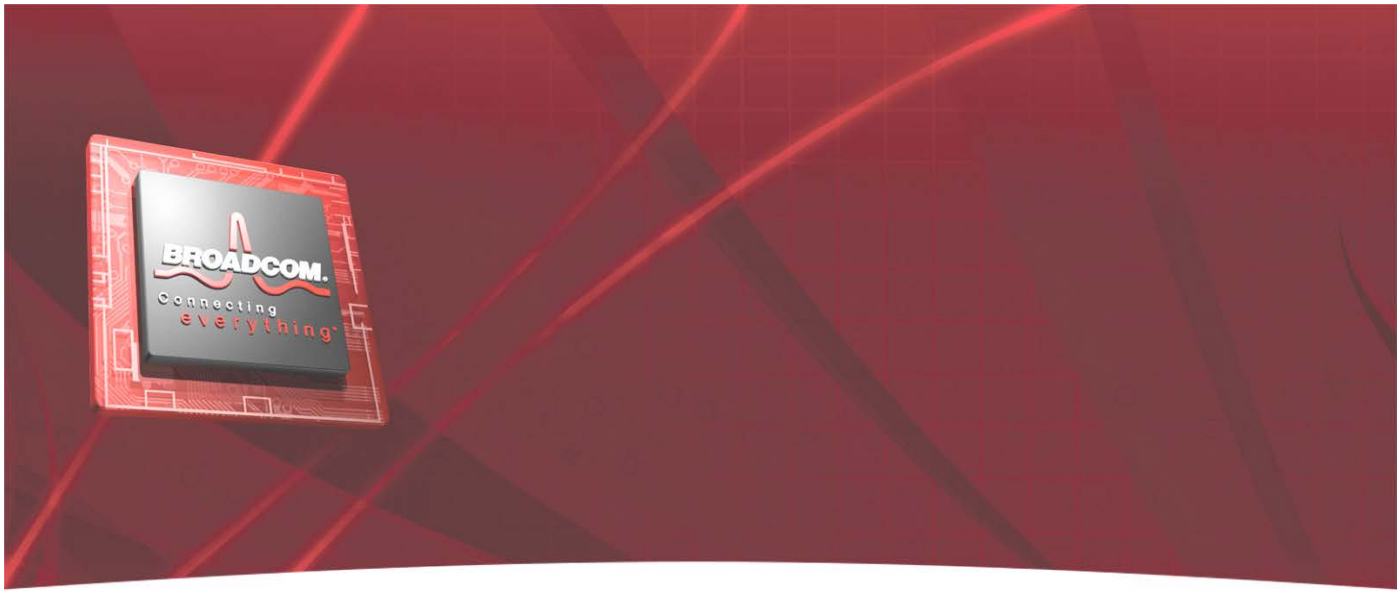

Reference Platform Installation Guide for Linux Systems

Broadcom Corporation
5300 California Avenue
Irvine, California, USA 92617
Phone: 949-926-5000
Fax: 949-926-5203

Web: www.broadcom.com

Revision History

Revision	Date	Change Description
STB_Platform-SWUM100-R	12/26/2008	Initial version
BroadcomReferencePlatformSetup	02/26/2009	Update kernel/rootfs build procedure
BroadcomReferencePlatformSetup	08/23/2011	Update kernel/rootfs build procedure
BroadcomReferencePlatformSetup	09/21/2012	Update kernel/rootfs build procedure
BroadcomReferencePlatformSetup	05/22/2013	General update and remove outdated material
BroadcomReferencePlatformSetup	07/16/2013	Updated for 28 nm platforms with BOLT
BroadcomReferencePlatformSetup	06/02/2015	Updated for Linux 3.14

Table of Contents

Introduction

This document describes how to set up the bootloader and Linux operating system on a Broadcom Set-top reference platform and how to configure a Linux toolchain and kernel source on a build server. Depending on the platform, primary bootloader can be either CFE (Common Firmware Environment) or BOLT (Broadcom Optimized Loader Technology). Newer platforms are shipped with BOLT bootloader. Please check accompanying release notes, to determine applicable bootloader. Other components of the Broadcom Set-Top Reference Software, including the Nexus API and Atlas reference application are documented separately.

Software Terms

The following terms are used in this document:
Bootloader – The code that is first executed when the Reference Board powers up. This is responsible for initial configuration of the hardware and loading the kernel.
Kernel – The operating system (OS) which controls the Reference Board and allows the Reference Software drivers and applications to run
Root File System – A file system containing general-purpose utilities and libraries which, along with the kernel, provide standard OS services to applications.
Toolchain – The compiler and linker that run on the Build Server and create binaries that can run on the Reference Board.

Prerequisites

The term “Reference Software” refers to the source and/or executable code delivered by Broadcom to control the features of the Broadcom Reference Platform
You must have the following to set up the Reference Platform:

- A Broadcom reference board
  - The Reference Software is not guaranteed to run on other systems. It may require significant modification.
- A PC on which you can install Linux for building Reference Software and other programs (the build server)
  - The build server can be a dedicated Linux machine, a dual-boot Linux/Windows machine, or an installation of Linux in a virtual machine.
  - See below for Linux version details.
  - You can install binaries with only a Windows PC and a TFTP server. This is documented in Configuring a Reference Board with a Windows PC.
  - These instructions only apply to Linux platforms.
- Knowledge of the “vi” UNIX editor
  - You must be able to edit text on the reference board.
- Root (or sudo) access to your Linux build server may be needed during configuration
- Connection of the Reference Board and build server to the same Ethernet network
- DHCP server running on the network that can be used by the Reference Board
- A bash shell on the build server
  - If you are not sure, type asdf and press Enter. You should see: bash: asdf: command not found. If it does not say bash: at the front, just run bash and try again.
- Release notes
  - Translate the following pseudo filenames to the actual names specified in the release notes:
- kernel_source.tgz
- toolchain_binary.tar.bz2
- vmlinuz-xxxx = Linux kernel where xxxx is the chip/rev. The exact form of the filename may vary.
- vmlinuz-initrd-xxxx = initrd Linux kernel.

