CPE
NAND Flash Support

Application Note

Table of Contents

1 Overview

NAND Flash is supported for the following Broadcom devices:

BCM6362	BCM63381	BCM6838	BCM4908
BCM6328	BCM63138	BCM6848	BCM62118
BCM63268	BCM63148	BCM6858	
BCM68360	BCM63158		

This Application Note provides the following information about the NAND Flash support.

- NAND Flash Update Procedure
- NAND Flash Partitions
- NAND Flash Dual Image Support
- NAND Flash Part Requirements
- NAND Image for NAND Programmer
- NAND Flash Source Files

To use NAND Flash, the reference design board must be strapped to boot from NAND Flash. Only one NAND Flash chip may be used. Reference platform software does not support NAND Flash and NOR Flash simultaneously; but if necessary, this can be achieved using source code modifications.

While a pure JFFS2 filesystem is supported, it exists essentially for legacy purposes, as well as the split UBI filesystem. New designs should preferably use the pureUBI file system.

2 NAND Flash Update Procedure

2.1 Setting Default Partition Sizes

The software allows up to four data partitions. By default, the 4th data partition will always be allocated with 4MB. 1, 2, and 3 are initialized to 0MB.

To override these defaults, change the PROFILE. When the software is built, it will create a default partition table based on the new values.

```
BRCM_MISC1_PARTITION_SIZE=0
BRCM_MISC2_PARTITION_SIZE=0
BRCM_MISC3_PARTITION_SIZE=0
BRCM_MISC4_PARTITION_SIZE=4
```

The data partition sizes can be changed by the CFERAM using b or x command. e.g.

```
CFE>     x
Press: <enter> to use current value
        '-' to go previous parameter
        '.' to clear the current value
        'x' to exit this command
Partition 1 Size (MB)             :  0M
Partition 2 Size (MB)             :  0M
Partition 3 Size (MB)             :  0M
Partition 4 Size (MB) (Data)      :  4M
```

Typically, when data partition sizes are changed, it's advisable to reprogram the binaries.

2.2 Preparing the Image

The following steps write a Linux router image to an uninitialized SPI or parallel NAND Flash part. The BCM63138 is used as an example. Most examples also use TFTP for file transfer, but the HTTP server is usually preferable.

1. On the Linux build PC, run **make menuconfig** from the base directory.
   ```
   $ cd SOURCE_DIR
   $ make menuconfig
   ```
2. In the menuconfig application, load the desired profile, visit the Root Filesystem submenu, and ensure that the block size for your flash is among the options selected. (See "Setting Default Partition Sizes" on page 4.)
3. On the Linux build PC, build the desired profile.
   ```
   $ make PROFILE=96313GW
   ```
   Or
   `$ make PROFILE=96313GW nandcfeimage` to produce a minimal CFE-only NAND image that is much faster to load.

2.3 JTAG Debugger

1. On the Linux build PC, copy the bcm963138GW_cferom_fs_flash_image_*.w image to a TFTP boot directory.
   ```
   $ cp targets/963138GW/bcm963138GW_nand_cferom_fs_image_*.w /tftpboot/.
   ```
2. Set the board strap for NAND boot with proper ECC setting and page size according to the NAND chip on board. The ECC must be at least as strong as required by the Flash device. For SoCs that support BCH-4, *BCH4* should always be used in preference to Hamming even when the Flash requires only 1-bit correction.
3. On the Linux build PC, build the BCM63138 CFE bootloader.
   ```
   $ cd SOURCE_DIR/cfe/build/broadcom/bcm63xx_rom
   $ make BRCM_CHIP=63138 BLD_NAND=1
   ```
4. Launch the JTAG debugger.
5. From the JTAG debugger, connect to the CPU and run any needed memory initialization script, then load the CFE RAM ELF file.
   The CFE RAM ELF file is located at SOURCE_DIR/cfe/build/broadcom/bcm63xx_ram/cfe63138.
6. From the JTAG debugger, start the CFE RAM ELF file. The CFE bootloader is started on the serial console. Configure the NVRAM parameters as required.
7. To download and flash the NAND Flash image, use the following command from the BCM63138 router serial console:
   - For 128 KB block size NAND Flash chips: `w bcm963138GW_nand_cferom_fs_image_128_ubi.w`

   When the image is flashed, the BCM63138 router will boot the Linux image that is stored on NAND Flash. The image with `cferom` in the name starts to flash the image at block 0. The (upgrade) image without `cferom` in the name starts to flash the image at block 1.

Subsequent image updates can be done from either the CFE bootloader or a Linux router.

To load from a TFTP server, enter the *c* command from the BCM63138 router serial console, to configure the Host IP address of the Linux PC where the TFTP server and the NAND Flash image are located.

To load from a web browser, configure a PC with a static IP address on the 192.168.1.0/24 subnet, navigate to http://192.168.1.1, and upload the image file via the form provided.

2.4 SPI Flash

The following steps are for customers who do not have a JTAG debugger. The procedure supports CPE boots from SPI Flash and upgrades the image to NAND Flash. The board BCM963138REF is used as an example.

