```
#
# (C) Copyright 2000 - 2013
# Wolfgang Denk, DENX Software Engineering, wd@denx.de.
#
# SPDX-License-Identifier:      GPL-2.0+
#

Summary:
========

This directory contains the source code for U-Boot, a boot loader for
Embedded boards based on PowerPC, ARM, MIPS and several other
processors, which can be installed in a boot ROM and used to
initialize and test the hardware or to download and run application
code.

The development of U-Boot is closely related to Linux: some parts of
the source code originate in the Linux source tree, we have some
header files in common, and special provision has been made to
support booting of Linux images.

Some attention has been paid to make this software easily
configurable and extendable. For instance, all monitor commands are
implemented with the same call interface, so that it's very easy to
add new commands. Also, instead of permanently adding rarely used
code (for instance hardware test utilities) to the monitor, you can
load and run it dynamically.


Status:
=======

In general, all boards for which a configuration option exists in the
Makefile have been tested to some extent and can be considered
"working". In fact, many of them are used in production systems.

In case of problems see the CHANGELOG and CREDITS files to find out
who contributed the specific port. The boards.cfg file lists board
maintainers.

Note: There is no CHANGELOG file in the actual U-Boot source tree;
it can be created dynamically from the Git log using:

        make CHANGELOG


Where to get help:
==================

In case you have questions about, problems with or contributions for
U-Boot you should send a message to the U-Boot mailing list at
<u-boot@lists.denx.de>. There is also an archive of previous traffic
on the mailing list - please search the archive before asking FAQ's.
Please see http://lists.denx.de/pipermail/u-boot and
http://dir.gmane.org/gmane.comp.boot-loaders.u-boot


Where to get source code:
=========================

The U-Boot source code is maintained in the git repository at
git://www.denx.de/git/u-boot.git ; you can browse it online at
http://www.denx.de/cgi-bin/gitweb.cgi?p=u-boot.git;a=summary

The "snapshot" links on this page allow you to download tarballs of
any version you might be interested in. Official releases are also
available for FTP download from the ftp://ftp.denx.de/pub/u-boot/
```

```
directory.

Pre-built (and tested) images are available from
ftp://ftp.denx.de/pub/u-boot/images/


Where we come from:
===================

- start from 8xxrom sources
- create PPCBoot project (http://sourceforge.net/projects/ppcboot)
- clean up code
- make it easier to add custom boards
- make it possible to add other [PowerPC] CPUs
- extend functions, especially:
  * Provide extended interface to Linux boot loader
  * S-Record download
  * network boot
  * PCMCIA / CompactFlash / ATA disk / SCSI ... boot
- create ARMBoot project (http://sourceforge.net/projects/armboot)
- add other CPU families (starting with ARM)
- create U-Boot project (http://sourceforge.net/projects/u-boot)
- current project page: see http://www.denx.de/wiki/U-Boot


Names and Spelling:
===================

The "official" name of this project is "Das U-Boot". The spelling
"U-Boot" shall be used in all written text (documentation, comments
in source files etc.). Example:

        This is the README file for the U-Boot project.

File names etc. shall be based on the string "u-boot". Examples:

        include/asm-ppc/u-boot.h

        #include <asm/u-boot.h>

Variable names, preprocessor constants etc. shall be either based on
the string "u_boot" or on "U_BOOT". Example:

        U_BOOT_VERSION          u_boot_logo
        IH_OS_U_BOOT            u_boot_hush_start


Versioning:
===========

Starting with the release in October 2008, the names of the releases
were changed from numerical release numbers without deeper meaning
into a time stamp based numbering. Regular releases are identified by
names consisting of the calendar year and month of the release date.
Additional fields (if present) indicate release candidates or bug fix
releases in "stable" maintenance trees.

Examples:
        U-Boot v2009.11     - Release November 2009
        U-Boot v2009.11.1   - Release 1 in version November 2009 stable tree
        U-Boot v2010.09-rc1 - Release candiate 1 for September 2010 release


Directory Hierarchy:
====================

/arch                   Architecture specific files
```

```
/arc                Files generic to ARC architecture
  /cpu              CPU specific files
    /arc700         Files specific to ARC 700 CPUs
  /lib              Architecture specific library files
/arm                Files generic to ARM architecture
  /cpu              CPU specific files
    /arm720t        Files specific to ARM 720 CPUs
    /arm920t        Files specific to ARM 920 CPUs
      /at91         Files specific to Atmel AT91RM9200 CPU
      /imx          Files specific to Freescale MC9328 i.MX CPUs
      /s3c24x0      Files specific to Samsung S3C24X0 CPUs
    /arm926ejs      Files specific to ARM 926 CPUs
    /arm1136        Files specific to ARM 1136 CPUs
    /pxa            Files specific to Intel XScale PXA CPUs
    /sa1100         Files specific to Intel StrongARM SA1100 CPUs
  /lib              Architecture specific library files
/avr32              Files generic to AVR32 architecture
  /cpu              CPU specific files
  /lib              Architecture specific library files
/blackfin           Files generic to Analog Devices Blackfin architecture
  /cpu              CPU specific files
  /lib              Architecture specific library files
/m68k               Files generic to m68k architecture
  /cpu              CPU specific files
    /mcf52x2        Files specific to Freescale ColdFire MCF52x2 CPUs
    /mcf5227x       Files specific to Freescale ColdFire MCF5227x CPUs
    /mcf532x        Files specific to Freescale ColdFire MCF5329 CPUs
    /mcf5445x       Files specific to Freescale ColdFire MCF5445x CPUs
    /mcf547x_8x     Files specific to Freescale ColdFire MCF547x_8x CPUs
  /lib              Architecture specific library files
/microblaze         Files generic to microblaze architecture
  /cpu              CPU specific files
  /lib              Architecture specific library files
/mips               Files generic to MIPS architecture
  /cpu              CPU specific files
    /mips32         Files specific to MIPS32 CPUs
    /mips64         Files specific to MIPS64 CPUs
  /lib              Architecture specific library files
/nds32              Files generic to NDS32 architecture
  /cpu              CPU specific files
    /n1213          Files specific to Andes Technology N1213 CPUs
  /lib              Architecture specific library files
/nios2              Files generic to Altera NIOS2 architecture
  /cpu              CPU specific files
  /lib              Architecture specific library files
/openrisc           Files generic to OpenRISC architecture
  /cpu              CPU specific files
  /lib              Architecture specific library files
/powerpc            Files generic to PowerPC architecture
  /cpu              CPU specific files
    /74xx_7xx       Files specific to Freescale MPC74xx and 7xx CPUs
    /mpc5xx         Files specific to Freescale MPC5xx CPUs
    /mpc5xxx        Files specific to Freescale MPC5xxx CPUs
    /mpc8xx         Files specific to Freescale MPC8xx CPUs
    /mpc824x        Files specific to Freescale MPC824x CPUs
    /mpc8260        Files specific to Freescale MPC8260 CPUs
    /mpc85xx        Files specific to Freescale MPC85xx CPUs
    /ppc4xx         Files specific to AMCC PowerPC 4xx CPUs
  /lib              Architecture specific library files
/sh                 Files generic to SH architecture
  /cpu              CPU specific files
    /sh2            Files specific to sh2 CPUs
    /sh3            Files specific to sh3 CPUs
    /sh4            Files specific to sh4 CPUs
  /lib              Architecture specific library files
/sparc              Files generic to SPARC architecture
  /cpu              CPU specific files
```

```
      /leon2            Files specific to Gaisler LEON2 SPARC CPU
      /leon3            Files specific to Gaisler LEON3 SPARC CPU
    /lib                Architecture specific library files
  /x86                  Files generic to x86 architecture
    /cpu                CPU specific files
    /lib                Architecture specific library files
/api                    Machine/arch independent API for external apps
/board                  Board dependent files
/common                 Misc architecture independent functions
/disk                   Code for disk drive partition handling
/doc                    Documentation (don't expect too much)
/drivers                Commonly used device drivers
/dts                    Contains Makefile for building internal U-Boot fdt.
/examples               Example code for standalone applications, etc.
/fs                     Filesystem code (cramfs, ext2, jffs2, etc.)
/include                Header Files
/lib                    Files generic to all architectures
  /libfdt               Library files to support flattened device trees
  /lzma                 Library files to support LZMA decompression
  /lzo                  Library files to support LZO decompression
/net                    Networking code
/post                   Power On Self Test
/spl                    Secondary Program Loader framework
/tools                  Tools to build S-Record or U-Boot images, etc.

Software Configuration:
=======================

Configuration is usually done using C preprocessor defines; the
rationale behind that is to avoid dead code whenever possible.

There are two classes of configuration variables:

* Configuration _OPTIONS_:
  These are selectable by the user and have names beginning with
  "CONFIG_".

* Configuration _SETTINGS_:
  These depend on the hardware etc. and should not be meddled with if
  you don't know what you're doing; they have names beginning with
  "CONFIG_SYS_".

Later we will add a configuration tool - probably similar to or even
identical to what's used for the Linux kernel. Right now, we have to
do the configuration by hand, which means creating some symbolic
links and editing some configuration files. We use the TQM8xxL boards
as an example here.


Selection of Processor Architecture and Board Type:
---------------------------------------------------

For all supported boards there are ready-to-use default
configurations available; just type "make <board_name>_config".

Example: For a TQM823L module type:

        cd u-boot
        make TQM823L_config

For the Cogent platform, you need to specify the CPU type as well;
e.g. "make cogent_mpc8xx_config". And also configure the cogent
directory according to the instructions in cogent/README.


Sandbox Environment:
--------------------
```

```
U-Boot can be built natively to run on a Linux host using the 'sandbox'
board. This allows feature development which is not board- or architecture-
specific to be undertaken on a native platform. The sandbox is also used to
run some of U-Boot's tests.

See board/sandbox/sandbox/README.sandbox for more details.


Configuration Options:
----------------------

Configuration depends on the combination of board and CPU type; all
such information is kept in a configuration file
"include/configs/<board_name>.h".

Example: For a TQM823L module, all configuration settings are in
"include/configs/TQM823L.h".


Many of the options are named exactly as the corresponding Linux
kernel configuration options. The intention is to make it easier to
build a config tool - later.


The following options need to be configured:

- CPU Type:     Define exactly one, e.g. CONFIG_MPC85XX.

- Board Type:   Define exactly one, e.g. CONFIG_MPC8540ADS.

- CPU Daughterboard Type: (if CONFIG_ATSTK1000 is defined)
                Define exactly one, e.g. CONFIG_ATSTK1002

- CPU Module Type: (if CONFIG_COGENT is defined)
                Define exactly one of
                CONFIG_CMA286_60_OLD
--- FIXME --- not tested yet:
                CONFIG_CMA286_60, CONFIG_CMA286_21, CONFIG_CMA286_60P,
                CONFIG_CMA287_23, CONFIG_CMA287_50

- Motherboard Type: (if CONFIG_COGENT is defined)
                Define exactly one of
                CONFIG_CMA101, CONFIG_CMA102

- Motherboard I/O Modules: (if CONFIG_COGENT is defined)
                Define one or more of
                CONFIG_CMA302

- Motherboard Options: (if CONFIG_CMA101 or CONFIG_CMA102 are defined)
                Define one or more of
                CONFIG_LCD_HEARTBEAT    - update a character position on
                                          the LCD display every second with
                                          a "rotator" |\-/|\-/

- Marvell Family Member
                CONFIG_SYS_MVFS         - define it if you want to enable
                                          multiple fs option at one time
                                          for marvell soc family

- MPC824X Family Member (if CONFIG_MPC824X is defined)
                Define exactly one of
                CONFIG_MPC8240, CONFIG_MPC8245

- 8xx CPU Options: (if using an MPC8xx CPU)
                CONFIG_8xx_GCLK_FREQ    - deprecated: CPU clock if
                                          get_gclk_freq() cannot work
```

```
                                    e.g. if there is no 32KHz
                                    reference PIT/RTC clock
                CONFIG_8xx_OSCLK        - PLL input clock (either EXTCLK
                                    or XTAL/EXTAL)

- 859/866/885 CPU options: (if using a MPC859 or MPC866 or MPC885 CPU):
                CONFIG_SYS_8xx_CPUCLK_MIN
                CONFIG_SYS_8xx_CPUCLK_MAX
                CONFIG_8xx_CPUCLK_DEFAULT
                        See doc/README.MPC866

                CONFIG_SYS_MEASURE_CPUCLK

                Define this to measure the actual CPU clock instead
                of relying on the correctness of the configured
                values. Mostly useful for board bringup to make sure
                the PLL is locked at the intended frequency. Note
                that this requires a (stable) reference clock (32 kHz
                RTC clock or CONFIG_SYS_8XX_XIN)

                CONFIG_SYS_DELAYED_ICACHE

                Define this option if you want to enable the
                ICache only when Code runs from RAM.

- 85xx CPU Options:
                CONFIG_SYS_PPC64

                Specifies that the core is a 64-bit PowerPC implementation (implements
                the "64" category of the Power ISA). This is necessary for ePAPR
                compliance, among other possible reasons.

                CONFIG_SYS_FSL_TBCLK_DIV

                Defines the core time base clock divider ratio compared to the
                system clock.  On most PQ3 devices this is 8, on newer QorIQ
                devices it can be 16 or 32.  The ratio varies from SoC to Soc.

                CONFIG_SYS_FSL_PCIE_COMPAT

                Defines the string to utilize when trying to match PCIe device
                tree nodes for the given platform.

                CONFIG_SYS_PPC_E500_DEBUG_TLB

                Enables a temporary TLB entry to be used during boot to work
                around limitations in e500v1 and e500v2 external debugger
                support. This reduces the portions of the boot code where
                breakpoints and single stepping do not work.  The value of this
                symbol should be set to the TLB1 entry to be used for this
                purpose.

                CONFIG_SYS_FSL_ERRATUM_A004510

                Enables a workaround for erratum A004510.  If set,
                then CONFIG_SYS_FSL_ERRATUM_A004510_SVR_REV and
                CONFIG_SYS_FSL_CORENET_SNOOPVEC_COREONLY must be set.

                CONFIG_SYS_FSL_ERRATUM_A004510_SVR_REV
                CONFIG_SYS_FSL_ERRATUM_A004510_SVR_REV2 (optional)

                Defines one or two SoC revisions (low 8 bits of SVR)
                for which the A004510 workaround should be applied.

                The rest of SVR is either not relevant to the decision
                of whether the erratum is present (e.g. p2040 versus
                p2041) or is implied by the build target, which controls
```

```
                whether CONFIG_SYS_FSL_ERRATUM_A004510 is set.

                See Freescale App Note 4493 for more information about
                this erratum.

                CONFIG_A003399_NOR_WORKAROUND
                Enables a workaround for IFC erratum A003399. It is only
                requred during NOR boot.

                CONFIG_SYS_FSL_CORENET_SNOOPVEC_COREONLY

                This is the value to write into CCSR offset 0x18600
                according to the A004510 workaround.

                CONFIG_SYS_FSL_DSP_DDR_ADDR
                This value denotes start offset of DDR memory which is
                connected exclusively to the DSP cores.

                CONFIG_SYS_FSL_DSP_M2_RAM_ADDR
                This value denotes start offset of M2 memory
                which is directly connected to the DSP core.

                CONFIG_SYS_FSL_DSP_M3_RAM_ADDR
                This value denotes start offset of M3 memory which is directly
                connected to the DSP core.

                CONFIG_SYS_FSL_DSP_CCSRBAR_DEFAULT
                This value denotes start offset of DSP CCSR space.

                CONFIG_SYS_FSL_SINGLE_SOURCE_CLK
                Single Source Clock is clocking mode present in some of FSL SoC's.
                In this mode, a single differential clock is used to supply
                clocks to the sysclock, ddrclock and usbclock.

                CONFIG_SYS_CPC_REINIT_F
                This CONFIG is defined when the CPC is configured as SRAM at the
                time of U-boot entry and is required to be re-initialized.

                CONFIG_DEEP_SLEEP
                Inidcates this SoC supports deep sleep feature. If deep sleep is
                supported, core will start to execute uboot when wakes up.

- Generic CPU options:
                CONFIG_SYS_GENERIC_GLOBAL_DATA
                Defines global data is initialized in generic board board_init_f().
                If this macro is defined, global data is created and cleared in
                generic board board_init_f(). Without this macro, architecture/board
                should initialize global data before calling board_init_f().

                CONFIG_SYS_BIG_ENDIAN, CONFIG_SYS_LITTLE_ENDIAN

                Defines the endianess of the CPU. Implementation of those
                values is arch specific.