Build System Configuration

Figure 1: System Configuration Broadcom Platform shows an example system configuration. The types of RF inputs and A/V outputs depend on the platform.

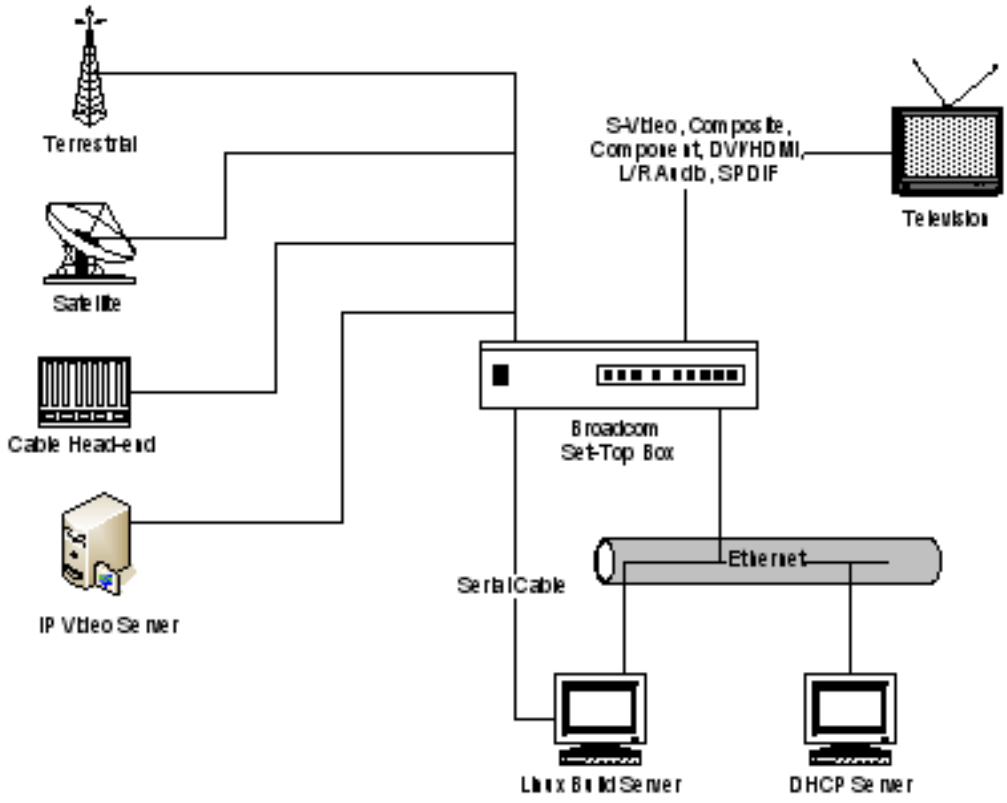

Figure 1: System Configuration Broadcom Platform

Unpacking the Release File

The reference software release is distributed as a tarball with a name of the form *refsw_release_unified_20130308.tgz,* which encodes the release date as *YYYYMMDD*. "Unified" means that the release supports most current set-top platforms. A release intended to support only a particular platform would have that platform's number in place of "unified". This file should be unpacked using Linux rather than WinZip or another Windows utility.

Move the tarball to a location where you have write permission, and unpack it by entering a command of the form:

```
tar zxvf refsw_release_unified_20130308.tgz
```

After unpacking it, view ReleaseNotes.html in a web browser. These notes describe the various files included in the release, point you to this document and list special issues and known bugs. The release may include platform-specific release notes (named with the platform number) for certain platforms for which additional information might be helpful.

Configuring a Reference Board with a Windows PC

It is possible to configure a Reference Board with only a Windows PC. You will not be able to build from source (which requires a Linux build server) and you will not be able to NFS mount. However, you can still load binaries using TFTP. The PumpKIN TFTP server can be downloaded and installed on a Windows machine, allowing it to serve files to the Reference Board, which can be installed or copied onto the board as described in Configuring the Reference Board and Making the Reference Board Auto-Boot.

Configuring the Linux Build Server and Network

Installing Linux on the Build Server

The build server is used to compile source code using the cross-compiling toolchain. The instructions in this document assume Red Hat Linux or Fedora Core, but Ubuntu can also be used. When you install Linux, use a server configuration, which should install the following tools:

- minicom (or other serial-capable terminal emulator)
  If it is more convenient to use the terminal emulator under Windows, Tera Term can be used.
- TFTP server
- NFS server (strongly recommended)
- FTP server (optional)

You will need root (or sudo) permission in order to configure your build server. Once everything is installed and running, you do not need to build as root.