1. On the Linux build PC, build 963138 bootloader to support the program image to NAND Flash.
   ```
   $ cd SOURCE_DIR/cfe/build/broadcom/bcm63xx_rom
   $ make BRCM_CHIP=63138 BLD_SPI_NAND=1
   ```
2. Change the bootstrap to boot from SPI Flash, but leave ECC switches set correctly as if booting from NAND.
3. Upgrade `SOURCE_DIR/cfe/build/broadcom/bcm63xx_rom/bcm963138_cfe.w` to the board.
4. When the image upgrade is complete, change the bootstrap to boot from NAND Flash.
5. Boot and adjust NVRAM settings as required.

3 NAND Flash Partitions

Figure 1 shows the JFFS2, JFFS2/UBIFS split, and pureUBI NAND Flash partitioning.

Figure 1: NAND Flash Partitions

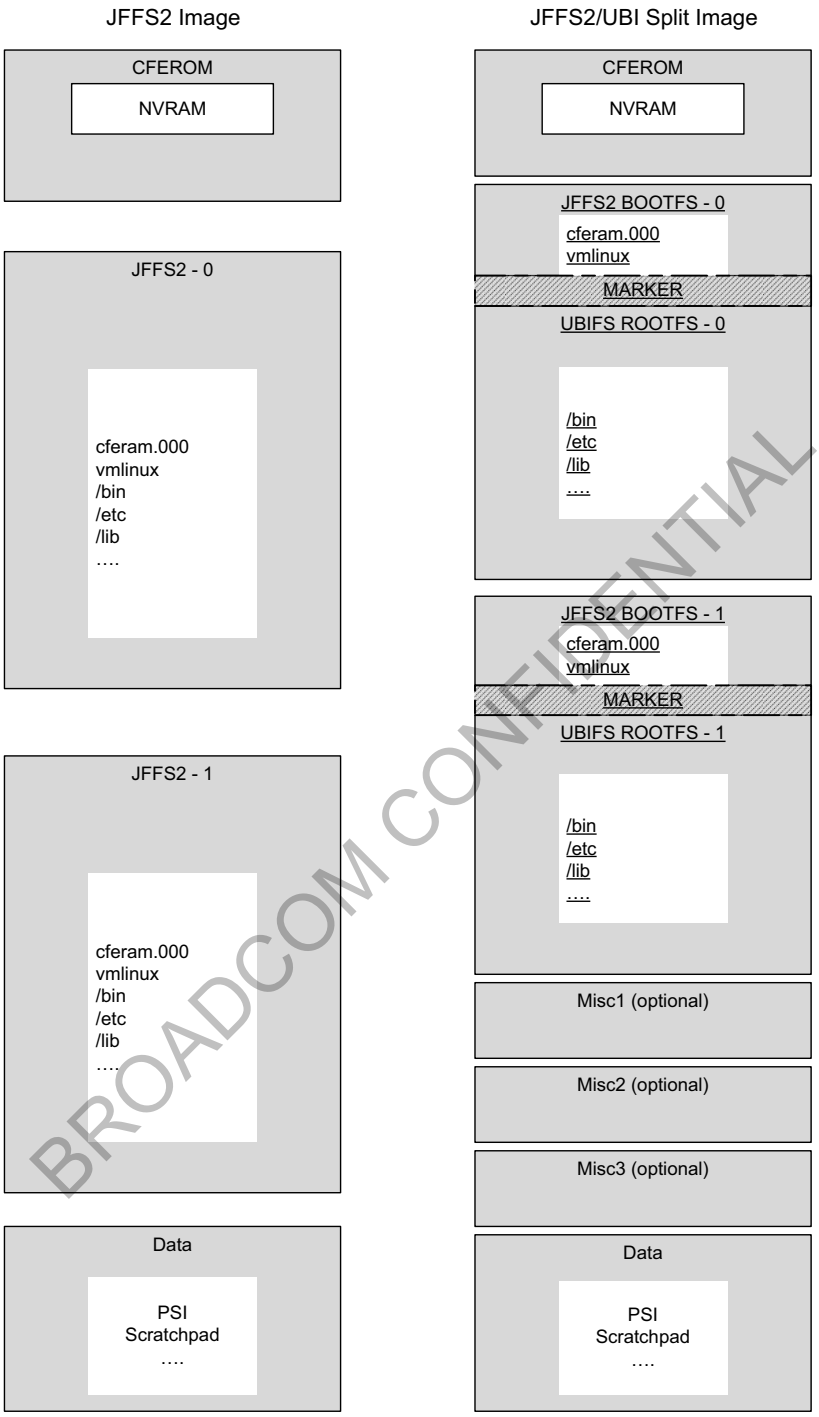
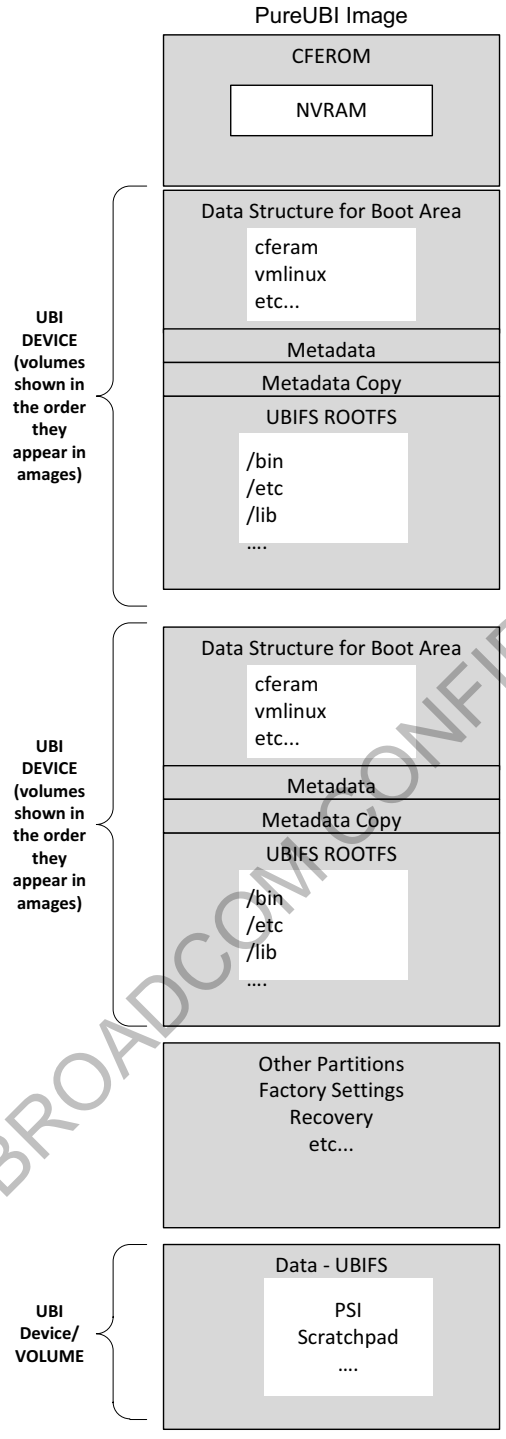


