Manual - 13.02.pdf	
Linux BSP Release Notes - 13.02.pdf	
Yocto Danny User Manual - 13.02.pdf	

You can find the release at: http://software.altera.com/linux_socfpga.

Prerequisites

SoC EDS 13.0 or above installed from (http://software.altera.com/soceds)

Build Yocto

Host Setup

The recommended development platform is a PC computer with minimum 2GB RAM and 20GB hard drive, with one of the following OS-es installed:

- CentOS 6.3,
- Ubuntu 12.04 LTS.

We have tested the Yocto package with above distributions. Be warned that Ubuntu has been tested with Yocto but is not an officially supported distribution for Altera ACDS. Other distributions may work as well but not guaranteed to be work-free. Generally the latest version is preferred.

CentOS

These are the required packages that need to be installed on a fresh DVD-based installation of CentOS 6.3. If a different installation method was used (e.g. from a CD) then more packages might be necessary.

```
$ sudo yum update
$ sudo yum groupinstall "Development Tools"
$ sudo yum install texi2html texinfo glibc-devel chrpath
```

If the host machine runs the 64bit version of the OS, then the following additional packages need to be installed:

```
$ sudo yum install glibc.i686 libgcc.i686 libstdc++.i686 glibc-
devel.i686 ncurses-libs.i686 zlib.i686
```

Ubuntu

These are the required packages that need to be installed on a fresh DVD-based installation of Ubuntu 12.04 LTS.

```
$ sudo apt-get update
$ sudo apt-get upgrade
$ sudo apt-get install sed wget cvs subversion git-core
coreutils unzip texi2html texinfo libsdl1.2-dev docbook-utils
gawk python-pysqlite2 diffstat help2man make gcc build-essential
g++ desktop-file-utils chrpath libgl1-mesa-dev libglu1-mesa-dev
mercurial autoconf automake groff libtool xterm
$ sudo apt-get install uboot-mkimage
```

If a different installation method was used (e.g from a CD) then more packages might be necessary.

If the host machine runs the 64bit version of the OS, then the following additional packages need to be installed:

```
$ sudo apt-get install ia32-libs
```

Setup Yocto

1. Download the linux-socfpga-13.02-src.bsx source package.
2. Install the package. It should be installed in a publicly accessible location, as this step can be shared by all users on the system (or on the filesystem if your company is using a network share). The default install location is /opt/altera-linux. However, if you wish to use this location, you will likely need root access in order to access this directory.

```
$ <path_to_downloaded_file>/linux-socfpga-13.02-src.bsx
/opt/altera-linux
```

3. 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.

```
$ /opt/altera-linux/bin/install_altera_socfpga_src.sh ~/yocto-
13.02
```

4. Create a build directory. By keeping this separate from your yocto source you can erase your 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. The script serves 2 purposes. First, it creates the new build directory using Altera's default configuration. Secondly, it sets some shell variables that are required for building. If you start a new shell you will need to run these commands to set these shell variables again.

```
$ cd ~/yocto-13.02/
$ source altera-init ~/yocto-13.02/build
```

Build kernel/rootfs/u-boot

In order to build u-boot from within the build directory:

```
$ bitbake u-boot
```

In order to build linux from within the build directory:

```
$ bitbake linux-altera
```

In order to build the root filesystem:

```
$ bitbake altera-image
```

The first time may take up to several hours depending on your host machine. Once finished, all images should be generated in ~/build/tmp/deploy/images.

Programming Flash

To boot the linux images on SoC FPGA development kit, you need to write the images you just built with Yocto into one of the three Flash devices: SDMMC, NAND and QSPI. For this guide, we will use SDMMC due to its easy detachability. For SDMMC boot, all boot images will be located inside SD/MMC card.

A script is provided with the release that will create an SD card image, ready to be deployed.

Requirements

The script relies on a tool, named **mkpimage**, which creates the correct preloader image that the SoCFPGA Boot ROM accepts. This tool is provided with the SoCEDS release and must be on your search path (PATH). To find out, run:

```
$ which mkpimage
```

The SoC EDS toolset provides the **embedded_command_shell.sh** script that sets all necessary PATH entries for the included tools. Please run it if the **mkpimage** is not in the PATH.