                CONFIG_SYS_FSL_DDR
                Freescale DDR driver in use. This type of DDR controller is
                found in mpc83xx, mpc85xx, mpc86xx as well as some ARM core
                SoCs.

                CONFIG_SYS_FSL_DDR_ADDR
                Freescale DDR memory-mapped register base.

                CONFIG_SYS_FSL_DDR_EMU
                Specify emulator support for DDR. Some DDR features such as
                deskew training are not available.

                CONFIG_SYS_FSL_DDRC_GEN1
```

```
Freescale DDR1 controller.

CONFIG_SYS_FSL_DDRC_GEN2
Freescale DDR2 controller.

CONFIG_SYS_FSL_DDRC_GEN3
Freescale DDR3 controller.

CONFIG_SYS_FSL_DDRC_GEN4
Freescale DDR4 controller.

CONFIG_SYS_FSL_DDRC_ARM_GEN3
Freescale DDR3 controller for ARM-based SoCs.

CONFIG_SYS_FSL_DDR1
Board config to use DDR1. It can be enabled for SoCs with
Freescale DDR1 or DDR2 controllers, depending on the board
implemetation.

CONFIG_SYS_FSL_DDR2
Board config to use DDR2. It can be eanbeld for SoCs with
Freescale DDR2 or DDR3 controllers, depending on the board
implementation.

CONFIG_SYS_FSL_DDR3
Board config to use DDR3. It can be enabled for SoCs with
Freescale DDR3 or DDR3L controllers.

CONFIG_SYS_FSL_DDR3L
Board config to use DDR3L. It can be enabled for SoCs with
DDR3L controllers.

CONFIG_SYS_FSL_DDR4
Board config to use DDR4. It can be enabled for SoCs with
DDR4 controllers.

CONFIG_SYS_FSL_IFC_BE
Defines the IFC controller register space as Big Endian

CONFIG_SYS_FSL_IFC_LE
Defines the IFC controller register space as Little Endian

CONFIG_SYS_FSL_PBL_PBI
It enables addition of RCW (Power on reset configuration) in built image.
Please refer doc/README.pblimage for more details

CONFIG_SYS_FSL_PBL_RCW
It adds PBI(pre-boot instructions) commands in u-boot build image.
PBI commands can be used to configure SoC before it starts the execution.
Please refer doc/README.pblimage for more details

CONFIG_SPL_FSL_PBL
It adds a target to create boot binary having SPL binary in PBI format
concatenated with u-boot binary.

CONFIG_SYS_FSL_DDR_BE
Defines the DDR controller register space as Big Endian

CONFIG_SYS_FSL_DDR_LE
Defines the DDR controller register space as Little Endian

CONFIG_SYS_FSL_DDR_SDRAM_BASE_PHY
Physical address from the view of DDR controllers. It is the
same as CONFIG_SYS_DDR_SDRAM_BASE for  all Power SoCs. But
it could be different for ARM SoCs.

CONFIG_SYS_FSL_DDR_INTLV_256B
```

```
                DDR controller interleaving on 256-byte. This is a special
                interleaving mode, handled by Dickens for Freescale layerscape
                SoCs with ARM core.

- Intel Monahans options:
                CONFIG_SYS_MONAHANS_RUN_MODE_OSC_RATIO

                Defines the Monahans run mode to oscillator
                ratio. Valid values are 8, 16, 24, 31. The core
                frequency is this value multiplied by 13 MHz.

                CONFIG_SYS_MONAHANS_TURBO_RUN_MODE_RATIO

                Defines the Monahans turbo mode to oscillator
                ratio. Valid values are 1 (default if undefined) and
                2. The core frequency as calculated above is multiplied
                by this value.

- MIPS CPU options:
                CONFIG_SYS_INIT_SP_OFFSET

                Offset relative to CONFIG_SYS_SDRAM_BASE for initial stack
                pointer. This is needed for the temporary stack before
                relocation.

                CONFIG_SYS_MIPS_CACHE_MODE

                Cache operation mode for the MIPS CPU.
                See also arch/mips/include/asm/mipsregs.h.
                Possible values are:
                        CONF_CM_CACHABLE_NO_WA
                        CONF_CM_CACHABLE_WA
                        CONF_CM_UNCACHED
                        CONF_CM_CACHABLE_NONCOHERENT
                        CONF_CM_CACHABLE_CE
                        CONF_CM_CACHABLE_COW
                        CONF_CM_CACHABLE_CUW
                        CONF_CM_CACHABLE_ACCELERATED

                CONFIG_SYS_XWAY_EBU_BOOTCFG

                Special option for Lantiq XWAY SoCs for booting from NOR flash.
                See also arch/mips/cpu/mips32/start.S.

                CONFIG_XWAY_SWAP_BYTES

                Enable compilation of tools/xway-swap-bytes needed for Lantiq
                XWAY SoCs for booting from NOR flash. The U-Boot image needs to
                be swapped if a flash programmer is used.

- ARM options:
                CONFIG_SYS_EXCEPTION_VECTORS_HIGH

                Select high exception vectors of the ARM core, e.g., do not
                clear the V bit of the c1 register of CP15.

                CONFIG_SYS_THUMB_BUILD

                Use this flag to build U-Boot using the Thumb instruction
                set for ARM architectures. Thumb instruction set provides
                better code density. For ARM architectures that support
                Thumb2 this flag will result in Thumb2 code generated by
                GCC.

                CONFIG_ARM_ERRATA_716044
                CONFIG_ARM_ERRATA_742230
                CONFIG_ARM_ERRATA_743622
```

```
                CONFIG_ARM_ERRATA_751472
                CONFIG_ARM_ERRATA_794072
                CONFIG_ARM_ERRATA_761320

                If set, the workarounds for these ARM errata are applied early
                during U-Boot startup. Note that these options force the
                workarounds to be applied; no CPU-type/version detection
                exists, unlike the similar options in the Linux kernel. Do not
                set these options unless they apply!

- CPU timer options:
                CONFIG_SYS_HZ

                The frequency of the timer returned by get_timer().
                get_timer() must operate in milliseconds and this CONFIG
                option must be set to 1000.

- Linux Kernel Interface:
                CONFIG_CLOCKS_IN_MHZ

                U-Boot stores all clock information in Hz
                internally. For binary compatibility with older Linux
                kernels (which expect the clocks passed in the
                bd_info data to be in MHz) the environment variable
                "clocks_in_mhz" can be defined so that U-Boot
                converts clock data to MHZ before passing it to the
                Linux kernel.
                When CONFIG_CLOCKS_IN_MHZ is defined, a definition of
                "clocks_in_mhz=1" is automatically included in the
                default environment.

                CONFIG_MEMSIZE_IN_BYTES         [relevant for MIPS only]

                When transferring memsize parameter to linux, some versions
                expect it to be in bytes, others in MB.
                Define CONFIG_MEMSIZE_IN_BYTES to make it in bytes.

                CONFIG_OF_LIBFDT

                New kernel versions are expecting firmware settings to be
                passed using flattened device trees (based on open firmware
                concepts).

                CONFIG_OF_LIBFDT
                 * New libfdt-based support
                 * Adds the "fdt" command
                 * The bootm command automatically updates the fdt

                OF_CPU - The proper name of the cpus node (only required for
                        MPC512X and MPC5xxx based boards).
                OF_SOC - The proper name of the soc node (only required for
                        MPC512X and MPC5xxx based boards).
                OF_TBCLK - The timebase frequency.
                OF_STDOUT_PATH - The path to the console device

                boards with QUICC Engines require OF_QE to set UCC MAC
                addresses

                CONFIG_OF_BOARD_SETUP

                Board code has addition modification that it wants to make
                to the flat device tree before handing it off to the kernel

                CONFIG_OF_BOOT_CPU

                This define fills in the correct boot CPU in the boot
                param header, the default value is zero if undefined.
```

```
                CONFIG_OF_IDE_FIXUP

                U-Boot can detect if an IDE device is present or not.
                If not, and this new config option is activated, U-Boot
                removes the ATA node from the DTS before booting Linux,
                so the Linux IDE driver does not probe the device and
                crash. This is needed for buggy hardware (uc101) where
                no pull down resistor is connected to the signal IDE5V_DD7.

                CONFIG_MACH_TYPE        [relevant for ARM only][mandatory]

                This setting is mandatory for all boards that have only one
                machine type and must be used to specify the machine type
                number as it appears in the ARM machine registry
                (see http://www.arm.linux.org.uk/developer/machines/).
                Only boards that have multiple machine types supported
                in a single configuration file and the machine type is
                runtime discoverable, do not have to use this setting.

- vxWorks boot parameters:

                bootvx constructs a valid bootline using the following
                environments variables: bootfile, ipaddr, serverip, hostname.
                It loads the vxWorks image pointed bootfile.

                CONFIG_SYS_VXWORKS_BOOT_DEVICE - The vxworks device name
                CONFIG_SYS_VXWORKS_MAC_PTR - Ethernet 6 byte MA -address
                CONFIG_SYS_VXWORKS_SERVERNAME - Name of the server
                CONFIG_SYS_VXWORKS_BOOT_ADDR - Address of boot parameters

                CONFIG_SYS_VXWORKS_ADD_PARAMS

                Add it at the end of the bootline. E.g "u=username pw=secret"

                Note: If a "bootargs" environment is defined, it will overwride
                the defaults discussed just above.

- Cache Configuration:
                CONFIG_SYS_ICACHE_OFF - Do not enable instruction cache in U-Boot
                CONFIG_SYS_DCACHE_OFF - Do not enable data cache in U-Boot
                CONFIG_SYS_L2CACHE_OFF- Do not enable L2 cache in U-Boot

- Cache Configuration for ARM:
                CONFIG_SYS_L2_PL310 - Enable support for ARM PL310 L2 cache
                                      controller
                CONFIG_SYS_PL310_BASE - Physical base address of PL310
                                        controller register space

- Serial Ports:
                CONFIG_PL010_SERIAL

                Define this if you want support for Amba PrimeCell PL010 UARTs.

                CONFIG_PL011_SERIAL

                Define this if you want support for Amba PrimeCell PL011 UARTs.

                CONFIG_PL011_CLOCK

                If you have Amba PrimeCell PL011 UARTs, set this variable to
                the clock speed of the UARTs.

                CONFIG_PL01x_PORTS

                If you have Amba PrimeCell PL010 or PL011 UARTs on your board,
                define this to a list of base addresses for each (supported)
```

```
                port. See e.g. include/configs/versatile.h

                CONFIG_PL011_SERIAL_RLCR

                Some vendor versions of PL011 serial ports (e.g. ST-Ericsson U8500)
                have separate receive and transmit line control registers.  Set
                this variable to initialize the extra register.

                CONFIG_PL011_SERIAL_FLUSH_ON_INIT

                On some platforms (e.g. U8500) U-Boot is loaded by a second stage
                boot loader that has already initialized the UART.  Define this
                variable to flush the UART at init time.

                CONFIG_SERIAL_HW_FLOW_CONTROL

                Define this variable to enable hw flow control in serial driver.
                Current user of this option is drivers/serial/nsl16550.c driver

- Console Interface:
                Depending on board, define exactly one serial port
                (like CONFIG_8xx_CONS_SMC1, CONFIG_8xx_CONS_SMC2,
                CONFIG_8xx_CONS_SCC1, ...), or switch off the serial
                console by defining CONFIG_8xx_CONS_NONE

                Note: if CONFIG_8xx_CONS_NONE is defined, the serial
                port routines must be defined elsewhere
                (i.e. serial_init(), serial_getc(), ...)

                CONFIG_CFB_CONSOLE
                Enables console device for a color framebuffer. Needs following
                defines (cf. smiLynxEM, i8042)
                        VIDEO_FB_LITTLE_ENDIAN  graphic memory organisation
                                                (default big endian)
                        VIDEO_HW_RECTFILL       graphic chip supports
                                                rectangle fill
                                                (cf. smiLynxEM)
                        VIDEO_HW_BITBLT         graphic chip supports
                                                bit-blit (cf. smiLynxEM)
                        VIDEO_VISIBLE_COLS      visible pixel columns
                                                (cols=pitch)
                        VIDEO_VISIBLE_ROWS      visible pixel rows
                        VIDEO_PIXEL_SIZE        bytes per pixel
                        VIDEO_DATA_FORMAT       graphic data format
                                                (0-5, cf. cfb_console.c)
                        VIDEO_FB_ADRS           framebuffer address
                        VIDEO_KBD_INIT_FCT      keyboard int fct
                                                (i.e. i8042_kbd_init())
                        VIDEO_TSTC_FCT          test char fct
                                                (i.e. i8042_tstc)
                        VIDEO_GETC_FCT          get char fct
                                                (i.e. i8042_getc)
                        CONFIG_CONSOLE_CURSOR   cursor drawing on/off
                                                (requires blink timer
                                                cf. i8042.c)
                        CONFIG_SYS_CONSOLE_BLINK_COUNT blink interval (cf. i8042.c)
                        CONFIG_CONSOLE_TIME     display time/date info in
                                                upper right corner
                                                (requires CONFIG_CMD_DATE)
                        CONFIG_VIDEO_LOGO       display Linux logo in
                                                upper left corner
                        CONFIG_VIDEO_BMP_LOGO   use bmp_logo.h instead of
                                                linux_logo.h for logo.
                                                Requires CONFIG_VIDEO_LOGO
                        CONFIG_CONSOLE_EXTRA_INFO
                                                additional board info beside
                                                the logo
```

```
                When CONFIG_CFB_CONSOLE_ANSI is defined, console will support
                a limited number of ANSI escape sequences (cursor control,
                erase functions and limited graphics rendition control).

                When CONFIG_CFB_CONSOLE is defined, video console is
                default i/o. Serial console can be forced with
                environment 'console=serial'.

                When CONFIG_SILENT_CONSOLE is defined, all console
                messages (by U-Boot and Linux!) can be silenced with
                the "silent" environment variable. See
                doc/README.silent for more information.

                CONFIG_SYS_CONSOLE_BG_COL: define the backgroundcolor, default
                        is 0x00.
                CONFIG_SYS_CONSOLE_FG_COL: define the foregroundcolor, default
                        is 0xa0.

- Console Baudrate:
                CONFIG_BAUDRATE - in bps
                Select one of the baudrates listed in
                CONFIG_SYS_BAUDRATE_TABLE, see below.
                CONFIG_SYS_BRGCLK_PRESCALE, baudrate prescale

- Console Rx buffer length
                With CONFIG_SYS_SMC_RXBUFLEN it is possible to define
                the maximum receive buffer length for the SMC.
                This option is actual only for 82xx and 8xx possible.
                If using CONFIG_SYS_SMC_RXBUFLEN also CONFIG_SYS_MAXIDLE
                must be defined, to setup the maximum idle timeout for
                the SMC.

- Pre-Console Buffer:
                Prior to the console being initialised (i.e. serial UART
                initialised etc) all console output is silently discarded.
                Defining CONFIG_PRE_CONSOLE_BUFFER will cause U-Boot to
                buffer any console messages prior to the console being
                initialised to a buffer of size CONFIG_PRE_CON_BUF_SZ
                bytes located at CONFIG_PRE_CON_BUF_ADDR. The buffer is
                a circular buffer, so if more than CONFIG_PRE_CON_BUF_SZ
                bytes are output before the console is initialised, the
                earlier bytes are discarded.

                'Sane' compilers will generate smaller code if
                CONFIG_PRE_CON_BUF_SZ is a power of 2

- Safe printf() functions
                Define CONFIG_SYS_VSNPRINTF to compile in safe versions of
                the printf() functions. These are defined in
                include/vsprintf.h and include snprintf(), vsnprintf() and
                so on. Code size increase is approximately 300-500 bytes.
                If this option is not given then these functions will
                silently discard their buffer size argument - this means
                you are not getting any overflow checking in this case.

- Boot Delay:   CONFIG_BOOTDELAY - in seconds
                Delay before automatically booting the default image;
                set to -1 to disable autoboot.
                set to -2 to autoboot with no delay and not check for abort
                (even when CONFIG_ZERO_BOOTDELAY_CHECK is defined).

                See doc/README.autoboot for these options that
                work with CONFIG_BOOTDELAY. None are required.
                CONFIG_BOOT_RETRY_TIME
                CONFIG_BOOT_RETRY_MIN
                CONFIG_AUTOBOOT_KEYED
```

```
                CONFIG_AUTOBOOT_PROMPT
                CONFIG_AUTOBOOT_DELAY_STR
                CONFIG_AUTOBOOT_STOP_STR
                CONFIG_AUTOBOOT_DELAY_STR2
                CONFIG_AUTOBOOT_STOP_STR2
                CONFIG_ZERO_BOOTDELAY_CHECK
                CONFIG_RESET_TO_RETRY

- Autoboot Command:
                CONFIG_BOOTCOMMAND
                Only needed when CONFIG_BOOTDELAY is enabled;
                define a command string that is automatically executed
                when no character is read on the console interface
                within "Boot Delay" after reset.