Figure 2: Flash Partitions



3.1 Raw CFE ROM Boot Partition

The first NAND Flash block contains a raw boot block with the CFE ROM code. It executes in place and determines which partition to continue to boot from. It then loads the CFE bootloader RAM image into memory from this partition and jumps to its entry point.

For Gen3 bootrom devices such as BCM4908/BCM62118/BCM63158/BCM6858, multiple copies of CFE ROM are placed to fill the first 1MB of Flash to provide backups in case there are bad blocks.

3.2 Image Partitions

There are three type of image system partitions that contain the files used in a Linux router image.

3.2.1 JFFS2 Images (deprecated)

JFFS2 images contain a single read-only filesystem that is mounted as the root filesystem. Two of the files are the CFE RAM image and the Linux kernel, vmlinux. The file extension on the CFE RAM image file is a number between 000 and 999. The partition with the CFERAM file bearing the higher number is the most recently flashed partition. This image is used for boot unless a flag is set indicating that an older image should be booted.

3.2.2 UBI Images (legacy designs only)

UBI images are actually "split" images. They contain a small read-only JFFS2 filesystem, known as the "boot" filesystem followed by a marker, one erase-block in size, followed by a root filesystem consisting of a UBIFS in a UBI volume.

3.2.3 pureUBI Images

PureUBI images differ in that they contain no JFFS2 and as such do not store any data in the Out of Band area of a Flash. CFE knows how to READ a raw-UBI volume, read and write NVRAM, and burn images to Flash. CFE should never have to WRITE to anything in Flash volumes but knows how to replace one string in a UBI image with another in the process of burning the image. The only portion modified during flashing is in a metadata volume so it is possible for the entire boot volume, Kernel, and root filesystem to remain unmodified and can be digitally signed.

Control of boot image selection operations (such as the OMCI "Activate" command and "Commit") are handled by Linux, writing the persistent metadata to Flash (UBI volumes) to indicate the committed image and using a volatile on-chip resource (register or RAM - depending on the device) to indicate that the boot is a "boot-once" event.

The layout is designed so that UBI/UBIFS and EMMC can be as similar as possible.

Metadata associated with an image (such as sequence number, boot state) is in a distinct volume from the bootfs or rootfs so that digital signatures can apply across each non-metadata volume in an image without concern that the contents of those volumes would change.

Images remain constructed so that a Flash device can be programmed using skipbb-only in a single write operation with or without CFEROM.

3.2.3.1 Metadata

Metadata is a small dynamic, non-auto-resize UBI volume that should appear last in the pureUBI image file. When creating images, it should contain for each entry, a 32-bit length, the entry name, and the entry value. For example:

```
<32-bit entry length>committed<null>1<32-bit CRC>
<32-bit entry length>cferam.000<null>021<32-bit CRC>
Etc..
<32-bit value 0>
```

Followed by padding, if necessary, to ensure that the volume will have space for additional content. This may be handled by ubinize options.

Metadata is updated during/after programming to change the sequence number entry (cferam.000). For example, to 101, where 101 is a sequence number analogous to the CFERAM file extension sequence numbers. In addition, the image may be committed by setting the committed entry to 1.

3.2.3.2 Boot Area

Boot area is a static UBI volume with a data structure containing lengths, names and binary objects (such as cferam.000, vmlinux.lz, etc.) and is terminated with a 32 bit zero. Additional entries could be built into the structure. The volume itself must be static so that signatures computed over the volume will not change.

This is a sequence of 8-byte-aligned records, each containing the following:

- 32-bit offset within volume to next record (zero for last record)
- 32-bit offset within volume to data start
- 32-bit data length
- Filename – null-terminated ASCII
- Padding (to data start)
- Data (8-byte aligned)
- 32-bit CRC
- Padding (to next record)

3.2.3.3 Root Filesystem

The root file system can be static (for read-only, signed root filesystems) or dynamic and auto-resize.

