U-Boot Porting Guide

01/15/13

Revision 347

TABLE OF CONTENTS

Index of Tables

1 Introduction

1.1 What is contained in this document?

This document describes the process of porting U-Boot to new OCTEON® hardware. It assumes that the reader is familiar with using U-Boot commands and with using the Linux operating system. It also assumes that the reader is familiar with the C programming language and has at least a basic understanding of microprocessors, the concept of virtual memory and communicating with memory-mapped I/O devices.

Familiarity with previous versions of U-Boot is helpful but it is not essential.

1.2 Note to the Reader

Porting U-Boot to a new board is not difficult. Getting U-Boot to boot up on a new board will likely take one to three days at the most, assuming that the board passes the Bringup and Diagnostic Kit (BDK). The number of files that actually need to be changed is generally quite small, usually a dozen or less. Most of the work is already done, it is just a matter of picking and choosing what features to use and knowing how things are connected to the boot bus and TWSI bus. The most difficult part may be configuring memory but the BDK should help determine the proper settings for the memory.

1.3 What is U-Boot?

U-Boot is an open-source multi-platform boot loader maintained by Wolfgang Denk at DENX Software Engineering. It provides a command line interface for performing various operations and support for booting various operating systems.

It has support for various filesystems, flash media and various I/O devices.

Think of it as being equivalent to the BIOS in a PC or Macintosh.

1.4 Assumptions

This document assumes that a command is a command that is entered on the U-Boot command line and not the Linux command line unless otherwise specified.

Environment variables are specific to the U-Boot environment, not the host environment.

This document also assumes that the reader is proficient with the C programming language and has some knowledge about hardware as well as a basic understanding of the MIPS platform. It is assumed that the reader is able to understand hardware datasheets and board schematics.

All directory references are from the U-Boot directory (`bootloader/u-boot`) unless otherwise specified.

1.5 What has changed since SDK 2.0?

Many of the changes were in moving files around as a result of moving OCTEON functionality out of the core U-Boot directories. Most of the OCTEON-specific files can now be found under `arch/mips/cpu/octeon` and `arch/mips/include/asm/arch-octeon`. All of the board specific code is under `board/octeon` with common board specific code under `board/octeon/common`.

- U-Boot has been upgraded from version 1.1.1 to version 2011.03. Note that U-Boot version numbers have changed. U-Boot is now released quarterly from the community with the version containing the year and month of release.
- Numerous files have been reformatted in order to support the U-Boot coding style guidelines. The U-Boot coding style is basically the same as the Linux kernel coding style. This reformatting is not yet complete and more files will likely be updated in the future. When writing code, please use the following coding style:
  - Indentation is 8 characters, using tabs, not spaces.
  - Block braces start on the same line as if/else/for/while statements.
  - Lines are limited to 80 columns where it makes sense to do so.

  To format existing code to the U-Boot coding style, use the `Lindent` script included with the Linux kernel in the `scripts` directory.
- The Simple Executive source files have moved. Instead of being located in the `lib_mips` directory, the c files are now located in `arch/mips/cpu/octeon/cvmx` and the h files are now linked under `arch/mips/include/asm/arch-octeon`. When including simple exec header files, use `<asm/arch/cvmx-xxxx.h>`. The `arch` directory is a symbolic link to the `arch-octeon` directory.
- The .c files for OCTEON-specific commands are now placed in the `arch/mips/cpu/octeon/commands` directory instead of the `common` directory. New OCTEON or board specific commands should *not* be placed in the `common` directory. Board-specific commands should be placed in the `board/octeon/<board_name>` directory. Currently there are no board-specific commands defined.
- The only OCTEON-specific file not located in these directories is `arch/mips/lib/board_octeon.c`. All of the other library files used are common to all MIPS CPUs. In the future there will be additional cleanup of `board_octeon.c` to make it more portable with the eventual goal of merging it with `board.c` which is generic for all MIPS platforms. The file `arch/mips/lib/board.c` is not used. If there were changes made to the old U-Boot `lib_mips/board.c` they may need to be applied to `arch/mips/lib/board_octeon.c` instead.
- Support for non-OCTEON MIPS products has been removed from all of the OCTEON MIPS code. There is no longer any reason to keep it. This includes support for boards like PURPLE and INCAIP.

- `CFG_CMD_<command>` is no longer a bitmask. Command definitions have been renamed `CONFIG_CMD_<command>`. Each command must be defined individually. Command options also often control whether different files are compiled by the Makefiles.
- `CFG_<OPTION>` is no longer used. Instead it is now `CONFIG_SYS_<OPTION>` or `CONFIG_<OPTION>` depending on what option it is. For example, `CFG_CMD_PING` is now `CONFIG_CMD_PING`.
- The `Makefile` no longer contains OCTEON board products in it. OCTEON boards are now configured in `boards.cfg`. There should be no reason to edit the main `Makefile` for adding additional boards.
- The board configuration header files have changed. Most of the common functionality is now located in `include/configs/octeon_common.h.` If a particular board needs to override any of these settings, it can use `#undef <OPTION>` and redefine it. Some defines have been removed as they are no longer needed and many have changed, see `CFG_<OPTION>` above.
- OCTEON-specific fields besides the flags have changed in `gd`. Now all OCTEON-specific fields are under `gd->ogd`. Many previous board files can be imported with minimal change, mostly changing `gd->xxx` to `gd->ogd.xxx` for OCTEON-specific fields. For example, `gd->board_desc` is now `gd->ogd.board_desc`.
- Additional flags have been added to `gd->flags` and the defines have changed. These are translated into the old U-Boot format before executing simple executive applications and the Linux kernel to maintain backwards compatibility. Expect this to change in the future when U-Boot for OCTEON will be migrated to use the flat device tree. This translation occurs at run-time in the function `octeon_translate_gd_to_linux_app_global_data()` in `arch/mips/cpu/octeon/octeon_boot.c`. This function is called by the `bootoct` and `bootoctlinux` commands.
- Common board functions use `__attribute__((__weak__))` so that individual boards can replace the functions without requiring `#ifdefs`. This reduces the number of files that need to be modified when adding new boards.
- The boot bus initialization now takes place in `board/octeon/common/octeon_boot_bus.c`. The functions defined there are all defined as `__attribute__((__weak__))` so if they need to be overridden for a specific board the function can be redefined and the linker will automatically choose it over the common function. See the NICPro2 board and the Embedded Planet EP6300C for examples. The NicPro2 board does not have any devices connected to the boot bus but still requires some minimal configuration of that bus.
- If a boot bus device needs to be initialized and is not defined in `octeon_boot_bus.c`, all that is needed is to create a function named `octeon_boot_bus_board_init()` within the board-specific .c file. See the Embedded Planet ep6300c board as an example. This board has a FPGA called the BCSR that needs to be initialized.

- PCI console and PCI bootcmd support have been moved. The main event loop no longer deals with it. U-Boot now takes advantage of the console MUX feature such that it is now possible to have the serial console, PCI console and PCI bootcmd all active simultaneously. By default only input is enabled for all three with output going only to the serial port. This can be changed by setting stdin, stdout and stderr. For example, one can connect the PCI console and type "`setenv stdout serial,pci; setenv stderr serial,pci`" and output will now appear simultaneously on both the serial and PCI console. Note that when typing the command in this instance that no data will be echoed until `stdout` has been set.
- PCI bootcmd support may be deprecated in the future and merged with PCI console support entirely. The command line utility will still be present however, but will use the PCI console interface instead of the bootcmd interface.
- For USB there are now several options. To speed up the U-Boot process, USB initialization can be disabled by setting the `disable_usb_scan` environment variable to any value. USB can be enabled at any point by typing the command "`usb start`". Note that the scan will not add USB storage devices unless the `usb start` command has been issued. If configuring the USB clock without a scan is still desired then set both the `disable_usb_scan` and `enable_usb_ehci_clock` environment variables to any value.
- Support for the non-TLB case has been removed. U-Boot will now always be linked and run at address 0xC0000000. TLB remapping occurs three times. The first time in `start.S` 0xBFC00000 is remapped to 0xC0000000 before calling any C code. The second time is when the flash is mapped to the boot bus area of 0x1XXX0000. The final time is when U-Boot is copied to the top of memory. The TLB remapping is the same as it was in the 2.0.0 SDK release of U-Boot.
- `DECLARE_GLOBAL_DATA_PTR` must be declared globally within each file now instead of inside each function, otherwise GCC may "optimize" the code too much and cause problems. While this has not been observed in practice, this is recommended by the U-Boot documentation.
- OCTEON-specific compact flash support has been moved to `arch/mips/cpu/octeon/octeon_cf.c`.
- Board-specific compact flash support has been moved to `board/octeon/common/octeon_board_cf.c` and all of the functions have been declared as weak to allow for board specific versions.
- Numerous configuration options such as command definitions control conditional compilation.

1.6 *What has changed since SDK 2.1*

The biggest change is the introduction of the flat device tree. When migrating from the SDK 2.1 to the SDK 2.2 a device tree file will need to be created. In many cases, if sections of the device tree are missing the settings from the configuration header files and cvmx-helper-board.c will be used.

It is still strongly recommended that the device tree be fully defined. Device tree support is continuing to evolve and future versions of the SDK, Linux and U-Boot will make further use of it.

- Flat Device Tree support
- BBGW Reference board added back
- ebb6600 board added
- octbootbus command added for testing boot bus settings (disabled by default)
- Various SATA fixes, switched to new SATA driver for Silicon Image.
- Many weak functions now use aliases so replacement functions can still call them.
- Several additional functions have been added.
- More fine tuning support for memory configuration on OCTEON II processors.
- 16-bit PSRAM support added
- Added "big bar" option for PCI to support 64-bit PCI devices, required for Silicon Image SIL3124.
- DFM is no longer initialized when not installed.
- QLM command expanded for ebb6600.
- QLM command can now apply changes to all lanes at once.
- Added hook functions for environment variable changes.
- Expanded trap dump support to display cause description string.

1.7 What Has Changed Since SDK 2.2

- Added EBB6100 board.
- Enhanced device tree support.
- Added MMC support.

1.8 What Has Changed Since SDK 2.3

- PHY support has been completely rewritten to utilize the new U-Boot PHY drivers.
- Most PHY initialization can now be defined by the flat device tree rather than the board specific C code.
- PHY APIs have changed. Do not use the `cvmx_` functions, use `phy_read` and `phy_write` instead.
- Moved from using `cvmx_gpio` to standard U-Boot GPIO routines.
- Added MMC and SD support for the CN61XX and CNF71XX processors.

- Added a generic MMC stage 1 and stage 2 bootloader which is also compatible with many SD devices.
- TWSI support has been changed to use the low-level driver. This allows support for larger transactions and finer control.
- More reliance on the flat device tree. All of the network settings can now be defined in the flat device tree.
- Writing files to the FAT filesystem is now supported.
- GPIO driver has been added and `gpio` command has been enabled.
- Ethernet driver has been moved to `drivers/net/octeon_eth.c`
- TWSI driver is now `drivers/i2c/octeon_i2c.c`.
- New OCTEON MDIO driver `drivers/net/octeon_mdio.c`.
- New device tree PHY support added in `board/octeon/common/octeon_board_phy.c`.
- Improved Vitesse PHY support is now unified for all boards which use it.
- USB performance improvements and bug fixes (EHCI timeout).
- New targets added: snic10e, snic10e_61, sff6100, evb7100, nic4e_66 as well as some old devices such as the Asus NAC38 and the WSX16 rack mountable devices.
- U-Boot now copies itself from flash to the L2 cache early on to speed up the boot process as long as the L2 cache is larger than the U-Boot image. In many cases the speed up is significant.
- Numerous bug fixes to the FAT filesystem.
- Simplified board-specific C files by making more functions common.
- 64-bit versions of common memory functions like `md64`, `mw64`, `mm64`, etc.
- lzma decompression support has been added.
- TFTP server support.
- TFTP put support.
- New `mdio` command with support for clause 45 10G PHY devices.
- Improved random MAC address generation.
- Enhanced NAND ECC support to handle multi-bit ECC.
- Added support for SGMII mode for NIC68_4 rev 3 boards.
- Automatic detection if NOR flash uses ALE. **Note**: `cavium,ale-mode = <1>` is still required in the device tree.

- Moved the serial driver to `drivers/serial/serial_octeon.c`.
- Moved the TWSI driver to `drivers/i2c/octeon_i2c.c`.
- Moved the Ethernet driver to `drivers/net/octeon_eth.c`.
- New GPIO driver in `drivers/gpio/octeon_gpio.c`.
- Removed the old unmaintained U-Boot version 1.1.1.
- New `octbootbus` command to display and change boot bus device settings.
- `gd->ogd.fdt_addr` is now `gd->fdt_blob` in order to be more compatible with other architectures.
- Support for redundant environment blocks in flash has been added.
- MD5, SHA1 and SHA256 are now accelerated.
- UART is initialized earlier to aid in debugging. Note that it is initialized to 115200 baud so any output before normal initialization will not work at other baudrates.
- New hooks have been added. `board_net_preinit()`, `board_mdio_init()` and `board_net_postinit()` for board-specific initialization of networking related components.
- NOTE: The octeon SPI driver is currently incomplete.

1.9 Further Information

Further information can be found in the U-Boot `README` and `README.OCTEON` files as well as in files located in the `doc` directory.

The ***OCTEON Programmer's Guide*** is an excellent reference.

The U-Boot web site has a documentation page at http://www.denx.de/wiki/U-Boot/Documentation.

The U-Boot mailing list is a good source of information and a great place to ask questions about U-Boot that are not related to OCTEON processors or OCTEON-specific changes. Asking questions about OCTEON processors or boards will likely yield no positive response.

There are numerous documents available at http://www.mips.com such as the ***MIPS64 Architecture For Programmers Volume III: The MIPS64 Privileged Resource Architecture*** document.

Documentation on the MIPS N32 ABI can be found in the ***MIPSpro N32 ABI Handbook*** which can be downloaded at http://techpubs.sgi.com/library/manuals/2000/007-2816-005/pdf/007-2816-005.pdf. If not found at this link, a search should turn up other copies of this document.

The OCTEON Hardware Reference Manual contains much of the needed information about the OCTEON processor.

The device tree web site at http://devicetree.org provides several documents covering the usage of the flat device tree. The best documents are the ePARR paper and the IEEE 1275 specification. Another good source is the Device Tree Usage page at http://devicetree.org/Device_Tree_Usage.

There is also a section in chapter 7 in the book ***Embedded Linux Primer: A Practical, Real-World Approach, Second Edition*** which covers the Flat Device Tree. This book can also be found online at http://my.safaribooksonline.com/book/operating-systems-and-server-administration/embedded-linux/9780137061129.

1.10 Feedback

Feedback on this document and the porting process is welcome and will help us make it easier in the future to port U-Boot to new hardware and to ease upgrading in the future.

Feedback can be sent to u-boot@cavium.com.

2 Before you begin

Information on the new board is essential. The following pieces of information are required:

- If the board is based on OCTEON II the board must pass the Bringup and Diagnostic Kit (BDK) before proceeding in order to make sure all of the memory settings are correct.
- A list of TWSI devices and their addresses, this includes the serial EEPROM, temperature sensor, real-time clock (if present) and any other devices that need to be configured by U-Boot.
- Configuration for the memory used on the board from the BDK.
- The size of the CFI (Common Flash Interface) compliant flash device and sector size.
- Type of network interfaces and specifications for the PHY chips used and the PHY configuration.
- A schematic of the board is helpful.
- Various data sheets for devices used on the board should be available.
- Read and become familiar with the device tree section.

3 Things ***NOT*** to do

There are a number of things that should be avoided in order to make maintaining U-Boot easier.

- **Do not modify an existing board.** Modifying an existing board will make it impossible to test against the reference board and will create major problems with support. It will also make it much more difficult to maintain the code when newer SDKs are released which overwrite the board files. It is much easier to create a new board, especially compared to older versions of the SDK. There is now a Linux shell script called `new_board.sh` designed to assist adding new boards.
- Do not modify code in the common or include directories. This code will be updated with each new release to follow code released by the open source U-Boot. The original U-Boot

source code can be found at http://www.denx.de/wiki/U-Boot/SourceCode. If there is no way to avoid updating these files the please contact CAVIUM so that future releases can provide hooks to avoid this.

- Avoid modifying code under `arch/mips/cpu` if possible. This code will be subject to frequent changes as new chips and features are added. The current exception is `arch/mips/cpu/lib_octeon_shared.c` for DDR2 and DDR3 memory initialization. Keep changes to this file to an absolute minimum since this file will also change in the future. Also, make sure to use `#ifdefs` or comments to make porting to future versions of U-Boot easier. A future version of U-Boot will implement a more portable way for adding board-specific entries. Instead, place all board-specific changes in the `board/octeon/<board_name>` directory.
- Avoid making unnecessary changes to the root `Makefile`, modify `boards.cfg` instead when adding new boards.
- Do not modify the new U-Boot code to allow old code to work. Instead modify the old code to fit into the new U-Boot framework wherever possible.
- Avoid adding new defines to change functionality. Instead use `__attribute__((__weak__))`. Please contact your FAE (field application engineer) or email u-boot@cavium.com when certain areas present problems so that future versions of U-Boot can take this into account to improve the porting process.
- Do not use any global variables until after U-Boot has been relocated. Until this time, only the `gd` data structure may be used. Attempts to use global variables before relocation will cause U-Boot to crash and debugging may be difficult. Sometimes these crashes are delayed until after DRAM is initialized making it difficult to debug.
- Do not add any significant data to the `gd` data structure. This data structure must reside entirely within cache until U-Boot has been relocated to DRAM.
- Do not use the device tree file from a different processor. Unlike the rest of U-Boot the device tree is both chip and board specific.
- Avoid using `OCTEON_IS_OCTEON2()` and instead use `!OCTEON_IS_OCTEON1PLUS()` to help make the code more future proof against future OCTEON III which will largely borrow from OCTEON II.

4 Virtual Memory Support in U-Boot

The OCTEON U-Boot implementation uses mapped virtual memory. This is unique to the OCTEON U-Boot and no other version does this. Mapped virtual memory is the most significant difference between the OCTEON U-Boot and standard U-Boot. Mapped memory was introduced in the SDK 2.0 and has not changed significantly in through SDK 3.0.

The changes required to the standard U-Boot code are minimal except for device drivers which perform DMA.

4.1 Introduction to Virtual Memory

U-Boot is a 32-bit application that was originally designed for 32-bit processors. While it is capable of running on 64-bit processors it always executes in 32-bit mode with 32-bit pointers.

U-Boot executes in two phases. During the first phase it executes out of flash, the L2 cache or from a RAM address chosen by the host Linux command `oct-remote-boot`. After the early initialization in U-Boot is complete it relocates itself to a new RAM location and begins executing from the new location.

The standard MIPS U-Boot normally executes out of KSEG0 which consists of the address range 0x80000000 through 0x9FFFFFFF or KSEG1 which consists of the address range 0xA0000000 through 0xBFFFFFFF. Both of these address ranges maps to physical addresses 0x00000000 through 0x1FFFFFFF. On the OCTEON processor, physical addresses 0x10000000 through 0x1FFFFFFF are reserved for the boot bus. This effectively limits U-Boot to being loaded in the first 256MB of RAM.

The standard MIPS U-Boot is normally linked to the address it will begin executing out of flash memory which is 0xBFC00000 or 0xB0000000, depending on the processor, in KSEG1. After RAM has been initialized it copies itself to RAM and updates its global offset table (GOT) to the location in RAM, typically in KSEG0. The RAM address is typically defined at compile time.

The global offset table is a feature of the ELF (Executable and Linkable Format) binary format that U-Boot uses. ELF executables are designed to allow for programs to be position independent by allowing the program loader to update addresses in the executable before execution begins. Since there is no program loader when U-Boot first begins executing, it must be linked to execute at the flash address it will start executing from.

The OCTEON U-Boot is linked at address 0xC0000000 instead of its address in flash. It relies on mapped memory so that 0xC0000000 will always map to the physical address where U-Boot is executing from. By using this mapping memory the OCTEON U-Boot is not restricted to a single location in flash nor is it restricted to the lower 256MB of physical memory. In fact, OCTEON U-Boot is not limited to 32-bit physical addresses.

An in-depth discussion on how virtual memory works on the OCTEON processor can be found in the ***OCTEON Programmer's Guide*** in the Software Overview chapter.

4.2 MIPS Translation Look-Aside Buffer

MIPS processors contain a small translation look-aside buffer (TLB) which is used to map virtual addresses to physical address. Most MIPS processors have between 16 and 64 entries in their TLB. The OCTEON can have between 32 and 128 entries. Each entry can map two contiguous virtual pages in memory to two physical pages. The physical pages do not need to be contiguous.

The OCTEON U-Boot is linked to execute out of the supervisory segment (SSEG) instead of its location in flash which is in KSEG1. SSEG consists of the address range 0xC0000000 through 0xDFFFFFFF. The OCTEON U-Boot is always linked at starting address 0xC0000000.

Before any C code executes, the assembly code in `arch/mips/cpu/octeon/start.S` clears the TLB and initializes the last entry to map two 4MB pages at the beginning of SSEG to the physical address U-Boot is currently executing from. The physical address can be any address in flash or RAM. This allows a single binary U-Boot image to execute from any location in flash or RAM without requiring it to be linked at a specific address.