If your build server does not have a TFTP server, you can do the following:

1. Go to rpmfind.net in a web browser.
2. Search for tftp-server.
3. Download a tftp-server rpm compatible with your Linux version.
4. Transfer the rpm to your Linux box.
5. As root, type a command of the form
   ```
   rpm –i tftp-server-0.28-2.i386.rpm
   ```

6. Restart xinetd to start the tftp server by typing
   ```
   service xinetd reload
   ```

Installing the Toolchain on the Build Server

Install the toolchain on the build server by unpacking the tarball in the standard location and editing your PATH. The specific filename of *toolchain_tarball.tar.gz* will vary. Please follow these steps:

```
mkdir –p /opt/toolchains
```

```
cd /opt/toolchains
```

```
tar jxvf toolchain_tarball.tar.bz2
```

You need to add the toolchain's bin directory to your PATH. The specific technique for doing this may vary depending on your system configuration. One method is to edit /etc/profile as root and add the following lines:

```
PATH=/opt/toolchains/toolchain_dir/bin:$PATH
```

```
export PATH
```

You do not have to put the toolchain at the head of your PATH, but it must be ahead of any other applicable cross-toolchain in your path. The reference software will be built using first mipsel-linux-uclibc or arm-linux toolchain in your PATH.

You will need to re-source this file in order to pick up the PATH change.

```
. /etc/profile
```

Verify you are using the right toolchain by doing the following:

```
which mipsel-linux-uclibc-gcc
```

or

```
which arm-linux-gcc
```

Installing the Kernel Source on the Build Server

The reference software assumes that you will place the kernel source in /opt/brcm/linux. You can specify another location for the kernel source by using the LINUX environment variable. Using LINUX is recommended, since reference software must be built against kernel source configured for the particular chip/platform, so it might be necessary to keep multiple copies of the configured kernel source to support builds for different chips. Another technique is to use a symlink for /opt/brcm/linux and move it to the correct kernel source.

The following instructions use "3.14-1.6" as an example, but you should replace that with whatever file names are appropriate. Be careful to not untar this source into /usr/src/linux, which may be the kernel source for your build server. Note that the root filesystem file and directory are called "uclinux-rootfs" rather than "rootfs" in Linux 3.8 and earlier.

For example:

```
mkdir -p /opt/brcm/linux-3.14-1.6
cd linux-3.14-1.6
tar jxvf stblinux-3.14-1.6.tar.bz2
tar jxvf rootfs-3.14-1.6.tar.bz2
cd /opt/brcm
ln -sf linux-3.14-1.6/linux linux
cd linux-3.14-1.6/rootfs
make images-7445d0
cd ../linux
export LINUX=`pwd`
```

You must build drivers with the same kernel source from which your kernel binary is built. Kernel module versioning will likely prevent drivers from loading if they have been built against mismatching kernel source.
It is required that the Linux kernel source be properly configured for a reference board platform before building a module. The safest way to do this is to build the kernel for your reference board platform as described above. The example above will build the root file system binary images in addition to configuring the kernel source and building a kernel image. If you plan to use the pre-built Linux binary images bundled with the release, it might be quicker to substitute a command like

```
make vmlinuz-7445d0
```

for

```
make images-7445d0
```

Please refer to the Release Notes for the latest information. It is important that you pick the right build target for your platform. You can use the naming of the pre-built kernel binaries bundled with the release as an example.

```
make help
```

will show available make options. The list of platform TARGETs is in [uclinux-]rootfs/misc/release_builds.

Configuring the TFTP Server

The bootloader retrieves kernel binaries from the build server using TFTP. This means that you will need a TFTP server on your build server. On Linux, TFTP is usually configured as part of the xinetd service. You should have the file /etc/xinetd.d/tftp with contents similar to these:

```
service tftp
{
      disable            = no
      socket_type        = dgram
      protocol           = udp
      wait               = yes
      user               = root
      server             = /usr/sbin/in.tftpd
      server_args        = -s /tftpboot
      per_source         = 11
      csp                = 100 2
}
```

Make sure you are using `/tftpboot` and `disable = no`. After configuring this file, you will need to start or restart xinetd as root:

```
service xinetd restart
```

This example makes the /tftpboot directory your root for tftp. If the directory does not exist yet, create the /tftpboot directory. You should put the following binary files from your release bundle into that directory:

- kernel binaries (for example, vmlinuz-xxxx and vmlinuz-initrd-xxxx)
- root file system binary (for example, jffs2-xxxx.img)
- bootloader binary (for example, cfe_xxxx_le.bin)

Configuring the NFS Server

It is typically most convenient to load and run the Reference Software drivers, libraries and applications on the Reference Board from an NFS mount.

Edit /etc/exports on your build server to add any desired mount points. If the directories do not exist yet, you should create them prior to restarting the NFS server. Here's an example which exports a directory named /opt/refsw for mounting by the set-top:

```
/opt/refsw *(rw,no_root_squash,no_all_squash)
```

To activate the export of the mount point by the server, stop and start the NFS server with root permissions.

Note: We've found stop and start to be more reliable than "restart".

```
service nfs stop
service nfs start
```

Note: If this is the first time you have started the NFS server, the stop command may fail. This is normal. Continue and execute the start command. This should not fail.

You can confirm that your NFS server is running by doing a loopback mount from the build server itself. You will need to do this with root permissions:

```
mkdir /mnt
mount localhost:/opt/nfsroot_uclibc /mnt
ls /mnt
umount /mnt
```

Note: You may already have a /mnt directory on your system (cdrom, floppy etc.) If this is the case, just skip the mkdir command.

Your NFS server may require DNS lookup before allowing a client to mount. If you get permissions errors when mounting from the reference board (error number -13), you may need to edit /etc/hosts on the build server and add in an entry for your reference board. Refer to this example:

```
# This is the IP address and hostname.domainname
# of the reference board
10.6.0.50   board1.myhouse   board1
```

Using an FTP Server

These instructions do not require an FTP server. NFS is sufficient for making file transfers. However, it is sometimes easier to get a file to the reference board using FTP (especially if you have a Windows machine). The uclibc root filesystem contains two FTP client applications you can use: ftpput and ftpget. It does not contain an interactive general-purpose FTP client.