                CONFIG_BOOTARGS
                This can be used to pass arguments to the bootm
                command. The value of CONFIG_BOOTARGS goes into the
                environment value "bootargs".

                CONFIG_RAMBOOT and CONFIG_NFSBOOT
                The value of these goes into the environment as
                "ramboot" and "nfsboot" respectively, and can be used
                as a convenience, when switching between booting from
                RAM and NFS.

- Bootcount:
                CONFIG_BOOTCOUNT_LIMIT
                Implements a mechanism for detecting a repeating reboot
                cycle, see:
                http://www.denx.de/wiki/view/DULG/UBootBootCountLimit

                CONFIG_BOOTCOUNT_ENV
                If no softreset save registers are found on the hardware
                "bootcount" is stored in the environment. To prevent a
                saveenv on all reboots, the environment variable
                "upgrade_available" is used. If "upgrade_available" is
                0, "bootcount" is always 0, if "upgrade_available" is
                1 "bootcount" is incremented in the environment.
                So the Userspace Applikation must set the "upgrade_available"
                and "bootcount" variable to 0, if a boot was successfully.

- Pre-Boot Commands:
                CONFIG_PREBOOT

                When this option is #defined, the existence of the
                environment variable "preboot" will be checked
                immediately before starting the CONFIG_BOOTDELAY
                countdown and/or running the auto-boot command resp.
                entering interactive mode.

                This feature is especially useful when "preboot" is
                automatically generated or modified. For an example
                see the LWMON board specific code: here "preboot" is
                modified when the user holds down a certain
                combination of keys on the (special) keyboard when
                booting the systems

- Serial Download Echo Mode:
                CONFIG_LOADS_ECHO
                If defined to 1, all characters received during a
                serial download (using the "loads" command) are
                echoed back. This might be needed by some terminal
                emulations (like "cu"), but may as well just take
                time on others. This setting #define's the initial
                value of the "loads_echo" environment variable.
```

```
- Kgdb Serial Baudrate: (if CONFIG_CMD_KGDB is defined)
               CONFIG_KGDB_BAUDRATE
               Select one of the baudrates listed in
               CONFIG_SYS_BAUDRATE_TABLE, see below.

- Monitor Functions:
               Monitor commands can be included or excluded
               from the build by using the #include files
               <config_cmd_all.h> and #undef'ing unwanted
               commands, or using <config_cmd_default.h>
               and augmenting with additional #define's
               for wanted commands.

               The default command configuration includes all commands
               except those marked below with a "*".

               CONFIG_CMD_AES            AES 128 CBC encrypt/decrypt
               CONFIG_CMD_ASKENV       * ask for env variable
               CONFIG_CMD_BDI            bdinfo
               CONFIG_CMD_BEDBUG       * Include BedBug Debugger
               CONFIG_CMD_BMP          * BMP support
               CONFIG_CMD_BSP          * Board specific commands
               CONFIG_CMD_BOOTD          bootd
               CONFIG_CMD_CACHE        * icache, dcache
               CONFIG_CMD_CLK          * clock command support
               CONFIG_CMD_CONSOLE        coninfo
               CONFIG_CMD_CRC32        * crc32
               CONFIG_CMD_DATE         * support for RTC, date/time...
               CONFIG_CMD_DHCP         * DHCP support
               CONFIG_CMD_DIAG         * Diagnostics
               CONFIG_CMD_DS4510       * ds4510 I2C gpio commands
               CONFIG_CMD_DS4510_INFO  * ds4510 I2C info command
               CONFIG_CMD_DS4510_MEM   * ds4510 I2C eeprom/sram commansd
               CONFIG_CMD_DS4510_RST   * ds4510 I2C rst command
               CONFIG_CMD_DTT          * Digital Therm and Thermostat
               CONFIG_CMD_ECHO           echo arguments
               CONFIG_CMD_EDITENV        edit env variable
               CONFIG_CMD_EEPROM       * EEPROM read/write support
               CONFIG_CMD_ELF          * bootelf, bootvx
               CONFIG_CMD_ENV_CALLBACK * display details about env callbacks
               CONFIG_CMD_ENV_FLAGS    * display details about env flags
               CONFIG_CMD_ENV_EXISTS   * check existence of env variable
               CONFIG_CMD_EXPORTENV    * export the environment
               CONFIG_CMD_EXT2         * ext2 command support
               CONFIG_CMD_EXT4         * ext4 command support
               CONFIG_CMD_FS_GENERIC   * filesystem commands (e.g. load, ls)
                                         that work for multiple fs types
               CONFIG_CMD_SAVEENV        saveenv
               CONFIG_CMD_FDC          * Floppy Disk Support
               CONFIG_CMD_FAT          * FAT command support
               CONFIG_CMD_FLASH          flinfo, erase, protect
               CONFIG_CMD_FPGA           FPGA device initialization support
               CONFIG_CMD_FUSE         * Device fuse support
               CONFIG_CMD_GETTIME      * Get time since boot
               CONFIG_CMD_GO           * the 'go' command (exec code)
               CONFIG_CMD_GREPENV      * search environment
               CONFIG_CMD_HASH         * calculate hash / digest
               CONFIG_CMD_HWFLOW       * RTS/CTS hw flow control
               CONFIG_CMD_I2C          * I2C serial bus support
               CONFIG_CMD_IDE          * IDE harddisk support
               CONFIG_CMD_IMI            iminfo
               CONFIG_CMD_IMLS           List all images found in NOR flash
               CONFIG_CMD_IMLS_NAND    * List all images found in NAND flash
               CONFIG_CMD_IMMAP        * IMMR dump support
               CONFIG_CMD_IOTRACE      * I/O tracing for debugging
               CONFIG_CMD_IMPORTENV    * import an environment
               CONFIG_CMD_INI          * import data from an ini file into the env
```

```
		CONFIG_CMD_IRQ		* irqinfo
		CONFIG_CMD_ITEST	  Integer/string test of 2 values
		CONFIG_CMD_JFFS2	* JFFS2 Support
		CONFIG_CMD_KGDB		* kgdb
		CONFIG_CMD_LDRINFO	* ldrinfo (display Blackfin loader)
		CONFIG_CMD_LINK_LOCAL	* link-local IP address auto-configuration
					  (169.254.*.*)
		CONFIG_CMD_LOADB	  loadb
		CONFIG_CMD_LOADS	  loads
		CONFIG_CMD_MD5SUM	* print md5 message digest
					  (requires CONFIG_CMD_MEMORY and CONFIG_MD5)
		CONFIG_CMD_MEMINFO	* Display detailed memory information
		CONFIG_CMD_MEMORY	  md, mm, nm, mw, cp, cmp, crc, base,
					  loop, loopw
		CONFIG_CMD_MEMTEST	* mtest
		CONFIG_CMD_MISC		  Misc functions like sleep etc
		CONFIG_CMD_MMC		* MMC memory mapped support
		CONFIG_CMD_MII		* MII utility commands
		CONFIG_CMD_MTDPARTS	* MTD partition support
		CONFIG_CMD_NAND		* NAND support
		CONFIG_CMD_NET		  bootp, tftpboot, rarpboot
		CONFIG_CMD_NFS		  NFS support
		CONFIG_CMD_PCA953X	* PCA953x I2C gpio commands
		CONFIG_CMD_PCA953X_INFO * PCA953x I2C gpio info command
		CONFIG_CMD_PCI		* pciinfo
		CONFIG_CMD_PCMCIA		* PCMCIA support
		CONFIG_CMD_PING		* send ICMP ECHO_REQUEST to network
					  host
		CONFIG_CMD_PORTIO	* Port I/O
		CONFIG_CMD_READ		* Read raw data from partition
		CONFIG_CMD_REGINFO	* Register dump
		CONFIG_CMD_RUN		  run command in env variable
		CONFIG_CMD_SANDBOX	* sb command to access sandbox features
		CONFIG_CMD_SAVES	* save S record dump
		CONFIG_CMD_SCSI		* SCSI Support
		CONFIG_CMD_SDRAM	* print SDRAM configuration information
					  (requires CONFIG_CMD_I2C)
		CONFIG_CMD_SETGETDCR	  Support for DCR Register access
					  (4xx only)
		CONFIG_CMD_SF		* Read/write/erase SPI NOR flash
		CONFIG_CMD_SHA1SUM	* print sha1 memory digest
					  (requires CONFIG_CMD_MEMORY)
		CONFIG_CMD_SOFTSWITCH	* Soft switch setting command for BF60x
		CONFIG_CMD_SOURCE	  "source" command Support
		CONFIG_CMD_SPI		* SPI serial bus support
		CONFIG_CMD_TFTPSRV	* TFTP transfer in server mode
		CONFIG_CMD_TFTPPUT	* TFTP put command (upload)
		CONFIG_CMD_TIME		* run command and report execution time (ARM specific)
		CONFIG_CMD_TIMER	* access to the system tick timer
		CONFIG_CMD_USB		* USB support
		CONFIG_CMD_CDP		* Cisco Discover Protocol support
		CONFIG_CMD_MFSL		* Microblaze FSL support
		CONFIG_CMD_XIMG		  Load part of Multi Image
		CONFIG_CMD_UUID		* Generate random UUID or GUID string

		EXAMPLE: If you want all functions except of network
		support you can write:

		#include "config_cmd_all.h"
		#undef CONFIG_CMD_NET

	Other Commands:
		fdt (flattened device tree) command: CONFIG_OF_LIBFDT

	Note:	Don't enable the "icache" and "dcache" commands
		(configuration option CONFIG_CMD_CACHE) unless you know
		what you (and your U-Boot users) are doing. Data
```

```
                cache cannot be enabled on systems like the 8xx or
                8260 (where accesses to the IMMR region must be
                uncached), and it cannot be disabled on all other
                systems where we (mis-) use the data cache to hold an
                initial stack and some data.


                XXX - this list needs to get updated!

- Regular expression support:
                CONFIG_REGEX
                If this variable is defined, U-Boot is linked against
                the SLRE (Super Light Regular Expression) library,
                which adds regex support to some commands, as for
                example "env grep" and "setexpr".

- Device tree:
                CONFIG_OF_CONTROL
                If this variable is defined, U-Boot will use a device tree
                to configure its devices, instead of relying on statically
                compiled #defines in the board file. This option is
                experimental and only available on a few boards. The device
                tree is available in the global data as gd->fdt_blob.

                U-Boot needs to get its device tree from somewhere. This can
                be done using one of the two options below:

                CONFIG_OF_EMBED
                If this variable is defined, U-Boot will embed a device tree
                binary in its image. This device tree file should be in the
                board directory and called <soc>-<board>.dts. The binary file
                is then picked up in board_init_f() and made available through
                the global data structure as gd->blob.

                CONFIG_OF_SEPARATE
                If this variable is defined, U-Boot will build a device tree
                binary. It will be called u-boot.dtb. Architecture-specific
                code will locate it at run-time. Generally this works by:

                        cat u-boot.bin u-boot.dtb >image.bin

                and in fact, U-Boot does this for you, creating a file called
                u-boot-dtb.bin which is useful in the common case. You can
                still use the individual files if you need something more
                exotic.

- Watchdog:
                CONFIG_WATCHDOG
                If this variable is defined, it enables watchdog
                support for the SoC. There must be support in the SoC
                specific code for a watchdog. For the 8xx and 8260
                CPUs, the SIU Watchdog feature is enabled in the SYPCR
                register.  When supported for a specific SoC is
                available, then no further board specific code should
                be needed to use it.

                CONFIG_HW_WATCHDOG
                When using a watchdog circuitry external to the used
                SoC, then define this variable and provide board
                specific code for the "hw_watchdog_reset" function.

- U-Boot Version:
                CONFIG_VERSION_VARIABLE
                If this variable is defined, an environment variable
                named "ver" is created by U-Boot showing the U-Boot
                version as printed by the "version" command.
                Any change to this variable will be reverted at the
```

```
                next reset.

- Real-Time Clock:

                When CONFIG_CMD_DATE is selected, the type of the RTC
                has to be selected, too. Define exactly one of the
                following options:

                CONFIG_RTC_MPC8xx       - use internal RTC of MPC8xx
                CONFIG_RTC_PCF8563      - use Philips PCF8563 RTC
                CONFIG_RTC_MC13XXX      - use MC13783 or MC13892 RTC
                CONFIG_RTC_MC146818     - use MC146818 RTC
                CONFIG_RTC_DS1307       - use Maxim, Inc. DS1307 RTC
                CONFIG_RTC_DS1337       - use Maxim, Inc. DS1337 RTC
                CONFIG_RTC_DS1338       - use Maxim, Inc. DS1338 RTC
                CONFIG_RTC_DS164x       - use Dallas DS164x RTC
                CONFIG_RTC_ISL1208      - use Intersil ISL1208 RTC
                CONFIG_RTC_MAX6900      - use Maxim, Inc. MAX6900 RTC
                CONFIG_SYS_RTC_DS1337_NOOSC     - Turn off the OSC output for DS1337
                CONFIG_SYS_RV3029_TCR   - enable trickle charger on
                                          RV3029 RTC.

                Note that if the RTC uses I2C, then the I2C interface
                must also be configured. See I2C Support, below.

- GPIO Support:
                CONFIG_PCA953X          - use NXP's PCA953X series I2C GPIO

                The CONFIG_SYS_I2C_PCA953X_WIDTH option specifies a list of
                chip-ngpio pairs that tell the PCA953X driver the number of
                pins supported by a particular chip.

                Note that if the GPIO device uses I2C, then the I2C interface
                must also be configured. See I2C Support, below.

- I/O tracing:
                When CONFIG_IO_TRACE is selected, U-Boot intercepts all I/O
                accesses and can checksum them or write a list of them out
                to memory. See the 'iotrace' command for details. This is
                useful for testing device drivers since it can confirm that
                the driver behaves the same way before and after a code
                change. Currently this is supported on sandbox and arm. To
                add support for your architecture, add '#include <iotrace.h>'
                to the bottom of arch/<arch>/include/asm/io.h and test.

                Example output from the 'iotrace stats' command is below.
                Note that if the trace buffer is exhausted, the checksum will
                still continue to operate.

                        iotrace is enabled
                        Start:  10000000        (buffer start address)
                        Size:   00010000        (buffer size)
                        Offset: 00000120        (current buffer offset)
                        Output: 10000120        (start + offset)
                        Count:  00000018        (number of trace records)
                        CRC32:  9526fb66        (CRC32 of all trace records)

- Timestamp Support:

                When CONFIG_TIMESTAMP is selected, the timestamp
                (date and time) of an image is printed by image
                commands like bootm or iminfo. This option is
                automatically enabled when you select CONFIG_CMD_DATE .

- Partition Labels (disklabels) Supported:
                Zero or more of the following:
                CONFIG_MAC_PARTITION   Apple's MacOS partition table.
```

```
                CONFIG_DOS_PARTITION   MS Dos partition table, traditional on the
                                       Intel architecture, USB sticks, etc.
                CONFIG_ISO_PARTITION   ISO partition table, used on CDROM etc.
                CONFIG_EFI_PARTITION   GPT partition table, common when EFI is the
                                       bootloader.  Note 2TB partition limit; see
                                       disk/part_efi.c
                CONFIG_MTD_PARTITIONS  Memory Technology Device partition table.

                If IDE or SCSI support is enabled (CONFIG_CMD_IDE or
                CONFIG_CMD_SCSI) you must configure support for at
                least one non-MTD partition type as well.

- IDE Reset method:
                CONFIG_IDE_RESET_ROUTINE - this is defined in several
                board configurations files but used nowhere!

                CONFIG_IDE_RESET - is this is defined, IDE Reset will
                be performed by calling the function
                        ide_set_reset(int reset)
                which has to be defined in a board specific file

- ATAPI Support:
                CONFIG_ATAPI

                Set this to enable ATAPI support.

- LBA48 Support
                CONFIG_LBA48

                Set this to enable support for disks larger than 137GB
                Also look at CONFIG_SYS_64BIT_LBA.
                Whithout these , LBA48 support uses 32bit variables and will 'only'
                support disks up to 2.1TB.

                CONFIG_SYS_64BIT_LBA:
                        When enabled, makes the IDE subsystem use 64bit sector addresses.
                        Default is 32bit.

- SCSI Support:
                At the moment only there is only support for the
                SYM53C8XX SCSI controller; define
                CONFIG_SCSI_SYM53C8XX to enable it.