4.3 Relocation and 64-bit Addressing

After U-Boot performs some early initialization where it typically executes out of flash or the L2 cache, it must copy itself to RAM.

By default, the standard U-Boot copies itself to a fixed location near the bottom of physical memory then updates the ELF GOT to the new address in KSEG0.

The OCTEON U-Boot instead copies itself to the top of physical memory and updates the TLB entry to map the new physical address to the virtual address 0xC0000000 in SSEG. It does not need to modify the GOT or perform any other relocation operations except for copying the `gd` and `bd` data structures.

Since the TLB is used, the physical addresses can be located anywhere and are no longer restricted to the lower 256MB of RAM. This allows for more of KSEG0 to be available for other purposes.

4.4 Virtual Memory Effect on Device Drivers

While using virtual memory has almost no impact on standard U-Boot operation, any device drivers that perform DMA must be aware of the physical to virtual address mapping.

For PCI devices, drivers should use the `virt_to_bus()` macro to convert virtual addresses to physical addresses. The Intel E1000 driver in `drivers/net/e1000.c` has been modified to work with the OCTEON U-Boot. All of the modifications can be found by searching for `CONFIG_OCTEON` in the code.

On OCTEON I and OCTEON Plus processors, `pci_virt_to_mem()` can return a 64-bit address which will cause problems if the PCI device does not support 64-bit addresses.

On OCTEON II and later if the `CONFIG_OCTEON_PCI_ENABLE_32BIT_MAPPING` option is set then the addresses returned will be 32-bit even when the physical address is not. The OCTEON is configured to remap PCI DMA accesses to the physical memory address range used by U-Boot. If this option is not set then the physical address space occupied by U-Boot will not be remapped. It is recommended that the `CONFIG_OCTEON_PCI_ENABLE_32BIT_MAPPING` option not be set unless it is required for a device that cannot perform 64-bit addressing.

For other non-PCI drivers such as the USB EHCI driver the above macro should not be used. Instead use the function `cvmx_virt_to_phys()`. Addresses returned may be 64-bit. The `pci_virt_to_mem()` macro cannot be used because DMA accesses will not be remapped.

Note that many U-Boot device drivers assume that the memory address stored in a pointer can be used for DMA. Even without using mapped virtual memory this will not work on MIPS unless the address is converted from KSEG0 to a physical address by stripping off the most significant bit.

4.5 Virtual Memory Example

For U-Boot, typically two 4MB pages are used for a total of 8MB. Most of this memory is needed by the heap, and the large heap size is needed for JFFS2 support. If JFFS2 support is removed then the heap size can be reduced down to 1MB..

The `octreginfo` command will display the TLB entry showing this mapping. For example, here is the output on an OCTEON EBB6800 board with 8GB of RAM installed:

```
127: Virtual=0xffffffffc0000000
        Page0=0x20f800000,C=0,D=1,V=1,G=1,RI=0,XI=0
        Page1=0x20fc00000,C=0,D=1,V=1,G=1,RI=0,XI=0
        ASID=  0 Size=4096KB
```

In this case,the last TLB entry (127) is used and two 4MB pages are created.

The flags indicate the following:

- The C flag indicates that the page is cacheable and coherent.
- The D flag indicates that the dirty bit is set which allows the page to be writeable.
- The V flag indicates that the page is valid.
- The G flag indicates that the page is global and not restricted to an ASID.
- The RI flag indicates that the page is not read inhibited.
- The XI flag indicates that the page is not executeable inhibited.

The 4MB page at 0xC0000000 is mapped to the 64-bit physical address 0x20F800000 and the address 0xC0400000 is mapped to 0x20FC00000. An in-depth discussion on how virtual memory works on the OCTEON processor can be found in the Software Overview chapter of the ***OCTEON Programmer's Guide***.

5 Quick Summary of Files and Directories to be Modified

The following files will need to be modified to support a new board. Use the `new_board.sh` script to create the new files. Most boards only require the below files and directories to be changed.

- `boards.cfg` – List of boards and whether PCI host mode is supported
- `board/octeon/<new_board>` - Directory containing board-specific files
- `board/octeon/<new_board>/<new_board>_board.c` – Board-specific code and commands
- `board/octeon/<new_board>/<new_board>.dts` – Device Tree Specification.
- `board/octeon/common/*.c` – Board helper functions which typically make use of the device tree for configuration purposes.
- `include/configs/octeon_<new_board>.h` – Board options and configuration

- `include/configs/octeon_<new_board>_shared.h` – Memory configuration and board EEPROM TWSI address (if it exists)
- `arch/mips/cpu/octeon/lib_octeon_shared.c` – Add memory settings for new board
- `cvmx-helper-board.c` – Modify to include network port information. This file is located in the Simple Executive directory (`OCTEON SDK/executive`)
- `cvmx-app-init.h` – Add new board ID. File is located in the Simple Executive directory `(OCTEON SDK/executive)`
- `MAKEALL` – Builds all boards to report all warnings and errors

6 Porting U-Boot to New Hardware

The easiest method to port to a new board is to select an existing board (donor board) and create a new directory, copying the files from the existing board. Choose an existing board based on the board's features rather than the particular OCTEON model. In most cases, the OCTEON U-Boot code determines the OCTEON model at run-time. The exception to this is that it is best to stay within the same family. Choose an OCTEON II based donor board if the new board will be using an OCTEON II processor or an OCTEON I/Plus based donor board if the new board will be using an OCTEON I or OCTEON Plus processor.

Usually, a good choice is to use the evaluation board for the processor being used. The generic_ram board is quite limited and may not be a good choice. See the OCTEON Development Board Details section on page 83 for a partial list of donor boards.

1. Start with the OCTEON Bringup and Diagnostic Kit (BDK) in order to make sure that the board is fully functional. This will aid in getting the correct settings for memory. Note that as of the SDK 2.1 that the BDK only supports OCTEON II (CN6XXX) devices and not OCTEON I or Plus devices (CN5XXX and earlier).

   All of the memory settings used in the BDK can be applied to U-Boot since both share the same memory initialization code.

2. Read and become familiar with the README file provided with U-Boot. It will answer many questions not answered in this document. The README file does not contain any changes or enhancements made by CAVIUM for the OCTEON processor.

3. Run the script `new_board.sh` in the main U-Boot directory from the Linux command line. Answer the questions and it will perform several steps automatically. **Do not modify an existing board.** The script will ask the following questions:

   a) What is the source or donor board to use?

   b) What is the name of the target board? The target board name must only contain the letters a-z, the numbers 0 to 9 and the underscore letter '_'. The name must start with a letter a-z.

   c) Does the new board support PCI or PCIe host mode?

After answering the questions it will use the donor board as a template for the new board and perform the following operations:

- Copies `include/configs/octeon_<donor board>.h` to `include/configs/octeon_<myboard>.h`.
- Copies `include/configs/octeon_<donor board>_shared.h` to `include/configs/octeon_<myboard>_shared.h`.
- Edits `include/configs/octeon_<myboard>.h` to include `octeon_<myboard>_shared.h`.
- Changes `__CONFIG_OCTEON_<DONOR BOARD>_SHARED_H__` to `__CONFIG_OCTEON_<MYBOARD>_SHARED_H__` in `include/configs/octeon_<myboard>_shared.h`.
- Adds a line for the new board to `boards.cfg`.
- Creates the directory `board/octeon/<myboard>`.
- Copies `board/octeon/<donor board>/<donor board>_board.c` to `board/octeon/<myboard>/<myboard>_board.c`.
- Copies `board/octeon/<donor board>/<donor board>.dts` to `board/octeon/<myboard>/<myboard.dts>`.
- Creates symbolic links in `board/octeon/<myboard>` for the `Makefile`, `config.mk` and `u-boot.lds`.

Running `new_board.sh` a second time will overwrite any changes made previously.

4. Edit `cvmx-app-init.h` in the simple executive directory by adding the new board to `cvmx_board_type_enum` and the function `cvmx_board_type_to_string()`.

   Use a board type between `CVMX_BOARD_TYPE_CUST_PRIVATE_MIN` and `CVMX_BOARD_TYPE_CUST_PRIVATE_MAX` or contact CAVIUM to obtain an entry for `cvmx_board_types_enum`. Add an entry to `cvmx_board_type_to_string`. Adding a string entry will allow U-Boot to automatically generate the correct prompt.

   Note that in SDK 2.2 and later this file will no longer need to be modified due to flat device tree support.

5. Edit the device tree file `board/octeon/<myboard>/<myboard>.dts` to change the board name.

6. Edit the file `cvmx-helper-board.c` if needed in the simple executive directory by duplicating any code for the chosen development board with the new board. With the addition of the device tree this is often no longer required.

   If the network and MII parameters are not defined in the device tree, edit `cvmx-helper-board.c` to modify the functions `cvmx_helper_get_mii_address()` and `__cvmx_helper_board_link_get()`. It is possible that several other functions in this file may need to be modified. Note that this is not necessary if the device tree is edited instead. The device tree is the preferred method. It is strongly recommended that instead of editing these functions that a device tree file be created instead.

7. Edit `arch/mips/cpu/octeon/lib_octeon_shared.c` to include `config/octeon_<myboard>_shared.h` and duplicate any places where `CONFIG_OCTEON_XXXX` is found with `CONFIG_OCTEON_<MYBOARD>`. Usually this involves adding the new board to `octeon_board_ddr_config_table[]` near the end of the file.
8. Attempt to build the new board using the Linux command `make octeon_<myboard>_config` followed by `make`. It should compile. If not, fix any errors until it does.
9. Load the new U-Boot image onto the evaluation board, assuming that the evaluation board was chosen as `XXXX` above. The `bootloaderupdate` U-Boot command should work to install the new bootloader. Note that the file `u-boot-octeon_<myboard>.bin` should be used. If the file fails to boot when booting out of flash then boot the failsafe image by toggling the switch for GPIO 0. The failsafe position is usually the off position.
10. Edit the include files to define the settings for the new board. If they conflict with the default settings in `include/configs/octeon_common.h`, rather than changing `octeon_common.h`, just `#undef` then re-define the setting. Feedback is also appreciated on any problems encountered here so future versions of `octeon_common.h` can add `#ifdefs` if the value has been previously defined. Note that the way commands are defined has changed from previous versions of the SDK and that most other settings have also changed names.
11. Edit the device tree file `board/octeon/<myboard>/myboard.dts`. See chapter 19 Flat Device Tree on page 42 for details.
12. Re-define any functions defined in the `board/octeon/common/` directory as needed for your board. All of the common functions have the weak attribute set so any new functions with the same name will take priority when linked. Do not edit the functions in `board/octeon/common` since this is not required. Instead replicate the needed functions in `board/octeon/<myboard>/<myboard>_board.c`.
13. Uncomment `PLATFORM_CPPFLAGS` in `board/octeon/<my board>`. Note that this may be a symbolic link to `board/octeon/common_make/config.mk`. This will enable address checking for all CSR register read and write operations. Remember to disable this feature before shipping since this increases the size of U-Boot and slows it down due to all of the additional checking.
14. Add the new board to the MAKEALL command.

While many OCTEON boards have a MCU present, the Embedded Planet EP6300C and SFF6100 boards do not. The EP6300C board also demonstrates how to add a board-specific device on the boot bus that is not used by other boards. Additionally, this board uses a 16-bit CFI flash rather than 8-bits. It does not make use of the True IDE mode that other boards utilize.

6.1 PCI and PCIe Target Boards

If the target board supports PCI/PCIe target mode then it is easiest to start out by copying the generic_ram board. The `lib_octeon_shared.c` file and `octeon_<myboard>.h` and `octeon_<myboard>_shared.h` files are used by the host Linux command `oct-remote-boot` and other remote utilities.

7 Porting from Earlier U-Boot Versions

7.1 Porting from U-Boot 1.1.1 (SDK 2.0) to U-Boot 2011.03 (SDK 2.1)

Prior to the SDK 2.1, U-Boot was based on version 1.1.1 which is very old. The SDK 2.1 moved to U-Boot version 2011.03.

1. Copy the board file `board/octeon_<board_name>_board.c` from the old U-Boot to `board/octeon/<board_name>/<board_name>_board.c`.
2. Create symbolic links from `board/octeon/common_make/*` into your directory for the `Makefile`, `u-boot.lds` and `config.mk` files. In most cases these can be used without modification. If modification is needed, then copy the file before modifying it instead of creating a symbolic link.
3. In `<board_name>_board.c` use search/replace and replace `gd->xxxx` with `gd->ogd.xxxx` for all OCTEON-specific fields, which is usually everything except `gd->flags`.
4. Update the include files to use `<asm/arch/xxxx>` for OCTEON-specific include files.
5. It might be easier to create the `octeon_<board_name>.h` file from scratch under `include/configs` rather than modify the existing include file from the old U-Boot. The easiest way is to diff between the custom `octeon_<board_name>.h` file and a similar OCTEON board file then make similar changes with the new U-Boot.
6. Add the appropriate `octeon_<board_name>_shared.h` file to `arch/mips/cpu/lib_octeon_shared.c` and support for the new board. `lib_octeon_shared.c` has not changed significantly in how configuration is handled. This will likely change in the future, however, as more portable methods are found to implement this.
7. Re-do any commands that were defined in the old header file since this has changed. Note: commands are no longer defined as a bitmap.

7.2 Porting from SDK 2.1 to SDK 2.2

Several changes were made in the SDK 2.2 release. The biggest change is the introduction of the flat device tree. The flat device tree provides a method of describing the hardware on a board by using a text file. This allows the Linux kernel and simple executive applications to be able to work with new hardware without requiring recompilation.

In order to migrate from SDK 2.1 to SDK 2.2 a flat device tree (.dts) file must be created under `board/octeon/<board name>`. The file should be named `<board name>.dts`.

Please see chapter 19 Flat Device Tree for details on the new device tree specification.

With the flat device tree modifications to the simple executive file `cvmx-helper-board.c` are no longer required.

7.3 Porting from SDK 2.2 to SDK 2.3

Very few changes were made between SDKs 2.2 and 2.3.

7.4 Porting from SDK 2.3 to SDK 3.0

The biggest change between SDK 2.3 and SDK 3.0 is how PHY devices are handled. Prior to SDK 3.0, all of the MDIO operations needed to initialize a PHY had to be performed in the board .c file. With SDK 3.0 most of these operations can now be specified in the flat device tree. U-Boot now fully utilizes the flat device tree for PHY initialization. For many boards the PHY initialization code can be completely eliminated as it can now take place automatically. If PHY initialization is still required then it should be implemented in `board_mdio_init()` instead of `checkboard()`. The default `board_mdio_init()` function is adequate for most boards. It makes use of the device tree to initialize the PHY devices.

Any code which previously used `cvmx_twsi_XXX` functions should now use the `i2c_read`/`i2c_write` functions provided by U-Boot. Be sure to set the bus using `i2c_set_bus_num()` first.

Any code which uses `cvmx_gpio_XXX` functions should now use the GPIO functions defined in `arch/mips/include/asm/gpio.h`.

Many of the settings which were previously required for boot bus devices are now handled entirely by the device tree.

The define `CONFIG_FLASH_USES_ALE` is no longer used. This is now automatically detected. It still must be defined in the device tree, however.

If a TLV EEPROM is present there are now helper functions to reduce the size of the code in early_board_init(). The functions are `octeon_board_get_clock_info()`, `octeon_board_get_descriptor()`, `octeon_board_get_mac_addr()` and `octeon_get_cpu_multiplier()`. The function which generates a random MAC address has also been significantly improved if the MAC address is not defined in the TLV_EEPROM.

8 Porting Details

The easiest way to port to a new board is to start with an existing board and copy it. A good example board is the Embedded Planet EP6300C because this board does not have a MCU yet implements most other features. The only drawback of this board is the fact that it does not implement True IDE mode if compact flash is desired. The best choice is likely the evaluation board being used. **Do not modify the code for existing boards.**

Most of the board specific functions are placed either within the board file or under `board/octeon/common/`. All of the predefined common functions are compiled with `__attribute__((__weak__))` to make it easy to replace the functions.

`board_fixup_fdt()`: This function performs any fixups needed to the flat device tree needed for the board. This function is optional. It is often used when certain features on a board can be changed such as pluggable networking modules. In this case based on the mode it prunes away the interfaces that are not used. In the device tree, fields that can be trimmed have the property "`cavium,qlm-trim`" set.

`octeon_boot_bus_board_init()`: This function is called to initialize any board specific boot bus devices that are not defined in `board/octeon/common/octeon_boot_bus.c`. The ep6300c has a FPGA called the BCSR which is enabled in this function. This function is optional and replaces a weakly defined function.

`octeon_cf_present()`: This function replaces a function defined in `octeon_board_cf.c` and returns if a CF card is inserted or not. It is not required. On this board the compact flash insertion status is obtained from a register on the BCSR FPGA.

`ide_set_reset()`: Since this board uses the FPGA to reset the IDE bus rather than a GPIO line, `ide_set_reset()` is defined which replaces the default one in `octeon_board_cf.c`.

`ide_led()`: If defined and `CONFIG_IDE_LED` is defined then this function will be called to enable and disable a LED for IDE disk access.

`checkboard()`: This is a required function that is called fairly late to check if the board is OK or not. It can also perform board-specific initialization. Return 0 for success.

`board_early_init_f()`: This function is defined for the ebb6300 board to perform some initialization very early in the boot process. It is optional and not all boards provide this function. It is called before the failsafe scan and boot bus initialization. Note that because it is called before the failsafe check it can be called twice. Since DRAM is not initialized, do not use any global variables, only the `gd` data strucsture is safe.

`early_board_init()`: This is a required function. This function is responsible for making sure any special features of the board that must be enabled early on are enabled. It also is responsible for filling in a number of required fields in the `gd->ogd` data structure and several `gd->flags`. It is called after the boot bus has been initialized but before the environment has been initialized. It is called before DRAM is initialized so it must not make use of any global variables.

On the EP6300C board, like all of CAVIUM's boards, there is a serial EEPROM that stores various parameters. These are read and used to fill in various fields in gd->ogd. If the board does not have a serial EEPROM or if the fields are missing then default values must be filled in. These values are typically stored in the board-specific configuration header file.

`octeon_led_str_write()`: The last function is `octeon_led_str_write()` which on this board is just an empty function since no LED display is present. If a display is present, this function is called to display an 8 character ASCII string.

`octeon_adjust_board_name()`: This function allows the name of the board which is used in the prompt to be changed at runtime. This is called fairly late after relocation to RAM has occurred. It is called for the EP6300C and EP6600C boards since the same U-Boot image is used for both boards.

`late_board_init()`: This function is called very late in the U-Boot process shortly before the prompt is displayed. This is available as a hook to perform any late initialization.

8.1 DRAM Configuration

A DRAM profile must be defined. See the `octeon_<board_name>_shared.h` configuration include files for examples for various boards. If no SPD is present on the board then the SPD values will have to be added in `octeon_<board_name>_shared.c` near the end of the file. The `octeon_<board>_shared.h` file will also have to be included into this file. The method of configuring DRAM is subject to change in future versions of U-Boot.

The DRAM configuration from the BDK should be used for U-Boot and should not require any modification.

Note that in addition to the BDK parameters that `CONFIG_OCTEON_DDR` must be defined to be either 2 or 3. For all OCTEON I and OCTEON Plus devices this must be defined as 2. For all OCTEON II devices this must be defined as 3.

8.1.1 Editing `include/configs/octeon_<board_name>_shared.h`

Most boards have a file named `include/configs/octeon_<board_name>_shared.h`. This file contains all of the DDR2 and DDR3 memory settings and the board serial EEPROM TWSI address, if present. This file is shared with the BDK and must be modified according to the board requirements. The file should be the same between the BDK and U-Boot.

8.2 Editing `include/configs/octeon_<board_name>.h`

The include file `include/configs/octeon_<board_name>.h` contains all of the board-specific configuration options other than those listed in section 8.1.1 above. Many options are already defined in the file `include/configs/octeon_common.h`. Edit this file as-needed to configure U-Boot for the target board. Most of the settings are listed in the U-Boot README file and in the Configuration Variables section on page 65. The configuration files for the EP6300C, EBB6300 and EBB6800 boards are the best documented boards at this time.

If changes are required to settings defined in `include/configs/octeon_common.h` then use `#undef` before redefining the settings. Do not edit `include/configs/octeon_common.h`.

9 Building U-Boot

All commands in this section are assumed to be from the Linux host command line. To build U-Boot for a particular board, type the following commands:

```
    % env-setup OCTEON_CN63XX
```

```
% cd bootloader/u-boot

% make clean

% make octeon_<board_name>_config

% make
```

A second method is also available. This method will configure the environment for the specified board as well as build the U-Boot binary image.

```
% make octeon_<board_name>
```

The `MAKEALL` command will build U-Boot for all of the boards and will display a list of all warnings and errors. Note that this does not preserve the finished binary of U-Boot.

One final method of building U-Boot is to type `make all` in the root SDK directory. This command will build U-Boot for all boards. The U-Boot binaries will be placed into the `target/bin` directory.

It is possible to perform parallel builds by adding the `-j` parameter to make, however it has been found that sometimes the build may fail. If an error occurs when trying to compile `board.c` in `arch/mips/lib` then the configuration has been lost or corrupted. If this happens, issue the following commands:

```
% make clean

% make octeon_<board_name>_config
```

This will reset the configuration back.

