Reference Platform Installation Guide for Linux Systems

September 21, 2012

BroadcomReferencePlatformSetup

REVISION HISTORY

Doc Number	Revision Date	Description/Author
STB_Platform-SWUM100-R	12/26/2008	Initial version
BroadcomReferencePlatformSetup	02/26/2009	Update kernel/rootfs build procedure from Mike Ward
BroadcomReferencePlatformSetup	08/23/2011	Update kernel/rootfs build procedure from Randall Jew
BroadcomReferencePlatformSetup	09/21/2012	Update kernel/rootfs build procedure from Mike Ward

Broadcom Corporation
5300 California Avenue
Irvine, CA 92617

Introduction

This document describes how to set up the CFE bootloader and Linux operating system on a Broadcom DTV/set-top reference platform and how to configure a Linux toolchain and kernel source on a build server. The CFE (Common Firmware Environment) is the primary bootloader for Broadcom set-top and HD DVD boxes, and DTV designs.

If you are running other software, like Brutus, Kylin, Settop API, or Nexus, please refer to their respective documents on how to build and run that software.

SOFTWARE TERMS

The follow 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 which controls the Reference Board and allows the Reference Software drivers and applications to run
- **Root Filesystem** – On some operating systems (such as Linux) it is customary to have at least a minimal disk layout for management of applications and libraries
- **Toolchain** – The compiler and linker that run on your Build Server and create binaries that can run on the Reference Board.

Section 2: Prerequisites

The term “Reference Software” refers to the source 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 (referred to herein as the Build Server)
  - This can be a dedicated Linux machine or a dual-boot Linux/Windows 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 .
  - These instructions only apply to Linux platforms. VxWorks has separate instructions not contained in this document.
- Knowledge of the “vi” UNIX editor
  - You must be able to edit text on the reference board.
  - There are many books on vi at virtually any bookstore.
  - You do not have to be an expert, but keystroke-by-keystroke instructions are not provided.
- Root access to your Linux build server and must be able to switch between root and user mode when instructed
- Connection of the Reference Board and build server to the same Ethernet network
- DHCP server running on the network that the Reference Board can use
- A bash shell
  - 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.tgz
    - vmlinuz-xxxx = uclibc-based kernel where xxxx is the chip; be aware that the exact kernel name may be different than just substituting xxxx.
    - vmlinuz-initrd-xxxx = uclibc-based initrd kernel where xxxx is the chip; be aware that the exact kernel name may be different than just substituting xxxx.
    - uclibc root filesystem tar ball binary (for installation on build server)

Section 3: Build System Configuration

System Configuration Broadcom Platform shows how your system configuration may look like. The types of RF inputs and A/V outputs depend on the platform.