3.2.3.4 Optional User Volume

If the root filesystem is read-only and a portion of the image that can be modified after imaging is desired, one or more user volumes can be accommodated.

3.3 CFEROM Behavior

CFEROM uses NVRAM for DDR initialization and to locate the two image partitions. It reads UBI volumes, instead of knowing how to read JFFS2.

After DDR-INIT, the CFEROM checks the reset_status register to determine if the reset was a soft-reset. For newer devices, (newer than BCM63268), CFEROM then looks in a (device-dependent) register that survives reboot to determine if the reboot was to boot to a specific image. If so, that image is selected regardless of metadata.

If the boot image was not specified for reboot, CFEROM locates the metadata volume in each of the two images' UBI devices. If the one with the higher sequence number is committed, it will be booted. If not, the other image is valid, and it will be booted.

3.4 CFERAM Behavior

CFERAM will boot VmLinux from the same boot volume as it was loaded from.

If presented with a JFFS2-type image (CFEROM+IMAGE), it will preserve the NVRAM and program the image just like a Flash programmer, updating the sequence number.

If presented with a pureUBI image, it will program it as an update image, replacing the CFERAM.000 entry with 123 (assuming the previous image was 122) and recalculating the CRC as the metadata volume goes by.

CFERAM does not preserve erase counts as this is for engineering, initial programming, and recovery only.

3.4.1 MISC1, MISC2, and MISC3 Partitions

If the sizes of MISC1, MISC2, and MIS3 partitions in the NVRAM are non-zero, space will be reserved between the end of the image area and the data partition. These partitions are generally ignored by CFE, though they can be imaged by CFE and Linux will create MTD devices for them.

Attaching them or mounting them is a function of startup scripts that can be customized.

3.5 Data Partition

The data partition contains files that require write access such as configuration data (psi) and data that is needed across reboots (scratch pad). This is deprecated as JFFS2. When using pureUBI if a JFFS2 data partition is already in existence it is left as is, however if there is not a data partition, a UBI data partition is created.

3.6 Raw Bad Block Table Partition

The bad block table partition maintains a list of NAND blocks that are bad.

4 NAND Flash Dual Image Support

The file system part of a NAND Flash image is flashed to the nonactive file system partition. The file system partition to boot from is read from the NVRAM configuration boot flag. The value of this field is *boot from latest image* or *boot from previous image*. This field is configured in the CFE serial console *c* command.

When an image is flashed, the extension of the JFFS2 file, cferam.xxx, is changed to a number that is greater than the highest number that is currently on the two file systems. This file name is used to identify the latest image.

5 Headered Images

If Secure boot is enabled, it is only responsible to get a secure CFEROM up and running. The CFEROM is then responsible to do security for CFERAM, and then CFERAM to do security for Linux.

Gen 1

BCM63268 supports Gen 1 secure image, if enabled, otherwise the image format is the same as was shown in NAND Flash Partitions.

Gen 2

Devices BCM6838, BCM6848, BCM63138, BCM63148, and BCM63381 support Gen 2 secure image, if enabled, otherwise the image format is as shown in NAND Flash Partitions. In addition, BCM63138 SPI NAND requires a headered image (BCM63148 does not support SPI NAND.)

1. Headered images are built by executing this command from the code root directory
   ```
   make menuconfig PROFILE=<name>
   ```
   Example:
   ```
   make menuconfig PROFILE=963138GW
   ```
2. Select "Secure Boot">"Enable Secure Boot".
3. Exit and save changes
4. Rebuild CFE/Linux and several _secureboot images should be available.

A secure headered image boot is shown in Table 3, essentially the same two image partition Flash layout that is shown earlier in this document, with the difference being that CFEROM may now be larger than a single block, taking up to either 512KB or 1Mb depending on the device.

Figure 3: Gen 2 Secure Headered Image

Block 0	63138/48: 64KB zero padding, other: CFE ROM XIP (Legacy, contains NVRAM, runs if secure boot is not enabled)		
64KB (0x10000)	63138/48: CFE ROM XIP (Legacy, contains NVRAM, runs if secure boot is not enabled), other: continuation of CFE ROM		
Block 1	CFEROM copy #1 (same CFEROM binary as above but with Header, Trailer)		
	Unauthenticated header		
		4 bytes	Magic number 112233 (0x0001B669)
		4 bytes	Magic number 445566 (0x0006CC7E)
		4 bytes	Version (1)
		4 bytes	Length
		4 bytes	CRC
	Authenticated header (ignored for non-secure image)		
		2 bytes	Type (0)
		2 bytes	Version (0)
		2 bytes	Length (0)
		2 bytes	Config bits (0)
		584 bytes	Manuf. Chain of trust (mfg market ID, mfg RSA public key modulus, encrypted BEK/BIV, signature)
		584 bytes	Operator chain of trust (oper market ID, oper RSA public key modulus, encrypted BEK/BIV, signature)
	Bootloader Binary		
	Trailer		
		256 bytes	Signature
		4 bytes	CRC
	Padding		
Block 2	CFEROM copy #2 (same as above)		
…	...		
Block N	CFEROM copy #N (same as above)		
...	Padding		
Block M (1MB or 512KB)	roofs first block (@1MB for BCM63138, BCM63148, @512 KB for BCM6838 BCM63381)		
Block M+2	rootfs second block		
…	...		