Sometimes `make clean` will be insufficient. If this occurs use the following commands:

```
% make distclean

% make octeon_<board_name>_config
```

The `make distclean` command will clear out any configuration that exists.

Before building U-Boot for a different board always run `make distclean` before making the new configuration.

10 Debugging

Many files have debugging support. To enable it, usually all that is required is to add `#define DEBUG` at the very beginning of the file. Note that it should be defined before `common.h` is included because many files use the `debug()` function which is defined in `common.h`.

The core of the U-Boot startup process is in `arch/mips/lib/board_octeon.c`. Defining DEBUG here will provide a lot of useful information to help debug where the boot process is failing. In this case, change `#undef DEBUG` to `#define DEBUG` at the very beginning of the file.

To debug issues with the boot bus consider enabling the `octbootbus` command and enabling debugging in `board/octeon/common/octeon_boot_bus.c`.

10.1 Trap Dumps

In some cases, a trap dump will be printed out the serial port if a crash occurs. Refer to the map file `System-octeon_<board>.map` to determine what function caused the trap. U-Boot uses the MIPS N32 application binary interface (ABI). The CPU register 0x1F (r31) may contain the address from where the function that crashed was called from. The registers 0x4 (a0) through 0xB (a7) contain the first 8 arguments to the function.

Note that the exception handler executes out of the failsafe version of U-Boot. U-Boot prior to the 2.1 SDK will not properly display the registers. If the failsafe version of U-Boot is older than 2.1 then only the status, cause, epc and badvaddr fields will be accurate, all other registers may not be accurate.

The `K0` and `K1` (`$26` and `$27`) registers will not be accurate. These are used internally by the trap handler.

If the trap is due to a NMI or a watchdog interrupt then this will be displayed after the status value. Future releases of the SDK may expand the information displayed to make debugging easier. Note that watchdog support will likely not be available until the SDK 2.2.

A write operation to some location in DRAM before DRAM has been initialized may cause the OCTEON to lock up when the DRAM is cleared. This can be difficult to track down. This is often caused by a write to a global variable.

The status and cause fields are defined in the Coprocessor 0 Registers section in the ***MIPS64 Architecture for Programmers Volume III*** PDF available from http://www.mips.com/. The `badvaddr` field will display the bad virtual address that caused the exception (if due to an invalid load or store address) and `epc` will contain the address of the failing instruction.

Note that not all cases will produce a trap. Also, since the trap handler is often contained in the failsafe bootloader, upgrading the failsafe bootloader may fix some trap dump issues.

10.2 NULL Pointers

NULL pointers are not trapped in U-Boot. This can make debugging particularly difficult since the result may not be visible until much later. For example, writing to a NULL pointer before U-Boot has been relocated to RAM can cause U-Boot to lock up when it later flushes the cache.

10.3 Debugging Memory

When debugging memory issues there are several helpful tools in U-Boot. The first is the U-Boot environment variables `ddr_verbose` and `ddr_trace_init`. Setting `ddr_verbose` will display verbose information during the initialization sequence and setting `ddr_trace_init` will display all register read and write accesses. These are used in `arch/mips/cpu/octeon/lib_octeon_shared.c`.

Another helpful option is to change `DO_SIMPLE_DRAM_TEST_FROM_FLASH` from 0 to 1 in `arch/mips/lib/board_octeon.c`. This will enable a memory test after initializing memory.

10.3.1 Debugging Memory over PCI or EJTAG

If the board is capable of booting over the PCI or PCIe bus it will make memory bring-up easier. After making changes to `include/configs/octeon_<myboard>_shared.h` recompile the files in `../../host/remote-utils` and use the `oct-remote-ddr` and `oct-remote-boot` Linux programs to initialize and test the memory. These commands can also be used with EJTAG. The `oct-pci-ddr` and `oct-pci-boot` Linux programs are shortcut scripts for `oct-remote-ddr` and `oct-remote-boot` respectively. The environment variables listed above will also work with the Linux command-line tools. Using EJTAG can be very slow. It is not unusual for the memory configuration to take an hour or more on an OCTEON CN63XX board, for example[1]. PCI and PCIe are *much* faster.

11 Important Files and Directories

11.1 Board-Specific Files and Directories

All of the board-specific files are in the `board/octeon/<board_name>` directory and `board/octeon/common` directory. In previous versions of U-Boot these files were located in `board/octeon_<board_name>` and there was no common directory.

All of the functions in the common files have the "weak" attribute set so that any board may override the weak functions during the link stage. Rather than modify the common code, replace the function in the board-specific file. All of the files in the common directory are compiled by default. It is not recommended that these files be modified or that additional files be added here.

Table 11.1: Common Board-Specific Files

Common Board Files	
`octeon_board_cf.c`	Compact flash code to initialize, reset and check if a Compact Flash card is present.
`octeon_board_display.c`	This code displays board information in the console.
`octeon_boot_bus.c`	This code initializes devices on the boot bus. Board-specific initialization occurs in the respective board files. This file also makes extensive use of the device tree. The `octbootbus` command can be enabled to allow viewing and changing of boot bus parameters from the command line.

[1]CAVIUM has had seen delays of this length using a Macraigor MP Demon EJTAG device connected by Ethernet. Other devices may vary.

Common Board Files	
`octeon_board_common.c`	This file contains many helper functions used by many boards.
`octeon_board_dhcp.c`	This contains the NetOurBootCommand DHCP functionality where a boot command may be specified by the DHCP server via option 224.
`octeon_board_display.c`	This contains the `display_board_info()` function which displays board information early in the U-Boot boot process.
`octeon_board_mmc.c`	This file contains the function `octeon_boot_bus_mmc_get_info()` which parses the device tree to configure the MMC device.
`octeon_board_pci.c`	This file contains code to initialize board-specific PCI or PCIe support. Currently the function `pci_init_board()` just calles `init_octeon_pcie()` or `init_octeon_pci().`
`octeon_board_phy.c`	This file contains code for parsing the device tree in order to configure various PHY devices. Currently it supports the Vitesse VSC848X, Marvell 88e11xx and 88e1240, and various Broadcom PHYs. For Vitesse PHY devices it also contains code to read the SFP+ serial EEPROM to configure the PHY device for copper or optical modules. Additionally, it also performs some QLM configuration needed for Vitesse.

Inside each board directory there is a file called `<board_name>_board.c`. This file contains all of the board-specific initialization code. Note that by replacing the symbolic link to the `Makefile` with a new `Makefile` it is possible to add multiple files for a given board. Another important file is the device tree file. The device tree file is named `<board_name>.dts`.

11.2 The `common` Directory

The `common` directory contains the standard U-Boot files. All of the commands and environment handling code is located here. Also, common code for heap management, USB, command-line parsing and other features are also located in this directory.

11.3 The `drivers` directory

The `drivers` directory contains most of the drivers within various subdirectories. This includes drivers for USB, the real-time clock, temperature sensors, voltage monitors, flash and more. Note that many platform-specific drivers are not located here but are in `arch/mips/cpu/octeon` instead.

11.4 The `lib` directory

The `lib` directory contains various library files for common operations. The flat device tree library is in `lib/libfdt`.

11.5 The `disk` Directory

The `disk` directory contains code for handling various partition types such as dos, efi, iso and Mac. Only the MS DOS partition type is supported in the OCTEON U-Boot.

11.6 The `fs` Directory

The `fs` directory holds most of the filesystem code. Currently the OCTEON U-Boot uses the ext2, fat and jffs2 filesystems. Note that the partitioning code is stored in the `disk` directory.

11.7 The `include` directories

There are several include directories and several symbolic links to those directories.

11.7.1 `arch/mips/include/asm`

This directory contains the MIPS-specific include files. Most of these files are generic for MIPS, though `mipsregs.h` has been modified to include a number of OCTEON-specific CP0 registers and `global_data.h` has been slightly modified. A symbolic link to this directory is created in `include` so that `include/asm` is a symbolic link to this directory.

Do not change any files in this directory.

11.7.2 `arch/mips/include/asm/arch-octeon`

This directory contains all of the OCTEON-specific include files including symbolic links to the simple executive include files.

The arch directory is set up to be a symbolic link to `arch-octeon`, so `arch/mips/include/asm/arch` is the same as `arch/mips/include/asm/arch-octeon`. Similarly, `include/asm/arch` is the equivalent of `arch/mips/include/asm/arch-octeon`.

Try and avoid changing any files in this directory.

11.7.3 `include`

This directory is the main include directory for U-Boot. Most of the common include files are located here. Avoid changing any files in this directory and all directories not part of `include/configs`. Adding new files for new common drivers that are not OCTEON-specific is acceptable here.

11.7.4 `include/configs`

This directory contains all of the board configuration files. There are two common files, `octeon_cmd_conf.h` and `octeon_common.h`. The `octeon_cmd_conf.h` file contains all of the common command definitions and `octeon_common.h` contains all of the definitions that are common to most boards.

Each board typically has two configuration files. The first is `octeon_<board_name>.h` and the second is `octeon_<board_name>_shared.h`. The first file must follow this naming convention and contains all of the board-specific configuration settings. The shared file contains the memory configuration as well as the board EEPROM TWSI address. It is used by `lib_octeon_shared.c`. If two boards are similar in their memory settings, both may use the same shared file.

Try and avoid changing `octeon_cmd_conf.h` and `octeon_common.h` and the files belonging to other boards. Create a new file for the new board.

11.8 Architecture-Specific Files and Directories

All of the OCTEON-specific code is located in `arch/mips/cpu/octeon` and `arch/mips/include/asm/arch-octeon` directory trees.

OCTEON-specific commands are located in `arch/mips/cpu/octeon/commands` and symbolic links to the simple executive .c files are placed in `arch/mips/cpu/octeon/cvmx`.

Table 11.2: `arch/mips/cpu/octeon` Files

arch/mips/cpu/octeon Files	
`cpu.c`	This file contains some CPU-specific functions for managing the translation look-aside buffer (TLB), initializing the network, flushing the cache and resetting the chip.
`dfm.c`	This file contains code for initializing the DFM memory.
`env_octeon_nand.c`	This file is used to process the environment stored in NAND flash when a NAND stage 2 bootloader is built. It is currently not yet supported.
`interrupts.c`	This file contains empty functions for enabling and disabling interrupts. Interrupts are not currently used.
`lib_octeon.c`	This contains miscellaneous functions. This file will be cleaned up in the future and the functions will be moved elsewhere.
`lib_octeon_shared.c`	This file is used both by U-Boot and by oct-remote-boot. Most of the functionality is related to initializing the DDR memory. This file will likely be separated into several files for handling DDR2 and DDR3 memory in the future.
`memcpy.c`	This contains an OCTEON-optimized version of the memcpy function.
`memset.c`	This contains an OCTEON-optimized version of the memset function.
`mips_linux.c`	This file is currently not used.
`octeon_bist.c`	This file contains the built-in self test functions which make use of the OCTEON

arch/mips/cpu/octeon Files	
	BIST features.
`octeon_boot.c`	This file contains functions used for booting simple executive applications. This file is also shared with applications other than U-Boot.
`octeon_cf.c`	This file contains OCTEON-specific functions for accessing the compact flash on the boot bus.
`octeon_crc32.c`	This contains the crc32 function optimized for the OCTEON.
`octeon_env.c`	This contains OCTEON environment related code. Currently it just contains a hook that gets called whenever an environment variable is changed and switches the console UART.
`octeon_eth.c`	This contains the OCTEON-specific Ethernet drivers.
`octeon_exec.c`	This contains functions that are called for go and exec commands to perform cleanup and replace the default U-Boot functions.
`octeon_fdt.c`	Octeon flat device tree support functions.
`octeon_flash.c`	This contains the code to initialize the environment from flash on the OCTEON processor. It differs from the common environment initialization by ignoring the environment in flash if the failsafe bootloader is booting.
`octeon_ipd_bp_enable.S`	This contains code to work around an errata for SPI-400 backpressure. It is currently not used.
`octeon_ipd_bp_verify.c`	This contains code to verify the SPI-400 backpressure work around. It is currently not used.
`octeon_nand.c`	This contains the OCTEON-specific NAND flash driver.
`octeon_pci.c`	This contains the OCTEON-specific PCI support for accessing configuration space and initializing the PCI bus.
`octeon_pci_console.c`	This contains the console support for the PCI and bootcmd console interfaces.
`octeon_pcie.c`	This contains the OCTEON-specific PCIe support for accessing configuration space and initializing the PCIe bus.
`octeon_wd.c`	This contains the OCTEON watchdog driver.
`start.S`	This contains the assembly language code called during initialization as well as several other routines used within U-Boot. The trap and NMI handler code is also

arch/mips/cpu/octeon Files	
	located in this file.
uasm.c	This contains a micro-assembler that is used by the watchdog driver to dynamically assemble a new boot vector for handling NMI interrupts.

Table 11.3: `arch/mips/cpu/octeon/commands` Files

arch/mips/cpu/octeon/commands Files	
cmd_ddr.c	This is currently not used as of version 2.1.0. It is for the NAC32 board which currently is not supported.
cmd_eeprom.c	This is currently not used as of version 2.1.0. It has been replaced with `cmd_octeon_tlveeprom.c`.
cmd_elf64.c	This adds 64-bit ELF support and is based off of the common cmd_elf.c file.
cmd_octeon_bootbus.c	This contains support for the `octbootbus` command to display and modify the boot bus configuration.
cmd_octeon_bootloader.c	This adds support for updating and verifying the OCTEON bootloader in flash.
cmd_octeon.c	This contains the `bootoct` and `dram_init` commands. The `dram_init` command currently is not used.
cmd_octeon_csr.c	This will add the `octcsr` command. Currently the command cannot be used as of version 2.1.0 because the database causes U-Boot to grow too large. Changes to `cvmx-csr-db.c` are needed in order to reduce the database size.
cmd_octeon_eraseenv.c	This file adds the `eraseenv` command which erases the flash environment sector.
cmd_octeon_linux.c	This implements the `bootoctelf` and `bootoctlinux` commands.
cmd_octeon_mem.c	This implements the commands `namedalloc`, `namedfree`, `namedprint`, `freeprint`, `flush_l2c`, `inv_icache` and `flush_dcache` commands.
cmd_mem64.c	This implements 64-bit versions of various memory commands such as `md64`, `mw64` and `mm64`.
cmd_octeon_nand.c	This implements the `octnand` command.

arch/mips/cpu/octeon/commands Files	
`cmd_octeon_reginfo.c`	This implements the `octreginfo` command which displays all of the CP0 registers and TLB entries used on core 0. It is similar to the `oct-remote-core` command on the host or in Linux.
`cmd_octeon_rw.c`	This implements all of the `read64` and `write64` commands to access 64-bit addressable memory.
`cmd_octeon_tftp.c`	This adds an alias for the `tftp` command since otherwise it would be confused with the new `tftpput` and `tftpserv` commands.
`cmd_octeon_tlveeprom.c`	This implements the `tlv_eeprom` command.
`cmd_qlm.c`	This implements the `qlm` command.

12 Memory Map

There are several stages involved in U-Boot. During the first stage, U-Boot executes in place out of flash memory without access to DRAM.

The bootloader then enables the local cache of the OCTEON processor to behave as local memory and maps the flash from 0x1FC00000 to the virtual address 0xC0000000 via a TLB entry.

After `board_init_f()` completes, U-Boot is copied to the top of physical memory and the stack, heap, `bd` and `gd` data structures are set up.

Table 12.1 below displays a virtual memory map as seen from within U-Boot.

Table 12.1:U-Boot Virtual Memory Map

U-Boot Virtual Memory Map
0xF0000000 – 0xFFFFFFFF PCI/PCIe configuration address space (port 1)
0xE0000000 - 0xEFFFFFFF PCI/PCIe configuration address space (port 0)
Unused
0xC0XXXXXX Board Data (`bd`) data structure
0xC0XXXXXX U-Boot Global Data (`gd`) data structure
0xC0XXXXXX U-Boot parameters (128K)

U-Boot Virtual Memory Map
0xC0XXXXXX U-Boot Stack (16K)
0xC0XXXXXX U-Boot Heap (4MB)
0xC0000500+ U-Boot Code
0xC0000500 U-Boot reset (start.S)
0xC0000000 U-Boot Code _start
Available
0x20000000 Load Address
0x1FC00000 – 0x1FFFFFFF 4MB initial NOR flash boot block
0x10000000 – 0x1FFFFFFF Boot Bus (256MB)
0x081000000 – 0x0FFFFFFF Available
0x00100000 – 0x080FFFFF Linux Reserved Address (128KB)
0x000FFFF0 – 0x000FFFFF Idle Core Loop
0x000A0000 – 0x000BFFFF Little Endian Flat Device Tree
0x00080000 – 0x0009FFFF Flat Device Tree
0x48108 – 0x0007FFFF Reserved low memory (~223k)
0x48100 – 0x48107 Bootmem Descriptor Address
0x48000 – 0x480FF PCI Console Read Buffer
0x40000 – 0x47FFF Debug Stack Area (32K) (32 * 1K)
0x20000 – 0x3FFFF Debug Save Area (128K) (32 * 512 * 8)
0x3000 – 0x1FFFF (reserved)
0x2000 – 0x2FFF U-Boot temporary environment copy for oct-remote-boot

U-Boot Virtual Memory Map
`0x1000 - 0x1FFF` U-Boot Environment passed from oct-remote-boot
`0x00000C00` Debug Trampoline
`0x00000800` Boot Vector Base (1K)
`0x00000000` - `0x000007FF` Reserved

13 Boot Sequence

Immediately out of reset, the CPU will branch to reset in `start.S`.

13.1 *Reset sequence in* `start.s`

1. Enable 64-bit mode for CSR access.
2. Speed up the flash.
3. Disable Authentik ROM.
4. Initialize the local L1 cache for scratch memory.
5. Initialize various CP0 registers
6. Initialize all of the TLB entries
7. Enable COP0 and COP2
8. Enable all read hardware locations
9. Enable unaligned memory access fixups
10. Set up scratch memory for stack and global data
11. Clear performance counters
12. Calculate GP address
13. Calculate address of `_start`
14. Check if running out of flash, cache or RAM.
15. If flash, check if OCTEON II and if the L2 cache is bigger than the U-Boot image.
16. If L2 cache is large enough and OCTEON II then write memcpy code to L2 cache then copy U-Boot to L2 cache.
17. Calculate physical and virtual address of U-Boot.
18. Add a TLB entry to map virtual address 0xC0000000 to the 4MB aligned physical address of U-Boot, whether that is in flash or RAM.

19. If running out of flash, configure the boot bus mapping for flash to 0x10000000 so it is aligned for a 4 megabyte physical page.
20. If not core 0, DRAM has been set up so branch to boot vector address.
21. Initialize the GOT (global offset table) pointer so global symbols can be used rather than PC relative addressing.
22. Branch to `board_init_f()` in `arch/mips/lib/board_octeon.c`

13.2 board_init_f()

The `board_init_f()` function is the first C function called from the assembly language start code in `start.S.` At this point execution can be either out of flash or RAM depending on if the board is being booted remotely or not. This function assumes that execution is out of flash but it works the same if it is running out of RAM. It performs all of the initialization prior to relocating to RAM.

1. Initialize the `gd` data structure
2. Initialize the TWSI bus
3. Start BIST testing
4. Run through the `init_sequence` array. If any function returns non-zero then an error is assumed to have occurred and the board will be reset.
    5. `board_early_init_f()` – this is an optional function to perform early board initialization. It should be implemented in `board/octeon/xxxx/xxxx_board.c`. The default function does not do anything.
    6. `board_fdt_init_f()` - This performs any early flat device tree operations that are board specific. Note that only read-only operations can be performed at this time because the device tree is stored in flash. The default function does not do anything.
    7. `failsafe_scan()` – This looks for a second image after the current one if running out of FLASH. If it is found then it branches to the normal image, otherwise it is assumed we are running in failsafe mode.
    8. `octeon_boot_bus_init()` initializes the boot bus. This is defined in `board/octeon/common/octeon_boot_bus.c` and is defined to be a weak function and may be replaced for specific boards.
    9. `octeon_boot_bus_moveable_init()` sets up support for the debugger and reset vectors by writing to local memory mapped to the boot bus for all of the additional cores.
    10. `timer_init()` initializes timer support. Note that this is the standard U-Boot MIPS code with only minor changes to not use global variables since DRAM may not be initialized.

11. `early_board_init()` performs early board initialization. This is defined in `board/octeon/xxxx/xxxx_board.c`. It initializes fields in `gd`, sets up voltages, clock values among other things.
12. `env_init()` initializes the environment variables. It will try and obtain the environment variables from either flash or RAM and depends on `CONFIG_ENV_IS_IN_FLASH, CONFIG_ENV_IS_IN_RAM` or `CONFIG_ENV_IS_NOWHERE`. If it cannot obtain the environment variables it will use the default environment variables.
13. `import_ram_env()` reads any environment variables passed via `oct-remote-boot`. If not booting out of RAM this function does not do anything, otherwise it will pre-pend any RAM environment variables to override existing environment variables.
14. `init_baudrate()` initializes the serial port baud rate to the default baud rate or the rate set in the environment.
15. `serial_init()` initializes the serial port.
16. `console_init_f()` enables console support.
17. `display_banner()` displays the U-Boot version information. The "version" command will display the same string.
18. `octeon_bist()` performs most of the OCTEON Built-In Self Test with the exception of PCI and USB. USB BIST tests occur later when USB is initialized.
19. `init_dram()` initializes DRAM and zeros it out if not booting remotely. It does not zero out the address used by oct-remote-boot to pass environment information if booting remotely.
20. `init_dfm()` initializes DFM memory if it exists.
21. `checkboard()` is defined in `board/octeon/xxxx/xxxx_board.c` and is used for late initialization and checking if a board is valid or not.
22. `display_board_info()` is defined in `board/octeon/common/octeon_board_display.c` and is a weak function that displays the board name, revision, serial number, core clock, I/O clock and DDR clock information.