                CONFIG_SYS_SCSI_MAX_LUN [8], CONFIG_SYS_SCSI_MAX_SCSI_ID [7] and
                CONFIG_SYS_SCSI_MAX_DEVICE [CONFIG_SYS_SCSI_MAX_SCSI_ID *
                CONFIG_SYS_SCSI_MAX_LUN] can be adjusted to define the
                maximum numbers of LUNs, SCSI ID's and target
                devices.
                CONFIG_SYS_SCSI_SYM53C8XX_CCF to fix clock timing (80Mhz)

                The environment variable 'scsidevs' is set to the number of
                SCSI devices found during the last scan.

- NETWORK Support (PCI):
                CONFIG_E1000
                Support for Intel 8254x/8257x gigabit chips.

                CONFIG_E1000_SPI
                Utility code for direct access to the SPI bus on Intel 8257x.
                This does not do anything useful unless you set at least one
                of CONFIG_CMD_E1000 or CONFIG_E1000_SPI_GENERIC.

                CONFIG_E1000_SPI_GENERIC
                Allow generic access to the SPI bus on the Intel 8257x, for
                example with the "sspi" command.

                CONFIG_CMD_E1000
```

```
                Management command for E1000 devices.  When used on devices
                with SPI support you can reprogram the EEPROM from U-Boot.

                CONFIG_E1000_FALLBACK_MAC
                default MAC for empty EEPROM after production.

                CONFIG_EEPRO100
                Support for Intel 82557/82559/82559ER chips.
                Optional CONFIG_EEPRO100_SROM_WRITE enables EEPROM
                write routine for first time initialisation.

                CONFIG_TULIP
                Support for Digital 2114x chips.
                Optional CONFIG_TULIP_SELECT_MEDIA for board specific
                modem chip initialisation (KS8761/QS6611).

                CONFIG_NATSEMI
                Support for National dp83815 chips.

                CONFIG_NS8382X
                Support for National dp8382[01] gigabit chips.

- NETWORK Support (other):

                CONFIG_DRIVER_AT91EMAC
                Support for AT91RM9200 EMAC.

                        CONFIG_RMII
                        Define this to use reduced MII inteface

                        CONFIG_DRIVER_AT91EMAC_QUIET
                        If this defined, the driver is quiet.
                        The driver doen't show link status messages.

                CONFIG_CALXEDA_XGMAC
                Support for the Calxeda XGMAC device

                CONFIG_LAN91C96
                Support for SMSC's LAN91C96 chips.

                        CONFIG_LAN91C96_BASE
                        Define this to hold the physical address
                        of the LAN91C96's I/O space

                        CONFIG_LAN91C96_USE_32_BIT
                        Define this to enable 32 bit addressing

                CONFIG_SMC91111
                Support for SMSC's LAN91C111 chip

                        CONFIG_SMC91111_BASE
                        Define this to hold the physical address
                        of the device (I/O space)

                        CONFIG_SMC_USE_32_BIT
                        Define this if data bus is 32 bits

                        CONFIG_SMC_USE_IOFUNCS
                        Define this to use i/o functions instead of macros
                        (some hardware wont work with macros)

                CONFIG_DRIVER_TI_EMAC
                Support for davinci emac

                        CONFIG_SYS_DAVINCI_EMAC_PHY_COUNT
                        Define this if you have more then 3 PHYs.
```

```
                CONFIG_FTGMAC100
                Support for Faraday's FTGMAC100 Gigabit SoC Ethernet

                        CONFIG_FTGMAC100_EGIGA
                        Define this to use GE link update with gigabit PHY.
                        Define this if FTGMAC100 is connected to gigabit PHY.
                        If your system has 10/100 PHY only, it might not occur
                        wrong behavior. Because PHY usually return timeout or
                        useless data when polling gigabit status and gigabit
                        control registers. This behavior won't affect the
                        correctnessof 10/100 link speed update.

                CONFIG_SMC911X
                Support for SMSC's LAN911x and LAN921x chips

                        CONFIG_SMC911X_BASE
                        Define this to hold the physical address
                        of the device (I/O space)

                        CONFIG_SMC911X_32_BIT
                        Define this if data bus is 32 bits

                        CONFIG_SMC911X_16_BIT
                        Define this if data bus is 16 bits. If your processor
                        automatically converts one 32 bit word to two 16 bit
                        words you may also try CONFIG_SMC911X_32_BIT.

                CONFIG_SH_ETHER
                Support for Renesas on-chip Ethernet controller

                        CONFIG_SH_ETHER_USE_PORT
                        Define the number of ports to be used

                        CONFIG_SH_ETHER_PHY_ADDR
                        Define the ETH PHY's address

                        CONFIG_SH_ETHER_CACHE_WRITEBACK
                        If this option is set, the driver enables cache flush.

- TPM Support:
                CONFIG_TPM
                Support TPM devices.

                CONFIG_TPM_TIS_I2C
                Support for i2c bus TPM devices. Only one device
                per system is supported at this time.

                        CONFIG_TPM_TIS_I2C_BUS_NUMBER
                        Define the the i2c bus number for the TPM device

                        CONFIG_TPM_TIS_I2C_SLAVE_ADDRESS
                        Define the TPM's address on the i2c bus

                        CONFIG_TPM_TIS_I2C_BURST_LIMITATION
                        Define the burst count bytes upper limit

                CONFIG_TPM_ATMEL_TWI
                Support for Atmel TWI TPM device. Requires I2C support.

                CONFIG_TPM_TIS_LPC
                Support for generic parallel port TPM devices. Only one device
                per system is supported at this time.

                        CONFIG_TPM_TIS_BASE_ADDRESS
                        Base address where the generic TPM device is mapped
                        to. Contemporary x86 systems usually map it at
                        0xfed40000.
```

```
                CONFIG_CMD_TPM
                Add tpm monitor functions.
                Requires CONFIG_TPM. If CONFIG_TPM_AUTH_SESSIONS is set, also
                provides monitor access to authorized functions.

                CONFIG_TPM
                Define this to enable the TPM support library which provides
                functional interfaces to some TPM commands.
                Requires support for a TPM device.

                CONFIG_TPM_AUTH_SESSIONS
                Define this to enable authorized functions in the TPM library.
                Requires CONFIG_TPM and CONFIG_SHA1.

- USB Support:
                At the moment only the UHCI host controller is
                supported (PIP405, MIP405, MPC5200); define
                CONFIG_USB_UHCI to enable it.
                define CONFIG_USB_KEYBOARD to enable the USB Keyboard
                and define CONFIG_USB_STORAGE to enable the USB
                storage devices.
                Note:
                Supported are USB Keyboards and USB Floppy drives
                (TEAC FD-05PUB).
                MPC5200 USB requires additional defines:
                        CONFIG_USB_CLOCK
                                for 528 MHz Clock: 0x0001bbbb
                        CONFIG_PSC3_USB
                                for USB on PSC3
                        CONFIG_USB_CONFIG
                                for differential drivers: 0x00001000
                                for single ended drivers: 0x00005000
                                for differential drivers on PSC3: 0x00000100
                                for single ended drivers on PSC3: 0x00004100
                        CONFIG_SYS_USB_EVENT_POLL
                                May be defined to allow interrupt polling
                                instead of using asynchronous interrupts

                CONFIG_USB_EHCI_TXFIFO_THRESH enables setting of the
                txfilltuning field in the EHCI controller on reset.

- USB Device:
                Define the below if you wish to use the USB console.
                Once firmware is rebuilt from a serial console issue the
                command "setenv stdin usbtty; setenv stdout usbtty" and
                attach your USB cable. The Unix command "dmesg" should print
                it has found a new device. The environment variable usbtty
                can be set to gserial or cdc_acm to enable your device to
                appear to a USB host as a Linux gserial device or a
                Common Device Class Abstract Control Model serial device.
                If you select usbtty = gserial you should be able to enumerate
                a Linux host by
                # modprobe usbserial vendor=0xVendorID product=0xProductID
                else if using cdc_acm, simply setting the environment
                variable usbtty to be cdc_acm should suffice. The following
                might be defined in YourBoardName.h

                        CONFIG_USB_DEVICE
                        Define this to build a UDC device

                        CONFIG_USB_TTY
                        Define this to have a tty type of device available to
                        talk to the UDC device

                        CONFIG_USBD_HS
                        Define this to enable the high speed support for usb
```

```
                        device and usbtty. If this feature is enabled, a routine
                        int is_usbd_high_speed(void)
                        also needs to be defined by the driver to dynamically poll
                        whether the enumeration has succeded at high speed or full
                        speed.

                        CONFIG_SYS_CONSOLE_IS_IN_ENV
                        Define this if you want stdin, stdout &/or stderr to
                        be set to usbtty.

                        mpc8xx:
                                CONFIG_SYS_USB_EXTC_CLK 0xBLAH
                                Derive USB clock from external clock "blah"
                                - CONFIG_SYS_USB_EXTC_CLK 0x02

                                CONFIG_SYS_USB_BRG_CLK 0xBLAH
                                Derive USB clock from brgclk
                                - CONFIG_SYS_USB_BRG_CLK 0x04

                If you have a USB-IF assigned VendorID then you may wish to
                define your own vendor specific values either in BoardName.h
                or directly in usbd_vendor_info.h. If you don't define
                CONFIG_USBD_MANUFACTURER, CONFIG_USBD_PRODUCT_NAME,
                CONFIG_USBD_VENDORID and CONFIG_USBD_PRODUCTID, then U-Boot
                should pretend to be a Linux device to it's target host.

                        CONFIG_USBD_MANUFACTURER
                        Define this string as the name of your company for
                        - CONFIG_USBD_MANUFACTURER "my company"

                        CONFIG_USBD_PRODUCT_NAME
                        Define this string as the name of your product
                        - CONFIG_USBD_PRODUCT_NAME "acme usb device"

                        CONFIG_USBD_VENDORID
                        Define this as your assigned Vendor ID from the USB
                        Implementors Forum. This *must* be a genuine Vendor ID
                        to avoid polluting the USB namespace.
                        - CONFIG_USBD_VENDORID 0xFFFF

                        CONFIG_USBD_PRODUCTID
                        Define this as the unique Product ID
                        for your device
                        - CONFIG_USBD_PRODUCTID 0xFFFF

- ULPI Layer Support:
                The ULPI (UTMI Low Pin (count) Interface) PHYs are supported via
                the generic ULPI layer. The generic layer accesses the ULPI PHY
                via the platform viewport, so you need both the genric layer and
                the viewport enabled. Currently only Chipidea/ARC based
                viewport is supported.
                To enable the ULPI layer support, define CONFIG_USB_ULPI and
                CONFIG_USB_ULPI_VIEWPORT in your board configuration file.
                If your ULPI phy needs a different reference clock than the
                standard 24 MHz then you have to define CONFIG_ULPI_REF_CLK to
                the appropriate value in Hz.

- MMC Support:
                The MMC controller on the Intel PXA is supported. To
                enable this define CONFIG_MMC. The MMC can be
                accessed from the boot prompt by mapping the device
                to physical memory similar to flash. Command line is
                enabled with CONFIG_CMD_MMC. The MMC driver also works with
                the FAT fs. This is enabled with CONFIG_CMD_FAT.

                CONFIG_SH_MMCIF
                Support for Renesas on-chip MMCIF controller
```

```
                        CONFIG_SH_MMCIF_ADDR
                        Define the base address of MMCIF registers

                        CONFIG_SH_MMCIF_CLK
                        Define the clock frequency for MMCIF

                CONFIG_GENERIC_MMC
                Enable the generic MMC driver

                CONFIG_SUPPORT_EMMC_BOOT
                Enable some additional features of the eMMC boot partitions.

                CONFIG_SUPPORT_EMMC_RPMB
                Enable the commands for reading, writing and programming the
                key for the Replay Protection Memory Block partition in eMMC.

- USB Device Firmware Update (DFU) class support:
                CONFIG_DFU_FUNCTION
                This enables the USB portion of the DFU USB class

                CONFIG_CMD_DFU
                This enables the command "dfu" which is used to have
                U-Boot create a DFU class device via USB.  This command
                requires that the "dfu_alt_info" environment variable be
                set and define the alt settings to expose to the host.

                CONFIG_DFU_MMC
                This enables support for exposing (e)MMC devices via DFU.

                CONFIG_DFU_NAND
                This enables support for exposing NAND devices via DFU.

                CONFIG_DFU_RAM
                This enables support for exposing RAM via DFU.
                Note: DFU spec refer to non-volatile memory usage, but
                allow usages beyond the scope of spec - here RAM usage,
                one that would help mostly the developer.

                CONFIG_SYS_DFU_DATA_BUF_SIZE
                Dfu transfer uses a buffer before writing data to the
                raw storage device. Make the size (in bytes) of this buffer
                configurable. The size of this buffer is also configurable
                through the "dfu_bufsiz" environment variable.

                CONFIG_SYS_DFU_MAX_FILE_SIZE
                When updating files rather than the raw storage device,
                we use a static buffer to copy the file into and then write
                the buffer once we've been given the whole file.  Define
                this to the maximum filesize (in bytes) for the buffer.
                Default is 4 MiB if undefined.

                DFU_DEFAULT_POLL_TIMEOUT
                Poll timeout [ms], is the timeout a device can send to the
                host. The host must wait for this timeout before sending
                a subsequent DFU_GET_STATUS request to the device.

                DFU_MANIFEST_POLL_TIMEOUT
                Poll timeout [ms], which the device sends to the host when
                entering dfuMANIFEST state. Host waits this timeout, before
                sending again an USB request to the device.

- USB Device Android Fastboot support:
                CONFIG_CMD_FASTBOOT
                This enables the command "fastboot" which enables the Android
                fastboot mode for the platform's USB device. Fastboot is a USB
                protocol for downloading images, flashing and device control
```

```
                used on Android devices.
                See doc/README.android-fastboot for more information.

                CONFIG_ANDROID_BOOT_IMAGE
                This enables support for booting images which use the Android
                image format header.

                CONFIG_USB_FASTBOOT_BUF_ADDR
                The fastboot protocol requires a large memory buffer for
                downloads. Define this to the starting RAM address to use for
                downloaded images.

                CONFIG_USB_FASTBOOT_BUF_SIZE
                The fastboot protocol requires a large memory buffer for
                downloads. This buffer should be as large as possible for a
                platform. Define this to the size available RAM for fastboot.

- Journaling Flash filesystem support:
                CONFIG_JFFS2_NAND, CONFIG_JFFS2_NAND_OFF, CONFIG_JFFS2_NAND_SIZE,
                CONFIG_JFFS2_NAND_DEV
                Define these for a default partition on a NAND device

                CONFIG_SYS_JFFS2_FIRST_SECTOR,
                CONFIG_SYS_JFFS2_FIRST_BANK, CONFIG_SYS_JFFS2_NUM_BANKS
                Define these for a default partition on a NOR device

                CONFIG_SYS_JFFS_CUSTOM_PART
                Define this to create an own partition. You have to provide a
                function struct part_info* jffs2_part_info(int part_num)

                If you define only one JFFS2 partition you may also want to
                #define CONFIG_SYS_JFFS_SINGLE_PART     1
                to disable the command chpart. This is the default when you
                have not defined a custom partition

- FAT(File Allocation Table) filesystem write function support:
                CONFIG_FAT_WRITE

                Define this to enable support for saving memory data as a
                file in FAT formatted partition.

                This will also enable the command "fatwrite" enabling the
                user to write files to FAT.

CBFS (Coreboot Filesystem) support
                CONFIG_CMD_CBFS

                Define this to enable support for reading from a Coreboot
                filesystem. Available commands are cbfsinit, cbfsinfo, cbfsls
                and cbfsload.

- FAT(File Allocation Table) filesystem cluster size:
                CONFIG_FS_FAT_MAX_CLUSTSIZE

                Define the max cluster size for fat operations else
                a default value of 65536 will be defined.

- Keyboard Support:
                CONFIG_ISA_KEYBOARD

                Define this to enable standard (PC-Style) keyboard
                support

                CONFIG_I8042_KBD
                Standard PC keyboard driver with US (is default) and
                GERMAN key layout (switch via environment 'keymap=de') support.
                Export function i8042_kbd_init, i8042_tstc and i8042_getc
```

```
                for cfb_console. Supports cursor blinking.

                CONFIG_CROS_EC_KEYB
                Enables a Chrome OS keyboard using the CROS_EC interface.
                This uses CROS_EC to communicate with a second microcontroller
                which provides key scans on request.

- Video support:
                CONFIG_VIDEO

                Define this to enable video support (for output to
                video).