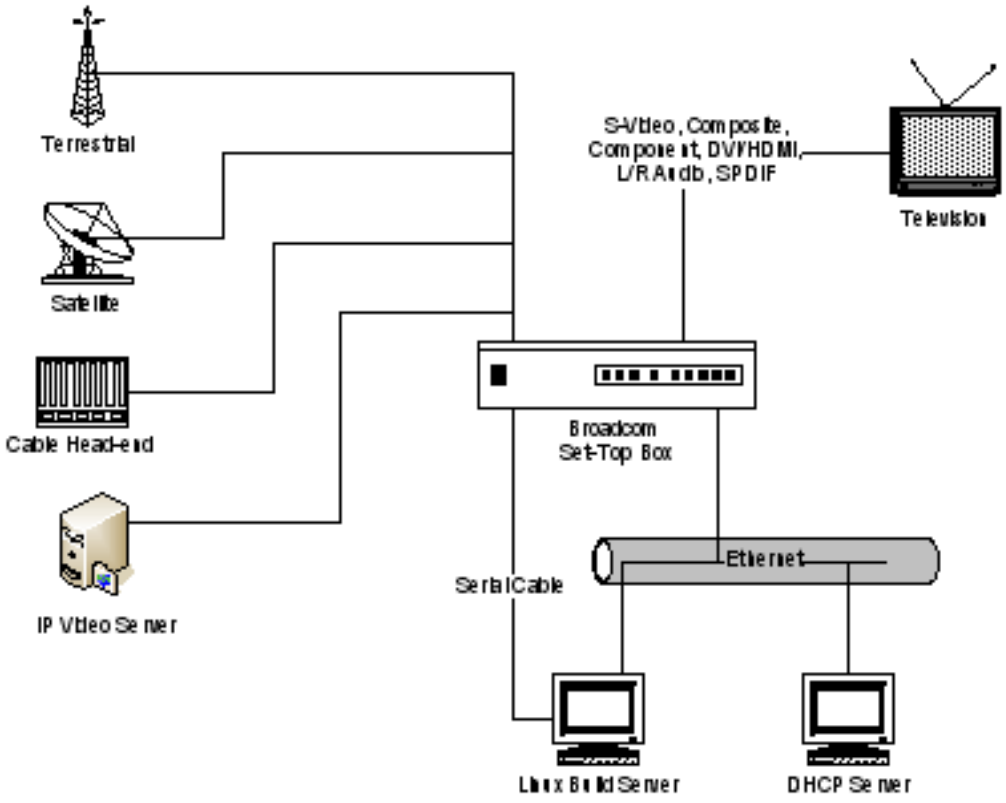

Figure 1: System Configuration Broadcom Platform

Section 4: Unpacking the Release File

When you receive a reference software release, it will come as a single tarball with a name like *refsw-20080328.97405.tgz,* which means *refsw-YYYYMMDD-PLATFORM.tgz*. This file should be unpacked using Linux and not WinZip or any Windows utility. If you use WinZip, you need to be sure CRs are not added, which will break the makefiles, and disrupt the files' symbolic links.

Move the tarball to a location you own (for instance, opt/refsw directory) and unpack it by entering the following:

```
tar zxvf refsw-20080328.97405.tgz
```

If you are using a 2.6.xx Linux kernel and above, the reference software may have been packaged using bzip2. If the release has a .bz2 extension, enter the following tar command:

```
tar jxvf pathname_to_file
```

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.

Throughout this document, you will be instructed how to access some of these files from your reference board. You will need to load the boot loader, kernel, and root filesystem binaries into a TFTP directory. You may need to have TFTP or NFS access to the reference software binary.

Section 5: 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 the Linux Build Server) and you will not be able to NFS mount. However, you can still load binaries using TFTP. Broadcom recommends getting the PumpKIN TFTP server. For details, see Configuring the Reference Board and Setting the CFE's autoboot Command (Option #2 is not applicable).

Section 6: Configuring the Linux Build Server and Network

INSTALLING LINUX ON THE BUILD SERVER

The build server is where you compile source code using the cross-compiling toolchain. Broadcom recommends Red Hat Linux or Fedora Core.

When you install Linux, use a server configuration, which should install the following tools:

- minicom
- TFTP server
- NFS server (strongly recommended)
- FTP server (optional)

You will need root permissions 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. Type **`tftp-server`**.
3. Get tftp-server-0.28-2.i386.rpm for your Linux version.
4. Transfer the rpm to your Linux box.
5. As root, type **`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 a standard location and editing your PATH. The specific filename and location of *toolchain_tarball.tar.gz* will vary. The name of the *toolchain_dir* will vary. The path of the elements stored inside the tarball may also vary. Please follow these steps:

```
tar ztvf toolchain_tarball.tar.gz
```

If the tarball contains files with /opt/toolchains/toolchain_dir/... in their path, then do the following:

```
cd /
tar zxvf toolchain_tarball.tar.gz
```

If the tarball contains files without /opt/toolchains/toolchain_dir/... in their path, then do the following:

```
mkdir –p /opt/toolchains
cd /opt/toolchains
tar zxvf toolchain_tarball.tar.gz
```

If the file name has a .bz2 extension, use the following tar command:

```
tar xjvf toolchain_tarball.tar.bz2
```

In all cases, the toolchain will install into /opt/toolchains/*toolchain_dir*, where the definition of *toolchain_dir* will vary.