Gen 3

Devices BCM4908, BCM4906, BCM6858, BCM63158, BCM62118, and BCM61800 use the Gen 3 secure boot procedure whereby all images are headered. A Gen 3 secure or unsecure headered image boot follows essentially the same two image partition Flash layout as described previously, but CFEROM now is stored using headered and potentially redundant images within the 1MB instead of a single block.

Figure 4: Gen 3 Unsecure Image

Offset			
0	64KB zero padding		
64KB (0x10000)	NVRAM (first 4KB of CFEROM which includes NVRAM)		
64KB (0x11400)	Bootloader		
	Unauthenticated header		
		4 bytes	Magic number 1183954 (0x0002CE92)
		4 bytes	Magic number 145257 (0x00023769)
		4 bytes	Version (1)
		4 bytes	Mode eligibility (1)
		4 bytes	Length (28)
		4 bytes	Image size
		4 bytes	CRC
	Authenticated header		
		4 bytes	Type (1)
		4 bytes	Unauthenticated header length (32)
		4 bytes	Bytes to authenticate (0)
		20 bytes	Offsets (Zero)
	CFEROM Binary		
	Trailer		
		4 bytes	CRC
	Padding		
1KB boundary	Bootloader copy #1		
…	...		
1KB boundary	Bootloader copy #N		
...	Padding		
1MB	roofs start		

Figure 5: Gen 3 Secure Image

Offset			
0	64KB zero padding		
64KB (0x10000)	NVRAM (first 4KB of CFEROM)		
64KB (0x11400)	Bootloader		
	Unauthenticated header		
		4 bytes	Magic number 1183954 (0x0002CE92)
		4 bytes	Magic number 145257 (0x00023769)
		4 bytes	Version (1)
		4 bytes	Mode eligibility (6)
		4 bytes	Unauthenticated header length (28, 0x1C)
		4 bytes	Image size
		4 bytes	CRC
	Authenticated header (only if secure mode)		
		4 bytes	Type (1)
		4 bytes	Authenticated header length
		4 bytes	Bytes to authenticate
		4 bytes each	COT offsets
		...	...
		Various each	COTs
		...	...
	CFEROM Binary		
	Trailer		
		256 bytes	Signature field use
		256 bytes	Signature manufacturing use
		4 bytes	CRC
	Padding		
1KB boundary	Bootloader copy #1		
…	...		
1KB boundary	Bootloader copy #N		
...	Padding		
1MB	roofs start		

6 Manually Changing Partition Sizes

Normally, in manufacturing, the desired MISC partition sizes are programmed in the NVRAM in advance and the partition table is loaded on first boot.

When repartitioning, only image 0 and cferom are preserved.

To change partition sizes, upgrade **twice** to guarantee that the current version is available on image 0, then use the "b" command to update parameters and select the appropriate sizes. After reboot, the new partitions will be available.

7 NAND Flash Part Requirements

In order to boot from a parallel NAND Flash, the chip's NAND Flash controller must be able to determine the Flash part's configuration. This is accomplished by retrieving the configuration from the ONFI parameter table if the NAND device supports ONFI. If not, the NAND Flash controller uses an internal list of NAND Flash chip identifiers and corresponding configurations. If the NAND Flash chip is not in the list, the NAND Flash controller uses the fourth byte of the chip ID as the page size and tries to determine the configuration. This means that a NAND Flash part that is not in the list may work, but must first be verified.

A parallel NAND flash part must meet the following requirements to be supported:

- The NAND Flash part supports ONFI

  or
- If the NAND Flash part does not support ONFI, it still may work. The NAND Flash controller uses an algorithm that does several things which include looking at NAND Flash chip ID byte four for the page size. If the NAND Flash controller correctly determines the configuration, the part will work.

To boot from NAND Flash, the BCM63xx/BCM68xx NAND Flash controller must be able to determine the Flash part's configuration. If the NAND Flash chip is identified in this manner, the NAND Flash controller uses the fourth byte of the chip ID as the page size and tries to determine the configuration. This means that a NAND Flash part that is not specifically identified, may work, but needs to be verified.

In addition:

- The DSL chip must be able to support the ECC level required by the NAND, and the NAND must have enough OOB/spare area bytes to support the ECC level required.
- The NAND MUST NOT do its own ECC, i.e., the DSL chip must be able to determine exactly how many bits are bad in a page without resorting to a device-specific method of doing so.
- NAND chips with number of operations (NOP) >= 4 will work. (NOP < 4 may work, but in this case they must be identified properly by the system).
- NAND Flash parts must be "CE Don't Care".
- NAND Flash parts must have guaranteed good block 0 (zero).
- NAND Flash size must be at least 128 MB with a block size of at least 128 KB.

If all of these conditions are met, and the NAND operates through some basic testing (boots to Linux command prompt, a few image updates from CFE and Linux, some Linux file manipulation) then it should operate correctly in the system.

BCH level and bytes required per 512 byte subpage for parallel NAND (SPI NAND does its own ECC.) Consider the space requirements for the bad block marker (1 byte) plus if using JFFS2 the clean marker (8 bytes), so for example a 2K page device with 16-byte spare area per 512 byte subpage x 4 subpages = 64 bytes spare area total, subtract 1 byte for the bad block marker, 8 bytes for the JFFS2 clean marker. This leaves a total of 55 bytes available in the OOB, which is not quite enough for BCH8 on some newer devices (14 x 4=56 required bytes.).