                CONFIG_VIDEO_CT69000

                Enable Chips & Technologies 69000 Video chip

                CONFIG_VIDEO_SMI_LYNXEM
                Enable Silicon Motion SMI 712/710/810 Video chip. The
                video output is selected via environment 'videoout'
                (1 = LCD and 2 = CRT). If videoout is undefined, CRT is
                assumed.

                For the CT69000 and SMI_LYNXEM drivers, videomode is
                selected via environment 'videomode'. Two different ways
                are possible:
                - "videomode=num"   'num' is a standard LiLo mode numbers.
                Following standard modes are supported  (* is default):

                      Colors    640x480 800x600 1024x768 1152x864 1280x1024
                -------------+---------------------------------------------
                      8 bits |  0x301*  0x303    0x305    0x161     0x307
                     15 bits |  0x310   0x313    0x316    0x162     0x319
                     16 bits |  0x311   0x314    0x317    0x163     0x31A
                     24 bits |  0x312   0x315    0x318      ?       0x31B
                -------------+---------------------------------------------
                (i.e. setenv videomode 317; saveenv; reset;)

                - "videomode=bootargs" all the video parameters are parsed
                from the bootargs. (See drivers/video/videomodes.c)


                CONFIG_VIDEO_SED13806
                Enable Epson SED13806 driver. This driver supports 8bpp
                and 16bpp modes defined by CONFIG_VIDEO_SED13806_8BPP
                or CONFIG_VIDEO_SED13806_16BPP

                CONFIG_FSL_DIU_FB
                Enable the Freescale DIU video driver.  Reference boards for
                SOCs that have a DIU should define this macro to enable DIU
                support, and should also define these other macros:

                        CONFIG_SYS_DIU_ADDR
                        CONFIG_VIDEO
                        CONFIG_CMD_BMP
                        CONFIG_CFB_CONSOLE
                        CONFIG_VIDEO_SW_CURSOR
                        CONFIG_VGA_AS_SINGLE_DEVICE
                        CONFIG_VIDEO_LOGO
                        CONFIG_VIDEO_BMP_LOGO

                The DIU driver will look for the 'video-mode' environment
                variable, and if defined, enable the DIU as a console during
                boot.  See the documentation file README.video for a
                description of this variable.

                CONFIG_VIDEO_VGA
```

```
                Enable the VGA video / BIOS for x86. The alternative if you
                are using coreboot is to use the coreboot frame buffer
                driver.


- Keyboard Support:
                CONFIG_KEYBOARD

                Define this to enable a custom keyboard support.
                This simply calls drv_keyboard_init() which must be
                defined in your board-specific files.
                The only board using this so far is RBC823.

- LCD Support:  CONFIG_LCD

                Define this to enable LCD support (for output to LCD
                display); also select one of the supported displays
                by defining one of these:

                CONFIG_ATMEL_LCD:

                        HITACHI TX09D70VM1CCA, 3.5", 240x320.

                CONFIG_NEC_NL6448AC33:

                        NEC NL6448AC33-18. Active, color, single scan.

                CONFIG_NEC_NL6448BC20

                        NEC NL6448BC20-08. 6.5", 640x480.
                        Active, color, single scan.

                CONFIG_NEC_NL6448BC33_54

                        NEC NL6448BC33-54. 10.4", 640x480.
                        Active, color, single scan.

                CONFIG_SHARP_16x9

                        Sharp 320x240. Active, color, single scan.
                        It isn't 16x9, and I am not sure what it is.

                CONFIG_SHARP_LQ64D341

                        Sharp LQ64D341 display, 640x480.
                        Active, color, single scan.

                CONFIG_HLD1045

                        HLD1045 display, 640x480.
                        Active, color, single scan.

                CONFIG_OPTREX_BW

                        Optrex   CBL50840-2 NF-FW 99 22 M5
                        or
                        Hitachi  LMG6912RPFC-00T
                        or
                        Hitachi  SP14Q002

                        320x240. Black & white.

                Normally display is black on white background; define
                CONFIG_SYS_WHITE_ON_BLACK to get it inverted.

                CONFIG_LCD_ALIGNMENT
```

```
                Normally the LCD is page-aligned (tyically 4KB). If this is
                defined then the LCD will be aligned to this value instead.
                For ARM it is sometimes useful to use MMU_SECTION_SIZE
                here, since it is cheaper to change data cache settings on
                a per-section basis.

                CONFIG_CONSOLE_SCROLL_LINES

                When the console need to be scrolled, this is the number of
                lines to scroll by. It defaults to 1. Increasing this makes
                the console jump but can help speed up operation when scrolling
                is slow.

                CONFIG_LCD_BMP_RLE8

                Support drawing of RLE8-compressed bitmaps on the LCD.

                CONFIG_I2C_EDID

                Enables an 'i2c edid' command which can read EDID
                information over I2C from an attached LCD display.

- Splash Screen Support: CONFIG_SPLASH_SCREEN

                If this option is set, the environment is checked for
                a variable "splashimage". If found, the usual display
                of logo, copyright and system information on the LCD
                is suppressed and the BMP image at the address
                specified in "splashimage" is loaded instead. The
                console is redirected to the "nulldev", too. This
                allows for a "silent" boot where a splash screen is
                loaded very quickly after power-on.

                CONFIG_SPLASHIMAGE_GUARD

                If this option is set, then U-Boot will prevent the environment
                variable "splashimage" from being set to a problematic address
                (see README.displaying-bmps).
                This option is useful for targets where, due to alignment
                restrictions, an improperly aligned BMP image will cause a data
                abort. If you think you will not have problems with unaligned
                accesses (for example because your toolchain prevents them)
                there is no need to set this option.

                CONFIG_SPLASH_SCREEN_ALIGN

                If this option is set the splash image can be freely positioned
                on the screen. Environment variable "splashpos" specifies the
                position as "x,y". If a positive number is given it is used as
                number of pixel from left/top. If a negative number is given it
                is used as number of pixel from right/bottom. You can also
                specify 'm' for centering the image.

                Example:
                setenv splashpos m,m
                        => image at center of screen

                setenv splashpos 30,20
                        => image at x = 30 and y = 20

                setenv splashpos -10,m
                        => vertically centered image
                           at x = dspWidth - bmpWidth - 9

- Gzip compressed BMP image support: CONFIG_VIDEO_BMP_GZIP
```

```
                If this option is set, additionally to standard BMP
                images, gzipped BMP images can be displayed via the
                splashscreen support or the bmp command.

- Run length encoded BMP image (RLE8) support: CONFIG_VIDEO_BMP_RLE8

                If this option is set, 8-bit RLE compressed BMP images
                can be displayed via the splashscreen support or the
                bmp command.

- Do compresssing for memory range:
                CONFIG_CMD_ZIP

                If this option is set, it would use zlib deflate method
                to compress the specified memory at its best effort.

- Compression support:
                CONFIG_GZIP

                Enabled by default to support gzip compressed images.

                CONFIG_BZIP2

                If this option is set, support for bzip2 compressed
                images is included. If not, only uncompressed and gzip
                compressed images are supported.

                NOTE: the bzip2 algorithm requires a lot of RAM, so
                the malloc area (as defined by CONFIG_SYS_MALLOC_LEN) should
                be at least 4MB.

                CONFIG_LZMA

                If this option is set, support for lzma compressed
                images is included.

                Note: The LZMA algorithm adds between 2 and 4KB of code and it
                requires an amount of dynamic memory that is given by the
                formula:

                        (1846 + 768 << (lc + lp)) * sizeof(uint16)

                Where lc and lp stand for, respectively, Literal context bits
                and Literal pos bits.

                This value is upper-bounded by 14MB in the worst case. Anyway,
                for a ~4MB large kernel image, we have lc=3 and lp=0 for a
                total amount of (1846 + 768 << (3 + 0)) * 2 = ~41KB... that is
                a very small buffer.

                Use the lzmainfo tool to determinate the lc and lp values and
                then calculate the amount of needed dynamic memory (ensuring
                the appropriate CONFIG_SYS_MALLOC_LEN value).

                CONFIG_LZO

                If this option is set, support for LZO compressed images
                is included.

- MII/PHY support:
                CONFIG_PHY_ADDR

                The address of PHY on MII bus.

                CONFIG_PHY_CLOCK_FREQ (ppc4xx)

                The clock frequency of the MII bus
```

```
                CONFIG_PHY_GIGE

                If this option is set, support for speed/duplex
                detection of gigabit PHY is included.

                CONFIG_PHY_RESET_DELAY

                Some PHY like Intel LXT971A need extra delay after
                reset before any MII register access is possible.
                For such PHY, set this option to the usec delay
                required. (minimum 300usec for LXT971A)

                CONFIG_PHY_CMD_DELAY (ppc4xx)

                Some PHY like Intel LXT971A need extra delay after
                command issued before MII status register can be read

- Ethernet address:
                CONFIG_ETHADDR
                CONFIG_ETH1ADDR
                CONFIG_ETH2ADDR
                CONFIG_ETH3ADDR
                CONFIG_ETH4ADDR
                CONFIG_ETH5ADDR

                Define a default value for Ethernet address to use
                for the respective Ethernet interface, in case this
                is not determined automatically.

- IP address:
                CONFIG_IPADDR

                Define a default value for the IP address to use for
                the default Ethernet interface, in case this is not
                determined through e.g. bootp.
                (Environment variable "ipaddr")

- Server IP address:
                CONFIG_SERVERIP

                Defines a default value for the IP address of a TFTP
                server to contact when using the "tftboot" command.
                (Environment variable "serverip")

                CONFIG_KEEP_SERVERADDR

                Keeps the server's MAC address, in the env 'serveraddr'
                for passing to bootargs (like Linux's netconsole option)

- Gateway IP address:
                CONFIG_GATEWAYIP

                Defines a default value for the IP address of the
                default router where packets to other networks are
                sent to.
                (Environment variable "gatewayip")

- Subnet mask:
                CONFIG_NETMASK

                Defines a default value for the subnet mask (or
                routing prefix) which is used to determine if an IP
                address belongs to the local subnet or needs to be
                forwarded through a router.
                (Environment variable "netmask")
```

```
- Multicast TFTP Mode:
                CONFIG_MCAST_TFTP

                Defines whether you want to support multicast TFTP as per
                rfc-2090; for example to work with atftp.  Lets lots of targets
                tftp down the same boot image concurrently.  Note: the Ethernet
                driver in use must provide a function: mcast() to join/leave a
                multicast group.

- BOOTP Recovery Mode:
                CONFIG_BOOTP_RANDOM_DELAY

                If you have many targets in a network that try to
                boot using BOOTP, you may want to avoid that all
                systems send out BOOTP requests at precisely the same
                moment (which would happen for instance at recovery
                from a power failure, when all systems will try to
                boot, thus flooding the BOOTP server. Defining
                CONFIG_BOOTP_RANDOM_DELAY causes a random delay to be
                inserted before sending out BOOTP requests. The
                following delays are inserted then:

                1st BOOTP request:      delay 0 ... 1 sec
                2nd BOOTP request:      delay 0 ... 2 sec
                3rd BOOTP request:      delay 0 ... 4 sec
                4th and following
                BOOTP requests:         delay 0 ... 8 sec

- DHCP Advanced Options:
                You can fine tune the DHCP functionality by defining
                CONFIG_BOOTP_* symbols:

                CONFIG_BOOTP_SUBNETMASK
                CONFIG_BOOTP_GATEWAY
                CONFIG_BOOTP_HOSTNAME
                CONFIG_BOOTP_NISDOMAIN
                CONFIG_BOOTP_BOOTPATH
                CONFIG_BOOTP_BOOTFILESIZE
                CONFIG_BOOTP_DNS
                CONFIG_BOOTP_DNS2
                CONFIG_BOOTP_SEND_HOSTNAME
                CONFIG_BOOTP_NTPSERVER
                CONFIG_BOOTP_TIMEOFFSET
                CONFIG_BOOTP_VENDOREX
                CONFIG_BOOTP_MAY_FAIL

                CONFIG_BOOTP_SERVERIP - TFTP server will be the serverip
                environment variable, not the BOOTP server.

                CONFIG_BOOTP_MAY_FAIL - If the DHCP server is not found
                after the configured retry count, the call will fail
                instead of starting over.  This can be used to fail over
                to Link-local IP address configuration if the DHCP server
                is not available.

                CONFIG_BOOTP_DNS2 - If a DHCP client requests the DNS
                serverip from a DHCP server, it is possible that more
                than one DNS serverip is offered to the client.
                If CONFIG_BOOTP_DNS2 is enabled, the secondary DNS
                serverip will be stored in the additional environment
                variable "dnsip2". The first DNS serverip is always
                stored in the variable "dnsip", when CONFIG_BOOTP_DNS
                is defined.

                CONFIG_BOOTP_SEND_HOSTNAME - Some DHCP servers are capable
                to do a dynamic update of a DNS server. To do this, they
                need the hostname of the DHCP requester.
```

```
                If CONFIG_BOOTP_SEND_HOSTNAME is defined, the content
                of the "hostname" environment variable is passed as
                option 12 to the DHCP server.

                CONFIG_BOOTP_DHCP_REQUEST_DELAY

                A 32bit value in microseconds for a delay between
                receiving a "DHCP Offer" and sending the "DHCP Request".
                This fixes a problem with certain DHCP servers that don't
                respond 100% of the time to a "DHCP request". E.g. On an
                AT91RM9200 processor running at 180MHz, this delay needed
                to be *at least* 15,000 usec before a Windows Server 2003
                DHCP server would reply 100% of the time. I recommend at
                least 50,000 usec to be safe. The alternative is to hope
                that one of the retries will be successful but note that
                the DHCP timeout and retry process takes a longer than
                this delay.

- Link-local IP address negotiation:
                Negotiate with other link-local clients on the local network
                for an address that doesn't require explicit configuration.
                This is especially useful if a DHCP server cannot be guaranteed
                to exist in all environments that the device must operate.

                See doc/README.link-local for more information.

- CDP Options:
                CONFIG_CDP_DEVICE_ID

                The device id used in CDP trigger frames.

                CONFIG_CDP_DEVICE_ID_PREFIX

                A two character string which is prefixed to the MAC address
                of the device.

                CONFIG_CDP_PORT_ID

                A printf format string which contains the ascii name of
                the port. Normally is set to "eth%d" which sets
                eth0 for the first Ethernet, eth1 for the second etc.

                CONFIG_CDP_CAPABILITIES

                A 32bit integer which indicates the device capabilities;
                0x00000010 for a normal host which does not forwards.

                CONFIG_CDP_VERSION

                An ascii string containing the version of the software.

                CONFIG_CDP_PLATFORM

                An ascii string containing the name of the platform.

                CONFIG_CDP_TRIGGER

                A 32bit integer sent on the trigger.

                CONFIG_CDP_POWER_CONSUMPTION

                A 16bit integer containing the power consumption of the
                device in .1 of milliwatts.

                CONFIG_CDP_APPLIANCE_VLAN_TYPE

                A byte containing the id of the VLAN.
```

```
- Status LED:   CONFIG_STATUS_LED

                Several configurations allow to display the current
                status using a LED. For instance, the LED will blink
                fast while running U-Boot code, stop blinking as
                soon as a reply to a BOOTP request was received, and
                start blinking slow once the Linux kernel is running
                (supported by a status LED driver in the Linux
                kernel). Defining CONFIG_STATUS_LED enables this
                feature in U-Boot.

                Additional options:

                CONFIG_GPIO_LED
                The status LED can be connected to a GPIO pin.
                In such cases, the gpio_led driver can be used as a
                status LED backend implementation. Define CONFIG_GPIO_LED
                to include the gpio_led driver in the U-Boot binary.

                CONFIG_GPIO_LED_INVERTED_TABLE
                Some GPIO connected LEDs may have inverted polarity in which
                case the GPIO high value corresponds to LED off state and
                GPIO low value corresponds to LED on state.
                In such cases CONFIG_GPIO_LED_INVERTED_TABLE may be defined
                with a list of GPIO LEDs that have inverted polarity.

- CAN Support:  CONFIG_CAN_DRIVER

                Defining CONFIG_CAN_DRIVER enables CAN driver support
                on those systems that support this (optional)
                feature, like the TQM8xxL modules.

- I2C Support:  CONFIG_SYS_I2C

                This enable the NEW i2c subsystem, and will allow you to use
                i2c commands at the u-boot command line (as long as you set
                CONFIG_CMD_I2C in CONFIG_COMMANDS) and communicate with i2c
                based realtime clock chips or other i2c devices. See
                common/cmd_i2c.c for a description of the command line
                interface.