This is the procedure for using ftpget:

```
# Print usage information
ftpget

# Using anonymous ftp
ftpget servername localfile remotefile

# Using a login
ftpget –u username –p password servername localfile remotefile
```

The local file name will be relative to your path on the reference board. The remote file name will be relative to wherever you login. Because it's not interactive, you might want to use a non-

reference board ftp client to determine exactly where the file is, and then use ftpget on the reference board.

Configuring the terminal emulator

The build server controls the CFE bootloader through a serial interface. Configure a terminal emulator such as minicom to talk with the reference board using the settings shown in Table 1.

Settings	Value
Serial Port	/dev/ttyS0
Baud rate	115200
Data bits, parity and stop bit	8N1
Hardware Flow Control	No

Table 1: Reference Board Serial Settings

Here are some quick directions for minicom:

1. Enter **`minicom -s`**
2. Select “Serial port setup.”
3. Set the values given in Table 1: Reference Board Serial Settings
4. Press **Esc** to exit this popup.
5. Select “Save setup as dfl.”
6. Select “Exit.”

If you want to run the kernel installation scripts as a normal user, you may have to run the following as root first:

```
chmod a+rw /dev/ttyS[01]
cp /usr/sbin/chat /usr/bin/chat
```

A terminal emulator can also be used on a Windows PC. Tera Term is a good choice.

Getting the IP Address of your Build Server

You will need the IP address of your build server. Use the command ifconfig to determine the IP address. Be aware that ifconfig is located in the /sbin directory, so you might have to use the command /sbin/ifconfig.

If your build server is using DHCP, this address might change, and you may need to modify your reference board CFE scripts. A static IP address for the build server is recommended.

Disabling the Build Server's Firewall

If you are running newer versions of Linux, such as Fedora, it may have the firewall enabled by default. This will block tftp and other services. Disable this firewall.
Go to the lower-left corner **Start** menu, select "System Settings", select "Security Level", and then disable the firewall.

Configuring a DHCP Server

Obtaining IP addresses with DHCP is very convenient. It is possible to avoid DHCP and use only static IP addresses, but these instructions assume you have a DHCP server on your network.

Configuring your own DHCP server can also be very hazardous. Unlike all other configuration steps documented here, the DHCP server can cause other people on the network to start having problems. Therefore if your network already has a DHCP server, you should use it and not configure your own. If you know that you do not have a DHCP server, you can use an inexpensive router that has DHCP support built-in.

Configuring the Reference Board

Connecting the Reference Board

For now, you need the following connections to your reference board:

- An Ethernet cable connected to a network that can route to your Build Server
  - For most platforms, the Broadcom set-top chip has an Ethernet port connected to the set-top "host" CPU. Some other platforms, such as the single-chip set-top/cable modem BCM7125 family, may lack a "host" Ethernet port, and will require a USB Ethernet dongle.
- A serial cable that is connected from COM1 (UART A or 0) on your reference board to your Build Server or PC. This should be a regular serial cable, not a NULL modem cable.

Later on, you will need to connect the RF input and the A/V outputs.

Verifying the Bootloader

All reference boards should be delivered with the either CFE or BOLT bootloader already installed in flash. Connect via COM1 to the reference board using minicom or other terminal emulator program. When you boot, you should see a bootloader prompt (CFE> or BOLT>) after resetting the box and the front panel LEDs should light up.

If Bootloader is not working:

- Make sure that COM1 on the outside of the reference board enclosure is connected to UART A on the board.
- Make sure your terminal program is talking to the same COM port that you plugged into on your Build Server.

If that fails and you have another known-good Broadcom reference board, try connecting it for comparison. Broadcom Set-Top reference boards all use the same serial port settings.

You can use the Windows application "Broadband Studio" with an I^2C-to-parallel or I^2C-to-USB connector to Flash the Bootloader. Broadband Studio can be requested from your FAE.

General Bootloader Help

Before we start booting and flashing kernels, please refer to this overview of CFE/BOLT commands:

Getting help on Bootloader commands

```
CFE/BOLT>help
```

Booting a compressed kernel from a TFTP server

```
CFE>boot –z -elf IPADDRESS:KERNEL
```

```
BOLT>boot IPADDRESS:KERNEL (note: -z –elf not required for BOLT based platforms)
```

Showing the flash partitions which are available (along with other devices)

```
CFE/BOLT>show devices
```

Copying a kernel into a flash partition

```
CFE/BOLT>flash -noheader IPADDRESS:KERNEL FLASHPARTITON
```

Booting a compressed kernel from a flash partition

```
CFE>boot –z -elf FLASHPARTITION:
```

```
BOLT>boot FLASHPARTITION: (note: -z –elf not required for BOLT based platforms)
```

Auto booting from a flash partition

```
CFE>setenv -p STARTUP 'boot -z -elf FLASHPARTITION:'
```

```
BOLT> setenv -p STARTUP 'boot FLASHPARTITION:'
```

Unsetting auto boot

```
CFE/BOLT>unsetenv STARTUP
```

Assigning a MAC Address

If your platform has an internal Ethernet controller (not USB Ethernet), then you may need to program a MAC address into Flash.
In, you will bring up the "eth0" interface using DHCP with the Bootloader command

```
ifconfig eth0 -auto
```

Bootloader will report back the MAC address. If it reads hwaddr 00-00-00-00-00-00, the board was not programmed with a valid MAC address. You can assign one by using Bootloader's macprog command.
Use
CFE/BOLT>help macprog
For details on using the command.