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. Edit /etc/profile as root and add the following lines:

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

You can enter the following lines instead of the lines listed above if your /etc/profile includes the pathmunge() function:

```
pathmunge /opt/toolchains/toolchain_dir/bin
```

You do not have to put the toolchain at the head of your PATH, but it must be ahead of any other toolchain in your path. The reference software will pick up the first mipsel-uclibc toolchain in your PATH and assume that is the one you want.

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-uclibc-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, especially because kernel versions can change over time and you may need multiple copies installed on your build server. Another technique is to use a symlink for /opt/brcm/linux and move it to the correct kernel source. For brevity, the symlink approach is assumed.

The following instructions use "3.3-1.2" 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.

For example:

```
mkdir -p /opt/brcm/linux-3.3-1.2
cd linux-3.3-1.2
tar jxvf stblinux-3.3-1.2.tar.bz2
tar jxvf uclinux-rootfs-3.3-1.2.tar.bz2
cd /opt/brcm
 ln -sf linux-3.3-1.2/linux linux
 cd linux-3.3-1.2/uclinux-rootfs
 make images-7425b0
 cd ../linux
 export LINUX=`pwd`
```

If you get a new release of software with a new Linux kernel, make sure to install the source and update the /opt/brcm/linux symlink if required. You must build drivers with the same kernel source from which your kernel binary is built. If you fail to do this, erratic behavior can result.

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. 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 CFE 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 the following contents:

```
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 filesystem binary (for example, jffs2-xxxx.img)
- bootloader binary (for example, cfe_xxxx_le.bin)

CONFIGURING THE NFS SERVER

You will need an NFS server for two reasons:

- Loading and running the Reference Software drivers, libraries and applications on the Reference Board often requires copying from an NFS mount.
- You may want to mount an NFS root filesystem during development

Edit /etc/exports on your build server and add the two mount points. We recommend you use /opt/nfsroot_uclibc for your root filesystem. If the directories do not exist yet, you can create them now. The other mount point will allow you to access the location of your source. Please refer to the following example:

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

This assumes you installed (copied over and extracted) your reference software release under /opt/refsw. Now stop and start the NFS server as root.

Note: Do not use restart because “stop and start” does not work in all cases.

```
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 as root:

```
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 MINICOM

The build server controls the CFE bootloader through a serial interface. Configure minicom to talk with the reference board using the settings from*Reference Board Settings* :

Table 1: *Reference Board Settings*

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

Here are some quick directions for minicom:

1. Enter **`minicom -s`**
2. Select "Serial port setup."
3. Set the values given in *Reference Board 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
```

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, you 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 it.

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 do not know, then you should not guess or assume.

If you know that you do not have a DHCP server, we recommend you buy a LinkSys router that has DHCP support built-in.

Section 7: 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 some platforms, Broadcom has an Ethernet port connected to either a BCM4413 or other networking chip.
  - For other platforms, Broadcom requires a USB Ethernet dongle.
- A serial cable that is connected from COM1 (UART A or 0) on your reference board to your Build Server.
  - 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 CFE BOOTLOADER

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

If CFE is not working, or if you accidentally got another ROM (like “No-OS Diags”):

- 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 hooking that up. They all use the same serial port settings.

You can use Broadband Studio with an I^2C-to-parallel or I^2C-to-usb connector to Flash your CFE. You will need to talk with your FAE to get this.

If that fails, your board is bad, and you must inform your FAE.