                ported i2c driver to the new framework:
                - drivers/i2c/soft_i2c.c:
                  - activate first bus with CONFIG_SYS_I2C_SOFT define
                    CONFIG_SYS_I2C_SOFT_SPEED and CONFIG_SYS_I2C_SOFT_SLAVE
                    for defining speed and slave address
                  - activate second bus with I2C_SOFT_DECLARATIONS2 define
                    CONFIG_SYS_I2C_SOFT_SPEED_2 and CONFIG_SYS_I2C_SOFT_SLAVE_2
                    for defining speed and slave address
                  - activate third bus with I2C_SOFT_DECLARATIONS3 define
                    CONFIG_SYS_I2C_SOFT_SPEED_3 and CONFIG_SYS_I2C_SOFT_SLAVE_3
                    for defining speed and slave address
                  - activate fourth bus with I2C_SOFT_DECLARATIONS4 define
                    CONFIG_SYS_I2C_SOFT_SPEED_4 and CONFIG_SYS_I2C_SOFT_SLAVE_4
                    for defining speed and slave address

                - drivers/i2c/fsl_i2c.c:
                  - activate i2c driver with CONFIG_SYS_I2C_FSL
                    define CONFIG_SYS_FSL_I2C_OFFSET for setting the register
                    offset CONFIG_SYS_FSL_I2C_SPEED for the i2c speed and
                    CONFIG_SYS_FSL_I2C_SLAVE for the slave addr of the first
                    bus.
                  - If your board supports a second fsl i2c bus, define
                    CONFIG_SYS_FSL_I2C2_OFFSET for the register offset
                    CONFIG_SYS_FSL_I2C2_SPEED for the speed and
                    CONFIG_SYS_FSL_I2C2_SLAVE for the slave address of the
```

```
    second bus.

- drivers/i2c/tegra_i2c.c:
  - activate this driver with CONFIG_SYS_I2C_TEGRA
  - This driver adds 4 i2c buses with a fix speed from
    100000 and the slave addr 0!

- drivers/i2c/ppc4xx_i2c.c
  - activate this driver with CONFIG_SYS_I2C_PPC4XX
  - CONFIG_SYS_I2C_PPC4XX_CH0 activate hardware channel 0
  - CONFIG_SYS_I2C_PPC4XX_CH1 activate hardware channel 1

- drivers/i2c/i2c_mxc.c
  - activate this driver with CONFIG_SYS_I2C_MXC
  - define speed for bus 1 with CONFIG_SYS_MXC_I2C1_SPEED
  - define slave for bus 1 with CONFIG_SYS_MXC_I2C1_SLAVE
  - define speed for bus 2 with CONFIG_SYS_MXC_I2C2_SPEED
  - define slave for bus 2 with CONFIG_SYS_MXC_I2C2_SLAVE
  - define speed for bus 3 with CONFIG_SYS_MXC_I2C3_SPEED
  - define slave for bus 3 with CONFIG_SYS_MXC_I2C3_SLAVE
If thoses defines are not set, default value is 100000
for speed, and 0 for slave.

- drivers/i2c/rcar_i2c.c:
  - activate this driver with CONFIG_SYS_I2C_RCAR
  - This driver adds 4 i2c buses

  - CONFIG_SYS_RCAR_I2C0_BASE for setting the register channel 0
  - CONFIG_SYS_RCAR_I2C0_SPEED for for the speed channel 0
  - CONFIG_SYS_RCAR_I2C1_BASE for setting the register channel 1
  - CONFIG_SYS_RCAR_I2C1_SPEED for for the speed channel 1
  - CONFIG_SYS_RCAR_I2C2_BASE for setting the register channel 2
  - CONFIG_SYS_RCAR_I2C2_SPEED for for the speed channel 2
  - CONFIG_SYS_RCAR_I2C3_BASE for setting the register channel 3
  - CONFIG_SYS_RCAR_I2C3_SPEED for for the speed channel 3
  - CONFIF_SYS_RCAR_I2C_NUM_CONTROLLERS for number of i2c buses

- drivers/i2c/sh_i2c.c:
  - activate this driver with CONFIG_SYS_I2C_SH
  - This driver adds from 2 to 5 i2c buses

  - CONFIG_SYS_I2C_SH_BASE0 for setting the register channel 0
  - CONFIG_SYS_I2C_SH_SPEED0 for for the speed channel 0
  - CONFIG_SYS_I2C_SH_BASE1 for setting the register channel 1
  - CONFIG_SYS_I2C_SH_SPEED1 for for the speed channel 1
  - CONFIG_SYS_I2C_SH_BASE2 for setting the register channel 2
  - CONFIG_SYS_I2C_SH_SPEED2 for for the speed channel 2
  - CONFIG_SYS_I2C_SH_BASE3 for setting the register channel 3
  - CONFIG_SYS_I2C_SH_SPEED3 for for the speed channel 3
  - CONFIG_SYS_I2C_SH_BASE4 for setting the register channel 4
  - CONFIG_SYS_I2C_SH_SPEED4 for for the speed channel 4
  - CONFIG_SYS_I2C_SH_BASE5 for setting the register channel 5
  - CONFIG_SYS_I2C_SH_SPEED5 for for the speed channel 5
  - CONFIF_SYS_I2C_SH_NUM_CONTROLLERS for nummber of i2c buses

- drivers/i2c/omap24xx_i2c.c
  - activate this driver with CONFIG_SYS_I2C_OMAP24XX
  - CONFIG_SYS_OMAP24_I2C_SPEED speed channel 0
  - CONFIG_SYS_OMAP24_I2C_SLAVE slave addr channel 0
  - CONFIG_SYS_OMAP24_I2C_SPEED1 speed channel 1
  - CONFIG_SYS_OMAP24_I2C_SLAVE1 slave addr channel 1
  - CONFIG_SYS_OMAP24_I2C_SPEED2 speed channel 2
  - CONFIG_SYS_OMAP24_I2C_SLAVE2 slave addr channel 2
  - CONFIG_SYS_OMAP24_I2C_SPEED3 speed channel 3
  - CONFIG_SYS_OMAP24_I2C_SLAVE3 slave addr channel 3
  - CONFIG_SYS_OMAP24_I2C_SPEED4 speed channel 4
  - CONFIG_SYS_OMAP24_I2C_SLAVE4 slave addr channel 4
```

```
- drivers/i2c/zynq_i2c.c
  - activate this driver with CONFIG_SYS_I2C_ZYNQ
  - set CONFIG_SYS_I2C_ZYNQ_SPEED for speed setting
  - set CONFIG_SYS_I2C_ZYNQ_SLAVE for slave addr

- drivers/i2c/s3c24x0_i2c.c:
  - activate this driver with CONFIG_SYS_I2C_S3C24X0
  - This driver adds i2c buses (11 for Exynos5250, Exynos5420
    9 i2c buses for Exynos4 and 1 for S3C24X0 SoCs from Samsung)
    with a fix speed from 100000 and the slave addr 0!

- drivers/i2c/ihs_i2c.c
  - activate this driver with CONFIG_SYS_I2C_IHS
  - CONFIG_SYS_I2C_IHS_CH0 activate hardware channel 0
  - CONFIG_SYS_I2C_IHS_SPEED_0 speed channel 0
  - CONFIG_SYS_I2C_IHS_SLAVE_0 slave addr channel 0
  - CONFIG_SYS_I2C_IHS_CH1 activate hardware channel 1
  - CONFIG_SYS_I2C_IHS_SPEED_1 speed channel 1
  - CONFIG_SYS_I2C_IHS_SLAVE_1 slave addr channel 1
  - CONFIG_SYS_I2C_IHS_CH2 activate hardware channel 2
  - CONFIG_SYS_I2C_IHS_SPEED_2 speed channel 2
  - CONFIG_SYS_I2C_IHS_SLAVE_2 slave addr channel 2
  - CONFIG_SYS_I2C_IHS_CH3 activate hardware channel 3
  - CONFIG_SYS_I2C_IHS_SPEED_3 speed channel 3
  - CONFIG_SYS_I2C_IHS_SLAVE_3 slave addr channel 3

additional defines:

CONFIG_SYS_NUM_I2C_BUSES
Hold the number of i2c busses you want to use. If you
don't use/have i2c muxes on your i2c bus, this
is equal to CONFIG_SYS_NUM_I2C_ADAPTERS, and you can
omit this define.

CONFIG_SYS_I2C_DIRECT_BUS
define this, if you don't use i2c muxes on your hardware.
if CONFIG_SYS_I2C_MAX_HOPS is not defined or == 0 you can
omit this define.

CONFIG_SYS_I2C_MAX_HOPS
define how many muxes are maximal consecutively connected
on one i2c bus. If you not use i2c muxes, omit this
define.

CONFIG_SYS_I2C_BUSES
hold a list of busses you want to use, only used if
CONFIG_SYS_I2C_DIRECT_BUS is not defined, for example
a board with CONFIG_SYS_I2C_MAX_HOPS = 1 and
CONFIG_SYS_NUM_I2C_BUSES = 9:

 CONFIG_SYS_I2C_BUSES   {{0, {I2C_NULL_HOP}}, \
                        {0, {{I2C_MUX_PCA9547, 0x70, 1}}}, \
                        {0, {{I2C_MUX_PCA9547, 0x70, 2}}}, \
                        {0, {{I2C_MUX_PCA9547, 0x70, 3}}}, \
                        {0, {{I2C_MUX_PCA9547, 0x70, 4}}}, \
                        {0, {{I2C_MUX_PCA9547, 0x70, 5}}}, \
                        {1, {I2C_NULL_HOP}}, \
                        {1, {{I2C_MUX_PCA9544, 0x72, 1}}}, \
                        {1, {{I2C_MUX_PCA9544, 0x72, 2}}}, \
                        }

which defines
        bus 0 on adapter 0 without a mux
        bus 1 on adapter 0 with a PCA9547 on address 0x70 port 1
        bus 2 on adapter 0 with a PCA9547 on address 0x70 port 2
        bus 3 on adapter 0 with a PCA9547 on address 0x70 port 3
```

```
                        bus 4 on adapter 0 with a PCA9547 on address 0x70 port 4
                        bus 5 on adapter 0 with a PCA9547 on address 0x70 port 5
                        bus 6 on adapter 1 without a mux
                        bus 7 on adapter 1 with a PCA9544 on address 0x72 port 1
                        bus 8 on adapter 1 with a PCA9544 on address 0x72 port 2

                If you do not have i2c muxes on your board, omit this define.

- Legacy I2C Support:   CONFIG_HARD_I2C

                NOTE: It is intended to move drivers to CONFIG_SYS_I2C which
                provides the following compelling advantages:

                - more than one i2c adapter is usable
                - approved multibus support
                - better i2c mux support

                ** Please consider updating your I2C driver now. **

                These enable legacy I2C serial bus commands. Defining
                CONFIG_HARD_I2C will include the appropriate I2C driver
                for the selected CPU.

                This will allow you to use i2c commands at the u-boot
                command line (as long as you set CONFIG_CMD_I2C in
                CONFIG_COMMANDS) and communicate with i2c based realtime
                clock chips. See common/cmd_i2c.c for a description of the
                command line interface.

                CONFIG_HARD_I2C selects a hardware I2C controller.

                There are several other quantities that must also be
                defined when you define CONFIG_HARD_I2C.

                In both cases you will need to define CONFIG_SYS_I2C_SPEED
                to be the frequency (in Hz) at which you wish your i2c bus
                to run and CONFIG_SYS_I2C_SLAVE to be the address of this node (ie
                the CPU's i2c node address).

                Now, the u-boot i2c code for the mpc8xx
                (arch/powerpc/cpu/mpc8xx/i2c.c) sets the CPU up as a master node
                and so its address should therefore be cleared to 0 (See,
                eg, MPC823e User's Manual p.16-473). So, set
                CONFIG_SYS_I2C_SLAVE to 0.

                CONFIG_SYS_I2C_INIT_MPC5XXX

                When a board is reset during an i2c bus transfer
                chips might think that the current transfer is still
                in progress.  Reset the slave devices by sending start
                commands until the slave device responds.

                That's all that's required for CONFIG_HARD_I2C.

                If you use the software i2c interface (CONFIG_SYS_I2C_SOFT)
                then the following macros need to be defined (examples are
                from include/configs/lwmon.h):

                I2C_INIT

                (Optional). Any commands necessary to enable the I2C
                controller or configure ports.

                eg: #define I2C_INIT (immr->im_cpm.cp_pbdir |=  PB_SCL)

                I2C_PORT
```

```
(Only for MPC8260 CPU). The I/O port to use (the code
assumes both bits are on the same port). Valid values
are 0..3 for ports A..D.

I2C_ACTIVE

The code necessary to make the I2C data line active
(driven).  If the data line is open collector, this
define can be null.

eg: #define I2C_ACTIVE (immr->im_cpm.cp_pbdir |=  PB_SDA)

I2C_TRISTATE

The code necessary to make the I2C data line tri-stated
(inactive).  If the data line is open collector, this
define can be null.

eg: #define I2C_TRISTATE (immr->im_cpm.cp_pbdir &= ~PB_SDA)

I2C_READ

Code that returns true if the I2C data line is high,
false if it is low.

eg: #define I2C_READ ((immr->im_cpm.cp_pbdat & PB_SDA) != 0)

I2C_SDA(bit)

If <bit> is true, sets the I2C data line high. If it
is false, it clears it (low).

eg: #define I2C_SDA(bit) \
        if(bit) immr->im_cpm.cp_pbdat |=  PB_SDA; \
        else    immr->im_cpm.cp_pbdat &= ~PB_SDA

I2C_SCL(bit)

If <bit> is true, sets the I2C clock line high. If it
is false, it clears it (low).

eg: #define I2C_SCL(bit) \
        if(bit) immr->im_cpm.cp_pbdat |=  PB_SCL; \
        else    immr->im_cpm.cp_pbdat &= ~PB_SCL

I2C_DELAY

This delay is invoked four times per clock cycle so this
controls the rate of data transfer.  The data rate thus
is 1 / (I2C_DELAY * 4). Often defined to be something
like:

#define I2C_DELAY  udelay(2)

CONFIG_SOFT_I2C_GPIO_SCL / CONFIG_SOFT_I2C_GPIO_SDA

If your arch supports the generic GPIO framework (asm/gpio.h),
then you may alternatively define the two GPIOs that are to be
used as SCL / SDA.  Any of the previous I2C_xxx macros will
have GPIO-based defaults assigned to them as appropriate.

You should define these to the GPIO value as given directly to
the generic GPIO functions.

CONFIG_SYS_I2C_INIT_BOARD

When a board is reset during an i2c bus transfer
```

```
chips might think that the current transfer is still
in progress. On some boards it is possible to access
the i2c SCLK line directly, either by using the
processor pin as a GPIO or by having a second pin
connected to the bus. If this option is defined a
custom i2c_init_board() routine in boards/xxx/board.c
is run early in the boot sequence.

CONFIG_SYS_I2C_BOARD_LATE_INIT

An alternative to CONFIG_SYS_I2C_INIT_BOARD. If this option is
defined a custom i2c_board_late_init() routine in
boards/xxx/board.c is run AFTER the operations in i2c_init()
is completed. This callpoint can be used to unreset i2c bus
using CPU i2c controller register accesses for CPUs whose i2c
controller provide such a method. It is called at the end of
i2c_init() to allow i2c_init operations to setup the i2c bus
controller on the CPU (e.g. setting bus speed & slave address).

CONFIG_I2CFAST (PPC405GP|PPC405EP only)

This option enables configuration of bi_iic_fast[] flags
in u-boot bd_info structure based on u-boot environment
variable "i2cfast". (see also i2cfast)

CONFIG_I2C_MULTI_BUS

This option allows the use of multiple I2C buses, each of which
must have a controller.  At any point in time, only one bus is
active.  To switch to a different bus, use the 'i2c dev' command.
Note that bus numbering is zero-based.

CONFIG_SYS_I2C_NOPROBES

This option specifies a list of I2C devices that will be skipped
when the 'i2c probe' command is issued.  If CONFIG_I2C_MULTI_BUS
is set, specify a list of bus-device pairs.  Otherwise, specify
a 1D array of device addresses

e.g.
        #undef  CONFIG_I2C_MULTI_BUS
        #define CONFIG_SYS_I2C_NOPROBES {0x50,0x68}

will skip addresses 0x50 and 0x68 on a board with one I2C bus

        #define CONFIG_I2C_MULTI_BUS
        #define CONFIG_SYS_I2C_MULTI_NOPROBES   {{0,0x50},{0,0x68},{1,0x54}}

will skip addresses 0x50 and 0x68 on bus 0 and address 0x54 on bus 1

CONFIG_SYS_SPD_BUS_NUM

If defined, then this indicates the I2C bus number for DDR SPD.