Flashing the Latest Bootloader

If you have a Bootloader installed you can generally use the old Bootloader to flash the new Bootloader and then boot the new Bootloader from flash. In some cases, the size of the Bootloader partition must be increased, and the new Bootloader can only be flashed using the Broadband Studio software. Unless the Release Notes warn otherwise, you can flash a new Bootloader from an existing one using these steps.

Place the new Bootloader binary into the /tftpboot directory, then run the following on your Reference Board:

```
CFE/BOLT>ifconfig eth0 –auto
CFE>flash –noheader y.y.y.y:cfe_xxxx.bin flash0.cfe or for BOLT platforms
BOLT>flash –noheader y.y.y.y:bolt.bin flash0.bolt
```

where `y.y.y.y` is the IP address of your Build Server. The file cfe_xxxx_le.bin/bolt.bin should be replaced by the actual filename of the new Bootloader in the /tftpboot directory of your build server.

Verify that jumpers are set for the right endianness. Refer to the hardware documentation if required.

Reboot your reference board and it should boot from Flash. You can verify that version information printed by the Bootloader matches the version listed in the release notes.

If this technique does not work, you can flash new Bootloader and/or erase the flash part(s) using Broadband Studio. Please contact your FAE to request a release of Broadband Studio (BBS) for your platform.

Building the Kernel and Root File System

This step is optional because release bundle includes binaries for the kernel and root file system. However, as noted above in the section "Installing the Kernel Source on the Build Server", the kernel source must be configured for the platform, and building it at least once is the easiest way to configure the source. If you want to customize or minimize the kernel and root file system, you will need to rebuild from source. When using binaries, you must keep the following in sync:

- The root file system binary must be matched with the kernel binary

- The toolchain used to compile applications must match the root file system binary
- The toolchain used to compile applications must match the toolchain used to compile drivers
- The kernel source used to compile drivers must match the kernel binary

When building from source, you have more flexibility, but you should be familiar with the Linux operating system before making code or configuration changes. You can always return to the released binaries as a baseline.

First make sure the kernel source has been installed on the build server as described in the section "Installing the Kernel Source on the Build Server".

Create a destination directory for the binaries. This is usually a tftp server directory, because you will be using tftp to load the files. Make sure you have write permission to this directory. To be safe, you may want to use the command `chmod o+w`, which would give others write permission. For example,

```
mkdir /tftpboot
chmod o+w /tftpboot
```

Next, you will need to select a platform string. This is different from the PLATFORM variable used later in this document to build the reference software. Check the misc/release_builds in the [uclinux-]rootfs directory for a list of platforms supported by the Linux release.

Check the Reference Software Release Notes. Some chips share kernel platforms with closely related ones. For example, the Bx revisions of 7241, 7428, and 7429 all share the 7429b0 kernel platform.

Once you have a writable /tftpboot directory and have selected the correct platform string, the build instructions are as follows:

```
cd rootfs
make TFTPDIR=<tftpdir> <TARGET>
```

Use

```
make help
```

for a list of <TARGET>s

```
Examples:
  make images-7445d0       - Build all images for 7445d0
  make vmlinuz-7445d0      - Build non-initrd kernel for 7445d0
```

Supported builds

Builds are per-chip, rather than per-board. Board differences are handled at runtime, as are minor chip variations (e.g. 7428 vs. 7429).

SMP is always enabled on chips that support it. To boot with SMP disabled, pass the "nosmp" option on the kernel command line. SMTC is not supported. UP builds can still be made by changing the kernel configuration, but UP binaries will not be included in the release.

NOR/NAND/SPI drivers are built into all kernels (as long as the chip supports it). The flash configuration is detected at runtime. There are no longer separate -nand builds.

Separate configuration files are no longer maintained for initramfs/non-initramfs or LE/BE. config.pl (see below) modifies the base configuration on the fly to set the appropriate options.
There are several "variant" builds supported at the time of this writing:

- -android: Enable Android
- -lxc: Enable LXC containers
- -ivc: USB Video Class
- -xfs: Enable XFS file system
- -perf: Performance counters and function tracer
- -kgdb: KGDB kernel
- -kdebug: Enable kernel debugging features (spinlock checks, full debug symbols, etc.)
- -opf: Oprofile kernel
- -small: Small rootfs image with most features disabled
- -gdb: Enable native gdb debugger on the target
- -netfilter: Enable netfilter and iptables
- -ipv6: Enable IPv6
- -nousb: Disable USB support (host/device drivers)
- -nomtd: Disable MTD (flash) support (drivers, mtd-utils)
- -nohdd: Disable hard disk support (fdisk, e2fsprogs, ATA/SCSI drivers)
- -nonet: Disable networking (drivers, ifconfig, etc.)

These modifiers can be combined, e.g. "images-7405d0-small-nohdd-netfilter" might make sense for an IPTV STB with no USB or SATA hard drive support.
Not all combinations are supported, and all should be considered untested. They are only provided as a starting point. To see what configuration options they are affecting, please refer to [uclinux-]rootfs/bin/config.pl .
Also note that changing the kernel configuration could affect binary compatibility with kernel modules (particularly drivers which may be distributed in binary form).

Basic build system usage (using the default settings)

Below are example commands for the 97445 D0.

Build rootfs, kernels, and flash images for 7445d0:

```
cd rootfs
make images-7445d0
```

Build initramfs (built-in rootfs) and non-initramfs kernels, but no flash images:

```
make kernels-7445d0
```

Build just the initramfs kernel:

```
make vmlinuz-initrd-7445d0
```

Build the kernel only, no rootfs or flash images:

```
make vmlinuz-7445d0
```

All images will be copied to the images/ directory (which are not erased by distclean/clean). To copy them elsewhere:
Install images to /tftpboot/$USER:
```
make install
```

Install to a custom location:
```
make install TFTPDIR=/tftpboot/newbuild
```

The `make install` target will overwrite any pre-existing image(s) in `$TFTPDIR` with the same name as any of the files in `images/` . To avoid this, use separate `$TFTPDIR` directories, or copy the files by hand.