The SD image script needs to be run with root privileges, using **sudo** command. By default, when **sudo** is invoked, it uses a default PATH variable, which may not have the SoC EDS PATH entries. In this case the tool would fail reporting it could not find the **mkpimage** tool. In order to avoid this problem, please instruct sudo to preserve the current PATH, by executing the following command:

```
$ sudo viso
```

and removing resetting environment and adding the PATH to the list of environment variables to be kept:

```
#Defaults env_reset
Defaults  env_keep = "... PATH"
```

Alternatively the sudo can be instructed from command line to keep the current path:

```
$ sudo PAT=$PATH …
```

Usage

The provided tool, named **make_sdimage.sh**, will create a 2GB SD card image, with three partitions:

- p1, being a FAT partition when the kernel and the device tree are located,
- p2, the Linux root file system, as an ext3 partition,

- p3, the partition used by the SoCFPGA ROM to load the preloader. The same partition is used by the preloader to load the u-boot image.

Here's how the script is used:

```
$ sudo make_sdimage.sh \
        -k uImage,socfpga.dtb \
        -p u-boot-spl-socfpga_cyclone5.bin \
        -b u-boot-socfpga_cyclone5.img \
        -r fs \
        -o sd_image.bin
```

Where:

-k accepts a comma separated list of files. Here, we show the kernel and the device tree blob.
-p the preloader raw binary, as generated by Yocto or the U-Boot Makefile
-b the bootloader image, as generated by Yocto or the U-Boot Makefile
-r the directory where the file system is located.
-o the image name

In case you need help, please run:

```
$ make_sdimage.sh -h.
```

The following presents a complete script usage:

```
$ cd ~/yocto/build/tmp/deploy/images
$ sudo /opt/altera-linux/bin/make_sdimage.sh \
     -k uImage,socfpga_cyclone5.dtb \
     -p u-boot-spl-socfpga_cyclone5.bin \
     -b u-boot-socfpga_cyclone5.img \
     -r ~/yocto/build/tmp/work/socfpga_cyclone5-poky-linux-
gnueabi/altera-image-1.0-r0/rootfs \
     -o sd_image.bin
```

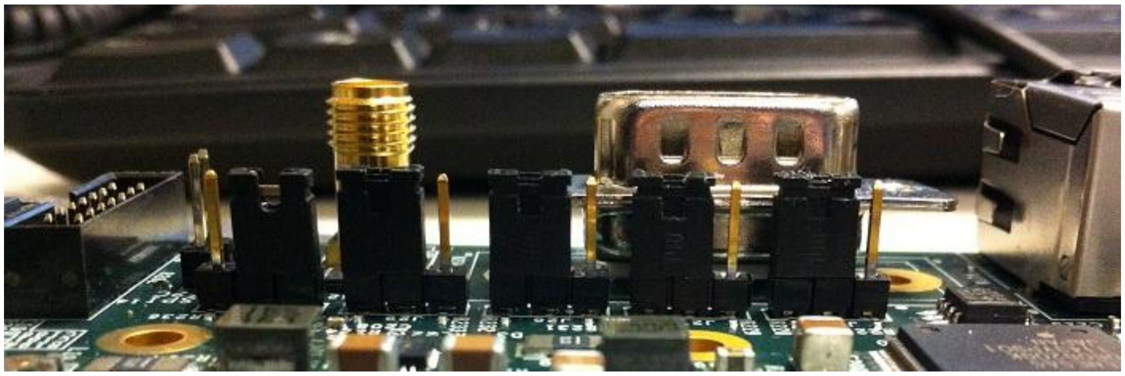
The log messages will be similar with the following:

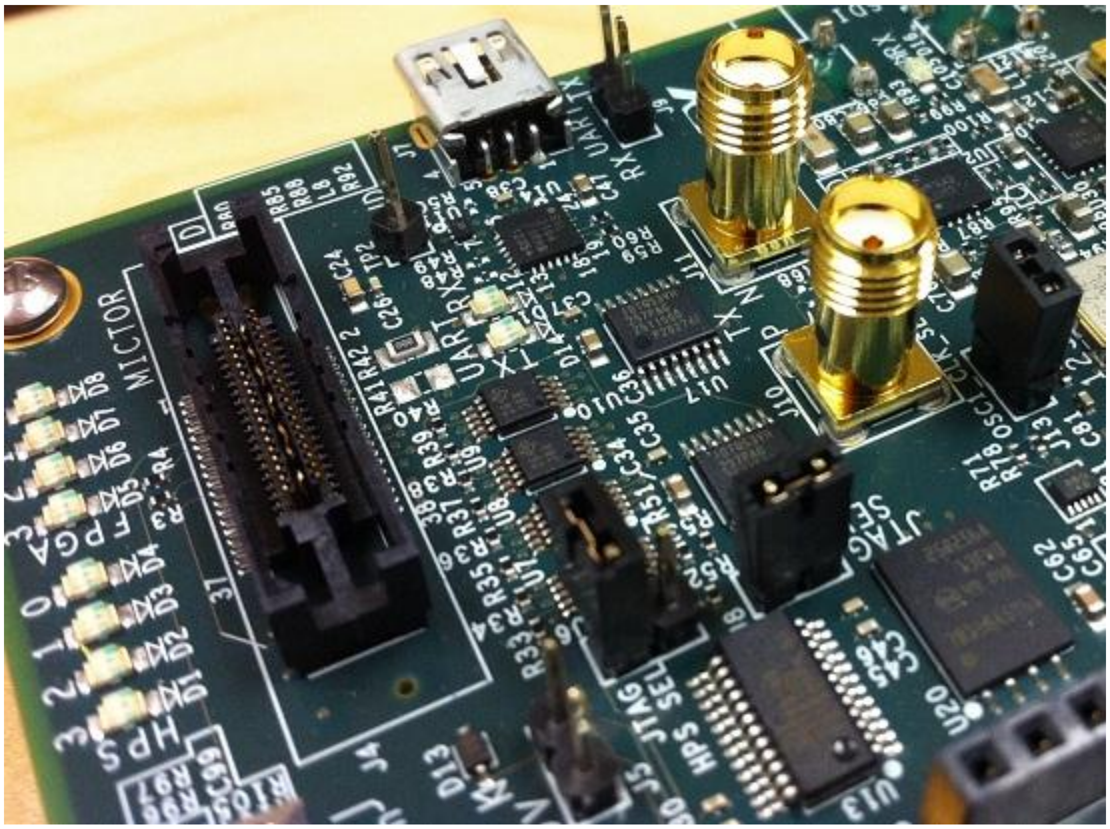
```
make_sdimage.sh: info: creating image file...
make_sdimage.sh: info: creating partition table...
make_sdimage.sh: info: clean up...
make_sdimage.sh: info: creating preloader/bootloader image...
make_preloader_img.sh: info: using preloader
/home/dumitru/yocto/build/tmp/deploy/images/u-boot-spl-
socfpga_cyclone5.bin
make_preloader_img.sh: info: clean up...
make_preloader_img.sh: info: done.
make_sdimage.sh: info: copying preloader image and bootloader to
partition...
make_sdimage.sh: info: copying OS files, etc...
make_sdimage.sh: info: creating root file system...
make_sdimage.sh: info: cleaning up (rfs)...
```

Booting Linux on the SoC FPGA CV Devkit

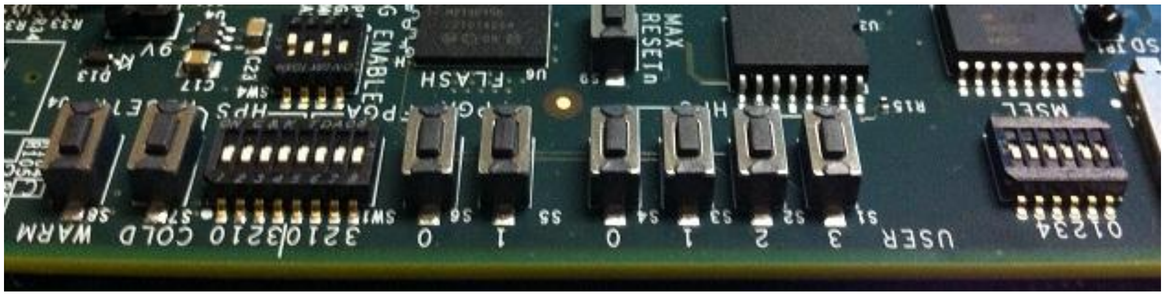
1. Make sure that following shunts/shorting jumpers are installed as described below. Pictures are shown for clarity here as well.
    - Clock select CLKSELx:
        - J26, J27: set toward the power switch
    - Boot select BOOTSELx:
        - J28, J29: set toward the power switch
        - J30: set away from the power switch
    - Rest of jumper settings:

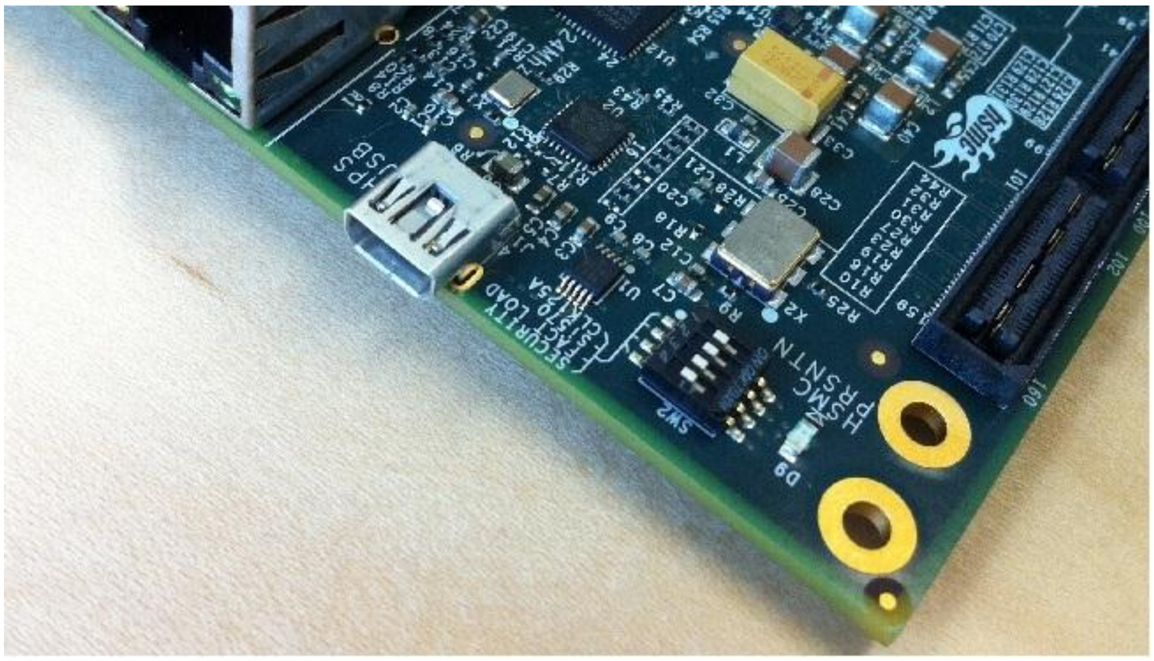
Number	Name	Setting
J5	9V	Open
J6	JTAG_HPS_SEL	Open
J8	JTAG_SEL	Shorted
J9	UART Signals	Open
J13	OSC1_CLK_SEL	Shorted
J15	JTAG_MIC_SEL	Open
J31	SPI_I2C	Open