23. Calculate top of memory now that DRAM is initialized.
24. Calculate amount of memory to reserve for U-Boot.
25. Zero out memory where U-Boot will be placed.
26. Calculate addresses for stack and heap.
27. Allocate space for `gd` and `gd->bd`.
28. Initialize `bd` structure.

29. Copy bd and gd structures to their future location.
30. Branch to assembly code to relocate U-Boot to the top of memory via relocate_code_octeon in start.S.

This function will likely change in the future and more functionality will hopefully be moved into the init_sequence array.

13.3 relocate_code_octeon()

This assembly function is defined in start.S. It is the same as relocate_code() except that the TLB page size is also passed as a parameter. It performs the following operations:

1. Copies code from current location to the new location at the top of memory.
2. Adds a TLB entry to map U-Boot's virtual address 0xC0000000 to the new physical location.
3. Branches to board_init_r() in arch/mips/lib/board_octeon.c

13.4 board_init_r()

The board_init_r() function continues initializing the board after U-Boot has been relocated to RAM. At this point global variables are available for use.

1. Set gd pointer to the new address and indicate that relocation is done.
2. Calculate relocation offset. Should be zero.
3. Remap flash on boot bus back to standard location since it no longer needs to be page aligned.
4. Initialize heap. After this point the heap may be used.
5. Use addrmap_set_entry for mapping virtual addresses to physical addresses.
6. Set up mapping for PCI BAR addresses. PCI interface 0 is mapped to 0xe0000000 and interface 1 is mapped to 0xf0000000. This is different than what it may have been before. Linux will remap it again later so this is only for within U-Boot.
7. Disable the watchdog timers on all cores.
8. Zero out all memory above DRAM_LATE_ZERO_OFFSET if CONFIG_NO_CLEAR_DDR is set. This is usually not used.
9. Clear any ECC error bits in the memory controllers.
10. Copy the flat device tree from flash to the low memory address reserved for it and increase the size of the flat device tree.
11. Initialize the SDK physical memory freelist.
12. Allocate a named block where U-Boot's code and data are located in physical memory to reserve it from the SDK.

13. Initialize the `cvmx_sysinfo` data structure, which is needed by Simple Executive functions.
14. Set up the command prompt.
15. If the environment is always in RAM, look for it and import it.
16. If booted out of RAM, look for environment passed by boot loader and import it again to override any variables previously set from defaults or from flash. Zero out the memory when done.
17. Allocate space for idle core loop for other cores.
18. Calculate address for reserved load address and reserve it from SDK.
19. Reserve memory for the Linux kernel.
20. Initialize idle loop code with wait loop.
21. Initialize pointer to bootloader bootmem descriptor.
22. Initialize and start the hardware watchdog if enabled.
23. Probe for NAND FLASH
24. Probe NOR flash
25. Initialize NAND flash and print out size.
26. Perform flat device tree fixups which also initializes the fdt /memory section.
27. Read `coremask_override` from serial EEPROM if present.
28. Set `numcores` environment variable.
29. Initialize Ethernet MAC addresses from `ethaddr` environment variable.
30. Set IP address from `ipaddr` environment variable
31. Initialize PCI busses and scan for devices.
32. Initialize `stdio`.
33. Configure `jumptable`. Note that the `jumptable` isn't used on OCTEON.
34. Initialize console after relocation.
35. Set `stdin`, `stdout` and `stder` environment variables. If a PCI console is possible then `stdin` is initialized to support serial, pci, bootcmd so it can accept input from any of the three. Note that `stdout` and `stderr` default to only serial output.
36. If `pci_console_active` is set, print out the board information again since the earlier output will have been lost.
37. Set the load address global variable.
38. Find the address and size of the environment and u-boot in flash and set the appropriate environment variables.

39. Initialize networking
40. Perform any miscellaneous platform initialization
41. Initialize and scan the IDE bus for compact flash.
42. Initialize the temperature sensor.
43. Initialize and scan for USB devices (does not scan for storage devices)
44. Verify that the boot vector is the correct size
45. Perform any late board initialization (if any).
46. Initialize the `serial#` environment variable.
47. Enter the main event loop.

14 The Global Data `gd` data structure

The `gd` data structure is designed to hold global data early on before U-Boot has been relocated to RAM. Until relocation occurs, global variables may not be used so the `gd` data structure is located in the local cache memory. After memory has been initialized the `gd` data structure is copied into RAM. The data structure is defined in `arch/mips/include/asm/global_data.h` and `arch/mips/include/asm/octeon_global_data.h`.

Be careful about adding entries to this data structure since it is subject to change in future releases. This data structure is also limited in size since it must first be located in the L1 cache along with the stack.

The pointer to `gd` is special and is always stored in the `k0` register.

To use `gd`, add the line

```
    DECLARE_GLOBAL_DATA_PTR;
```

at the beginning of the .c file. Do not declare this within a function like it was done in previous versions of U-Boot.

All OCTEON-specific fields except flags are stored in the `ogd` structure within gd.

Before starting Linux and simple executive applications the function `octeon_translate_gd_to_linux_app_global_data()` in `arch/mips/cpu/octeon/octeon_boot.c` is called to translate it to the old `gd` format before copying it to a special location in memory for compatibility with older applications and Linux kernels.

15 Porting U-Boot PCI Drivers

Unlike other platforms, virtual memory is used in U-Boot so that U-Boot can be loaded at the top of memory. This means that when programming addresses for DMA operations that the addresses must be translated. The macro `pci_virt_to_mem()` should be used.

On OCTEON II processors, memory used by U-Boot's code, stack and heap will be mapped starting at address 0 so that 32-bit addresses will be returned. On OCTEON I processors this functionality does not currently work and is still being debugged. In order to use PCI drivers on OCTEON I processors the macro will return a 64-bit address and requires that the PCI device supports 64-bit addressing.

Currently only two PCI drivers have been modified to work on the OCTEON processor, the Intel E1000 driver and the Silicon Image sil3124 driver. The E1000 driver can be found in `drivers/net/e1000.c` and `drivers/net/e1000.h`. The Silicon Image sil3124 driver may be found in `drivers/block/sata_sil3124.c`. Note that as of SDK 2.1 that the Silicon Image sil3124 driver does not work with the sil3124 PCI-X chip. It does work with various Silicon Image PCIe SATA controllers such as the SIL3132 and SIL3531 chips. The sil3124 driver is still under development and is subject to future fixes and cleanup. This driver does not contain #ifdef CONFIG_OCTEON since currently it only supports the Cavium U-Boot.

The supplied E1000 driver differs from the stock U-Boot 2011.03 E1000 driver in that a patch has been applied which adds support for some newer E1000 adapters. All of the OCTEON-specific changes can be found by searching for `#ifdef CONFIG_OCTEON`. The only change required was that the addresses now use the `pci_virt_to_mem` macro for DMA addresses. The old code would not have worked on a MIPS processor anyway since it did not account for the fact that KSEG0 is not the same as a physical address.

Note that the E1000 driver suffers since it has only a single receive buffer which is sensitive to network traffic while processing transactions. This can cause packets to be dropped. It is therefore better if one of the OCTEON network interfaces is used instead for management purposes.

16 USB EHCI Support

The standard U-Boot EHCI driver needed to be modified in order to work on the OCTEON II processor. All of the changes are surrounded by `#ifdef CONFIG_OCTEON`.

On OCTEON, all addresses used for DMA must first be translated by using `virt_to_phys()`. This is different from the PCI driver since addresses used by PCI devices may be translated by the hardware. This is not the case for the EHCI hardware where there is no address translation layer.

The USB driver assumes that several key data structures reside within the same 4GB segment. This is a limitation of EHCI but should not present a problem. Where possible the EHCI driver has also been enhanced to support 64-bit addressing.

17 Adding New Commands

Adding new commands in U-Boot is relatively easy. To add a command do the following:

1. Add a file under `board/octeon/<board_name>` that will be used for the command or modify an existing file in the same directory. Try and avoid adding the command elsewhere since this command is only specific to `<board_name>`. The OCTEON-specific commands in `arch/mips/cpu/octeon/commands` are used for multiple OCTEON boards and are not specific to any one board or vendor. The commands in the `common`

directory are supplied with the open-source U-Boot release and also should not be modified.

2. Add the new file to `board/octeon/<board_name>/Makefile`. If `Makefile` is a symbolic link, replace it with a copy of `board/octeon/common_make/Makefile`.
3. Include `config.h`, `command.h`, `common.h` and any other needed include files.
4. Create a new function with the function prototype:

```
int do_mycommand(cmd_tbl_t *cmdtp, int flag,
                 int argc, char * const argv[]);
```

5. Fill in the macro U_BOOT_CMD, for example:

```
U_BOOT_CMD(my_command, 1, 1, do_mycommand,
           "short help for my command",
           "usage for my command\n");
```

The first parameter is the string name of the command, without quotes.
The second parameter is the maximum number of arguments for the command.
The third parameter indicates whether or not the command should be repeated on enter.
The fourth parameter is the name of the function to call.

18 U-Boot Networking and TCP/IP

The TCP/IP networking stack in U-Boot is extremely limited. The only time packets can be received is when a response is expected. This works fine for DHCP, NFS, TFTP and other client/server type applications. Most of the functionality can be found in `net/net.c`.

U-Boot will not respond to ICMP ping requests and currently only UDP and Cisco Discovery Protocol are supported.

Cisco Discovery Protocol (CDP) is not fully configured. See the CDP options in the U-Boot `README` file for details.

All of the network-related code can be found in the `net` directory. The OCTEON Ethernet drivers can be found in `arch/mips/cpu/octeon/octeon_eth.c`.

Most network operations can be canceled by typing `<CONTROL> + C` at the command prompt.

19 Flat Device Tree

Starting in SDK 2.2, the flat device tree has been added to U-Boot. The flat device tree allows the hardware to be defined in a blob (binary large object) that is linked to U-Boot and passed to the Linux kernel and simple executive applications. It allows for new hardware to be supported without requiring that applications or the Linux kernel be recompiled.

The OCTEON device tree closely follows the Power.org ePAPR standard. Note that as of the 2.2 SDK that not everything is supported and not all required fields are initialized in the tree. The OCTEON specific sections are documented in the OCTEON Linux kernel `Documentation/devicetree` directory.

The easiest way to start working with the flat device tree is to copy a device tree file from a similar board based on the same processor. The dts file is much more sensitive to the exact processor used than other parts of U-Boot since there is no abstraction for register addresses or interrupt controllers.

The basic unit of the device tree is a cell. A cell contains a 32-bit value. For 64-bit addresses and values two cells are used.

The most common sections of the device tree that need to be edited are the boot bus, twsi bus, mdio bus, USB and pip interfaces. NAND flash and PCI are not currently defined or used in the flat device tree.

At the beginning of the flat device tree is the model. The model is set to the manufacturer and board name separated by a comma. For example, the Cavium ebb6600 board has model="cavium,ebb6600"; set. The compatible field should also be set and should be set to the same name as the model.

The #address-cells field specifies the number of cells required to contain an address in all children. The default is two cells for 64-bit addressing. The #size-cells field specifies how many cells are needed to contain a size. Again, two cells are needed to contain 64-bit sizes.

The interrupt-parent field specifies the interrupt controller used. Most OCTEON devices use the ciu unit but the CN6800 uses the ciu2 unit. The ciu or ciu2 section will be defined later. The &ciu and &ciu2 values are references to labels used later.

19.1.1 soc – System on Chip

The soc@0 field refers to the system on chip. All other OCTEON parameters are stored within this.

The compatible field should be set to "simple-bus".

19.2 Boot Bus Definitions in the Flat Device Tree

The boot bus defines the addresses for all of the chip enables as well as the timing parameters for various devices.

The easiest way to describe the boot bus section of the flat device tree is by example.

The following is an example of the flat device tree section from the Embedded Planet EP6300c board:

```
bootbus: bootbus@1180000000000 {
      compatible = "cavium,octeon-3860-bootbus";
      reg = <0x11800 0x00000000 0x0 0x200>;
      /* The chip select number and offset */
#address-cells = <2>;
      /* The size of the chip select region */
      #size-cells = <1>;
      ranges = <0 0  0       0x1bc00000  0x4400000>, /*CFI*/
             <1 0  0x10000 0x20000000  0>,
             <2 0  0       0x1b020000  0x10000>, /* BCSR */
```

```
                         <3 0  0       0x1b040000  0x10000>, /* CF */
                         <4 0  0x10000 0x30000000  0>,    /* NAND */
                         <5 0  0x10000 0x40000000  0>,
                         <6 0  0x10000 0x50000000  0>,
                         <7 0  0x10000 0x60000000  0>;

                  cavium,cs-config@0 {    /* CFI flash */
                        compatible = "cavium,octeon-3860-bootbus-config";
                        cavium,cs-index  = <0>;
                        cavium,t-adr     = <10>;
                        cavium,t-ale     = <34>;
                        cavium,t-ce      = <50>;
                        cavium,t-oe      = <50>;
                        cavium,t-we      = <35>;
                        cavium,t-rd-hld  = <25>;
                        cavium,t-wr-hld  = <35>;
                        cavium,t-pause   = <0>;
                        cavium,t-wait    = <300>;
                        cavium,t-page    = <25>;
                        cavium,t-rd-dly  = <0>;

                        cavium,pages     = <0>;
                        cavium,bus-width = <16>;
                        cavium,ale-mode  = <1>;

                  };
                  cavium,cs-config@2 {    /* BCSR */
                        compatible = "cavium,octeon-3860-bootbus-config";
                        cavium,cs-index = <2>;
                        cavium,t-adr    = <300>;
                        cavium,t-ce     = <300>;
                        cavium,t-oe     = <300>;
                        cavium,t-we     = <300>;
                        cavium,t-rd-hld = <300>;
                        cavium,t-wr-hld = <300>;
                        cavium,t-pause  = <300>;
                        cavium,t-wait   = <300>;
                        cavium,t-page   = <300>;
                        cavium,t-rd-dly = <300>;

                        cavium,pages     = <0>;
                        cavium,bus-width = <8>;
                  };
                  cavium,cs-config@3 {    /* Compact Flash */
                        compatible = "cavium,octeon-3860-bootbus-config";
                        cavium,cs-index = <3>;
                        cavium,t-adr  = <0>;
                        cavium,t-ce   = <30>;
                        cavium,t-oe   = <125>;
                        cavium,t-we   = <150>;
                        cavium,t-rd-hld = <100>;
                        cavium,t-wr-hld = <30>;
                        cavium,t-pause  = <0>;
                        cavium,t-wait   = <30>;
                        cavium,t-page   = <300>;
                        cavium,t-rd-dly = <0>;
                        cavium,pages     = <0>;
                        cavium,bus-width = <16>;
                };

                flash0: nor@0,0 {
                        compatible = "cfi-flash";
                        reg = <0 0 0x4000000>;
                        cavium,bus-width = <16>;
                        #address-cells = <1>;
                        #size-cells = <1>;

                        partition@0 {
                                label = "bootloader";
                                reg = <0x0 0x200000>;
                                read-only;
                        };
                        partition@200000 {
                                label = "kernel";
                                reg = <0x200000 0x400000>;
                        };
                        partition@600000 {
                                label = "cramfs";
                                reg = <0x600000 0x800000>;
                        };
                        partition@e00000 {
                                label = "jffs2big";
                                reg = <0xe00000 0x31e0000>;
                        };
                        partition@3fe0000 {
                                label = "environment";
                                reg = <0x3fe0000 0x20000>;
                                read-only;
                        };
                };

                bcsr-fpga@2,0 {
                        compatible = "embedded-planet,bcsr";
                        reg = <2 0 0x10000>;
                        cavium,bus-width = <8>;
                };

                compact-flash@3,0 {
                        compatible = "cavium,ebt3000-compact-flash";
                        reg = <3 0 0x10000>;
                        cavium,bus-width = <16>;
                };
        };
```

If one looks at the boot bus, section it starts with the line:

```
bootbus: bootbus@1180000000000 {
```

This indicates that the boot bus registers start at physical address 0x0001180000000000.

The next line says:

```
compatible = "cavium,octeon-3860-bootbus";
```

This is used for all boot bus implementations. All of them are compatible with the CN3860 boot bus.

The next line says:

```
reg = <0x11800 0x00000000 0x0 0x200>;
```

This provides the 64-bit address of the boot bus registers and the size of the block of registers. The address should match the above. Note that the address and size are made up of 2 words each to form 64-bit values by using pairs of 32-bit values.

```
#address-cells = <2>;
```

This indicates that all future addresses will be made of two 32-bit cells.

```
#size-cells = <1>;
```

This indicates that all future sizes will be made up of a single 32-bit cell.

```
ranges = <0 0  0       0x1bc00000  0x4400000>,
         <1 0  0x10000 0x20000000  0>,
         <2 0  0       0x1b020000  0x10000>, /* BCSR */
         <3 0  0       0x1b040000  0x10000>, /* CF */
         <4 0  0x10000 0x30000000  0>,   /* NAND */
         <5 0  0x10000 0x40000000  0>,
         <6 0  0x10000 0x50000000  0>,
         <7 0  0x10000 0x60000000  0>;
```

The ranges section has several fields. The first field which occupies the first two cells is the child bus address. In this case the high order address specifies the chip enable and the low order address is not used and should be zero. The next two fields are the parent bus address, the physical address to be assigned to the chip enable. The last field is the 32-bit length.

Chip enable 0 is usually connected to a NOR (CFI) flash device. On the EP6300C board this is a 64MB device. An extra 4MB is always added to the range for the NOR flash in order for it to properly wrap around so that address 0x1fc00000 always contains the first 4MB of the flash device. No extra space should be added to any other devices. The NOR flash is mapped to address 0x1bc00000 which occupies 68MB of space at the end of the 32-bit boot bus and the length indicates the size of 68MB.

Chip enables 1, 5, 6 and 7 are not used on this board. They are assigned addresses from the 4GB boot bus window starting at address 0x10000000000000 but have their lengths set to 0 to indicate that they are not used.

Chip enable 2 is connected to a FPGA called the BCSR on this board and it is assigned to address 0x1b020000 and assigned a length of 64KB. Note that the length should be specified in multiples of 64KB since that is what the boot bus address decoder supports.

Chip enable 3 is used by the compact-flash. On this board it does not support true-IDE mode. For compact flash it should not specify the offset since the compact-flash driver will automatically add it.

Chip enable 4 is connected to a NAND flash. It is treated as if it does not exist because NAND flash uses a separate mechanism for access. NAND flash is autodiscovered and is currently not defined in the flat device tree. This may change at a later date.

The next section starting with `cavium,cs-config@0` contains timing and mode information for chip select 0. On the EP6300C board the NOR flash uses ALE mode and a bus width of 16. Most OCTEON development boards do not use ALE mode and use a bus width of 8. The `@0` indicates that this is for chip select 0.

Details of the timing parameters can be found in the OCTEON Hardware Reference Manual in the section covering the MIO_BOOT_REG_TIM0 register.

Unlike this register, however, timing values are specified in nanoseconds. Make sure that the values chosen are not too large or too small. Values that are too large will be truncated to 63 clock cycles.

See Table 19.1:Boot Bus Timing and Mode Parameters below on page 47 for a list of timing and mode parameters available for the boot bus.

Table 19.1:Boot Bus Timing and Mode Parameters

`cavium,t-adr`	Address time.
`cavium,t-ale`	Address latch enable time.
`cavium,t-ce`	Chip enable time.
`cavium,t-oe`	Output enable time.
`cavium,t-we`	Write enable time.
`cavium,t-rd-hld`	Read hold time.
`cavium,t-wr-hld`	Write hold time.
`cavium,t-pause`	Pause time.
`cavium,t-wait`	Wait time. Must be non-zero when wait-mode is 1.
`cavium,pages`	Page Size. 0 = 8 bytes, 1 = 2 bytes, 2 = 4 bytes and 3 = 8 bytes.
`cavium,bus-width`	Set to 8 or 16 to specify the bus width.
`cavium,wait-mode`	Set to 1 to enable wait mode or 0 to disable wait mode.
`cavium,page-mode`	Set to 1 to enable page mode or 0 to disable page mode.

There are two more timing sections so that the BCSR and compact-flash are also covered by the timing parameters.

For NOR flash timing parameters it is best to choose those from a board with a similar device and copy it. Currently only two sets of values are chosen. One set is chosen for boards which use an 8-bit NOR flash without ALE. Most boards fall into this category. The other values are used by boards which use ALE such as the EP6300C board.

The next chip select configuration is for the BCSR FPGA. Due to lack of information from Embedded Planet regarding the timing information conservative values of 300ns are used. These should be close to the values used by default in MIO_BOOT_REG_TIM2.