The following are examples of variant builds.
Build a non-initramfs kernel with KGDB enabled:
```
make vmlinuz-7335b0-kgdb
```

Build the Oprofile initramfs kernel:
```
make vmlinuz-initrd-7335b0-opf
```

Build an initramfs kernel with the -small rootfs:
```
make vmlinuz-initrd-7335b0-small
```

Build for big-endian instead of LE:
```
make images-7335b0_be
```

BE variant build:
```
make images-7335b0_be-small-nohdd
```

Note that the uClinux build system does not support "make -j" (parallel builds). If "make -j" is used, the results will be undefined.

Customization

Set up the defaults for 7335b0, but don't build anything yet:
```
make defaults-7335b0
```

OPTIONAL: edit the kernel configuration:
```
make menuconfig-linux
```

OPTIONAL: edit the busybox configuration:
```
make menuconfig-busybox
```

OPTIONAL: edit the uClibc configuration:
```
make menuconfig-uclibc
```

OPTIONAL: edit the vendor configuration (rootfs utilities):
```
make menuconfig-vendor
```

Individual linux/busybox/vendor/uclibc options can also be changed from the command line.
Enable tcpdump and ntfs-3g; disable JFFS2 kernel support:

```
perl -w bin/config.pl vendor CONFIG_USER_TCPDUMP_TCPDUMP=y CONFIG_USER_NTFS_3G=y
perl -w bin/config.pl linux CONFIG_JFFS2_FS=n
```

Fix up dependencies:

```
make oldconfig
```

After customizing the configuration, any of the following items can be built.
(Re)build rootfs + kernels + flash images using the new configuration:

```
make
```

synonym:

```
make images
```

(Re)build rootfs + initramfs kernel:

```
make initrd_kernel
```

(Re)build non-initramfs kernel:

```
make kernel
```

These builds may be rerun multiple times. The variant builds are supported the same way.
Start off with the -small rootfs:

```
make defaults-7335b0-small
```

Edit the busybox configuration to reinstate some missing features

```
make menuconfig-busybox
```

Build all images

```
make
```

Configuring the Kernel and Root File System

Using a USB Ethernet Dongle

Using the reference board requires Ethernet connectivity. Platforms without an internal or on-board Ethernet controller, or platforms such as the BCM7125 family which has an Ethernet controller connected to the internal cable modem, rather than to the set-top host CPU, will require a USB Ethernet dongle. A couple of examples of USB Ethernet dongles that work are D-Link DUB-E100, and TRENDnet TU2-ET100.

Choose a Kernel and Root File System Configuration

There are alternate ways you can configure the Linux kernel and root file system using CFE. The choices include:

Loading the kernel

- From TFTP (for development only)
- From Flash (most stable)

Loading the root file system

- From Flash (most stable)
- From NFS (not documented by, or supported by Broadcom)
- Using an initrd root file system, in which the root file system is compiled into the kernel and loaded into a ramdisk at boot time
- From the hard drive

The following sections document the different root file system configurations. Select the one that meets your needs.

Option #1: Flashing Your Kernel and Root File System

Booting the initrd Kernel

The initrd kernel is a Linux kernel with a RAM disk root file system compiled into it. It allows you to boot without having a root file system and configure a new root file system either in flash or on the hard drive.

Here is an example command to boot the initrd kernel:

```
CFE/BOLT>ifconfig eth0 -auto
```

```
CFE>boot –z -elf y.y.y.y:vmlinuz-initrd-xxxx
```

or

```
BOLT> boot y.y.y.y:vmlinuz-initrd-xxxx
```

Login as root, no password.

where y.y.y.y is the IP address of your Build Server and xxxx is the chip name. The file vmlinuz-initrd-xxxx should already be in the /tftpboot directory of your build server, (or other directory configured to be shared by the TFTP server).

Flashing the root File System

Next, run the stbutil utility on the Reference Board to load the root file system via TFTP to either flash or the hard drive. You can select the option you want for the destination. This example assumes you will write the root file system to Flash.

```
stbutil y.y.y.y:
```

Select option

```
2) Install UBIFS rootfs to flash (RW/RO)
```

where y.y.y.y is the IP address of your Build Server and no directory is specified after the colon because you have placed the root file system binary into /tftpboot. If you placed them in a subdirectory under /tftpboot, add that after the colon.

Stbutil options

The following shows the options of stbutil. If an option is shown as "(not available)" the set-top's hardware configuration may not allow it. For example, a hard disk drive or USB thumb drive may not be connected. Here's an example stbutil menu.

```
# stbutil

stbutil v5.0

------------

Using TFTP server:     stb-bld-00.broadcom.com
Using TFTP path:       nightly/2631
Linux build target:    7342a0_be

Chip ID register:      BCM7342A0
Board name:            BCM97342A0
CPU:                   Broadcom BMIPS4380
Primary Linux flash:   nor

1) Install non-initrd kernel image to flash (not available)
2) Install UBIFS rootfs to flash (RW/RO)
3) Install JFFS2 rootfs to flash (RW/RO)
4) Install SQUASHFS rootfs to flash (RO)
5) Format/partition entire HDD, then install rootfs (not available)
6) Update rootfs on first HDD partition (not available)
7) Install kernel/rootfs to USB thumb drive (not available)
q) Exit

Selection:
```

Option 1: stbutil supports writing the kernel image itself (not just the rootfs) to flash. This feature requires that CFE define a kernel flash partition which ends on an eraseblock boundary, as partitions that do not end on an eraseblock boundary are automatically marked read-only by mtdpart. For the most part, the NAND partition maps are suitable, but the NOR partition maps are not.