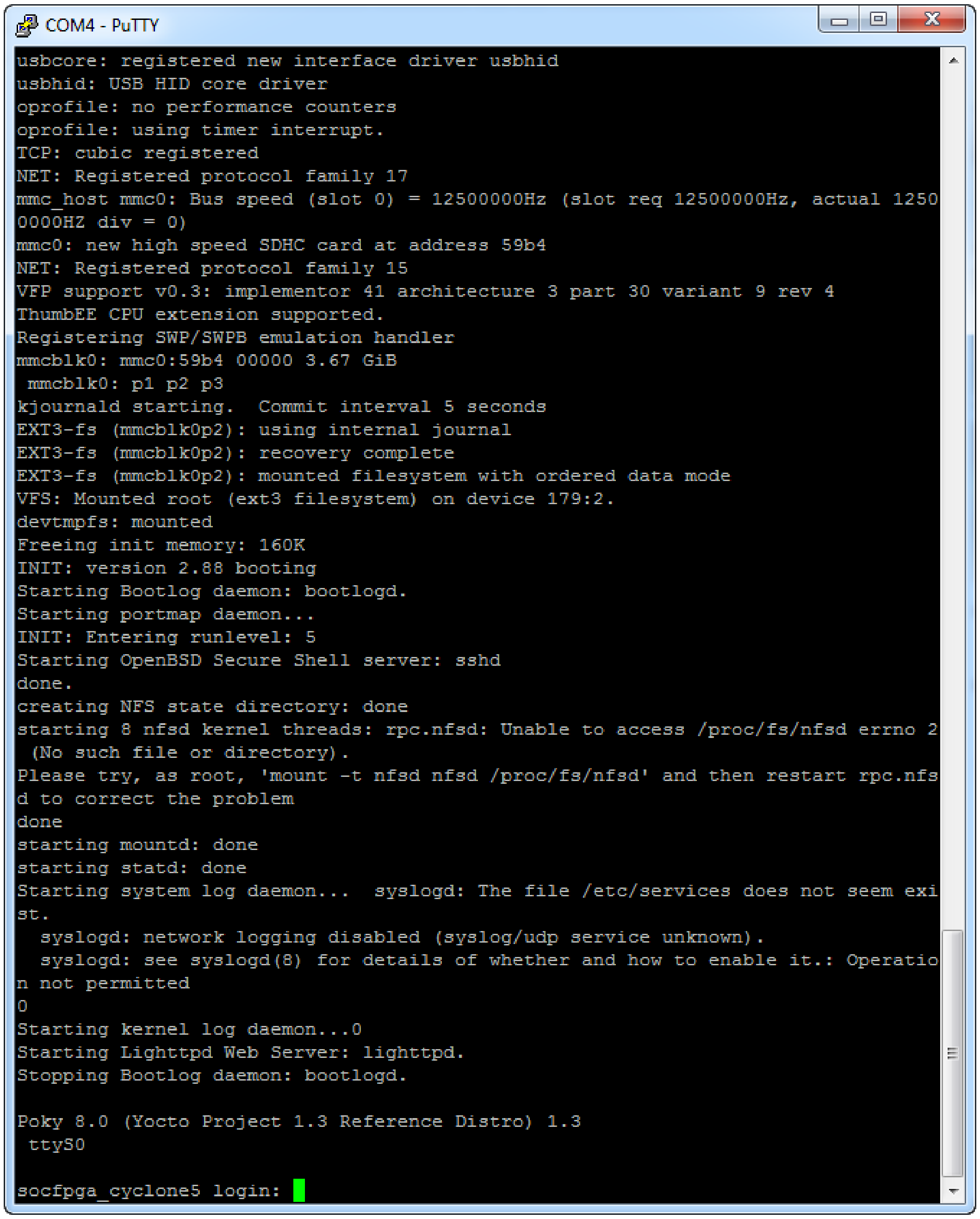


2. Make sure the DIP switches are configured as described below and shown in the following pictures:
   - SW1 = all switches set toward the board edge.
   - SW2 = all switches set away from the corner of the board.
   - SW3 = all switches set toward the board edge.
   - SW4 = JTAG ENABLE.
     - Each switch enables a connection to the scan chain when its corresponding switch is set away from the board edge (off).
     - Set for programming the FPGA using the on-board USB Blaster II = ON/OFF/ON/OFF, leaving the FPGA and MAX connected to JTAG.





3. Use a mini-USB to USB cable to connect “UART” on the board to the host PC (May need to install Cypress UART-to-USB driver)
4. Power on the board (19V power supply!)
5. Open a serial terminal program (i.e. minicom) and set the baudrate to 57600/8-N-1
6. With the microSD card slotted in, observe Linux booting on the UART console.