The last chip select is for the compact-flash. There are several different sets of values defined for various boards based on how it is implemented. The EP6300C board does not use True-IDE mode and also uses only a single chip select for the compact-flash. Other boards may use two chip selects and may use a bus width of 8 or 16.

The next three sections define the various devices on the boot bus and are device specific.

The following defines the beginning of the NOR flash section:

```
flash0: nor@0,0 {
```

It is defined to be compatible with `“cfi-flash”`. The exact wording in the compatible section is important.

```
reg = <0 0 0x4000000>;
```

The first cell contains the chip select, in this case 0. The second cell is unused and should also be defined as zero. The third cell contains the length in bytes. In this case it is set to 64MB and *does not* contain the extra 4MB in the `ranges` section above.

```
cavium,bus-width = <16>;
#address-cells = <1>;
#size-cells = <1>;
```

`This defines the width of the bus. It must match the width defined in the cavium,cs-config@0 section above it.`
All following address and size cells contain a single 32-bit value.

Below this are the flash partitions. The partitions should match the values used in the configuration header file in `MTDPARTS_DEFAULT`.

```
partition@0 {
      label = "bootloader";
      reg = <0x0 0x200000>;
      read-only;
};
```

The `partition@0` declares that the partition starts at offset 0 in the flash. The `label` provides a text label to the partition. The `reg` parameter specifies the start offset and the size of the partition in the flash and `read-only` indicates that this partition is read-only. Four additional partitions are also defined.

The first partition should be read-only and be large enough to contain both the failsafe and standard versions of U-Boot and the last partition should be defined at the end where the environment is stored. Both of these partitions should be read-only. Typically the first partition is configured to be 2 megabytes in size in order to store both a failsafe and a standard copy of U-Boot.

19.2.1 Compact Flash

There are a number of options specific to compact flash.

The parameters described in table 19.2 below are used in the `compact-flash@<chip select>,0` section where `<chip select>` is either chip select 0 for True IDE mode or the common chip select in the traditional mode.

Table 19.2:Compact Flash Device Tree Parameters

Parameter Name	Description
`compatible`	This should be set to "`cavium,ebt3000-compact-flash`".
`reg`	Reg can take either one or two triplets depending on how many chip selects are used. In True-IDE mode the first triplet should be `<chip_select0 0 0x10000>` and the second, if present, should be `<chip_select1 0 0x10000>`. In the traditional mode the first triplet is the common chip select and the optional second chip select is the attribute chip select.
`cavium,bus-width`	This can be either `<8>` or `<16>`. In True-IDE mode this must be `<16>`. Note that all of the compact flash `cavium,cs-config@<chip select>` sections must match this value.
`cavium,true-ide`	Add this property if True-IDE mode is supported. This enables DMA support.
`cavium,dma-engine-handle`	Set this to the DMA engine to use for True-IDE mode. Typically this is set to `<@dma0>`.

Here is an example of what would be present for a True-IDE mode compact flash, taken from the EBB6600 board:

```
            bootbus: bootbus@1180000000000 {
                  compatible = "cavium,octeon-3860-bootbus";
                  reg = <0x11800 0x00000000 0x0 0x200>;
                  /* The chip select number and offset */
                  #address-cells = <2>;
                  /* The size of the chip select region */
                  #size-cells = <1>;
                  /* Chip Select, offset (always 0), 64-bit address, length
                  * Use 0 length to disable chip select.
                  */
                 ranges = <0 0  0       0x1f400000  0xc00000>,
                        <1 0  0x10000 0x30000000  0>,
                        <2 0  0x10000 0x40000000  0>,
                        <3 0  0x10000 0x50000000  0>,
                        <4 0  0       0x1d020000  0x10000>,
                        <5 0  0       0x1d040000  0x10000>,
                        <6 0  0       0x1d050000  0x10000>,
                        <7 0  0x10000 0x90000000  0>;
                 cavium,cs-config@5 {
                       compatible = "cavium,octeon-3860-bootbus-config";
                       cavium,cs-index = <5>;
                       cavium,t-adr  = <0>;
                       cavium,t-ce   = <300>;
                       cavium,t-oe   = <125>;
                       cavium,t-we   = <150>;
                       cavium,t-rd-hld = <100>;
                       cavium,t-wr-hld = <300>;
                       cavium,t-pause  = <0>;
                       cavium,t-wait   = <300>;
                       cavium,t-page   = <300>;
                       cavium,t-rd-dly = <0>;

                       cavium,pages     = <0>;
                       cavium,bus-width = <16>;
                 };
                 cavium,cs-config@6 {
                       compatible = "cavium,octeon-3860-bootbus-config";
                       cavium,cs-index = <6>;
                       cavium,t-adr  = <0>;
                       cavium,t-ce   = <30>;
                       cavium,t-oe   = <125>;
                       cavium,t-we   = <150>;
                       cavium,t-rd-hld = <100>;
                       cavium,t-wr-hld = <30>;
                       cavium,t-pause  = <0>;
                       cavium,t-wait   = <30>;
                       cavium,t-page   = <300>;
                       cavium,t-rd-dly = <0>;

                       cavium,pages     = <0>;
                       cavium,wait-mode;
                       cavium,bus-width = <16>;
                 };
                 compact-flash@5,0 {
                       compatible = "cavium,ebt3000-compact-flash";
                       reg = <5 0 0x10000>, <6 0 0x10000>;
                       cavium,bus-width = <16>;
                       cavium,true-ide;
                       cavium,dma-engine-handle = <&dma0>;
                 };
           };
```

For the timing parameters it is best to use the settings for an existing board. The difference between this and a non-True-IDE configuration is the timing parameters are different and the `cavium,true-ide` and `cavium,dma-engine-handle` parameters are missing and that the bus width might be 8 bits instead of 16 bits.

19.2.2 Custom Boot Bus Devices

Support has been enhanced in SDK 2.3 in order to support additional boot bus devices. One example of this is the Embedded Planet EP6300c board which has a FPGA called the BCSR. During the latter initialization stages a user-defined function called `octeon_add_user_boot_bus_fdt_devices()` is called. If defined, this function should call `octeon_boot_bus_add_fdt_handler()` with the name of the device in the boot bus flat device tree section, a parameter and a function to call to initialize it.

A generic function has been created called `octeon_boot_bus_generic_init()` which is able to handle many common devices. If this function is used, the parameter should be either NULL or a NULL-terminated array of compatible strings.

Immediately after all of the devices on the boot bus have been configured a user-defined function called `octeon_boot_bus_board_post_init()` is called. The purpose of this function is to perform any post initialization of board-specific devices.

A function called `octeon_boot_bus_get_dev_info()` has been defined which will output the address and optionally size of a device on the boot bus which is useful for addressing devices. By using this function it is no longer necessary to use defines to hard-code addresses in the configuration header file and instead use just the device tree.

19.3 TWSI Bus Definitions

The flat device tree supports multiple TWSI buses as well as I^{2}C muxes, switches, temperature sensors, real-time clocks, GPIOs, eeproms and more.

It is safe to copy the TWSI sections from any other OCTEON device tree file. Currently U-Boot does not use these sections. The Linux kernel, however, does use this to define the devices used and their addresses on the bus. Additionally, the Linux kernel supports devices behind I^{2}c muxes.

19.3.1 Temperature Sensors

Temperature sensors should always be called "`tmp@<hex address>`". For example, a TI TMP421 will typically be called "`tmp@4c`". The compatible field should specify the manufacturer and part number, for example, `compatible = "ti,tmp421";` specifies that the temperature sensor is compatible with a Texas Instruments TMP421 device. The `reg` field specifies the I2C address of the device.

19.3.2 Real-Time Clock

Real-time clocks should always be called "`rtc@<hex address>`". For example, a Dallas Semiconductor DS1337 would likely be called "`rtc@68`". Like the temperature sensor, the compatible field should specify the manufacturer and part number. If there are multiple manufacturers, it should specify the most common or original manufacturer. The `reg` field should specify the address of the device.

19.3.3 Board Serial EEPROM

Currently the board serial EEPROM is called "tlv-eeprom@<hex address>". This name may change in the future as more device types are standardized for the device tree. Currently all OCTEON boards use a device compatible with the Atmel AT24C256 device. The `compatible` and `reg` fields are similar to other device types.

19.3.4 General Purpose I/O Controllers

I2C GPIO controllers are supported in the device tree. These can be used by U-Boot, the Linux kernel as well as by Simple Executive applications. There are cases where they can be used to control MDIO muxes, for example with the EBB6600 board.

19.3.5 I^{2}C Muxes and Switches

Currently I2C muxes and switches are handled the same way within the Linux kernel where only a single output will be enabled at any given time. An example of using an I^{2}C switch can be found by looking at the device tree file for the EBB6600 board. The NXP PCA9548 I^{2}C switch is located at address 0x70 and has 8 outputs. Outputs 3 and 4 are each connected to NXP PCA8574 I^{2}C GPIO devices, output 3 using address 0x38 and output for using address 0x3e. The gpio1 device is used elsewhere in the device tree to control a MDIO MUX.

19.3.6 SFP+ Modules

On boards which use the Vitesse 10G Ethernet PHY devices U-Boot and Linux are responsible for reading the SFP+ EEPROM module connected to the I^{2}C bus. An example of this can be found in the snic10e, snic10e_61 and nic10e_66 device tree files.

19.4 The System Management Interface (SMI)

The SMI/MDIO interface is used to communicate with PHY devices. The total number of SMI devices varies depending on the OCTEON model chosen. The best choice is to copy all of the parameters from a board based on the same or a similar OCTEON chip.

19.5 PHY Configuration

There are a number of different options available for PHY configuration. Usually PHYs are added under the smi mdio section. A PHY can have several properties:

Table 19.3:PHY Device Tree Properties

Property	Description
`reg`	5-bit MDIO address of PHY or 16-bit address for clause 45 PHYs. Note that it is possible

Property	Description
	for multiple PHYs to use the same address if they are connected to a MUX so that only one can be active at a time.
`compatible`	Manufacturer and model of the PHY. For example, "`marvell,88e1149r`" or "`broadcom,bcm8706`".
`interrupt-parent`	Interrupt handler. This is usually set to <&gpio>.
`interrupts`	GPIO pin and flag. The flag is usually set to 8 to indicate that the interrupt is active low. If a PHY device has multiple interrupt lines, they can all be defined here.
`marvell,reg-init`	List of operations to perform to initialize a PHY device. This can configure the LEDs and clock, for example. For Marvell, the first field is the register address. The second is the page number, the third is a mask to AND with the register and the last field is the value to OR with the register.
`broadcom,reg-init`	List of operations to perform to initialize a PHY device. This can configure the LEDs and clock, for example. For Broadcom, the first field is the register address. The second is a mask to AND with the register and the last field is the value to OR with the register.
`cavium,qlm-trim`	This contains a field used to trim the device tree during run-time. Some boards can have different types of interfaces that are determined during boot-up. In these cases all of the possible interface types are defined and during boot-up any trim parameter that does not match is removed from the device tree. For an example where this is performed, see the EP6300C board or any of the EBB boards. The function that performs this operation is called `board_fixup_fdt()` which is defined in the appropriate board file, for example in `board/octeon/ep6300c/ep6300c_board.c`.
`mod_abs`	This indicates the Vitesse GPIO signal used for the mod_abs signal.
`tx_fault`	This is the Vitesse GPIO signal used for the tx_fault signal.
`txon`	This is the Vitesse GPIO signal that enables the laser for optical modules.
`inta`	This is the Vitesse GPIO signal connected to inta.
`rx_equalization`	This is used to adjust the rx equalization between the Vitesse PHY and the OCTEON chip. The values used for the snic boards are for short trace lengths.
`tx_preemphasis`	This is used to adjust the tx preemphasis between the Vitesse PHY and the OCTEON chip. The values used for the snic boards are for short trace lengths.
`txout_driver_ctrl1`	This is used to adjust the transmit driver slew rate between the Vitesse PHY and the OCTEON chip. The values used for the snic boards are for short trace lengths.

Property	Description
`sfp-eeprom`	This is used to indicate which I2C SFP EEPROM is associated with the Vitesse PHY.
`reset`	If a GPIO line can be used to reset a PHY then reset may be defined.
`vitesse,reg-init`	This contains a series of MDIO operations to initialize the PHY. The first field is the device ID, typically 1 – 7 or in some cases 0x1e. The next field is the register address. The third field is a mask to preserve and the last field is the value to or with it after the mask.

19.5.1 Example PHY Configuration from the Embedded Planet EP6300C

The EP6300C board has a rather simple PHY configuration. The two management ports are connected to SMI0 and the four SGMII ports and XAUI port are connected to SMI1.

```
		smi0: mdio@1180000001800 {
			compatible = "cavium,octeon-3860-mdio";
			#address-cells = <1>;
			#size-cells = <0>;
			reg = <0x11800 0x00001800 0x0 0x40>;

			phy0: ethernet-phy@1 {
				compatible = "broadcom,bcm5482";
				reg = <1>;
				broadcom,reg-init =
					<0x18 0 0xf0e7>,
					<0x1c 0 0x8c00>,
					/* Adjust LED drive. */
					<0x0d 0 0xb431>,
					<0x0e 0 0xb860>;
					interrupt-parent = <&gpio>;
					interrupts = <1 8>;
			};

			phy1: ethernet-phy2@ {
				compatible = "broadcom,bcm5482";
				reg = <2>;
				broadcom,reg-init =
					<0x18 0 0xf0e7>,
					<0x1c 0 0x8c00>,
					/* Adjust LED drive. */
					<0x0d 0 0xb4ae>,
					<0x0e 0 0xb863>;
					interrupt-parent = <&gpio>;
					interrupts = <1 8>; /* Pin 1, active low */
			};
		};
```

In the above example, the two management PHYs are connected to SMI0 and are labeled phy0 and phy1. Phy0 uses MDIO address 1 and phy1 uses MDIO address 2.

The `compatible` field says that both phys are Broadcom BCM5482s although they are both integrated into the same dual-phy chip.

The `reg` field gives the MDIO address of each phy, with phy0 being 1 and phy1 being 2.

The `broadcom,reg-init` parameter specifies how each PHY is to be initialized before being used. For phy0, the first operation writes the value 0xf0e7 to PHY register 0x18, 0x8c00 to register 0x1c, 0xb431 to register 0x0d and 0xb860 to register 0x0e. If a read-modify-write operation were to be performed, the mask would contain a value other than zero. The phy1 initialization is similar though the LED configuration is slightly different due to how the connector is wired up.

The `interrupt-parent` field specifies that interrupts are handled by GPIO and the `interrupt` field indicates the GPIO pin number and flag. In this case both PHYs are connected to GPIO pin 1 and interrupts are active low (8).

The second SMI interface handles either four SGMII ports or a single XAUI port depending on how jumper J3 is set.

```
		smi1: mdio@1180000001900 {
			compatible = "cavium,octeon-3860-mdio";
			#address-cells = <1>;
			#size-cells = <0>;
			reg = <0x11800 0x00001900 0x0 0x40>;
			/* XAUI */
			phy6: ethernet-phy@7 {
				reg = <7>;
				compatible = "broadcom,bcm8706";
				interrupt-parent = <&gpio>;
				interrupts = <2 8>, <3 8>; /* Pin 2 & 3, active low */
				cavium,qlm-trim = "0,xaui";
			};
			phy2: ethernet-phy@3 {
				reg = <3>;
				compatible = "broadcom,bcm5466";
				interrupt-parent = <&gpio>;
				interrupts = <1 8>; /* Pin 1, active low */
				cavium,qlm-trim = "0,sgmii";
			};
			phy3: ethernet-phy@4 {
				reg = <4>;
				compatible = "broadcom,bcm5466";
				interrupt-parent = <&gpio>;
				interrupts = <1 8>; /* Pin 1, active low */
				cavium,qlm-trim = "0,sgmii";
			};
			phy4: ethernet-phy@5 {
				reg = <5>;
				compatible = "broadcom,bcm5466";
				interrupt-parent = <&gpio>;
				interrupts = <1 8>; /* Pin 1, active low */
				cavium,qlm-trim = "0,sgmii";
			};
			phy5: ethernet-phy@6 {
				reg = <6>;
				compatible = "broadcom,bcm5466";
				interrupt-parent = <&gpio>;
				interrupts = <1 8>; /* Pin 1, active low */
				cavium,qlm-trim = "0,sgmii";
```

```
                                };
                        };
```

In the above example, PHYs 2 through 5 are the SGMII ports and PHY 6 is the XAUI port. PHYs 2 through 5 are assigned MDIO addresses 3 through 6 respectively. Unlike the management ports, the Broadcom BCM5466 quad-PHY does not require any initialization.

The `interrupt-parent` and `interrupts` fields are the same for the SGMII ports as the management ports. The XAUI port uses two other GPIO lines for interrupts.

The `cavium,qlm-trim = "0,sgmii";` and `cavium,qlm-trim = "0,xaui";` parameters are used at run-time to remove the PHYs that are not used. The trimming occurs in the function `board_fixup_fdt()` in `board/octeon/ep6300c/ep6300c_board.c`.

19.5.2 Run-time PHY Pruning

In some cases the type of PHY is not known until run-time. In this case a parameter called cavium,qlm-trim is used. If a board supports multiple types of PHYs on an interface then the function board_fixup_fdt() should be defined in the board c file. Several OCTEON development boards support this feature including the Embedded Planet EP6300C and all of the EBB boards such as the EBB6300.

19.5.3 Vitesse PHY Initialization

Vitesse PHYs are a bit different than the other PHY devices in that some registers are common between multiple PHYs on the same chip. Due to this, these must be placed under an `ethernet-phy-nexus` node in the device tree.

In addition to the numerous PHY parameters defined for the Vitesse PHY initialization can include a sequence of PHY register operations using the `vitesse,reg-init` property. Each operation is defined by four tuples. The first is the device ID. The second is the register address. The third tuple is a mask to be applied and the last is the value to be applied. Each operation looks something like:

reg[id,address] = (reg[id,address] & mask) | value

19.5.4 Marvell PHY Initialization

Many Marvell PHYs require initialization operations before they work properly. It is possible to define these initialization operations from within the device tree file.

Initialization is performed with the `marvell,reg-init` property. Each operation is defined by four tuples.

The first tuple defines the register number. The second tuple defines the page number. The third defines a mask operation to AND with the register and the fourth defines the value to OR with the register. The Marvell PHY is described on the device tree web site at http://devicetree.org/Compatible%3Dmarvell,88e1149r.

19.5.5 Broadcom PHY Initialization

Broadcom PHYs can be far more complicated than Marvell PHYs and have multiple sub-registers. Due to this a simpler initialization method was chosen. For Broadcom PHYs the broadcom,reg-init property takes only three tuples. The first tuple is the register number. The second tuple contains the mask to AND with the register and the third contains the value to OR with the register.

19.6 The MIX Definitions

The MIX interface handles management Ethernet ports. Only the CN52XX, CN56XX and CN6XXX devices have a MIX interface. The CN52XX, CN63XX and CN66XX devices support up to two MIX interfaces and the CN56XX and CN68XX devices support one MIX interface.

It is best to copy this section from the device tree for a board which uses the same processor. The only parameter that may need to be changed is `phy-handle`.

The following is an example mix section from the EBB6600 board:

```
            mix0: ethernet@1070000100000 {
                  compatible = "cavium,octeon-5750-mix";
                  reg = <0x10700 0x00100000 0x0 0x100>, /* MIX */
                        <0x11800 0xE0000000 0x0 0x300>, /* AGL */
                        <0x11800 0xE0000400 0x0 0x400>, /* AGL_SHARED  */
                        <0x11800 0xE0002000 0x0 0x8>;   /* AGL_PRT_CTL */
                  cell-index = <0>;
                  interrupts = <0 62>, <1 46>;
                  local-mac-address = [ 00 00 00 00 00 00 ];
                  phy-handle = <&phy0>;
            };

            mix1: ethernet@1070000100800 {
                  compatible = "cavium,octeon-5750-mix";
                  reg = <0x10700 0x00100800 0x0 0x100>, /* MIX */
                        <0x11800 0xE0000800 0x0 0x300>, /* AGL */
                        <0x11800 0xE0000400 0x0 0x400>, /* AGL_SHARED  */
                        <0x11800 0xE0002008 0x0 0x8>;   /* AGL_PRT_CTL */
                  cell-index = <1>;
                  interrupts = <1 18>, < 1 46>;
                  local-mac-address = [ 00 00 00 00 00 00 ];
                  phy-handle = <&phy1>;
            };
```

The above example should work for any CN6300 or CN6600 OCTEON. The only parameter that may need changing is the `phy-handle` parameter. Note that the interrupts are different for the CN68XX compared to the other OCTEON devices.

19.7 The Packet Input Processor Interface (pip) Definitions

The pip section defines all of the network interfaces for a chip other than management ports. Each interface represents a different QLM or port type.