GENERAL CFE HELP

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

```
# Getting help on CFE commands
CFE>help
# Booting a compressed kernel from a TFTP server
CFE>boot –z -elf IPADDRESS:KERNEL

# Showing the flash partitions which are available (along with other devices)
CFE>show devices

# Copying a kernel into a flash partition
CFE>flash -noheader IPADDRESS:KERNEL FLASHPARTITON

# Booting a compressed kernel from a flash partition
CFE>boot –z -elf FLASHPARTITION:

# Autobooting from a flash partition
CFE>setenv -p STARTUP 'boot -z -elf FLASHPARTITION:'

# Unsetting autoboot
CFE>unsetenv STARTUP
```

ASSIGNING A MAC ADDRESS

If your platform has an internal Ethernet controller (that is, not USB Ethernet or BCM4413), then you may need to program a MAC address into Flash.

In Option #1: Flashing Your Kernel and Root File System of this document, you will bring up the “eth0” interface using DHCP with the following CFE command `ifconfig eth0 -auto`. CFE will report back the MAC address. If it reads FF:FF:FF:FF:FF:FF:FF:FF, then you do not have an address. You can assign one by using CFE’s macprog command. Type **`help macprog`** for instructions.

FLASHING THE LATEST CFE

The latest uclibc kernel requires the latest CFE in order to boot compressed kernels. If you have an old CFE installed, you can flash the new CFE and then boot from flash. Place the new CFE binary into the /tftpboot directory, then run the following on your Reference Board:

```
CFE>ifconfig eth0 –auto
CFE>flash –noheader y.y.y.y:cfe_xxxx.bin flash0.cfe
```

where `y.y.y.y` is the IP address of your Build Server. The file cfe_xxxx_le.bin should already be in the /tftpboot directory of your build server.

On some platforms, you have the option of switching between Flashed and ROM-based CFE. If so, please set the appropriate chip select jumper. Refer to the hardware documentation if required.

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 the build date printed by CFE with your release notes.

If this technique does not work, you can also burn a new CFE ROM or flash using Broadband Studio. Please contact your FAE for this program.

Section 8: Building the Kernel and Root File System

This step is optional because Broadcom includes binaries in the release bundle. However, if you want to customize or minimize the kernel and root filesystem, you will need to rebuild from source. When using binaries, you must keep the following in sync:

- The root filesystem binary must be matched with the kernel binary
- The toolchain used to compile applications must match the root filesystem 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 know what you are doing. You can always return to the released binaries as a baseline.

First you must untar the kernel source and root filesystem source in adjacent directories. In the following example the `2.6.31-3.3` should be substituted with your kernel release:

```
mkdir 2.6.31-3.3
cd 2.6.31-3.3
tar zxvf stblinux_source.tgz
tar zxvf uclinux_rootfilesystem_source.tgz
```

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. Otherwise you might get to the end of the build and find that no binaries were copied in the directory.

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

2.6.18 UCLINUX-ROOTFS

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 build.mk and build-be.mk for the full list of platforms. *Platform Strings* provides some examples:

Table 2: *Platform Strings*

Platform	*Kernel Platform String*
BCM97038 B0	7038b0
BCM97038 C0	7030c0
BCM97401 A0	7401a0

BCM97110	7110
BCM97320	7320
BCM97315	7315
BCM97400 D0, SMP kernel	7400d0-smp

Check the Reference Software Release Notes. Some chip revisions share kernel platforms with others.

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

```
cd uclinux-rootfs
make distclean
make TFTPDIR=<tftpdir> <platform string>
```

For **2.6.3X Kernels**, more details examples of commands are in section 2.6.3x uclinux-rootfs changes from 2.6.18

In order to build the root filesystem, you must prepend the string **rootfs-** to the Kernel platform string. For example, to build the root filesystem for the BCM7401a0, the last instruction above would be as follows:

```
make TFTPDIR=<tftpdir> rootfs-7401a0
```

To build the kernel, use this instruction:

```
make TFTPDIR=<tftpdir> 7401a0
```

Once the root filesystem is built, it can be found under /opt/uclinux-rootfs/images/7401x/initramfs. This path cannot be used for booting with an NFS root filesystem, because it does not have real device nodes. Please refer to Option #2: NFS Root File System.