COM4 - PuTTY

```
usbcore: registered new interface driver usbhid
usbhid: USB HID core driver
oprofile: no performance counters
oprofile: using timer interrupt.
TCP: cubic registered
NET: Registered protocol family 17
mmc_host mmc0: Bus speed (slot 0) = 12500000Hz (slot req 12500000Hz, actual 1250
0000HZ div = 0)
mmc0: new high speed SDHC card at address 59b4
NET: Registered protocol family 15
VFP support v0.3: implementor 41 architecture 3 part 30 variant 9 rev 4
ThumbEE CPU extension supported.
Registering SWP/SWPB emulation handler
mmcblk0: mmc0:59b4 00000 3.67 GiB
 mmcblk0: p1 p2 p3
kjournald starting.  Commit interval 5 seconds
EXT3-fs (mmcblk0p2): using internal journal
EXT3-fs (mmcblk0p2): recovery complete
EXT3-fs (mmcblk0p2): mounted filesystem with ordered data mode
VFS: Mounted root (ext3 filesystem) on device 179:2.
devtmpfs: mounted
Freeing init memory: 160K
INIT: version 2.88 booting
Starting Bootlog daemon: bootlogd.
Starting portmap daemon...
INIT: Entering runlevel: 5
Starting OpenBSD Secure Shell server: sshd
done.
creating NFS state directory: done
starting 8 nfsd kernel threads: rpc.nfsd: Unable to access /proc/fs/nfsd errno 2
 (No such file or directory).
Please try, as root, 'mount -t nfsd nfsd /proc/fs/nfsd' and then restart rpc.nfs
d to correct the problem
done
starting mountd: done
starting statd: done
Starting system log daemon...  syslogd: The file /etc/services does not seem exi
st.
  syslogd: network logging disabled (syslog/udp service unknown).
  syslogd: see syslogd(8) for details of whether and how to enable it.: Operatio
n not permitted
0
Starting kernel log daemon...0
Starting Lighttpd Web Server: lighttpd.
Stopping Bootlog daemon: bootlogd.

Poky 8.0 (Yocto Project 1.3 Reference Distro) 1.3
 ttyS0

socfpga_cyclone5 login: 
```

7. Login as “root” with no password and you are good to go!

“Hello World” Linux application using solo Linaro toolchain

Build “Hello World” Linux application

1. In your host linux machine, write a simple Hello World application called helloworld.c

```
#include <stdio.h>

int main(int argc,
void** argv){
    printf(“Hello
World!\n”);
    return 0;
}
```

2. Set up linux host environment for linaro cross-compiler. Ex

```
% export PATH=<your extracted bsp package location>/linaro/gcc-
linaro-arm-linux-gnueabihf-4.7-2012.11-20121123_linux/bin>:$PATH
```

An example of the command if you installed at the default /opt/altera-linux:

```
% export PATH=/opt/altera-linux/linaro/gcc-linaro-arm-linux-
gnueabihf-4.7-2012.11-20121123_linux/bin:$PATH
```

3. Build helloworld.c with the Linaro cross-compiler

```
% arm-linux-gnueabihf-gcc –o helloworld helloworld.c
```

4. Connect the target board to the same network as your host machine
5. Set up Ethernet interface on the target (if the DHCP server in your network did not automatically configured an IP address)

```
% ifconfig eth0 <static IP address in the same subnet as your
host machine and not used>
```

6. Set up TFTP server on the host and copy helloworld ELF file to the server folder (e.g. /tftpboot)
7. Transfer helloworld ELF to the running eLinux by issuing following command on the target

```
% tftp –g <host IP> -r helloworld
```

8. Run helloworld on the target:

```
% chmod +x helloworld
% ./helloworld
```

Debug "Hello World" Linux application

1. Run GDBSERVER on the helloworld ELF in the target

```
% cd <directory of helloworld ELF>
% /usr/bin/gdbserver host:<port of your choice> ./helloworld
```

2. Run GDB client in the host

```
% cd <directory of helloworld ELF>
% arm-linux-gnueabihf-gdb ./helloworld
```

3. In GDB command line, connect to the target via TCP/IP

```
% target remote <target IP>:<the port you chose>
```

4. Now you can use execute standard GDB operations such as breakpoint and single stepping. Typical commands are:
   - 's' for step into a function
   - 'n' for next instruction
   - 'b' for breakpoint