The Embedded Planet EP6300C is a good example and is shown below:

```
        pip: pip@11800a0000000 {
            compatible = "cavium,octeon-3860-pip";
            #address-cells = <1>;
            #size-cells = <0>;
            reg = <0x11800 0xa0000000 0x0 0x2000>;

            interface@X {
                cavium,qlm-trim = "0,sgmii";
                compatible = "cavium,octeon-3860-pip-interface";
                #address-cells = <1>;
                #size-cells = <0>;
                reg = <0>; /* interface */

                ethernet@0 {
                    compatible = "cavium,octeon-3860-pip-port";
                    reg = <0x0>; /* Port */
                    local-mac-address = [ 00 00 00 00 00 00 ];
                    phy-handle = <&phy2>;
                };
                ethernet@1 {
                    compatible = "cavium,octeon-3860-pip-port";
                    reg = <0x1>; /* Port */
                    local-mac-address = [ 00 00 00 00 00 00 ];
                    phy-handle = <&phy3>;
                };
                ethernet@2 {
                    compatible = "cavium,octeon-3860-pip-port";
                    reg = <0x2>; /* Port */
                    local-mac-address = [ 00 00 00 00 00 00 ];
                    phy-handle = <&phy4>;
                };
                ethernet@3 {
                    compatible = "cavium,octeon-3860-pip-port";
                    reg = <0x3>; /* Port */
                    local-mac-address = [ 00 00 00 00 00 00 ];
                    phy-handle = <&phy5>;
                };
            };

            interface@Y {
                cavium,qlm-trim = "0,xaui";
                compatible = "cavium,octeon-3860-pip-interface";
                #address-cells = <1>;
                #size-cells = <0>;
                reg = <0>; /* interface */

                ethernet@0 {
                    compatible = "cavium,octeon-3860-pip-port";
                    reg = <0x0>; /* Port */
                    local-mac-address = [ 00 00 00 00 00 00 ];
                    phy-handle = <&phy6>;
                };
            };
        };
```

The Embedded Planet 6300C board is based on an OCTEON CN6300 with either four SGMII gigabit Ethernet ports or a single XAUI port depending on if jumper J3 is removed or not.

The pip section should be copied from the device tree file for a board using the same processor.

The `compatible`, `#address-cells`, `#size-cells` and `reg` parameter should not be changed.

The `interface@X` section is used to define all of the SGMII ports on QLM 2 which is considered interface 0 for the CN6300 and CN6600.

The `cavium,qlm-trim = "0,sgmii";` parameter is used to remove the interface if the SGMII module is not connected to the QLM. It is used the same way as it is for the MDIO sections. The zero indicates that this is connected to interface 0 of the PIP.

The `compatible` field should always be set to `"cavium,octeon-3860-pip-interface"`.

Similarly, the `#address-cells` and `#size-cells` should not be changed.

The `reg = <0>;` parameter indicates that `interface@X` is connected to interface 0 of the PIP.

Next, all of the Ethernet ports for `interface@X` are defined. The naming convention is `ethernet@<port number>`.

The `compatible` parameter should always be set to `"cavium,octeon-3860-pip-interface"`.

The `reg` parameter should be set to the port number.

The `local-mac-address` parameter allows for a MAC address to be defined for the port. If no MAC address is defined then the MAC address will be assigned using the information stored in the gd data structure which is typically initialized in `early_board_init()` in `board/octeon/<board name>/<board name>_board.c`.

The `phy-handle` parameter indicates which PHY to use for the port. The `phy` labels are assigned in the mdio/smi sections. In this case, `ethernet@0` through `ethernet@3` use `phy2` through `phy5`.

The `interface@Y` section covers the case when a XAUI module is connected to QLM 2.

To see an example where two QLMs are used see the EBB6600 or EBB6800 device tree files.

19.8 MDIO MUX Support

There are cases where a MUX may be required on the MDIO bus. Currently the only board which uses this is the EBB6600 board where a MUX was required due to the large number of MDIO devices.

In the case of the EBB6600, a MUX is used to select between QLM1 and QLM2. The MDIO addresses are the same for both QLMs.

The MUX is controlled by two GPIO lines coming off of an I2C GPIO device.

The following example is taken from the EBB6600 device tree. The only difference is that three of the four SGMII PHYs have been removed from each QLM to reduce the size of the example.

```
      mdio-mux {
            compatible = "cavium,mdio-mux-sn74cbtlv3253", "cavium,mdio-mux";
            gpios = <&gpio1 3 0>, <&gpio1 4 0>;
            parent-bus = <&smi1>;

            mdio@2 {
                  cell-index = <2>;
                  #address-cells = <1>;
                  #size-cells = <0>;

                  phy21: ethernet-phy@1 {
                        reg = <1>;
                        compatible = "marvell,88e1149r";
                        marvell,reg-init = <3 0x10 0 0x5777>,
                              <3 0x11 0 0x00aa>,
                              <3 0x12 0 0x4105>,
                              <3 0x13 0 0x0a60>;
                        interrupt-parent = <&gpio>;
                        interrupts = <10 8>; /* Pin 10, active low */
                        cavium,qlm-trim = "1,sgmii";
                  };

                  phy25: ethernet-phy@5 {
                        reg = <5>;
                        compatible = "broadcom,bcm8706";
                        interrupt-parent = <&gpio>;
                        interrupts = <9 8>; /* Pin 9, active low */
                        cavium,qlm-trim = "1,xaui";
                  };
            };

            mdio@3 {
                  cell-index = <3>;
                  #address-cells = <1>;
                  #size-cells = <0>;

                  phy11: ethernet-phy@1 {
                        reg = <1>;
                        compatible = "marvell,88e1149r";
                        marvell,reg-init = <3 0x10 0 0x5777>,
                              <3 0x11 0 0x00aa>,
                              <3 0x12 0 0x4105>,
                              <3 0x13 0 0x0a60>;
                        interrupt-parent = <&gpio>;
                        interrupts = <12 8>; /* Pin 12, active low */
                        cavium,qlm-trim = "0,sgmii";
                  };

                  phy15: ethernet-phy@5 {
                        reg = <5>;
                        compatible = "broadcom,bcm8706";
                        interrupt-parent = <&gpio>;
                        interrupts = <11 8>; /* Pin 11, active low */
                        cavium,qlm-trim = "0,xaui";
                  };
            };
      };
```

19.9 The UART Definitions

It is safe to copy the UART definition from any other OCTEON device tree file. The only field that should be changed is `current-speed`. Most OCTEON devices only support two UART interfaces, but the CN5200 supports up to three. It is recommended that this section be copied from a board with the same processor.

19.10 The USB Control (`uctl` and `usbn`) Definitions

For OCTEON II devices the uctl section must be filled in. For older OCTEON devices the section is called usbn. This section should be copied from an existing OCTEON II board. There are only two settings that need to be changed. The `refclk-frequency` property must be set to the frequency used for the USB clock. If an external oscillator is used it can be 12000000, 24000000 or 48000000. If a crystal is used it must be 12000000.

The other property is `refclk-type`. This can be set to either "`crystal`" or "`external`". No other values are allowed.

All other fields should be left alone.

19.11 MMC (`mmc`) Definitions

Starting with the OCTEON CN61XX support has been added for MMC memory.

It is suggested to copy this section from an existing device tree such as the EBB6100 device tree and modify the slot information as needed.

Up to four slots are supported.

The reg parameter should be set as shown in Table 19.4 below according to the chip select used for the slot.

Table 19.4: MMC Chip Select to `reg` Value

Chip Selects	`reg` Value
1 and 5	0
2 and 5	1
3 and 5	2
4 and 5	3

The `spi-max-frequency` field specifies the maximum frequency the MMC bus can be run for the board. The EBB6100 board is limited to 26MHz due to the number of devices and layout of the board. For boards with fewer devices on the boot bus higher speeds can be used, up to 52MHz.

The `cavium,bus-max-width` field indicates the maximum bus width supported. This can be 1, 4 or 8. Note that the CN61XX revision 1.0 does not properly calculate checksum values if the bus width of the device is not 8 and pull-down resistors are not present. Revision 1.1 and later do not have this limitation.

The voltage-ranges property should be set to <3300 3300>.

The `wp-gpios` property specifies the GPIOs used to read the write-protect switch. On the ebb6100 board this is not hooked up and the sff6100 board is missing a pull-up resistor. The second tuple is the GPIO line and the last tuple is 1 if the signal is active low or 0 if active high.

The `cd-gpios` property specifies the GPIOs used to read if a card is present or not. On the ebb6100 board this is not hooked up and the sff6100 board is missing a pull-up resistor. The second item is the GPIO line and the last item is 1 if the signal is active low or 0 if active high.

The `power-gpios` property specifies the GPIO lines that control the power or reset line going to the MMC/SD slot. Typically this would be GPIO 8 for the first slot. The second item is the GPIO line and the last is whether it is active low (1) or active high (0). Typically this would be active high.

20 Command Configuration Variables

The following configuration variables are used to enable various commands in the U-Boot shell. For more information on the usage of the commands, use the built-in help command in the U-Boot shell. All of the command source code can be found in the `common` and `arch/mips/cpu/octeon/commands` directories. Many of the U-Boot commands are also documented in the `README` file.

Table 20.1: Command Configuration Variables

Command Definitions	
CONFIG_CMD_ASKENV	Enables the `askenv` command to get environment variables from stdin.
CONFIG_CMD_BDI	Prints the Board Info data structure in `gd->bd`.
CONFIG_CMD_CONSOLE	Enables the `coninfo` command.
CONFIG_CMD_CRAMFS	Enables the `cramfsls` and `cramfsload` commands. Note that `CONFIG_CRAMFS_CMDLINE` must also be defined.
CONFIG_CMD_DATE	Enables the `date` command to display and set the time and date on the real-time clock chip.
CONFIG_CMD_DHCP	Enables the `dhcp` command to configure the network.
CONFIG_CMD_DNS	Looks up the IP address of a hostname
CONFIG_CMD_DTT	Displays temperature information
CONFIG_CMD_ECHO	Echos any parameters to stdout.
CONFIG_CMD_EDITENV	Allows editing of environment variables.
CONFIG_CMD_EEPROM	Allows reading and writing to an EEPROM.
CONFIG_CMD_ENV	Enables various environment variable commands.
CONFIG_CMD_EXT2	Enables support for accessing files on the EXT2 filesystem. Note that files larger than 70MB may have problems in U-Boot. This enables commands such as `ext2ls` and `ext2load`.
CONFIG_CMD_FAT	Enables support for accessing files on FAT16 and FAT32 filesystems. This enables commands such as `fatls` and `fatload`..
CONFIG_CMD_FLASH	Enables the `flinfo` command.
CONFIG_CMD_I2C	This enables the `i2c` command for accessing the TWSI bus.
CONFIG_CMD_IDE	This enables the `ide` command.
CONFIG_CMD_IMLS	This enables the `imls` command to list all images found in flash.
CONFIG_CMD_ITEST	This enables the `itest` command (integer test) that returns true or false on an integer compare.
CONFIG_CMD_JFFS2	This enables support for the JFFS2 filesystem.
CONFIG_CMD_KGDB	This enables support for using KGDB with U-Boot. This is not supported as of

Command Definitions

	the 2.1 release but may be supported in a later release.
CONFIG_CMD_LOADB	This enables the `loadb` and loady commands to load images over the serial port using kermet and y-modem.
CONFIG_CMD_LOADS	This enables the `loads` command to load S-records over the serial port.
CONFIG_CMD_MD5SUM	This enables the `md5sum` command.
CONFIG_CMD_MEMORY	This enables most memory-related commands such as `md` and `mm` and more.
CONFIG_CMD_MII	This enables the `mii` command. This command may have issues as of the 2.1 release and does not work on all interface types. This command will be enhanced in a later release.
CONFIG_CMD_MISC	This enables the `sleep` command.
CONFIG_CMD_MMC	This enables the `mmc` command to support MMC devices.
CONFIG_CMD_MTDPARTS	This command defines flash partitions.
CONFIG_CMD_NAND	This enables the `nand` command to work with the NAND subsystem.
CONFIG_CMD_NET	This enables numerous network-related commands such as `bootp`, `tftpboot`, `dhcp`, `nfs`, `ping`, `cdp`, `etc`. Note that additional defines are needed for individual commands. Most network-related options are documented in the U-Boot `README` file.
CONFIG_CMD_NFS	This enables NFS support for loading files.
CONFIG_CMD_OCTEON_BOOTBUS	This enables the `octbootbus` command. This command can be used to display or modify boot bus settings. It should be used only for board bringup and diagnostic purposes.
CONFIG_CMD_OCTEON_ERASEENV	This enables the eraseenv command to erase the environment stored in NOR flash.
CONFIG_CMD_OCTEON_NAND	This enables the OCTEON-specific `octnand` command.
CONFIG_CMD_OCTEON_REGINFO	This enables the OCTEON-specific `octreginfo` command.
CONFIG_CMD_OCTEON_TLVEEPROM	This enables the OCTEON-specific `tlv_eeprom` command.
CONFIG_CMD_PCI	This enables the `pci` command. It should only be added if `OCTEON_PCI_HOST=1`.
CONFIG_CMD_PING	This enables the `ping` command.
CONFIG_CMD_QLM	This enables the OCTEON `qlm` command to display or modify Quad-Lane

Command Definitions	
	Module (QLM) settings. This command is dangerous and should be disabled for end-users.
CONFIG_CMD_RUN	This will run a command in the specified environment variable.
CONFIG_CMD_SATA	This will add the `sata` command and enable SATA support for commands such as `ext2ls`, `ext2load`, `fatls` and `fatload`. This must be defined if SATA support is enabled.
CONFIG_CMD_SAVEENV	This saves the environment to flash.
CONFIG_CMD_SAVES	Save data as an S-record over the serial line to the host.
CONFIG_CMD_SDRAM	This will print out all of the SPD information for a memory DIMM. Note that it currently does not work for DDR3 DIMMs and will be updated when U-Boot enables DDR3 support for this command.
CONFIG_CMD_SETEXPR	This enables the `setexpr` command.
CONFIG_CMD_SHA1SUM	This enables the `sha1sum` command.
CONFIG_CMD_SNTP	This enables the `sntp` command to set the time of the real-time clock from a Network Time Protocol (NTP) server.
CONFIG_CMD_SOURCE	This enables the `source` command.
CONFIG_CMD_STRINGS	This enables the `strings` command to display strings located in memory.
CONFIG_CMD_UBI	This enables the `ubi` command which is used for UBI volume management.
CONFIG_CMD_UBIFS	This enables the `ubifs` command used for mounting and accessing UBIFS volumes.
CONFIG_CMD_UNZIP	This enables the `unzip` command. Additionally, it also enables the deprecated `gunzip` command.
CONFIG_CMD_USB	This enables the `usb` command, and is required for USB support.
CONFIG_CMD_XIMG	This enables the `imgxtract` command for extracting a part of a multi-part image such as a FIT image.

21 Configuration Variables

Table 21.1: General Configuration Variables

In many cases there are two board-specific files, `octeon_<board>.h` and `octeon_<board>_shared.h`. The shared file contains memory settings used by the board and in some cases SPD values. It also contains the board EEPROM TWSI address. Most of the

settings below are part of the standard U-Boot distribution and are documented there. Note that some of these defines are used only by the Makefiles for conditional compilation, this is especially true for commands.	
General Options	
`__HAVE_ARCH_CRC32`	Use the OCTEON-specific CRC32 functions to improve speed and reduce memory.
`__HAVE_ARCH_MEMCPY`	Use architecture specific memcpy. This will speed up memcpy on the OCTEON.
`__HAVE_ARCH_MEMSET`	Use architecture specific memset. This will speed up memset on the OCTEON.
`BOARD_MCU_AVAILABLE`	This indicates if a MCU is available on the board. The TWSI address is specified in `BOARD_MCU_TWSI_ADDR`.
`CONFIG_64BIT`	This should always be defined. It enables 64-bit support for ELF images and 64-bit register support.
`CONFIG_64BIT_PHYS_ADDR`	This defines `phys_t` in `arch/mips/include/asm/types.h` to be 64-bit.
`CONFIG_ADDR_MAP`	This enables address mapping support with `addrmap_phys_to_virt()`, `addrmap_set_entry()` and `addrmap_virt_to_phys()`.
`CONFIG_ADDR_MAP_BOOT_BUS_IDX`	This is used to map the boot bus to a physical address.
`CONFIG_ADDR_MAP_CODE_IDX`	This is used to map the virtual U-Boot address to a physical address.
`CONFIG_ADDR_MAP_FIRST_256MB_IDX`	This is used to map the first 256MB to a physical address.
`CONFIG_ADDR_MAP_PCI0_BASE`	This maps the first PCI configuration space to a physical address. The virtual address is typically 0xE0000000 to 0xEFFFFFFF
`CONFIG_ADDR_MAP_PCI1_BASE`	This maps the second PCI configuration space to a physical address. The virtual address is typically 0xF0000000 to 0xFFFFFFFF
`CONFIG_ADDR_MAP_REST_MEM_IDX`	This maps the virtual addresses 0x20000000 to 0x7FFFFFFF to physical addresses.
`CONFIG_AUTO_COMPLETE`	This enables autocomplete support on the command line by pressing tab.
`CONFIG_BAUDRATE`	This defines the default baudrate to use. This is usually set to 115200.
`CONFIG_BOOTDELAY`	This specifies a default delay before booting the default application or operating system. This allows the operator to interrupt the boot sequence in order to issue commands. Even if zero, an operator can

	still interrupt boot-up by pressing a key before this check takes place if CONFIG_ZERO_BOOTDELAY_CHECK is defined.
`CONFIG_BZIP2`	Enables BZIP2 compression support for the `bootm` command.
`CONFIG_CRAMFS_CMDLINE`	This enables the `cramfsls` and `cramfsload` commands. This is in addition to `CONFIG_CMD_CRAMFS`.
`CONFIG_CONSOLE_IS_IN_ENV`	Console information is stored in the environment.
`CONFIG_DOWNLOAD_BAUDRATE`	This defines the default baudrate to use when downloading files.
`CONFIG_ENV_OVERWRITE`	Set this to 1 to allow the Ethernet address and other values to be overwritten during the boot process.
`CONFIG_FIT`	Enables FIT images for command scripts
`CONFIG_GZIP`	Enables GZIP compression support. Note that it currently is not accelerated. This is required for the unzip command.
`CONFIG_L2_ONLY`	This configures U-Boot to use the layer 2 cache for memory and disables using DRAM. This option is currently not used or tested.
`CONFIG_OF_LIBFDT`	This enables flat device tree support. This option is required.
`CONFIG_LMB`	Enables procedures for maintaining information about logical memory blocks.
`CONFIG_LZMA`	Enables LZMA compression support for the `bootm` command.
`CONFIG_LZO`	Enables LZO compression support required for UBIFS.
`CONFIG_MD5`	Enables MD5 hashing support
`CONFIG_MEMSIZE_IN_BYTES`	This defines whether to pass the memory size in megabytes or bytes to the Linux kernel. If defined then the size is passed in bytes. This is the default.
`CONFIG_MMC`	This enables support for MMC devices.
`CONFIG_MMC_MBLOCK`	This enables support for multi-block MMC commands.
`CONFIG_OCTEON_DEFAULT_CONSOLE_UART`	This defines the default console UART. It defaults to 0 and is defined in `include/configs/octeon_common.h`.
`CONFIG_OCTEON_DISABLE_FAILSAFE_SCAN`	Disables scanning the NOR flash for a non-failsafe version of U-Boot.
`CONFIG_OCTEON_DDR`	This defines the version of DDR memory. For all OCTEON I and OCTEON Plus devices set this to 2. For all OCTEON II devices set this to 3.
`CONFIG_OCTEON_L2_RAM_SIZE`	This defines the amount of L2 RAM that is available. It is typically

	2MB though on the OCTEON CN68XX this increases to 5MB. This is only used if CONFIG_L2_ONLY is defined which disables DRAM support to only execute out of L2 memory. CONFIG_L2_ONLY is not tested or supported at this time.
CONFIG_OCTEON_MIN_CONSOLE_UART	This sets the minimum console UART number supported. It defaults to 0 and is defined in include/configs/octeon_common.h.
CONFIG_OCTEON_MAX_CONSOLE_UART	This sets the maximum console UART number supported. It defaults to 0 and is defined in include/configs/octeon_common.h.
CONFIG_OCTEON_MMC	This enables the OCTEON MMC driver.
CONFIG_RBTREE	This enables red-black tree library support. It is required for UBIFS.
CONFIG_RTC_DS1337	This enables support for the Dallas Semiconductor DS1337 real-time clock chip. Other RTC chips can be defined as well if they are used. If a RTC is present, define CONFIG_CMD_DATE and CONFIG_BOOTP_NTPSERVER to add support for accessing the RTC and using a NTP server. If no RTC chip is present, do not define CONFIG_CMD_DATE.
CONFIG_SHA1	Enables SHA1 hashing support.
CONFIG_SHA256	Enables SHA256 hashing support
CONFIG_SYS_BAUDRATE_TABLE	This defines the table of baudrates that are supported.
CONFIG_SYS_BIITMAPSZ	Memory mapped by startup code of the Linux kernel where boot arguments, the flat device tree and other data needed by the kernel is stored. It should be the same as OCTEON_RESERVED_LOW_MEM_SIZE.
CONFIG_SYS_BOOTPARAMS_LEN	Amount of memory allocated for storing parameters for the Linux kernel. This is currently set to 128 kilobytes.
CONFIG_SYS_CACHELINE_SIZE	Size of a cache line. Always 128 for OCTEON.
CONFIG_SYS_CBSIZE	Console I/O buffer size.
CONFIG_SYS_CONSOLE_INFO_QUIET	Do not display console information when booting up.
CONFIG_SYS_DCACHE_SIZE	Size of data cache. This is not currently used with OCTEON.
CONFIG_SYS_HZ	Default timer rate. Currently set to 1000.
CONFIG_SYS_ICACHE_SIZE	Size of instruction cache. This is not currently used with OCTEON.
CONFIG_SYS_LOAD_ADDR	Default load address to use. Usually set to 0x20000000.