Table 1: BCH Levels OOB Bytes Used

Chip	NAND Ctrl Rev	Hamming	BCH4	BCH8	BCH12	BCH24	BCH40	BCH60	JFFS2 Notes
OOB bytes reserved by NAND controller per 512-byte subpage	—	—	16	27	—	—	—	—	—
6368	v2.1	3	—	—	—	—	—	—	—
6328/6362	v2.2	3	—	—	—	—	—	—	—
63168/268	v4.0	3	7	13	20	—	—	—	BCH8 can fit in 64-byte spare area
6838/6818/6828	v5.0	3	7	14	21	42	—	—	BCH8 requires 108-byte spare area
63381/63138	v7.0	3	7	14	21	42	70	105	BCH8 requires 108-byte spare area
6848/6858/63148/158/62118/4908	v7.1	3	7	14	21	42	70	105	BCH8 requires 108-byte spare area

8 SPI NAND Flash

SPI NAND Flash is supported on BCM63381, BCM63138, BCM6838, BCM6848, BCM6858, BCM4908, BCM62118.

BCM63148 does not support SPI NAND.

BCM63138 SPI NAND requires a headered image which is built as follows:

1. At the root directory of the code execute the command:
   "make menuconfig PROFILE=963138GW"
2. Select "Secure Boot">"Enable Secure Boot", making sure not to change any other option.
3. Exit and save changes
4. Rebuild CFE/Linux and there should be several _secureboot images available.
5. Use one of the images to program the SPI NAND.

For SPI NAND the requirement is that either the part is explicitly supported in our code, or we default to 2KB page size, 128KB block size, 1Gb total size, and one bit ECC.

9 NAND Image for NAND Programmer

Customers may need to build an image with spare area data if they use a NAND programmer to load the image in the product line. Broadcom provides a utility to build such an image.

9.1 Spare Area Format

The spare area contains the JFFS2 clean marker, bad block indicator (BI), and Error Correction Code (ECC). The JFFS2 clean marker is 8 bytes of fixed pattern 0x19, 0x85, 0x20, 0x03, 0x00, 0x00, 0x00, 0x08. BI is 0xFF if good, and non-0xFF if bad. The ECC calculation may use the Hamming Code, BCH-4, BCH-8, or BCH-12 algorithms, depending on the chip and its configuration. The ECC must be at least as strong as required by the Flash device. For SoCs that support BCH-4, *BCH4* should always be used in preference to Hamming even when the Flash requires only 1-bit correction.

Each character in the following layouts represents a byte in the spare area. The key is:

- B = location of bad block indicator
- E = location of ECC
- 0 = location for spare area data, the first eight locations contain the JFFS2 cleanmarker.

Table 2: Spare Area Formats

Spare Size	ECC Algorithm	Layout
64 bytes	Hamming	B,B,0,0,0,0,E,E,E,0,0,0,0,0,0,0,0,0,0,0,0,0,E,E,E,0,0,0,0,0,0,0 0,0,0,0,0,0,E,E,E,0,0,0,0,0,0,0,0,0,0,0,0,0,E,E,E,0,0,0,0,0,0,0
6 bytes	Hamming	0,0,0,0,0,B,E,E,E,0,0,0,0,0,0,0
16 bytes	BCH-4	0,0,0,0,0,B,0,0,0,E,E,E,E,E,E,E
128 bytes	Hamming	B,B,0,0,0,0,E,E,E,0,0,0,0,0,0,0,0,0,0,0,0,0,E,E,E,0,0,0,0,0,0,0 0,0,0,0,0,0,E,E,E,0,0,0,0,0,0,0,0,0,0,0,0,0,E,E,E,0,0,0,0,0,0,0 0,0,0,0,0,0,E,E,E,0,0,0,0,0,0,0,0,0,0,0,0,0,E,E,E,0,0,0,0,0,0,0 0,0,0,0,0,0,E,E,E,0,0,0,0,0,0,0,0,0,0,0,0,0,E,E,E,0,0,0,0,0,0,0
128 bytes	BCH-4	B,0,0,0,0,0,0,0,0,E,E,E,E,E,E,E,9,0,0,0,0,0,0,0,0,E,E,E,E,E,E,E 0,0,0,0,0,0,0,0,0,E,E,E,E,E,E,E,9,0,0,0,0,0,0,0,0,E,E,E,E,E,E,E 0,0,0,0,0,0,0,0,0,E,E,E,E,E,E,E,9,0,0,0,0,0,0,0,0,E,E,E,E,E,E,E 0,0,0,0,0,0,0,0,0,E,E,E,E,E,E,E,9,0,0,0,0,0,0,0,0,E,E,E,E,E,E,E
64 bytes	BCH-4	B,0,0,0,0,0,0,0,0,E,E,E,E,E,E,E,9,0,0,0,0,0,0,0,0,E,E,E,E,E,E,E 0,0,0,0,0,0,0,0,0,E,E,E,E,E,E,E,9,0,0,0,0,0,0,0,0,E,E,E,E,E,E,E
64 bytes	BCH-8	B,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E,9,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E 0,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E,9,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E
128 bytes	BCH-8	B,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E,9,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E 0,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E,9,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E 0,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E,9,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E 0,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E,9,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E