If not defined, then U-Boot assumes that SPD is on I2C bus 0.

CONFIG_SYS_RTC_BUS_NUM

If defined, then this indicates the I2C bus number for the RTC.
If not defined, then U-Boot assumes that RTC is on I2C bus 0.

CONFIG_SYS_DTT_BUS_NUM

If defined, then this indicates the I2C bus number for the DTT.
If not defined, then U-Boot assumes that DTT is on I2C bus 0.

CONFIG_SYS_I2C_DTT_ADDR:
```

```
                If defined, specifies the I2C address of the DTT device.
                If not defined, then U-Boot uses predefined value for
                specified DTT device.

                CONFIG_SOFT_I2C_READ_REPEATED_START

                defining this will force the i2c_read() function in
                the soft_i2c driver to perform an I2C repeated start
                between writing the address pointer and reading the
                data.  If this define is omitted the default behaviour
                of doing a stop-start sequence will be used.  Most I2C
                devices can use either method, but some require one or
                the other.

- SPI Support:  CONFIG_SPI

                Enables SPI driver (so far only tested with
                SPI EEPROM, also an instance works with Crystal A/D and
                D/As on the SACSng board)

                CONFIG_SH_SPI

                Enables the driver for SPI controller on SuperH. Currently
                only SH7757 is supported.

                CONFIG_SPI_X

                Enables extended (16-bit) SPI EEPROM addressing.
                (symmetrical to CONFIG_I2C_X)

                CONFIG_SOFT_SPI

                Enables a software (bit-bang) SPI driver rather than
                using hardware support. This is a general purpose
                driver that only requires three general I/O port pins
                (two outputs, one input) to function. If this is
                defined, the board configuration must define several
                SPI configuration items (port pins to use, etc). For
                an example, see include/configs/sacsng.h.

                CONFIG_HARD_SPI

                Enables a hardware SPI driver for general-purpose reads
                and writes.  As with CONFIG_SOFT_SPI, the board configuration
                must define a list of chip-select function pointers.
                Currently supported on some MPC8xxx processors.  For an
                example, see include/configs/mpc8349emds.h.

                CONFIG_MXC_SPI

                Enables the driver for the SPI controllers on i.MX and MXC
                SoCs. Currently i.MX31/35/51 are supported.

- FPGA Support: CONFIG_FPGA

                Enables FPGA subsystem.

                CONFIG_FPGA_<vendor>

                Enables support for specific chip vendors.
                (ALTERA, XILINX)

                CONFIG_FPGA_<family>

                Enables support for FPGA family.
                (SPARTAN2, SPARTAN3, VIRTEX2, CYCLONE2, ACEX1K, ACEX)
```

```
                CONFIG_FPGA_COUNT

                Specify the number of FPGA devices to support.

                CONFIG_CMD_FPGA_LOADMK

                Enable support for fpga loadmk command

                CONFIG_CMD_FPGA_LOADP

                Enable support for fpga loadp command - load partial bitstream

                CONFIG_CMD_FPGA_LOADBP

                Enable support for fpga loadbp command - load partial bitstream
                (Xilinx only)

                CONFIG_SYS_FPGA_PROG_FEEDBACK

                Enable printing of hash marks during FPGA configuration.

                CONFIG_SYS_FPGA_CHECK_BUSY

                Enable checks on FPGA configuration interface busy
                status by the configuration function. This option
                will require a board or device specific function to
                be written.

                CONFIG_FPGA_DELAY

                If defined, a function that provides delays in the FPGA
                configuration driver.

                CONFIG_SYS_FPGA_CHECK_CTRLC
                Allow Control-C to interrupt FPGA configuration

                CONFIG_SYS_FPGA_CHECK_ERROR

                Check for configuration errors during FPGA bitfile
                loading. For example, abort during Virtex II
                configuration if the INIT_B line goes low (which
                indicated a CRC error).

                CONFIG_SYS_FPGA_WAIT_INIT

                Maximum time to wait for the INIT_B line to deassert
                after PROB_B has been deasserted during a Virtex II
                FPGA configuration sequence. The default time is 500
                ms.

                CONFIG_SYS_FPGA_WAIT_BUSY

                Maximum time to wait for BUSY to deassert during
                Virtex II FPGA configuration. The default is 5 ms.

                CONFIG_SYS_FPGA_WAIT_CONFIG

                Time to wait after FPGA configuration. The default is
                200 ms.

- Configuration Management:
                CONFIG_IDENT_STRING

                If defined, this string will be added to the U-Boot
                version information (U_BOOT_VERSION)

- Vendor Parameter Protection:
```

```
                U-Boot considers the values of the environment
                variables "serial#" (Board Serial Number) and
                "ethaddr" (Ethernet Address) to be parameters that
                are set once by the board vendor / manufacturer, and
                protects these variables from casual modification by
                the user. Once set, these variables are read-only,
                and write or delete attempts are rejected. You can
                change this behaviour:

                If CONFIG_ENV_OVERWRITE is #defined in your config
                file, the write protection for vendor parameters is
                completely disabled. Anybody can change or delete
                these parameters.

                Alternatively, if you #define _both_ CONFIG_ETHADDR
                _and_ CONFIG_OVERWRITE_ETHADDR_ONCE, a default
                Ethernet address is installed in the environment,
                which can be changed exactly ONCE by the user. [The
                serial# is unaffected by this, i. e. it remains
                read-only.]

                The same can be accomplished in a more flexible way
                for any variable by configuring the type of access
                to allow for those variables in the ".flags" variable
                or define CONFIG_ENV_FLAGS_LIST_STATIC.

- Protected RAM:
                CONFIG_PRAM

                Define this variable to enable the reservation of
                "protected RAM", i. e. RAM which is not overwritten
                by U-Boot. Define CONFIG_PRAM to hold the number of
                kB you want to reserve for pRAM. You can overwrite
                this default value by defining an environment
                variable "pram" to the number of kB you want to
                reserve. Note that the board info structure will
                still show the full amount of RAM. If pRAM is
                reserved, a new environment variable "mem" will
                automatically be defined to hold the amount of
                remaining RAM in a form that can be passed as boot
                argument to Linux, for instance like that:

                        setenv bootargs ... mem=\${mem}
                        saveenv

                This way you can tell Linux not to use this memory,
                either, which results in a memory region that will
                not be affected by reboots.

                *WARNING* If your board configuration uses automatic
                detection of the RAM size, you must make sure that
                this memory test is non-destructive. So far, the
                following board configurations are known to be
                "pRAM-clean":

                        IVMS8, IVML24, SPD8xx, TQM8xxL,
                        HERMES, IP860, RPXlite, LWMON,
                        FLAGADM, TQM8260

- Access to physical memory region (> 4GB)
                Some basic support is provided for operations on memory not
                normally accessible to U-Boot - e.g. some architectures
                support access to more than 4GB of memory on 32-bit
                machines using physical address extension or similar.
                Define CONFIG_PHYSMEM to access this basic support, which
                currently only supports clearing the memory.
```

```
- Error Recovery:
                CONFIG_PANIC_HANG

                Define this variable to stop the system in case of a
                fatal error, so that you have to reset it manually.
                This is probably NOT a good idea for an embedded
                system where you want the system to reboot
                automatically as fast as possible, but it may be
                useful during development since you can try to debug
                the conditions that lead to the situation.

                CONFIG_NET_RETRY_COUNT

                This variable defines the number of retries for
                network operations like ARP, RARP, TFTP, or BOOTP
                before giving up the operation. If not defined, a
                default value of 5 is used.

                CONFIG_ARP_TIMEOUT

                Timeout waiting for an ARP reply in milliseconds.

                CONFIG_NFS_TIMEOUT

                Timeout in milliseconds used in NFS protocol.
                If you encounter "ERROR: Cannot umount" in nfs command,
                try longer timeout such as
                #define CONFIG_NFS_TIMEOUT 10000UL

- Command Interpreter:
                CONFIG_AUTO_COMPLETE

                Enable auto completion of commands using TAB.

                Note that this feature has NOT been implemented yet
                for the "hush" shell.


                CONFIG_SYS_HUSH_PARSER

                Define this variable to enable the "hush" shell (from
                Busybox) as command line interpreter, thus enabling
                powerful command line syntax like
                if...then...else...fi conditionals or `&&' and '||'
                constructs ("shell scripts").

                If undefined, you get the old, much simpler behaviour
                with a somewhat smaller memory footprint.


                CONFIG_SYS_PROMPT_HUSH_PS2

                This defines the secondary prompt string, which is
                printed when the command interpreter needs more input
                to complete a command. Usually "> ".

        Note:

                In the current implementation, the local variables
                space and global environment variables space are
                separated. Local variables are those you define by
                simply typing `name=value'. To access a local
                variable later on, you have write `$name' or
                `${name}'; to execute the contents of a variable
                directly type `$name' at the command prompt.
```

```
                Global environment variables are those you use
                setenv/printenv to work with. To run a command stored
                in such a variable, you need to use the run command,
                and you must not use the '$' sign to access them.

                To store commands and special characters in a
                variable, please use double quotation marks
                surrounding the whole text of the variable, instead
                of the backslashes before semicolons and special
                symbols.

- Commandline Editing and History:
                CONFIG_CMDLINE_EDITING

                Enable editing and History functions for interactive
                commandline input operations

- Default Environment:
                CONFIG_EXTRA_ENV_SETTINGS

                Define this to contain any number of null terminated
                strings (variable = value pairs) that will be part of
                the default environment compiled into the boot image.

                For example, place something like this in your
                board's config file:

                #define CONFIG_EXTRA_ENV_SETTINGS \
                        "myvar1=value1\0" \
                        "myvar2=value2\0"

                Warning: This method is based on knowledge about the
                internal format how the environment is stored by the
                U-Boot code. This is NOT an official, exported
                interface! Although it is unlikely that this format
                will change soon, there is no guarantee either.
                You better know what you are doing here.

                Note: overly (ab)use of the default environment is
                discouraged. Make sure to check other ways to preset
                the environment like the "source" command or the
                boot command first.

                CONFIG_ENV_VARS_UBOOT_CONFIG

                Define this in order to add variables describing the
                U-Boot build configuration to the default environment.
                These will be named arch, cpu, board, vendor, and soc.

                Enabling this option will cause the following to be defined:

                - CONFIG_SYS_ARCH
                - CONFIG_SYS_CPU
                - CONFIG_SYS_BOARD
                - CONFIG_SYS_VENDOR
                - CONFIG_SYS_SOC

                CONFIG_ENV_VARS_UBOOT_RUNTIME_CONFIG

                Define this in order to add variables describing certain
                run-time determined information about the hardware to the
                environment.  These will be named board_name, board_rev.

                CONFIG_DELAY_ENVIRONMENT

                Normally the environment is loaded when the board is
                intialised so that it is available to U-Boot. This inhibits
```

```
                that so that the environment is not available until
                explicitly loaded later by U-Boot code. With CONFIG_OF_CONTROL
                this is instead controlled by the value of
                /config/load-environment.

- DataFlash Support:
                CONFIG_HAS_DATAFLASH

                Defining this option enables DataFlash features and
                allows to read/write in Dataflash via the standard
                commands cp, md...

- Serial Flash support
                CONFIG_CMD_SF

                Defining this option enables SPI flash commands
                'sf probe/read/write/erase/update'.

                Usage requires an initial 'probe' to define the serial
                flash parameters, followed by read/write/erase/update
                commands.

                The following defaults may be provided by the platform
                to handle the common case when only a single serial
                flash is present on the system.

                CONFIG_SF_DEFAULT_BUS           Bus identifier
                CONFIG_SF_DEFAULT_CS            Chip-select
                CONFIG_SF_DEFAULT_MODE          (see include/spi.h)
                CONFIG_SF_DEFAULT_SPEED         in Hz

                CONFIG_CMD_SF_TEST

                Define this option to include a destructive SPI flash
                test ('sf test').

                CONFIG_SPI_FLASH_BAR            Ban/Extended Addr Reg

                Define this option to use the Bank addr/Extended addr
                support on SPI flashes which has size > 16Mbytes.

                CONFIG_SF_DUAL_FLASH            Dual flash memories

                Define this option to use dual flash support where two flash
                memories can be connected with a given cs line.
                currently Xilinx Zynq qspi support these type of connections.

- SystemACE Support:
                CONFIG_SYSTEMACE

                Adding this option adds support for Xilinx SystemACE
                chips attached via some sort of local bus. The address
                of the chip must also be defined in the
                CONFIG_SYS_SYSTEMACE_BASE macro. For example:

                #define CONFIG_SYSTEMACE
                #define CONFIG_SYS_SYSTEMACE_BASE 0xf0000000

                When SystemACE support is added, the "ace" device type
                becomes available to the fat commands, i.e. fatls.

- TFTP Fixed UDP Port:
                CONFIG_TFTP_PORT

                If this is defined, the environment variable tftpsrcp
                is used to supply the TFTP UDP source port value.
                If tftpsrcp isn't defined, the normal pseudo-random port
```

```
                number generator is used.

                Also, the environment variable tftpdstp is used to supply
                the TFTP UDP destination port value.  If tftpdstp isn't
                defined, the normal port 69 is used.

                The purpose for tftpsrcp is to allow a TFTP server to
                blindly start the TFTP transfer using the pre-configured
                target IP address and UDP port. This has the effect of
                "punching through" the (Windows XP) firewall, allowing
                the remainder of the TFTP transfer to proceed normally.
                A better solution is to properly configure the firewall,
                but sometimes that is not allowed.

- Hashing support:
                CONFIG_CMD_HASH

                This enables a generic 'hash' command which can produce
                hashes / digests from a few algorithms (e.g. SHA1, SHA256).

                CONFIG_HASH_VERIFY

                Enable the hash verify command (hash -v). This adds to code
                size a little.

                CONFIG_SHA1 - support SHA1 hashing
                CONFIG_SHA256 - support SHA256 hashing

                Note: There is also a sha1sum command, which should perhaps
                be deprecated in favour of 'hash sha1'.

- Freescale i.MX specific commands:
                CONFIG_CMD_HDMIDETECT
                This enables 'hdmidet' command which returns true if an
                HDMI monitor is detected.  This command is i.MX 6 specific.

                CONFIG_CMD_BMODE
                This enables the 'bmode' (bootmode) command for forcing
                a boot from specific media.

                This is useful for forcing the ROM's usb downloader to
                activate upon a watchdog reset which is nice when iterating
                on U-Boot.  Using the reset button or running bmode normal
                will set it back to normal.  This command currently
                supports i.MX53 and i.MX6.

- Signing support:
                CONFIG_RSA

                This enables the RSA algorithm used for FIT image verification
                in U-Boot. See doc/uImage.FIT/signature.txt for more information.

                The signing part is build into mkimage regardless of this
                option.

- bootcount support:
                CONFIG_BOOTCOUNT_LIMIT

                This enables the bootcounter support, see:
                http://www.denx.de/wiki/DULG/UBootBootCountLimit

                CONFIG_AT91SAM9XE
                enable special bootcounter support on at91sam9xe based boards.
                CONFIG_BLACKFIN
                enable special bootcounter support on blackfin based boards.
                CONFIG_SOC_DA8XX
                enable special bootcounter support on da850 based boards.
```

```
                CONFIG_BOOTCOUNT_RAM
                enable support for the bootcounter in RAM
                CONFIG_BOOTCOUNT_I2C
                enable support for the bootcounter on an i2c (like RTC) device.
                        CONFIG_SYS_I2C_RTC_ADDR = i2c chip address
                        CONFIG_SYS_BOOTCOUNT_ADDR = i2c addr which is used for
                                                    the bootcounter.
                        CONFIG_BOOTCOUNT_ALEN = address len

- Show boot progress:
                CONFIG_SHOW_BOOT_PROGRESS

                Defining this option allows to add some board-
                specific code (calling a user-provided function
                "show_boot_progress(int)") that enables you to show
                the system's boot progress on some display (for
                example, some LED's) on your board. At the moment,
                the following checkpoints are implemented:

- Detailed boot stage timing
                CONFIG_BOOTSTAGE
                Define this option to get detailed timing of each stage
                of the boot process.

                CONFIG_BOOTSTAGE_USER_COUNT
                This is the number of available user bootstage records.
                Each time you call bootstage_mark(BOOTSTAGE_ID_ALLOC, ...)
                a new ID will be allocated from this stash. If you exceed
                the limit, recording will stop.