CONFIG_SYS_MALLOC_LEN	Amount of memory to allocate for the heap in bytes. Currently 4MB is allocated for JFFS2 which can require a significant amount of memory.
CONFIG_SYS_MAX_PROMPT_LEN	This defines the maximum size of the prompt.
CONFIG_SYS_MAXARGS	Maximum number of command-line arguments to support
CONFIG_SYS_MEMTEST_END	Ending address for the `mtest` command.
CONFIG_SYS_MEMTEST_START	Starting address for the `mtest` command.
CONFIG_SYS_MIPS_TIMER_FREQ	This defines the MIPS timer frequency. It should not be changed. It is currently defined as `(gd->ogd.cpu_clock_mhz * 1000000)`
CONFIG_SYS_MMC_SET_DEV	This enables support for multiple MMC devices.
CONFIG_SYS_NUM_ADDR_MAP	This defines the maximum number of address maps to support.
CONFIG_SYS_PBSIZE	Print buffer size.
CONFIG_SYS_PROMPT	This defines the prompt to use. If not defined then the prompt will be computed automatically.
CONFIG_SYS_SDRAM_BASE	This is the base address for memory. It should be set to the beginning of KSEG0, 0x80000000.
CONFIG_TIMESTAMP	This displays the time stampof when U-Boot was compiled when displaying the version information.
CONFIG_ZERO_BOOTDELAY_CHECK	This allows a keystroke to interrupt the bootcmd even when the boot delay is set to zero.
DEFAULT_CPU_REF_FREQUENCY_MHZ	This is the CPU reference clock frequency if it cannot be determined automatically. It is usually set to 50.
DEFAULT_ECLK_FREQ_MHZ	This is the default DRAM refresh clock if the speed cannot be auto-detected. For DDR3 boards this is typically set to 400 or 533. The simulator uses the value of 600 to improve performance.
<board>_DEF_DRAM_FREQ	This is the default frequency to use for the DDR memory if it is not set in the board EEPROM. This is used in `board/octeon/<board>/<board>_board.c`.
CONFIG_OCTEON_SERIAL	Enables the OCTEON serial port driver. This is required.

Table 21.2: Boot Bus Configuration and Compact Flash

Boot Bus Configuration	
There are a number of common devices defined in the OCTEON U-Boot for the boot bus. Other than flash these are Compact Flash, an 8 character LED display and a PAL, though some boards may have additional features such as the Embedded Planet EP6300C. Some boards have no boot bus devices such as the NicPRO2 board. NOTE that the PAL, Compact Flash, BCSR	
All of these devices must have their base address and chip select defined.	
`OCTEON_CHAR_LED_CHIP_SEL`	Boot bus chip select number connected to the 8 character alpha-numeric LED display. NOTE: This is deprecated and should now be defined in the device tree.
`OCTEON_CHAR_LED_BASE_ADDR`	Base address for the 8-character alpha-numeric LED display on the boot bus. NOTE: This is deprecated and should now be defined in the device tree.
`OCTEON_PAL_CHIP_SEL`	Chip select number connected to the PAL device. NOTE: This is deprecated and should now be defined in the device tree.
`OCTEON_PAL_BASE_ADDR`	Base address for the PAL device on the boot bus. NOTE: This is deprecated and should now be defined in the device tree.
Compact Flash Support	
There are a number of defines used for Compact Flash and it may be implemented as True IDE mode or the old CF mode. The old CF mode can be implemented as either an 8 or 16-bit interface, but most CAVIUM development boards use the 16-bit bus.	
Some defines are common for both True IDE mode and the old IDE mode. Some boards also provide a method of resetting the CF adapter.	
`CONFIG_IDE_RESET`	This enables IDE reset support. If IDE reset is connected to a GPIO pin then `OCTEON_CF_RESET_GPIO` should be set to the GPIO number.
`CONFIG_LBA48`	This enables LBA 48-bit addressing. It is usually safe to enable this.
`CONFIG_OCTEON_NO_BOOT_BUS`	This is set if the boot bus is not used or supported.
`CONFIG_SYS_64BIT_LBA`	This enables 64-bit LBA addressing. It is usually safe to enable this. It can be enabled along with `CONFIG_LBA48`.
`CONFIG_SYS_ATA_BASE_ADDR`	This should be set to zero. It more or less is needed to make the compiler happy since other methods are used to determine the address.
`CONFIG_SYS_IDE_MAXBUS`	This is defined for the maximum number of IDE buses. Usually this is

Boot Bus Configuration	
	set to 1. For the EBB6600 revision 2.0 this is set to 2.
`CONFIG_SYS_IDE_MAXDEVICE`	This should be defined to the maximum number of devices supported on the IDE bus. Usually this is set to 1.
`OCTEON_CF_16_BIT_BUS`	This must be defined if the IDE bus is 16-bits. In True IDE mode this must be defined. In non-True IDE mode it depends on the board. NOTE: This is deprecated and should now be defined in the device tree.
`OCTEON_CF_RESET_GPIO`	This should be set to the GPIO number connected to the CF reset line. Note that on some boards, like the Embedded Planet EP6300C that the reset is handled differently and does not use a GPIO line.
True IDE Mode	
True IDE mode emulates a 16-bit IDE interface and is capable of performing DMA. It is preferable to the non-IDE mode which is not capable of DMA. These fields are deprecated and should be supported entirely by the device tree.	
`OCTEON_CF_TRUE_IDE_CS0_ADDR`	This should be set to the address to assign on the boot bus for the task file registers. NOTE: This is deprecated and should now be defined in the device tree.
`OCTEON_CF_TRUE_IDE_CS0_CHIP_SEL`	This should be configured for the chip select used for the task file registers. NOTE: This is deprecated and should now be defined in the device tree.
`OCTEON_CF_TRUE_IDE_CS1_ADDR`	This should be set to the address to assign on the boot bus for the alternate status register and device control register. NOTE: This is deprecated and should now be defined in the device tree.
`OCTEON_CF_TRUE_IDE_CS1_CHIP_SEL`	This should be configured for the chip select used for the alternate status register and device control register. NOTE: This is deprecated and should now be defined in the device tree.
Non-True IDE mode	
Non-True IDE mode is compatible with the PC card interface defined for Compact Flash. It is slower and is unable to take advantage of DMA. All of these fields except `CONFIG_SYS_ATA_DATA_OFFSET` and `CONFIG_SYS_ATA_REG_OFFSET` are deprecated and no longer required due to support in the device tree.	
`CONFIG_SYS_ATA_DATA_OFFSET`	Address offset from the base at which data is repeated so it can be read as a block. This is usually initialized as 0x400.
`CONFIG_SYS_ATA_REG_OFFSET`	Offset of IDE registers. This is usually initialized as zero.
`OCTEON_CF_ATTRIB_BASE_ADDR`	Base address to assign to the compact flash attribute registers. NOTE: This is deprecated and should now be defined in the device tree.

Boot Bus Configuration	
OCTEON_CF_ATTRIB_CHIP_SEL	This defines the GPIO connect to the compact flash attribute chip select. This may or may not be connected to the compact flash card. NOTE: This is deprecated and should now be defined in the device tree.
OCTEON_CF_COMMON_BASE_ADDR	Common compact flash base address. It is usually `OCTEON_CF_ATTRIB_BASE_ADDR | (1 << 11)`. NOTE: This is deprecated and should now be defined in the device tree.
OCTEON_CF_COMMON_CHIP_SEL	This defines the GPIO connect to the compact flash for the common chip select. NOTE: This is deprecated and should now be defined in the device tree.

Table 21.3: Networking Options

Networking Settings	
CONFIG_BOOTP_BOOTFILESIZE	Allows the server to specify the size of the bootfile.
CONFIG_BOOTP_BOOTPATH	Allows the server to specify the boot path.
CONFIG_BOOTP_DNS	Allows BOOTP to specify the primary domain name server.
CONFIG_BOOTP_DNS2	Allows BOOTP to specify the secondary domain name server.
CONFIG_BOOTP_GATEWAY	Allows BOOTP to set the default gateway.
CONFIG_BOOTP_HOSTNAME	Allows BOOTP to assign the hostname sent from the server.
CONFIG_BOOTP_MASK	Mask of settings for BOOTP. Usually this is set to `CONFIG_BOOTP_DEFAULT`
CONFIG_BOOTP_NTPSERVER	Allows the server to specify a NTP server. This only makes sense when there is a real-time clock.
CONFIG_BOOTP_SEND_HOSTNAME	Allows the client to send the hostname to the server.
CONFIG_BOOTP_SUBNETMASK	Allows the server to set the subnet mask.
CONFIG_BOOTP_TIMEOFFSET	Allows BOOTP to set the time offset.
CONFIG_CMD_NET	This is the main define to enable networking support. This enables commands like `ping`, `dhcp`, `tftp`, etc.
CONFIG_MII_CLAUSE_45	Define this to enable MII clause 45 support for XAUI
CONFIG_NET_MULTI	This should always be enabled for networking support. This allows multiple network interfaces to be supported and will cause DHCP to move to the next interface if the current one does not respond.

Networking Settings	
CONFIG_NET_RETRY_COUNT	Specify the number of times DHCP should retry before moving to the next interface. The default is 5.
CONFIG_NETCONSOLE	Define this to enable netconsole support. “nc” must be added to `stdin`, `stdout` and `stderr`.
CONFIG_TFTP_TSIZE	Enables support so TFTP output is dependent on the size of the file. This restricts the hash output to a single line to show progress.
CONFIG_UDP_CHECKSUM	This enables UDP checksum generation and checking.
OCTEON_INTERNAL_ENET	Define this to enable non-management Ethernet support for U-Boot.
OCTEON_MGMT_ENET	Define this to enable management Ethernet ports for devices that support this.
OCTEON_NO_NETWORK	Define this to disable all OCTEON network support including management. This is currently only used by the generic RAM version of U-Boot.
OCTEON_RGMII_ENET	Define this to enable RGMII Ethernet support
OCTEON_SGMII_ENET	Define this to enable SGMII Ethernet support
OCTEON_SPI_IXF18201_ENET	Define this to enable SPI IXF18201 support
OCTEON_SPI4000_ENET	Define this to enable SPI4000 support
OCTEON_XAUI_ENET	Define this to enable XAUI 10Gbps Ethernet support
CONFIG_PHY_GIGE	Enables support for gigabit PHY devices.
CONFIG_PHYLIB_10G	Enables support for clause 4510G PHY devices.
CONFIG_PHYLIB	Enables new PHY support. This is required for networking.
CONFIG_PHY_MARVELL	Enables the Marvell PHY driver.
CONFIG_PHY_BROADCOM	Enables the Broadcom PHY driver.

Table 21.4: USB Options

USB Options	
CONFIG_OCTEON_USB_OCTEON	If this is defined before including octeon_common.h then all of the default options to enable USB support including storage support will be defined. This should only be defined for OCTEON I devices such as the CN3XXX or CN5XXX families.
CONFIG_OCTEON_USB_OCTEON2	If this is defined before including octeon_common.h then all of the

USB Options	
	default options to enable USB support including storage support will be defined. This should only be defined for OCTEON II devices such as the CN6XXX family.
CONFIG_USB_BIN_FIXUP	Fixes output from some USB storage devices whose binary output can cause some consoles to freeze.
CONFIG_USB_STORAGE	Enables support for USB storage devices.
For the CN5XXX and earlier parts, the following should be defined in addition to *CONFIG_CMD_USB:*	
CONFIG_USB_OCTEON	Enables the Octeon USB driver for CN5XXX and earlier devices.
Octeon II and later devices include both an OHCI and EHCI compliant controller. Since U-Boot can only support either OHCI or EHCI and not both, the decision was made to support EHCI since virtually all modern USB storage devices support EHCI. For the CN6XXX series of Octeon CPUs, do not define *CONFIG_USB_OCTEON* *but define the following instead:*	
CONFIG_EHCI_DESC_BIG_ENDIAN	This variable is defined for big endian descriptors and should be defined for OCTEON.
CONFIG_SYS_USB_EHCI_MAX_ROOT_PORTS	Defines the number of root ports. This should be set to 1 or 2, depending on the number of ports available.
CONFIG_USB_EHCI	Enables the generic EHCI driver. This driver has been slightly modified in order to support virtual memory and the 64-bit physical addresses used in the Octeon U-Boot.
CONFIG_USB_EHCI_OCTEON2	Enables the low-level Octeon II USB support. This driver enables the clock and performs configuration that is not in the EHCI driver.

Table 21.5: TWSI Options

TWSI Options	
CONFIG_OCTEON_I2C	Enables the OCTEON I^{2}C (TWSI) driver.
BOARD_ISP_TWSI_ADDR	Address of ispPAC POWR1220AT8 In-System Programmable Power Supply Monitoring, Sequencing and Margining Controller.
BOARD_MCU_TWSI_ADDR	TWSI address of the MCU, if present.
CONFIG_CMD_OCTEON_TLVEEPROM	This enables the tlv_eeprom command which displays and sets the TLV fields stored in the board EEPROM.
CONFIG_DTT_SENSORS	This defines which sensors are active and the order to display

TWSI Options	
	them.
CONFIG_DTT_TMP42X	This enables support for the Texas Instruments TMP421, TMP422 or TMP423 temperature sensor.
CONFIG_I2C_MULTI_BUS	Define this if more than one TWSI bus is used. CONFIG_SYS_MAX_I2C_BUS must also be defined.
CONFIG_I2C_MUX	Define this if a TWSI MUX is present and should be supported by the I2C command. Currently this is not tested or supported.
CONFIG_ISPPAC_POWER	Defined if an ispPAC POWR1220AT8 is present.
CONFIG_SYS_DEF_EEPROM_ADDR	This should be set to CONFIG_SYS_I2C_EEPROM_ADDR.
CONFIG_SYS_DTT_BUS_NUM	TWSI bus number if not zero.
CONFIG_SYS_DTT_TMP42X	This is used to contain the initialization values for the TMP42X temperature sensor. See one of the include files like octeon_ebb6800.h for details.
CONFIG_SYS_EEPROM_PAGE_WRITE_BITS	Defines the number of bytes that can be written at once into a page of TWSI EEPROM. The name is a bit misleading. The value of 5 specifies 2^5 or 32 bytes. Check the manufacturer data sheet for details. Do not define this if page writes are not supported. It is not required and only speeds up write operations.
CONFIG_SYS_EEPROM_PAGE_WRITE_DELAY_MS	Defines the delay required after writing a page to the TWSI EEPROM. It is device dependent. The default value is 5ms which should be adequate for most devices.
CONFIG_SYS_I2C_DTT_ADDR	TWSI address of the temperature sensor, if present
CONFIG_SYS_I2C_EEPROM_ADDR	This should be set to OCTEON_XXXX_EEPROM_TWSI_ADDR where XXXX is the board name. This is typically defined in octeon_xxxx_shared.h.
CONFIG_SYS_I2C_EEPROM_ADDR_LEN	This defines the number of bytes to use for the address when communicating with the board serial EEPROM. This is set to 2 for the 24LC256 which is used on all CAVIUM OCTEON development boards.
CONFIG_SYS_I2C_RTC_ADDR	This specifies the I2C (TWSI) address of the RTC chip if one is present. If no RTC chip is present then do not define this.
CONFIG_SYS_I2C_SPEED	This defines the default TWSI bus speed. It is usually initialized to 100000.
CONFIG_SYS_MAX_I2C_BUS	Maximum supported TWSI bus. This must be defined if

TWSI Options	
	CONFIG_I2C_MULTI_BUS is defined.
CONFIG_SYS_RTC_BUS_NUM	If multiple TWSI buses are used, this specifies the bus number used for the real-time clock if it is non-zero.

Table 21.6: Environment Options

Environment Settings	
CONFIG_ENV_ADDR	This contains the address of the environment. For the environment stored in flash it should be set to (CONFIG_SYS_FLASH_BASE + CONFIG_SYS_FLASH_SIZE – CONFIG_ENV_SIZE). For the environment stored in RAM it should be set to U_BOOT_RAM_ENV_ADDR.
CONFIG_ENV_IS_IN_FLASH	Define this to 1 if the environment is stored in flash. The environment will usually be stored in the last sector. If this is selected then CONFIG_ENV_IS_IN_RAM and CONFIG_ENV_IS_NOWHERE cannot be selected.
CONFIG_ENV_IS_IN_NAND	Define this if the environment is stored in NAND flash. If set then CONFIG_ENV_OFFSET, CONFIG_ENV_SIZE and CONFIG_ENV_RANGE must be set. CONFIG_ENV_RANGE should be the size of two blocks.
CONFIG_ENV_IS_IN_RAM	Define this to 1 if the environment is stored in RAM. This should only be defined on boards that always boot remotely with oct-remote-boot. If this is selected then CONFIG_ENV_IS_IN_FLASH and CONFIG_ENV_IS_NOWHERE cannot be selected.
CONFIG_ENV_IS_NOWHERE	Define this as 1 if the environment is not defined in flash or RAM. If this is selected then CONFIG_ENV_IS_IN_RAM and CONFIG_ENV_IS_IN_FLASH cannot be selected.
CONFIG_ENV_OFFSET	This is the offset in flash where the environment is stored. For NAND flash this should be at least 4MB.
CONFIG_ENV_RANGE	This defines the size of the range of NAND that can be used to store the environment. It should be a minimum of the size of two blocks.
CONFIG_ENV_SIZE	Define this to be the maximum size of the environment. For flash devices this is usually defined to be the size of the last sector. If CONFIG_ENV_IS_IN_RAM is defined

Environment Settings	
	then this must be set to `U_BOOT_RAM_ENV_SIZE`. For NAND flash this should be smaller than the block size.
`CONFIG_VERSION_VARIABLE`	Enables the ver environment variable which contains the version information.

Table 21.7: Flash Options

Flash Settings	
`CONFIG_FLASH_CFI_DRIVER`	Define this to add the CFI flash driver. This must be defined to use CFI.
`CONFIG_OCTEON_ENABLE_DT_FLASH_CHECKS`	This enables U-Boot to verify that all of the flash settings are correct. Since the flash settings are split between the configuration header file and the flat device tree it is a common mistake for at least one of these to be misconfigured. If this is set then a number of consistency checks are made.
`CONFIG_FLASH_CFI_MTD`	This enables MTD partitioning for CFI flash devices.
`CONFIG_JFFS2_LZO`	Enables JFFS2 LZO compression support
`CONFIG_JFFS2_NAND`	Enables NAND support for JFFS2
`CONFIG_JFFS2_SUMMARY`	Define this to enable JFFS2 summary support. Note that this is somewhat broken in U-Boot but should be fixed in the near future.
`CONFIG_MTD_DEVICE`	This enables MTD partitioning support
`CONFIG_MTD_NAND_ECC_JFFS2`	Enables ECC support for JFFS2
`CONFIG_MTD_PARTITIONS`	This enables MTD partitions that are compatible with Linux
`CONFIG_MTD_UBI`	Enables MTD support for UBI volumes in NAND flash.
`CONFIG_OCTEON_DISABLE_JFFS2`	When including `config_octeon_nand.h` this disables support for the JFFS2 filesystem. JFFS2 has been superceded with UBIFS.
`CONFIG_OCTEON_DISABLE_UBI`	When including `config_octeon_nand.h` this disables the UBI filesystem and volume support.
`CONFIG_OCTEON_EEPROM_TUPLES`	Set to 1 if there is a serial EEPROM which holds OCTEON board information. Undefine if a board serial EEPROM is not present.
`CONFIG_OCTEON_FLASH`	This enables OCTEON flash support via conditional compilation.