Option 2, UBIFS, is supported on all flash types. Several UBIFS images are generated for each build, in order to accommodate flash devices with different eraseblock and page sizes. stbutil will attempt to load a UBIFS image from the server over TFTP, appropriate for the detected flash geometry. In order to limit the size of the binary distribution, images are not included for larger flash devices. If a UBIFS image is not found, stbutil will create a new, empty UBIFS, and then load the UBIFS with the root file system contents from nfsroot-${plat}.tar.bz2 where ${plat} is the kernel platform (chip).

Option 3, JFFS2, is supported on NOR flash only. The summary feature is enabled, in order to improve mount times.

Option 4, SQUASHFS, is supported for all flash types. On NOR, the image is written directly to the flash. On NAND, the SQUASHFS image is written on top of a newly created UBI (not UBIFS) volume.

In this case, UBI provides bad block remapping and handles read disturb - a superset of the romblock functionality found in Linux 2.6.18.
Option 5 copies the initramfs rootfs to a SATA hard drive.

Option 6 reformats the first disk partition only (sda1), then refreshes the rootfs contents from the initramfs. The contents of sda3/sda4 (/opt and /data) are left undisturbed. Thus, sda3/sda4 would be good locations to store application binaries and video streams.

Option 7 installs the kernel and rootfs to a USB thumb drive. The thumb drive must be inserted and detected prior to running stbutil. A boot command is provided at the end, to allow CFE to load the kernel directly from the thumb drive.

After installing a kernel or rootfs image, stbutil provides sample boot instructions.
The command line options can be displayed through "stbutil -h". These options allow the user to specify a different TFTP HOST:PATH, copy the files from a local directory instead of TFTP, and/or automate the process.
The default TFTP HOST:PATH now points to /tftpboot/$USER on the machine on which the image was built. Released images from Broadcom still point to stb-irva-01:/tftpboot/<version> .
The default TFTP parameters and build target name are now stored in /etc/brcmstb.conf . This is auto generated by the build system.
stbutil requires bash as well as several other utilities. Therefore it is non-functional in -small builds, and would be severely limited in the case of -nomtd, -nohdd, -nonet, etc. However, it may still provide a useful demonstration of how to write out flash images.

Booting the Regular Kernel with a Flash root File System

After using stbutil to write a root file system to flash, you can boot the regular kernel (as opposed to the initrd kernel). Reboot the reference board, then boot the Linux kernel from CFE.

For 2.6.18 kernels

```
CFE>ifconfig eth0 -auto

CFE>boot –z -elf y.y.y.y:vmlinuz-xxxx ’rootfstype=jffs2

root=/dev/mtdblock0 ro’
```

For 2.6.3x and later kernels

```
CFE>boot -z -elf y.y.y.y:vmlinuz-xxxx 'ubiroot bmem=xx@xx'"
```

For newer kernels supporting BOLT

```
BOLT>ifconfig eth0 -auto

BOLT> boot y.y.y.y:vmlinuz-xxxx 'console=ttyS0,115200 earlyprintk ubi.mtd=flash1.rootfs
rootfstype=ubifs root=ubi0:rootfs rw'
```

where y.y.y.y is the IP address of your Build Server and xxxx is the chip name. The file vmlinuz-xxxx should already be in the /tftpboot directory.

The parameters that come at the end of the boot command are sent to the kernel and instruct it where to find the root file system.

Flashing the Kernel

After you have successfully booted the kernel once, you may want to write it to flash it so that it will boot faster in the future. Generally, you cannot flash the initrd kernel, because the initrd image is larger than the flash partitions intended for the kernel. Therefore, the root file system should be installed first, using stbutil from an initrd kernel booted over TFTP, so that regular kernel will have access to a root file system.

CFE divides the Flash into partitions. In most cases the partitions are labeled “flash0.kernel” and then it is easy to know which to use. In other cases, they are labeled “flash0.avail1” and you will have to select. You will need to pick a partition that has at least 1 MB but assume that the largest partition (12 MB or more) is going to be used for the Flash file system. This command will list the partitions:

```
CFE>show devices
```

Use this command boot from flash using a read-only Flash file system. The kernel boot parameters should be what you used earlier to boot the kernel. The following example assumes a flash root file system

```
CFE/BOLT>ifconfig eth0 -auto

CFE/BOLT>flash -noheader y.y.y.y:vmlinuz-xxxx flash0.kernel
```

Use this command boot from Flash using a read-only flash file system. The kernel boot parameters should be what you used earlier to boot the kernel. The following example assumes you are using a Flash root file system.

For 2.6.18 kernels

```
CFE>boot –z -elf flash0.kernel: ’rootfstype=jffs2 root=/dev/mtdblock0 ro’
```

For 2.6.3x and later kernels

```
CFE>boot –z -elf flash0.kernel: ‘ubiroot bmem=xx@xx’
```

For newer kernels supporting BOLT

```
CFE>boot –z -elf flash0.kernel: 'console=ttyS0,115200 earlyprintk ubi.mtd=flash1.rootfs
rootfstype=ubifs root=ubi0:rootfs rw'
```

See Making the Reference Board Auto-Boot for directions on how to make this boot command execute automatically when the reference board boots.

Option #2: Using INITRD Root File System

A fast and easy way to get running is to boot the initrd kernel and not create another file system. You can NFS mount your build server and run your application across the mount. If you do this, be aware of a couple things:

- You cannot flash the initrd kernel. You must boot from the network.
- You must remount the hard drive after every reboot.