Table 2: Spare Area Formats (Continued)

Spare Size	ECC Algorithm	Layout
216 bytes	BCH-8	B,0,0,0,0,0,0,0,0,0,0,0,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E 0,0,0,0,0,0,0,0,0,0,0,0,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E 0,0,0,0,0,0,0,0,0,0,0,0,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E 0,0,0,0,0,0,0,0,0,0,0,0,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E 0,0,0,0,0,0,0,0,0,0,0,0,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E 0,0,0,0,0,0,0,0,0,0,0,0,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E 0,0,0,0,0,0,0,0,0,0,0,0,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E 0,0,0,0,0,0,0,0,0,0,0,0,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E
216 bytes	BCH-12	B,0,0,0,0,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E 0,0,0,0,0,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E 0,0,0,0,0,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E 0,0,0,0,0,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E 0,0,0,0,0,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E 0,0,0,0,0,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E 0,0,0,0,0,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E 0,0,0,0,0,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E
256 bytes	BCH-4	B,0,0,0,0,0,0,0,0,E,E,E,E,E,E,E,9,0,0,0,0,0,0,0,0,E,E,E,E,E,E,E 0,0,0,0,0,0,0,0,0,E,E,E,E,E,E,E,9,0,0,0,0,0,0,0,0,E,E,E,E,E,E,E 0,0,0,0,0,0,0,0,0,E,E,E,E,E,E,E,9,0,0,0,0,0,0,0,0,E,E,E,E,E,E,E 0,0,0,0,0,0,0,0,0,E,E,E,E,E,E,E,9,0,0,0,0,0,0,0,0,E,E,E,E,E,E,E 0,0,0,0,0,0,0,0,0,E,E,E,E,E,E,E,9,0,0,0,0,0,0,0,0,E,E,E,E,E,E,E 0,0,0,0,0,0,0,0,0,E,E,E,E,E,E,E,9,0,0,0,0,0,0,0,0,E,E,E,E,E,E,E 0,0,0,0,0,0,0,0,0,E,E,E,E,E,E,E,9,0,0,0,0,0,0,0,0,E,E,E,E,E,E,E 0,0,0,0,0,0,0,0,0,E,E,E,E,E,E,E,9,0,0,0,0,0,0,0,0,E,E,E,E,E,E,E
256 bytes	BCH-8	B,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E,9,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E 0,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E,9,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E 0,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E,9,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E 0,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E,9,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E 0,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E,9,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E 0,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E,9,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E 0,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E,9,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E 0,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E,9,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E

Table 2: Spare Area Formats (Continued)

Spare Size	ECC Algorithm	Layout
432 bytes	BCH-8	B,0,0,0,0,0,0,0,0,0,0,0,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E 0,0,0,0,0,0,0,0,0,0,0,0,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E 0,0,0,0,0,0,0,0,0,0,0,0,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E 0,0,0,0,0,0,0,0,0,0,0,0,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E 0,0,0,0,0,0,0,0,0,0,0,0,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E 0,0,0,0,0,0,0,0,0,0,0,0,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E 0,0,0,0,0,0,0,0,0,0,0,0,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E 0,0,0,0,0,0,0,0,0,0,0,0,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E 0,0,0,0,0,0,0,0,0,0,0,0,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E 0,0,0,0,0,0,0,0,0,0,0,0,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E 0,0,0,0,0,0,0,0,0,0,0,0,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E 0,0,0,0,0,0,0,0,0,0,0,0,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E 0,0,0,0,0,0,0,0,0,0,0,0,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E 0,0,0,0,0,0,0,0,0,0,0,0,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E 0,0,0,0,0,0,0,0,0,0,0,0,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E 0,0,0,0,0,0,0,0,0,0,0,0,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E
432 bytes	BCH-12	B,0,0,0,0,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E 0,0,0,0,0,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E 0,0,0,0,0,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E 0,0,0,0,0,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E 0,0,0,0,0,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E 0,0,0,0,0,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E 0,0,0,0,0,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E 0,0,0,0,0,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E 0,0,0,0,0,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E 0,0,0,0,0,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E 0,0,0,0,0,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E 0,0,0,0,0,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E 0,0,0,0,0,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E 0,0,0,0,0,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E 0,0,0,0,0,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E 0,0,0,0,0,0,0,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E,E

Example: In the system using 16 bytes spare area and Hamming ECC algorithm, a dump of the spare area using the *e s* command in CFE shows:

0x19,0x85,0x20,0x03,0x00,0xff,0x3c,0x0f – 0x00,0x00,0x00,0x08,0xff,0xff,0xff,0xff

9.2 NAND Image Build Utility

The spare area data is not contained in the image generated by the make PROFILE=XXX. A utility, *createnfimg*, can be run that calculates the spare area and creates an image that contains both the page data and the spare area data.

The tool is located in the hostTools/createnfimg folder from the release source. Contact Broadcom support if it is not available. Run the following command to build the tool:

```
gcc -o createnfimg createnfimg.c
```

This program takes in a binary file and breaks it into 512-byte or 2048-byte pages. For each 512-byte partial page, it creates the NAND OOB area contents with a Hamming or BCH ECC and optional JFFS2 cleanmarker. Each page is written out, followed by the OOB area, which is 16 bytes for a 512-byte page or 64 bytes for a 2048-byte page. The output file name is the input file name with a `.out` extension.