Flash Settings

`CONFIG_OCTEON_FLASH_USES_ALE`	Define this if the flash is wired to use ALE. This prevents the U-Boot startup code from speeding up the ALE time which would otherwise fail. This is used on the Embedded Planet EP6300C board. This configuration option was removed in the 3.0 SDK and it is now detected at run-time.
`CONFIG_SYS_CFI_FLASH_STATUS_POLL`	This enables polling support for the flash. This can speed up certain operations and is compatible with most flash devices.
`CONFIG_SYS_FLASH_BASE`	This defines the base address to use for flash. It is usually set to `0x1FC00000 - CONFIG_SYS_FLASH_SIZE`.
`CONFIG_SYS_FLASH_CFI`	Define this to 1 if a CFI flash is present. This must be defined to use CFI.
`CONFIG_SYS_FLASH_CFI_WIDTH`	Can be set to FLASH_CFI_8BIT or FLASH_CFI_16BIT. This defines which bus width to start with when detecting the CFI flash. It is optional and the CFI driver should automatically detect the correct width. It might offer a minor performance increase when set.
`CONFIG_SYS_FLASH_EMPTY_INFO`	This makes the `flinfo` command print out whether a sector is empty or not. It can slow down the print command significantly if the flash is slow.
`CONFIG_SYS_FLASH_ERASE_TOUT`	This defines the flash erase timeout
`CONFIG_SYS_FLASH_PROTECTION`	This causes the portions of flash containing the bootloader and environment to be protected by default.
`CONFIG_SYS_FLASH_SIZE`	This defines the size of the CFI flash in bytes.
`CONFIG_SYS_FLASH_USE_BUFFER_WRITE`	This enables buffered write support for CFI flash. This significantly speeds up write operations and all modern CFI flash chips should support this.
`CONFIG_SYS_FLASH_WRITE_TOUT`	This defines the flash write timeout.
`CONFIG_SYS_JFFS2_SORT_FRAGMENTS`	If you boot from a partition which is mounted writable, and you update your boot environment by replacing single files on that partition, you should also define CONFIG_SYS_JFFS2_SORT_FRAGMENTS. Scanning the JFFS2 filesystem takes *much* longer with this feature, though. Sorting is done while inserting into the fragment list, which is more or less a bubble sort. That algorithm is known to be O(n^2), thus you should really consider if you can avoid it!
`CONFIG_SYS_MAX_FLASH_BANKS`	Maximum number of CFI flash banks (chips) to support.
`CONFIG_SYS_MAX_FLASH_SECT`	Maximum number of CFI flash sectors to support

Flash Settings	
CONFIG_SYS_MAX_NAND_DEVICE	Maximum number of NAND devices to support
CONFIG_SYS_MONITOR_LENGTH	This defines what offset U-Boot should use to start scanning for the non-failsafe bootloader. It is currently set to 192K which may be rather small for modern boards that use a recent version of U-Boot for the failsafe.
CONFIG_SYS_NAND_BASE	Base address for NAND flash. For OCTEON based boards this must be set to 0.
CONFIG_SYS_NAND_FORCE_WIDTH	This can be used to force the width of a NAND device. It can be set to 8 or 16.
CONFIG_SYS_NAND_MAX_CHIPS	This should be the same as `CONFIG_SYS_MAX_NAND_DEVICE`. It defaults to 1 if not set.
CONFIG_SYS_NAND_ONFI_DETECTION	This enables NAND flash ONFI detection and should be enabled for boards with ONFI compliant flash. It is safe to enable this even if the flash does not support ONFI (i.e. Samsung flash).
CONFIG_SYS_NO_FLASH	If defined this indicates that there is no CFI flash on the board. Currently this is only used for the simulator and the NIC Pro2 card.
FAILSAFE_BASE	This is the base address to use for the failsafe U-Boot in flash. It is always initialized to 0x1FC00000. It is currently only used for the `bootloaderupdate` command when updating the failsafe bootloader. The syntax of this command is undocumented.
LOOKUP_STEP	Step size to use when searching for an alternative bootloader from the failsafe. The default value is `64*1024` and should not be changed.
MTDIDS_DEFAULT	This specifies aliases used for NAND and NOR flash partitioning. It may be passed to the Linux kernel as well.
MTDPARTS_DEFAULT	This specifies the default partitioning of the flash. It may be passed to the Linux kernel.
CONFIG_BCH	This enables multi-bit error correction.

Table 21.8: MMC Options

MMC Settings	
CONFIG_MMC	Enables MMC support
CONFIG_MMC_MBLOCK	Enables support for multi-block commands to the MMC device
CONFIG_OCTEON_MMC	Enables the OCTEON MMC driver

MMC Settings	
CONFIG_SYS_MMC_SET_DEV	Enables support for multiple MMC devices

Table 21.9: DFM Memory Options

DFM Settings	
DFM_CFG_ADDR_MIRROR	Set to 1 to enable address mirroring
DFM_CFG_CAS_LATENCIES	This contains a bitmap of the supported CAS latencies, shifted right by 4. So if bit 0 were set to 1 then a CAS latency of 4 is supported. If bit 1 then CAS 5, etc.
DFM_CFG_COL_BITS	Number of column bits for DFM memory
DFM_CFG_DRAM_WIDTH	Width of each DFM memory chip
DFM_CFG_NUM_BANKS	Number of DFM memory banks
DFM_CFG_NUM_RANKS	Number of ranks for DFM memory
DFM_CFG_ROW_BITS	Number of row bits for DFM memory
DFM_CFG_TAAMIN	
DFM_CFG_TCKMIN	
DFM_CFG_TFAW	tFAW minimum four activate window delay in ps
DFM_CFG_TRAS	tRAS minimum active to precharge delay in ps
DFM_CFG_TRC	tRC minimum active to active/refresh delay in ps
DFM_CFG_TRCD	tRCD minimum RAS to CAS delay in ps
DFM_CFG_TRFC	tRFC minimum refresh recovery delay in ps
DFM_CFG_TRP	tRP minimum row precharge delay in ps
DFM_CFG_TRRD	tRRD minimum row active to row active delay in ps
DFM_CFG_TRTP	tRTP minimum internal read to precharge command delay
DFM_CFG_TWR	tWR write recovery time in ps
DFM_CFG_TWTR	tWTR internal write to read command delay in ps
DFM_EARLY_DQX	Set to 1 in the rare case when ADD/CMD signals are routed shorter than the shortest DQS/DQ signal.

DFM Settings	
`OCTEON_CN63XX_DFM_ODT_CONFIGURATION`	This defines the ODT configuration to use for OCTEON CN63XX devices. It should have one entry per supported rank.
`OCTEON_CONFIG_DFM`	Define this as 1 if the board contains DFM memory. DFM memory is supported by the OCTEON CN63XX.

Table 21.10: PCI Options

PCI Settings	
`CONFIG_CONSOLE_MUX`	This should be defined if either `CONFIG_SYS_PCI_CONSOLE` or `CONFIG_OCTEON_BOOTCMD` is defined. This allows for multiple inputs and outputs to be supported simultaneously.
`CONFIG_E1000`	Enables support for the Intel E1000 network adapter. Both the PCI and PCIe adapters are supported but some development boards may have issues with the PCI version.
`CONFIG_OCTEON_BOOTCMD`	This enables input for the oct-remote-bootcmd utility. It is also used by oct-remote-load, oct-remote-boot and several other utilities. In the future this support may be merged into the PCI console. This is another input-only method to the console.
`CONFIG_OCTEON_PCI_BIG_BAR`	Set this to 1 if all PCI devices support 64-bit addressing. Note that this applies to PCI and PCIX only and not PCI Express. This is required to use the Silicon Image 3124 PCI/PCIX SATA controller.
`CONFIG_OCTEON_PCI_ENABLE_32BIT_MAPPING`	Setting this for OCTEON II and later devices enables PCI memory mapping so that PCI devices that only support 32-bit addressing are capable of performing DMA to the physical memory addresses used by U-Boot. Setting this will prevent PCI devices from being able to DMA to memory addresses outside of U-Boot's physical address space. For example, if this is set then the Silicon Image SATA controller cannot read to the default load address 0x20000000. Only set this if needed. Most modern PCI and PCIe devices are capable of 64-bit addressing.
`CONFIG_OCTEON_PCI_HOST`	This should be set to 1 or 0 depending on if the device behaves as a PCI or PCIe host on at least one interface. If a device has no PCI devices connected or will only be in target mode then this should be set to 0. Note that this is usually set in boards.cfg. This should also be defined before including `octeon_common.h`.

PCI Settings	
CONFIG_PCI	This enables the PCI and PCIe drivers used in host mode.
CONFIG_PCI_PNP	This enables "plug-n-play" support for PCI and will allow U-Boot to automatically assign resources. Note that usually the operating system will re-assign the PCI resources.
CONFIG_PCI_SCAN_SHOW	This displays all of the PCI devices found during the PCI scan during the boot process.
CONFIG_PCIAUTO_SKIP_HOST_BRIDGE	This prevents the host bridge from being auto-configured.
CONFIG_SATA_SIL	Enables support for the Silicon Image SIL3132 and related PCIe SATA controllers. Note that for OCTEON devices with PCI and not PCIe that CONFIG_OCTEON_PCI_BIG_BAR must be defined as 1.
CONFIG_SYS_PCI_64BIT	This enables 64-bit addressing for the BAR address space.
CONFIG_SYS_PCI_CONSOLE	This adds support for directing stdin, stdout and stderr to the PCI console. By default, boards that can be PCI targets have this defined and stdin is enabled for the PCI console as well as the serial console. By default stdout and stderr are not assigned to the PCI console since they will block until a PCI console is connected.
CONFIG_SYS_SATA_MAX_DEVICE	This specifies the maximum number of SATA devices supported. For the Silicon Image PCI express SATA controllers this is typically set to 1 or 2.
OCTEON_FIRST_PCIE_BUSNO	This defines the starting PCIE bus number. It is usually initialized to 0 in include/configs/octeon_common.h. For boards using IDT switches this should be set to 1.

Table 21.11: Disk Filesystem Options

Disk Filesystem Settings	
CONFIG_DOS_PARTITION	This enables DOS partitioning support for compact flash and USB drives.
CONFIG_FAT_WRITE	This enables FAT write support. It allows creating new files or overwriting existing files.
CONFIG_CMD_EXT2	This enables EXT2 filesystem support.
CONFIG_CMD_FAT	This enables FAT filesystem support.

22 OCTEON Development Board Details

In order to choose a board to base development off of it helps to know what features existing boards offer. The top differences between boards are:

1. OCTEON I or OCTEON II
2. DDR memory type
3. Flash type and size
4. PHY type and port configuration
5. Boot bus options
6. Compact Flash
7. USB type

The OCTEON model is not a significant factor when porting U-Boot. Most of the OCTEON model differences are handled automatically at run-time.

22.1 OCTEON EBB6800 Development Board

This board is based around the OCTEON CN6800 processor and has the following features:

- 8 DDR3 memory slots supporting 4 DDR3 memory interfaces
- CFI compliant 8MB NOR flash
- True-IDE Compact Flash port
- Two EHCI/OHCI compliant USB ports
- PCIe host mode
- PCIe target mode
- 5 QLM interfaces
- 8-character alpha-numeric LED display
- PAL device
- MCU
- 1 gigabit Ethernet management port
- 4GB NAND flash
- TMP421 temperature sensor
- Two TWSI buses with bus 1 connected to DDR3 SPDs
- TWSI Board EEPROM
- Dallas Semiconductor DS1337 real-time clock
- ispPAC POWR1220AT8 Voltage Controller Chip

22.2 OCTEON EBB6600 Development Board

This board is based around the OCTEON CN6600 processor and has the following features:

- 2 DDR3 memory slots supporting 2 DDR3 memory interfaces
- CFI compliant 8MB NOR flash
- True-IDE Compact Flash port
- Two EHCI/OHCI compliant USB ports
- PCIe host mode
- PCIe target mode
- 3 QLM interfaces, two which may be used for SGMII and XAUI.
- 8-character alpha-numeric LED display
- MCU
- 2 gigabit Ethernet management ports
- 4GB NAND flash
- TMP421 temperature sensor
- Two TWSI buses
- TWSI Board EEPROM
- Dallas Semiconductor DS1337 real-time clock
- ispPAC POWR1220AT8 Voltage Controller Chip

22.2.1 Special Notes

- The MDIO buses for QLMs 1 and 2 are selected by using a TWSI controlled GPIO line behind a TWSI switch.
- Revision 2 of this board contains two Compact Flash.

22.3 OCTEON EBB6100 Development Board

This board is based around the CN6100 OCTEON processor and has the following features:

- 2 DDR3 memory slots supporting 2 DDR3 memory interfaces
- CFI compliant 8MB NOR flash
- True-IDE Compact Flash port
- Two EHCI/OHCI compliant USB ports
- PCIe host mode
- PCIe target mode
- 3 QLM interfaces, two which may be used for SGMII and PCIe, and one for XAUI.
- 8-character alpha-numeric LED display

- MCU
- 2 gigabit Ethernet management ports
- TMP421 temperature sensor
- Two TWSI buses
- TWSI Board EEPROM
- Dallas Semiconductor DS1337 real-time clock
- 8-bit MMC Slot
- ispPAC POWR1220AT8 Voltage Controller Chip

22.4 Embedded Planet EP6300C Development Board

This board is based around the OCTEON CN6300 and has the following features:

- 2 DDR3 slots with 1 memory interfaces
- CFI compliant 64MB NOR flash configured in 16-bit mode.
- PCIe host mode
- PCIe target mode
- 16-bit compact flash interface that does *not* use the True IDE mode
- BCSR FPGA on boot bus
- Two EHCI/OHCI compliant USB ports
- 2 gigabit Ethernet management ports
- 2 serial ports
- 4GB NAND flash
- TMP421 temperature sensor
- TWSI Board EEPROM
- 4 SGMII ports
- 1 XAUI port
- DFM memory

22.4.1 Special Notes

- This board lacks a MCU.
- The compact flash detection and reset is handled by the BCSR FPGA. This is initialized in `board/octeon/ep6300c/ep6300c_board.c` by `octeon_boot_bus_board_init()`. The compact flash presence is detected by the function `octeon_cf_present()` in the same file. Reset is handled by `ide_set_reset()` and a LED is lit during access to the compact flash with `ide_led()`, also in the same file.

- This board is capable of booting from NAND flash. The OCTEON U-Boot as of SDK version 2.1.0 does not yet support this.
- There is a CN66XX version of this board. It is identical except for the CPU.

22.5 OCTEON EBB6300 Development Board

This board is based around the OCTEON CN6300 processor and has the following features:

- 2 DDR3 memory slots supporting 1 DDR3 memory interface
- CFI compliant 4MB NOR flash
- PCIe host mode
- PCIe target mode
- 3 QLM interfaces
- 8-character alpha-numeric LED display
- PAL device
- MCU
- True-IDE Compact Flash port
- Two EHCI/OHCI compliant USB ports
- 2 gigabit Ethernet management ports
- 2 serial ports
- 4GB NAND flash
- TMP421 temperature sensor
- TWSI Board EEPROM
- Dallas Semiconductor DS1337 real-time clock
- rapidIO support via QLM
- XAUI via QLM
- SGMII via QLM

22.5.1 Special Notes

On this board the MCU must be notified early on that the OCTEON is running. This is performed in the function `board_early_init_f()` in the file `board/octeon/ebb6300/ebb6300_board.c`.

22.6 OCTEON SFF6100

This board is based around the OCTEON CN6100 and has the following features:

- PCIe 4x host slot (requires bracket removal for cards)
- Two mini PCIe 1x slots (NOTE: there is a work-around for a mis-wired reset line)

- 8 SGMII ports
- 2 SATA 2 ports with integrated Silicon Image SIL3132.
- USB 2.0 port.
- Management Ethernet Port
- Real-time Clock
- MMC/SD slot with power controlled by GPIO.

Note: not all combinations of features may be active at the same time.

22.7 OCTEON NIC10e_66

This board is based around the OCTEON CN6600 and has the following features:

- PCIe target mode
- 1 DDR3 memory interface
- 1 serial port
- 4GB NAND flash
- CFI compliant 8MB NOR flash
- SAA56004X temperature sensor
- TWSI bus
- TWSI Board EEPROM
- 2 XAUI ports connected to a Vitesse VSC8488 dual 10G PHY
- DFM memory

22.7.1 Special Notes

- This board lacks a MCU.
- The NIC10e_66 is designed to boot over the PCIe bus

22.8 OCTEON NIC4e

This board is based around the OCTEON CN6300 and is very similar to the NIC10e card. The major difference is that it has four gigabit Ethernet ports instead of two XAUI ports. It has the following features:

- PCIe target mode
- 1 DDR3 memory interface
- One EHCI/OHCI compliant USB port
- 1 serial port
- 4GB NAND flash

- CFI compliant 8MB NOR flash
- SAA56004X temperature sensor
- TWSI bus
- TWSI Board EEPROM
- 4 gigabit Ethernet ports
- DFM memory

22.8.1 Special Notes

- This board lacks a MCU.
- The NIC4e is designed to boot over the PCIe bus
- This board does not have an alpha-numeric LED so `octeon_led_str_write()` is empty.

22.9 OCTEON NIC2e

This board is based around the OCTEON CN6300 and is very similar to the NIC10e and NIC4e cards. The major difference is that it has two gigabit Ethernet ports instead of two XAUI ports or four gigabit Ethernet ports. It has the following features:

- PCIe target mode
- 1 DDR3 memory interface
- One EHCI/OHCI compliant USB port
- 1 serial port
- 4GB NAND flash
- CFI compliant 8MB NOR flash
- SAA56004X temperature sensor
- TWSI bus
- TWSI Board EEPROM
- 4 gigabit Ethernet ports
- DFM memory

22.9.1 Special Notes

- This board lacks a MCU.
- The NIC2e is designed to boot over the PCIe bus

22.10 Octeon EBT5800 Development Board

The EBT5800 board is based on the Octeon CN5800 and has the following features:

- 2 DDR2 slots

- 8MB CFI compliant NOR flash
- PCI-X target mode
- 4 SGMII gigabit Ethernet interfaces
- MCU
- 16-bit Compact Flash (not True-IDE)
- 8 character alpha-numeric display
- 2 serial ports
- TWSI bus
- TWSI bus serial EEPROM
- PAL

22.11 OCTEON NICPro2

The NICPro2 board is based on the Octeon CN5800 and has the following features:

- PCI-X target mode
- 4 DDR2 slots with 2 memory interfaces
- 4 RGMII gigabit Ethernet interfaces
- 2 serial ports
- TWSI bus
- TWSI bus serial EEPROM
- LM83 temperature sensor

22.11.1 Special Notes

The NICPro2 board does not have any devices connected to the boot bus. It does not have any flash. It can only be booted remotely over the PCI-X bus.

22.12 OCTEON EBH5610 Development Board

The EBH5610 board is based on the Octeon CN5600 or CN5700 and has the following features:

- 4 DDR2 memory slots with 2 memory interfaces
- 8MB CFI compliant NOR flash
- True-IDE compact flash port
- PCIe host mode
- PCIe target mode
- 1 management Ethernet port
- 4 SGMII gigabit Ethernet ports
- 1 XAUI port

- 2 serial ports
- 1 non-EHCI compliant USB port
- TWSI bus
- TWSI bus serial EEPROM
- MCU
- Dallas Semiconductor DS1337 real-time clock
- PAL
- ispPAC POWR1220AT8 Voltage Controller
- 8 character alpha-numeric LED display

22.13 OCTEON EBH5600 Development Board

The EBH5600 is based on the OCTEON CN5600 or CN5700 and has the following features:

- 4 DDR2 memory slots with 2 memory interfaces
- 8MB CFI compliant NOR flash
- PCIe host connector
- 2 PCI-X host connectors via PLX PEX8114 PCIe to PCI-X bridge
- 1 management Ethernet port
- 12 SGMII gigabit Ethernet ports
- 2 serial ports
- 1 non-EHCI compliant USB port
- TWSI bus
- TWSI bus serial EEPROM
- MCU
- Dallas Semiconductor DS1337 real-time clock
- PAL
- ispPAC POWR1220AT8 Voltage Controller
- True-IDE compact flash port
- 8 character alpha-numeric LED display

22.13.1 Special Notes

The PCI-X connectors on this board have problems with some devices, notably the Intel E1000 PCI gigabit Ethernet card. This card may work if two PCI devices are connected. This is caused by an incorrect resistor value.

22.14 OCTEON CN3010 EVB HS5 Development Board

This development board can have either a CN3010 or a CN5010 OCTEON processor. It has the following features:

- 1 DDR2 memory connector (note that only half of the memory is used since only 32-bits are used).
- Compact flash (non-True-IDE mode)
- Two PCI connectors
- One mini-PCI connectors
- 2 serial ports
- 1 non-EHCI compliant USB port
- 5 1 Gigabit Ethernet ports
- PCM telephone support
- Real-time clock
- 8-character alpha-numeric display
- MCU

22.15 OCTEON Generic RAM

The OCTEON generic RAM U-Boot is designed to work on most OCTEON boards when booted remotely. It is a very stripped-down version of U-Boot that lacks support for networking, USB, flash, TWSI and anything on the boot bus. It requires using `oct-remote-boot` to initialize memory and load.

It supports the UART and the PCI console only.

It does not perform any memory initialization. It relies on external tools to initialize memory. If `oct-pci-boot` is unable to initialize the memory then the generic RAM U-Boot cannot be used.

It does not initialize the HFA memory on the CN63XX or CN66XX OCTEON processors. It also does not support any temperature sensors, real-time clock chips or any other peripherals.

By not initializing the temperature sensor on some OCTEON boards such as the NIC 10E, NIC4E , NIC2E and NICPro2 the boards will not shut down if they overheat which **may void the warranty**.

Networking is not supported. Networking will not work in the Linux driver or in simple executive applications due to the fact that the MII addresses and PHY types are unknown and not initialized. There is no easy way to obtain this information at run-time.

Boot bus devices are not initialized. This means that compact flash will not work in Linux. Similarly, neither NOR nor NAND flash will work in simple executive or in Linux.

There is no guarantee that the generic RAM U-Boot will be maintained for future devices or in future versions of the SDK. This version of U-Boot is designed more for testing purposes and should not be used as a basis for a new board. It is only tested occasionally since it is not an official U-Boot version.

Currently the OCTEON specific portion of U-Boot is able to determine which device is being used at run-time so that it is able to seamlessly support various OCTEON processors, however, this may not hold in the future when new devices are released.

Additionally, if certain work-arounds or fixes are required for specific boards then they cannot be supported by this version of U-Boot.

Device Tree support may introduce further problems in later releases of U-Boot which may prevent the generic RAM version from being supported. The device tree used by the Linux kernel is very board-specific.

Future OCTEON processors may not work with the generic RAM version.

Finally, the generic RAM version of U-Boot is not well tested or supported compared to the board-specific versions of U-Boot.