Here is an example command to boot the initrd kernel.

```
CFE/BOLT>ifconfig eth0 -auto

CFE>boot –z -elf y.y.y.y:vmlinuz-initrd-xxxx

or

BOLT>boot y.y.y.y:vmlinuz-initrd-xxxx
```

(Login as root, no password)

Option #3: Using Hard Drive Root File System

Boot the initrd kernel, run stbutil, and select the option that installs the root file system to the hard drive. It will repartition and reformat your hard drive. When this is done, you simply need to use different kernel boot parameters. Enter commands like these.

```
CFE/BOLT>ifconfig eth0 -auto

CFE>boot –z -elf y.y.y.y:vmlinuz-xxxx ‘root=/dev/sda1’

 or

BOLT>boot y.y.y.y:vmlinuz-xxxx ‘root=/dev/sda1’
```

Option #4: NFS Root File System

While it is possible to use a root file system over NFS, the other options are more useful for set-top development. This configuration is not regularly tested by Broadcom. More information can be found in general Linux documentation online.

Kernel Boot Parameters

The following options may be added as command line arguments to the Linux kernel by enclosing them in single quotes in the boot command given to the Bootloader.

`bmem` – specify reserved A/V buffer memory
bmem option is not supported with BOLT bootloader with kernel 3.8-0.5 and later.

If no options are specified, Linux will create a default bmem region covering all but the first 64MB of lower memory. Linux will own 0-64MB, plus any upper memory or high memory.
`bmem=0` will disable all reserved memory.
`bmem=xxM@yyM` creates an XX megabyte region starting at the YY megabyte boundary. This replaces the default region.

Example:

```
bmem=64M@192M bmem=64M@512M
```

This example creates a 64MB region at the end of lower memory (192M-256M) and another 64MB region at the beginning of upper memory (512M-576M). Everything else on the system (512M-128M = 384M) is owned by Linux (assuming a system with 512MB total memory).
The `bmem` argument is available on the 2.6.3x and later kernels. It replaces the `mem` argument used on 2.6.18 and earlier kernel versions. The `mem` argument worked with the opposite sense, specifying memory for Linux, with unspecified memory

`pci=off`, `nousb`, `noflash`, `nosmp` - Disable PCI (includes SATA), USB, MTD, or SMP.

`console=ttySx,115200` - Console on UARTB/UARTC (ttyS1, ttyS2) is fully supported. Early printk can be enabled (default) or disabled.

`debug` - Show KERN_DEBUG messages

`initcall_debug` - Show each initcall entry/exit (lots of output)

`bcmrac` - No longer supported. All RAC/L2/LMB options are set up in CFE.

`sata_brcmstb.s2=1` - Enable SATA2 PHY rates (replaces `bcmsata2=1`)
`sata_brcmstb.ssc=1` - Enable SATA interpolation (replaces `bcmssc=1`)

`brcmnand.cmd=<cmd>` - pass CMD to brcmnand driver. Example commands include: `rescan`, `showbbt` . Replaces "`brcmnand=`".

`nandcs` - Comma-separated list of NAND chip selects. This overrides the defaults set by CFE in the NAND registers. Example: `nandcs=1`

Arguments specifying root file system:
`root=/dev/sda1` - rootfs (nominally ext3 or ext4) on SATA
`ro` - mount rootfs read-only (default is read/write)
`root=/dev/mtdblock0 rootfstype=jffs2` - Boot from the jffs2 file system on the MTD rootfs partition (NOR only)
`ubi.mtd=rootfs rootfstype=ubifs root=ubi0:rootfs` - Attach UBI to the MTD rootfs partition, then boot from the UBIFS file system on the UBI 'rootfs' volume.
`ubiroot` - Alias for the previous "UBI" command line, to save typing. The Broadcom BSP (prom.c) will expand this into the appropriate argument.

`nfsroot=SERVER:ROOTDIR ip=dhcp` - Mount the NFS-exported directory ROOTDIR on host SERVER as the root file system, after obtaining an IP for the STB via DHCP.
`mtdparts` - Override the default MTD partition tables from the kernel command line. Example for NOR flash: "`mtdparts=physmap-flash.0:8M(rootfs),52M(data),4M(cfe)`". "`physmap-flash.0`" is the device name for NOR; for NAND, use "`brcmnand.0`".

The following is an example of a complete boot command with kernel boot parameters (arguments).

```
CFE> boot -z -elf stb-sj1-01.sj.broadcom.com:2637-2.0/vmlinuz-initrd-7425a0 'nosmp
bmem=192M@64M bmem=192M@512M'
```

Making the Reference Board Auto-Boot

Setting the auto-boot Command

Whatever method you have of booting the kernel, you can have that boot command execute automatically when the reference board boots. The following example uses a flashed root file system as an example.

```
CFE> setenv -p STARTUP "boot -z -elf flash0.kernel: 'bmem=192M@64M bmem=512M@512M
ubiroot"
```

Or

```
BOLT>setenv –p STARTUP “boot flash0.kernel: 'console=ttyS0,115200 earlyprintk
ubi.mtd=flash1.rootfs rootfstype=ubifs root=ubi0:rootfs rw'”
```

Undoing auto-boot

Reboot the box and press **Ctrl-C** in the terminal program before the Bootloader finishes coming up. You should see a CFE> or BOLT> prompt that means the auto boot was aborted once. To remove the auto boot permanently, use:

```
CFE/BOLT>unsetenv STARTUP
```

To verify that STARTUP is no longer set in the CFE environment, use:

```
CFE/BOLT>printenv
```