Here is a summary of the procedure. You must:

- Put the kernel source adjacent to the root filesystem source by untarring them in the same directory.
- Create a TFTPDIR directory with write permissions.
- Select an appropriate kernel platform string.
- For 2.6.18 kernels. use the root filesystem's build.mk file, which sets the environment variables. Use build-be.mk if you are building for a big-endian platform.

2.6.3X UCLINUX-ROOTFS CHANGES FROM 2.6.18

The uclinux-rootfs build configuration system has been overhauled in order to better meet the needs of the Broadcom STB platform. Here are some guidelines for using the new system.

SUPPORTED BUILDS

Builds are now per-chip, rather than per-board. Board differences are handled at runtime, as are minor chip variations (e.g. 7406 vs. 7405). There are no separate DOCSIS kernels in 2.6.3x.

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:

- -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
- -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 the non-free drivers).

BASIC BUILD SYSTEM USAGE (USING THE DEFAULT SETTINGS)

Below are example commands for the 97335 B0:

```
# Build rootfs, kernels, and flash images for 7335b0
cd uclinux-rootfs
make images-7335b0

# Build initramfs (builtin rootfs) and non-initramfs kernels, but no flash images
make kernels-7335b0

# Build just the initramfs kernel
make vmlinuz-initrd-7335b0
```

```
# Build the kernel only, no rootfs or flash images
make vmlinuz-7335b0
```

All images will be copied to the images/ directory (which is 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.

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 un-defined.

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 CON-
FIG_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 (e.g. after changing the configuration again, or after modifying files under user/lib/linux-2.6.x).

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
```

Section 9: Configuring the Kernel and Root Filesystem

GETTING THE USB ETHERNET DONGLE

Your reference board requires Ethernet connectivity. For platforms without an internal or on-board Ethernet controller, this requires a USB Ethernet dongle. Unfortunately, Broadcom supports a very limited set of dongles using our existing Linux kernel. The set of supported dongles may expand with future kernels.

For USB 1.0 platforms, Broadcom recommends the 3Com 3C460B. It can be purchased at:
http://www.pcpartscollection.com/3com3c10mbus.html.

CHOOSE A KERNEL AND ROOT FILE SYSTEM

There are many ways you can configure your Linux kernel and root filesystem system using CFE. The methods generally include:

- Loading the kernel
  - From TFTP (for development only)
  - From Flash (most stable)
- Loading the root filesystem
  - From Flash (most stable)
  - From NFS (for development only)
  - Using an initrd root filesystem, in which the root filesystem is compiled into the kernel and loaded into a ramdisk at boot time.
  - From the hard drive

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

KERNEL BOOT PARAMETERS

See **Section 10:**

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 filesystem compiled into it. It allows you to boot without having a root filesystem and configure a new root filesystem either in flash or on the hard drive. If you want to use an NFS root filesystem, you can ignore the initrd directions.

Here is an example of how to boot the initrd kernel:

```
CFE>ifconfig eth0 -auto
CFE>boot –z -elf 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.

FLASHING THE ROOT FILESYSTEM

Now run the stbutil utility on the Reference Board to load the root filesystem from your NFS mount 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 filesystem to Flash:

```
stbutil y.y.y.y:
Select option 2
```

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 filesystem binary into /tftpboot. If you placed them in a subdirectory under /tftpboot, then add that after the colon. I

This will configure a read-only Flash filesystem. When you reboot, the /etc and /var mount points will be loaded from images stored in Flash and placed in a RAM disk so that the kernel has write access to them. The flash remains read-only.

Changes to stbutil

stbutil has been rewritten for 2.6.3x. The new interface appears below:

```
# 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 thumbdrive (not available)
8) Install nonfree drivers
q) Exit

Selection:
```

Option 1: stbutil now 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. This needs to match the flash device or ubiformat will generate an error.

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 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 thumbdrive. The thumbdrive 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 thumbdrive.

Option 8 installs the nonfree drivers from the network. MoCA drivers are built into the released rootfs images, but in some cases they may not be present. WLAN drivers are not built into any images by default. For more information, see the Nonfree section of networking.html .