Table 3: Useful Options for the NAND Image Build Utility

Option	Description
-b <bch_lvl>	Level of ECC correction bit. Default = 4. ■ 1 for 1-bit Hamming ■ 4 for 4-bit BCH ■ 8 for 8-bit BCH
-m <field_order>	Order of the finite field for BCH[a]. Default = 13.
-c <partial_page_size>	Partial page size in bytes. Default = 512.
-i <infile>	Name of the input file. Output files will be `infile.out`.
-p <page_size>	Page size in bytes. Default = 2048.
-l <0|1>	Set little-endian clean marker for ARM based chips (BCM68138, BCM68148).
-d <pad_amount>	OOB padding amount to fill after ECC is finished, for example with newer NAND devices that have 128 bytes OOB per page, treat part like OOB=64 (-r 16) and set this value to 64 to pad 64 bytes after the OOB 64 bytes. Default = 0.
-r <oob_size>	OOB size in bytes per 512 byte subpage. Default = 16
-j <0|1r>	1 = Add JFFS2 cleanmarker to spare area. Default = 1
-n <pages_per_block>	Pages per block. Default = 64.
-t <0|1>	Trim input file to page boundary (removes extraneous signature bytes). Default = 1.

a. The BCH finite field order, the *m* parameter:
– is 13 for the BCM6368, BCM6816, BCM6362, BCM6328, and BCM63268.
– is 14 for the BCM6828, BCM6838, BCM6848, BCM6858, BCM6818, BCM63138, BCM63148, BCM4908 and newer chips.

Usage examples:

BCM63268

- Large page BCH-8:
```
./createnfimg -b 8 -P 2048 -m 13 -i bcm963268GW_nand_cferom_fs_image_128_ubi.w
```

BCM63138

- Large page BCH-8:
```
./createnfimg -b 8 -P 2048 -m 14 -i bcm963138GW_nand_cferom_fs_image_128_ubi.w
```

9.3 NAND Flash Source Files

The following files have changes for NAND Flash support.

- hostTools/scripts/defconfig-bcm.template
- hostTools/scripts/gendefconfig
  These files contain kernel configuration information to configure NAND Flash MTD driver when JFFS2 is configured in make menuconfig.
- hostTools/jffs2/*
  Source files for the mkfs.jffs2 utility, which creates the JFFS2 file system used with NAND Flash images.

- hostTools/createnfimg/*
  Source files for the NAND image build utility.
- cfe/cfe/board/bcm63xx_rom/src/bcm63xx_main.c
  This file contains bootup code that reads the CFE RAM image from JFFS2.
- cfe/cfe/board/bcm63xx_ram/src/*
  These files contain NAND Flash support for booting Linux and flash image update.
- targets/buildFS2
  This script creates the JFFS2 file system.
- shared/opensource/flash/nandflash.c
  NAND Flash driver that is used by the CFE bootloader.
- kernel/linux/drivers/mtd/brcmnand/*
  NAND Flash MTD driver that is used by the Linux router.

9.4 SPI NAND Flash Source Files

SPI NAND is an extension of NAND, and in addition, uses the Linux NAND infrastructure. Most of the files used for NAND are also used for SPI NAND, with the exception of noted replacements. The following files have changes for SPI NAND Flash support:

- shared/opensource/flash/nandflash.c → shared/opensource/flash/spinandflash.c
  SPI NAND Flash driver that is used by the CFE bootloader.
- kernel/linux/drivers/mtd/brcmnand/* → kernel/linux/drivers/mtd/nand/bcm63xx_spinand.c & nand_ids.c, kernel/linux/drivers/mtd/maps/bcm963xx_mtd.c
  SPI NAND Flash MTD driver that is used by the Linux router.

Revision History

CPE-AN1103; January 15, 2018

- Updated: “Raw CFE ROM Boot Partition” on page 9
- Added: “pureUBI Images” on page 9
- Added: “Metadata” on page 10
- Added: “Root Filesystem” on page 10
- Added: “Optional User Volume” on page 10
- Added: “CFEROM Behavior” on page 10

CPE-AN1102; May 1, 2016

- Updated: “NAND Flash Update Procedure” on page 7
- Updated: “NAND Flash Partitions” on page 9
- Updated: Table 1: “NAND Flash Partitions,” on page 10
- Updated: “NAND Flash Parts” on page 12
- Updated: “Spare Area Format” on page 13
- Updated: “NAND Flash Source Files” on page 17
- Updated: “SPI NAND Flash Source Files” on page 17
- Added support for BCM6848, BCM6858, BCM4908
- Added: “Image Partitions” on page 11
- Added: “Manually Changing Partition Sizes” on page 12
- Removed: Supported NAND Flash Devices table
- Removed: Supported SPI NAND Flash Devices table
- Removed: “Preparing the Image”

CPE-AN1101; February 23, 2015

- Updated: “Preparing the Image”
- Updated: Figure 1: “NAND Flash Partitions”
- Updated: “NAND Flash Parts”
- Updated: Table 2: “Useful Options for the NAND Image Build Utility”
- Added: Table 2: “Supported SPI NAND Flash Devices”

CPE-AN1100; March 23, 2014

Initial release.