After installing a kernel or rootfs image, stbutil now 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 autogenerated 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 Filesystem

Now that you have a root filesystem in flash, you can boot the regular kernel (as opposed to the initrd kernel). Do the following:

```
# Reboot the Reference Board
```

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 kernels

```
CFE>boot -z -elf flash0.kernel: 'ubiroot bmem=xx@xx'"
```

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 load its root filesystem.

If you used a Flash root filesystem, you can go to Flashing the Kernel

FLASHING THE KERNEL

After you have successfully booted the kernel once, you may want to Flash it so that it will boot faster in the future. You cannot Flash the initrd kernel, which means it is required to have your root filesystem installed already.

CFE divides up 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 filesystem. Here is how to list the partitions:

```
CFE>show devices
```

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

```
CFE>ifconfig eth0 -auto
CFE>flash -noheader y.y.y.y:vmlinuz-xxxx flash0.kernel
```

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

For 2.6.18 kernels

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

For 2.6.3x kernels

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

See Setting the CFE's autoboot Command for directions on how to make this boot command execute automatically when the reference board boots.

OPTION #2: NFS ROOT FILE SYSTEM

INSTALLING THE NFS ROOT FILESYSTEM ON THE BUILD SERVER

The rootfs build process also creates a file called uclinux-rootfs/images/<platform>/initramfs_data.cpio.gz. This file may be used to set up an NFS root filesystem. The following commands can be used. You must be root when executing the commands. Take care, as omitting or misspelling the --no-absolute-filenames flag will most likely corrupt the Linux OS on your PC or server

```
mkdir nfsroot
cd nfsroot
zcat <path>/initramfs_data.cpio.gz | cpio --no-absolute-filenames -id
chown -R root.root *
```

You should have already set your NFS mount points so that nfsroot is in a shared directory.

BOOTING THE REGULAR KERNEL WITH AN NFS ROOT FILESYSTEM

The only thing you must do on the Reference Board is to use the correct kernel boot parameters when you boot the kernel from CFE.

```
# Reboot the Reference Board

CFE>ifconfig eth0 -auto
CFE>boot -z -elf y.y.y.y:vmlinuz-xxxx 'root=/dev/nfs nfsroot=y.y.y.y:/opt/nfsroot
ip=dhcp rw'
```

The nfsroot path /opt/nfsroot must contain the subdirectories /etc, /dev, /lib, etc.

The kernel will use DHCP to obtain an IP address in order to mount the filesystem. The filesystem will be mounted with read/write access.

You can Flash the kernel and use an NFS root filesystem. See Flashing the Kernel for instructions. Just use the NFS root filesystem boot parameters listed here.

OPTION #3: USING INITRD ROOT FILE SYSTEM

A fast and easy way to get running is to boot the initrd kernel and not create another filesystem. 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.
- You must recreate device nodes after every reboot.

Here is an example of how to boot the initrd kernel:

```
CFE>ifconfig eth0 -auto
CFE>boot –z -elf y.y.y.y:vmlinuz-initrd-xxxx

# Login as root, no password.
```

OPTION #4: USING HARD DRIVE ROOT FILE SYSTEM

Boot the initrd kernel, run stbutil, and select the option that installs the root filesystem 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 these commands:

```
CFE>ifconfig eth0 -auto
CFE>boot –z -elf y.y.y.y:vmlinuz-xxxx ’rootfstype=ext2 root=/dev/hda1 rw’
```

Section 10: Kernel Boot Parameters

See the Linux kernel release notes for other Linux boot parameters for configuring memory, static IPs, MIPS RAC and many other options.

2.6.3X MEMORY CHANGES

The reserved memory facility has been overhauled. The "mem=" command line parameter should no longer be used for this purpose. Please see "bmem" under the command line arguments section.

Support for 512MB of DRAM is handled by a few customizations scattered throughout the MIPS code (CONFIG_BRCM_UPPER_MEMORY). The DISCONTIGMEM facility is no longer used. The SPARSEMEM facility may be used if desired, but it is optional. SPARSEMEM will save approximately 2MB of system memory because it does not create "struct page" arrays for the memory hole at 256MB.

Chips that have cache aliases will support up to 512MB through CONFIG_BRCM_UPPER_MEMORY . Chips that do not have cache aliases will support 1GB and beyond through the standard HIGHMEM facility.

Both cached and uncached kseg2 mappings are provided for upper memory (256MB-512MB), so it can be used the same way as standard "lower" memory (the 256MB of RAM in kseg0/kseg1).

Nonstandard "one-off" memory configurations to support a specific customer requirement (e.g. 7440 with 1GB, 3563 with 128MB+64MB) are no longer included, in the interest of making the reference kernel code simpler and more robust.

Per-chip feature definitions are centralized in arch/mips/brcmstb/Kconfig . The intent is to leverage common platform features as much as possible instead of duplicating code or treating each product as a special case.

VM overcommit is now the default on 2.6.3x. In the past, it was disabled in the rcS script. Consequently, the default stack size hack (8MB->1MB) is no longer needed in the kernel code.

Standard 2.6 kernel facilities (e.g. platform_device, plat_* hooks) are used wherever possible, in lieu of changing the kernel code. Large diffs to the kernel code are avoided where possible; most of these have been replaced with fcuntion calls out to the BSP. This is intended to simplify porting to other kernel versions, and to cleanly separate the BSP code from core kernel functionality.

The MIPS COUNT/COMPARE timers are no longer used to calculate sub-millisecond time offsets in SMP mode. There is no reliable mechanism for keeping these timers in sync on an SMP system, which often caused gettimeofday() to lose monotonicity (i.e. run backwards). The alternative is to use the 27Mhz WKTMR clocksource, available on 3548 and newer chips.

The EMAC driver now performs background link negotiation, and supports all ethtool ioctls that are provided by the standard MII library.

Ethernet MAC addresses are no longer read from a fixed location in flash. CFE is expected to pass this information through the callback mechanism (see below). MAC addresses are assigned by the core BSP code, and passed to each driver through the platform_device (platform_data) mechanism.

A "brcmstb" platform device has been created to provide access to STB platform features and information. This replaces the changes to /proc/cpuinfo in 2.6.18 and previous kernels.

Initial RAC and cache initialization is handled entirely by the CFE MIPS init. "bcmrac" and other related kernel support is no longer present in 2.6.3x.

The SATA driver will not support port multipliers.

SMTC mode (MIPS multithreading) is not supported on 2.6.3x.

USEFUL COMMAND LINE OPTIONS

bmem - for reserved A/V buffer memory

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 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.

The "mem" argument should no longer be used for this purpose.

The application must handle all cache coherence for BMEM.

BMEM supports O_DIRECT I/O and get_user_pages() but does not handle cache coherence for this case.

UNCAC_ADDR(), CAC_ADDR(), virt_to_phys(), and phys_to_virt() all fully support upper memory. The kernel now maintains both cached and uncached (fixed) mappings to the upper 256MB.

Due to the way the boot time allocator works, BMEM must come out of `ZONE_NORMAL` memory. If any part of a BMEM range falls in `ZONE_HIGHMEM` , it will not be honored.

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, show-bbt . Replaces "brcmnand=".

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

root, rootfstype, and friends

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 filesystem 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 filesystem 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 filesystem, 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".

Examples:

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

Section 11: Making the Reference Board Auto-Boot

SETTING THE CFE'S AUTOBOOT 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 filesystem as an example.

```
CFE setenv -p STARTUP "boot -z -elf flash0.kernel: 'rootfstype=jffs2
root=/dev/mtdblock0 ro'"
```

UNDOING AUTOBOOT

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

```
CFE>unsetenv STARTUP
```

Enter this to verify that it is gone:

```
CFE>printenv
```