                CONFIG_BOOTSTAGE_REPORT
                Define this to print a report before boot, similar to this:

                Timer summary in microseconds:
                       Mark    Elapsed  Stage
                          0          0  reset
                  3,575,678  3,575,678  board_init_f start
                  3,575,695         17  arch_cpu_init A9
                  3,575,777         82  arch_cpu_init done
                  3,659,598     83,821  board_init_r start
                  3,910,375    250,777  main_loop
                 29,916,167 26,005,792  bootm_start
                 30,361,327    445,160  start_kernel

                CONFIG_CMD_BOOTSTAGE
                Add a 'bootstage' command which supports printing a report
                and un/stashing of bootstage data.

                CONFIG_BOOTSTAGE_FDT
                Stash the bootstage information in the FDT. A root 'bootstage'
                node is created with each bootstage id as a child. Each child
                has a 'name' property and either 'mark' containing the
                mark time in microsecond, or 'accum' containing the
                accumulated time for that bootstage id in microseconds.
                For example:

                bootstage {
                        154 {
                                name = "board_init_f";
                                mark = <3575678>;
                        };
                        170 {
                                name = "lcd";
                                accum = <33482>;
                        };
                };
```

```
               Code in the Linux kernel can find this in /proc/devicetree.

Legacy uImage format:

  Arg   Where                   When
    1   common/cmd_bootm.c      before attempting to boot an image
   -1   common/cmd_bootm.c      Image header has bad     magic number
    2   common/cmd_bootm.c      Image header has correct magic number
   -2   common/cmd_bootm.c      Image header has bad     checksum
    3   common/cmd_bootm.c      Image header has correct checksum
   -3   common/cmd_bootm.c      Image data   has bad     checksum
    4   common/cmd_bootm.c      Image data   has correct checksum
   -4   common/cmd_bootm.c      Image is for unsupported architecture
    5   common/cmd_bootm.c      Architecture check OK
   -5   common/cmd_bootm.c      Wrong Image Type (not kernel, multi)
    6   common/cmd_bootm.c      Image Type check OK
   -6   common/cmd_bootm.c      gunzip uncompression error
   -7   common/cmd_bootm.c      Unimplemented compression type
    7   common/cmd_bootm.c      Uncompression OK
    8   common/cmd_bootm.c      No uncompress/copy overwrite error
   -9   common/cmd_bootm.c      Unsupported OS (not Linux, BSD, VxWorks, QNX)

    9   common/image.c          Start initial ramdisk verification
  -10   common/image.c          Ramdisk header has bad     magic number
  -11   common/image.c          Ramdisk header has bad     checksum
   10   common/image.c          Ramdisk header is OK
  -12   common/image.c          Ramdisk data   has bad     checksum
   11   common/image.c          Ramdisk data   has correct checksum
   12   common/image.c          Ramdisk verification complete, start loading
  -13   common/image.c          Wrong Image Type (not PPC Linux ramdisk)
   13   common/image.c          Start multifile image verification
   14   common/image.c          No initial ramdisk, no multifile, continue.

   15   arch/<arch>/lib/bootm.c All preparation done, transferring control to OS

  -30   arch/powerpc/lib/board.c        Fatal error, hang the system
  -31   post/post.c             POST test failed, detected by post_output_backlog()
  -32   post/post.c             POST test failed, detected by post_run_single()

   34   common/cmd_doc.c        before loading a Image from a DOC device
  -35   common/cmd_doc.c        Bad usage of "doc" command
   35   common/cmd_doc.c        correct usage of "doc" command
  -36   common/cmd_doc.c        No boot device
   36   common/cmd_doc.c        correct boot device
  -37   common/cmd_doc.c        Unknown Chip ID on boot device
   37   common/cmd_doc.c        correct chip ID found, device available
  -38   common/cmd_doc.c        Read Error on boot device
   38   common/cmd_doc.c        reading Image header from DOC device OK
  -39   common/cmd_doc.c        Image header has bad magic number
   39   common/cmd_doc.c        Image header has correct magic number
  -40   common/cmd_doc.c        Error reading Image from DOC device
   40   common/cmd_doc.c        Image header has correct magic number
   41   common/cmd_ide.c        before loading a Image from a IDE device
  -42   common/cmd_ide.c        Bad usage of "ide" command
   42   common/cmd_ide.c        correct usage of "ide" command
  -43   common/cmd_ide.c        No boot device
   43   common/cmd_ide.c        boot device found
  -44   common/cmd_ide.c        Device not available
   44   common/cmd_ide.c        Device available
  -45   common/cmd_ide.c        wrong partition selected
   45   common/cmd_ide.c        partition selected
  -46   common/cmd_ide.c        Unknown partition table
   46   common/cmd_ide.c        valid partition table found
  -47   common/cmd_ide.c        Invalid partition type
   47   common/cmd_ide.c        correct partition type
  -48   common/cmd_ide.c        Error reading Image Header on boot device
   48   common/cmd_ide.c        reading Image Header from IDE device OK
```

```
  -49   common/cmd_ide.c        Image header has bad magic number
   49   common/cmd_ide.c        Image header has correct magic number
  -50   common/cmd_ide.c        Image header has bad     checksum
   50   common/cmd_ide.c        Image header has correct checksum
  -51   common/cmd_ide.c        Error reading Image from IDE device
   51   common/cmd_ide.c        reading Image from IDE device OK
   52   common/cmd_nand.c       before loading a Image from a NAND device
  -53   common/cmd_nand.c       Bad usage of "nand" command
   53   common/cmd_nand.c       correct usage of "nand" command
  -54   common/cmd_nand.c       No boot device
   54   common/cmd_nand.c       boot device found
  -55   common/cmd_nand.c       Unknown Chip ID on boot device
   55   common/cmd_nand.c       correct chip ID found, device available
  -56   common/cmd_nand.c       Error reading Image Header on boot device
   56   common/cmd_nand.c       reading Image Header from NAND device OK
  -57   common/cmd_nand.c       Image header has bad magic number
   57   common/cmd_nand.c       Image header has correct magic number
  -58   common/cmd_nand.c       Error reading Image from NAND device
   58   common/cmd_nand.c       reading Image from NAND device OK

  -60   common/env_common.c     Environment has a bad CRC, using default

   64   net/eth.c               starting with Ethernet configuration.
  -64   net/eth.c               no Ethernet found.
   65   net/eth.c               Ethernet found.

  -80   common/cmd_net.c        usage wrong
   80   common/cmd_net.c        before calling NetLoop()
  -81   common/cmd_net.c        some error in NetLoop() occurred
   81   common/cmd_net.c        NetLoop() back without error
  -82   common/cmd_net.c        size == 0 (File with size 0 loaded)
   82   common/cmd_net.c        trying automatic boot
   83   common/cmd_net.c        running "source" command
  -83   common/cmd_net.c        some error in automatic boot or "source" command
   84   common/cmd_net.c        end without errors
```

FIT uImage format:

```
  Arg   Where                   When
  100   common/cmd_bootm.c      Kernel FIT Image has correct format
 -100   common/cmd_bootm.c      Kernel FIT Image has incorrect format
  101   common/cmd_bootm.c      No Kernel subimage unit name, using configuration
 -101   common/cmd_bootm.c      Can't get configuration for kernel subimage
  102   common/cmd_bootm.c      Kernel unit name specified
 -103   common/cmd_bootm.c      Can't get kernel subimage node offset
  103   common/cmd_bootm.c      Found configuration node
  104   common/cmd_bootm.c      Got kernel subimage node offset
 -104   common/cmd_bootm.c      Kernel subimage hash verification failed
  105   common/cmd_bootm.c      Kernel subimage hash verification OK
 -105   common/cmd_bootm.c      Kernel subimage is for unsupported architecture
  106   common/cmd_bootm.c      Architecture check OK
 -106   common/cmd_bootm.c      Kernel subimage has wrong type
  107   common/cmd_bootm.c      Kernel subimage type OK
 -107   common/cmd_bootm.c      Can't get kernel subimage data/size
  108   common/cmd_bootm.c      Got kernel subimage data/size
 -108   common/cmd_bootm.c      Wrong image type (not legacy, FIT)
 -109   common/cmd_bootm.c      Can't get kernel subimage type
 -110   common/cmd_bootm.c      Can't get kernel subimage comp
 -111   common/cmd_bootm.c      Can't get kernel subimage os
 -112   common/cmd_bootm.c      Can't get kernel subimage load address
 -113   common/cmd_bootm.c      Image uncompress/copy overwrite error

  120   common/image.c          Start initial ramdisk verification
 -120   common/image.c          Ramdisk FIT image has incorrect format
  121   common/image.c          Ramdisk FIT image has correct format
  122   common/image.c          No ramdisk subimage unit name, using configuration
 -122   common/image.c          Can't get configuration for ramdisk subimage
```

```
  123   common/image.c          Ramdisk unit name specified
 -124   common/image.c          Can't get ramdisk subimage node offset
  125   common/image.c          Got ramdisk subimage node offset
 -125   common/image.c          Ramdisk subimage hash verification failed
  126   common/image.c          Ramdisk subimage hash verification OK
 -126   common/image.c          Ramdisk subimage for unsupported architecture
  127   common/image.c          Architecture check OK
 -127   common/image.c          Can't get ramdisk subimage data/size
  128   common/image.c          Got ramdisk subimage data/size
  129   common/image.c          Can't get ramdisk load address
 -129   common/image.c          Got ramdisk load address

 -130   common/cmd_doc.c        Incorrect FIT image format
  131   common/cmd_doc.c        FIT image format OK

 -140   common/cmd_ide.c        Incorrect FIT image format
  141   common/cmd_ide.c        FIT image format OK

 -150   common/cmd_nand.c       Incorrect FIT image format
  151   common/cmd_nand.c       FIT image format OK

- legacy image format:
                CONFIG_IMAGE_FORMAT_LEGACY
                enables the legacy image format support in U-Boot.

                Default:
                enabled if CONFIG_FIT_SIGNATURE is not defined.

                CONFIG_DISABLE_IMAGE_LEGACY
                disable the legacy image format

                This define is introduced, as the legacy image format is
                enabled per default for backward compatibility.

- FIT image support:
                CONFIG_FIT
                Enable support for the FIT uImage format.

                CONFIG_FIT_BEST_MATCH
                When no configuration is explicitly selected, default to the
                one whose fdt's compatibility field best matches that of
                U-Boot itself. A match is considered "best" if it matches the
                most specific compatibility entry of U-Boot's fdt's root node.
                The order of entries in the configuration's fdt is ignored.

                CONFIG_FIT_SIGNATURE
                This option enables signature verification of FIT uImages,
                using a hash signed and verified using RSA. See
                doc/uImage.FIT/signature.txt for more details.

                WARNING: When relying on signed FIT images with required
                signature check the legacy image format is default
                disabled. If a board need legacy image format support
                enable this through CONFIG_IMAGE_FORMAT_LEGACY

                CONFIG_FIT_DISABLE_SHA256
                Supporting SHA256 hashes has quite an impact on binary size.
                For constrained systems sha256 hash support can be disabled
                with this option.

- Standalone program support:
                CONFIG_STANDALONE_LOAD_ADDR

                This option defines a board specific value for the
                address where standalone program gets loaded, thus
                overwriting the architecture dependent default
                settings.
```

```
- Frame Buffer Address:
                CONFIG_FB_ADDR

                Define CONFIG_FB_ADDR if you want to use specific
                address for frame buffer.  This is typically the case
                when using a graphics controller has separate video
                memory.  U-Boot will then place the frame buffer at
                the given address instead of dynamically reserving it
                in system RAM by calling lcd_setmem(), which grabs
                the memory for the frame buffer depending on the
                configured panel size.

                Please see board_init_f function.

- Automatic software updates via TFTP server
                CONFIG_UPDATE_TFTP
                CONFIG_UPDATE_TFTP_CNT_MAX
                CONFIG_UPDATE_TFTP_MSEC_MAX

                These options enable and control the auto-update feature;
                for a more detailed description refer to doc/README.update.

- MTD Support (mtdparts command, UBI support)
                CONFIG_MTD_DEVICE

                Adds the MTD device infrastructure from the Linux kernel.
                Needed for mtdparts command support.

                CONFIG_MTD_PARTITIONS

                Adds the MTD partitioning infrastructure from the Linux
                kernel. Needed for UBI support.

- UBI support
                CONFIG_CMD_UBI

                Adds commands for interacting with MTD partitions formatted
                with the UBI flash translation layer

                Requires also defining CONFIG_RBTREE

                CONFIG_UBI_SILENCE_MSG

                Make the verbose messages from UBI stop printing.  This leaves
                warnings and errors enabled.

- UBIFS support
                CONFIG_CMD_UBIFS

                Adds commands for interacting with UBI volumes formatted as
                UBIFS.  UBIFS is read-only in u-boot.

                Requires UBI support as well as CONFIG_LZO

                CONFIG_UBIFS_SILENCE_MSG

                Make the verbose messages from UBIFS stop printing.  This leaves
                warnings and errors enabled.

- SPL framework
                CONFIG_SPL
                Enable building of SPL globally.

                CONFIG_SPL_LDSCRIPT
                LDSCRIPT for linking the SPL binary.
```

```
CONFIG_SPL_MAX_FOOTPRINT
Maximum size in memory allocated to the SPL, BSS included.
When defined, the linker checks that the actual memory
used by SPL from _start to __bss_end does not exceed it.
CONFIG_SPL_MAX_FOOTPRINT and CONFIG_SPL_BSS_MAX_SIZE
must not be both defined at the same time.

CONFIG_SPL_MAX_SIZE
Maximum size of the SPL image (text, data, rodata, and
linker lists sections), BSS excluded.
When defined, the linker checks that the actual size does
not exceed it.

CONFIG_SPL_TEXT_BASE
TEXT_BASE for linking the SPL binary.

CONFIG_SPL_RELOC_TEXT_BASE
Address to relocate to.  If unspecified, this is equal to
CONFIG_SPL_TEXT_BASE (i.e. no relocation is done).

CONFIG_SPL_BSS_START_ADDR
Link address for the BSS within the SPL binary.

CONFIG_SPL_BSS_MAX_SIZE
Maximum size in memory allocated to the SPL BSS.
When defined, the linker checks that the actual memory used
by SPL from __bss_start to __bss_end does not exceed it.
CONFIG_SPL_MAX_FOOTPRINT and CONFIG_SPL_BSS_MAX_SIZE
must not be both defined at the same time.

CONFIG_SPL_STACK
Adress of the start of the stack SPL will use

CONFIG_SPL_RELOC_STACK
Adress of the start of the stack SPL will use after
relocation.  If unspecified, this is equal to
CONFIG_SPL_STACK.

CONFIG_SYS_SPL_MALLOC_START
Starting address of the malloc pool used in SPL.

CONFIG_SYS_SPL_MALLOC_SIZE
The size of the malloc pool used in SPL.

CONFIG_SPL_FRAMEWORK
Enable the SPL framework under common/.  This framework
supports MMC, NAND and YMODEM loading of U-Boot and NAND
NAND loading of the Linux Kernel.

CONFIG_SPL_OS_BOOT
Enable booting directly to an OS from SPL.
See also: doc/README.falcon

CONFIG_SPL_DISPLAY_PRINT
For ARM, enable an optional function to print more information
about the running system.

CONFIG_SPL_INIT_MINIMAL
Arch init code should be built for a very small image

CONFIG_SPL_LIBCOMMON_SUPPORT
Support for common/libcommon.o in SPL binary

CONFIG_SPL_LIBDISK_SUPPORT
Support for disk/libdisk.o in SPL binary

CONFIG_SPL_I2C_SUPPORT
```

```
Support for drivers/i2c/libi2c.o in SPL binary

CONFIG_SPL_GPIO_SUPPORT
Support for drivers/gpio/libgpio.o in SPL binary

CONFIG_SPL_MMC_SUPPORT
Support for drivers/mmc/libmmc.o in SPL binary

CONFIG_SYS_MMCSD_RAW_MODE_U_BOOT_SECTOR,
CONFIG_SYS_U_BOOT_MAX_SIZE_SECTORS,
CONFIG_SYS_MMC_SD_FAT_BOOT_PARTITION
Address, size and partition on the MMC to load U-Boot from
when the MMC is being used in raw mode.

CONFIG_SYS_MMCSD_RAW_MODE_KERNEL_SECTOR
Sector to load kernel uImage from when MMC is being
used in raw mode (for Falcon mode)

CONFIG_SYS_MMCSD_RAW_MODE_ARGS_SECTOR,
CONFIG_SYS_MMCSD_RAW_MODE_ARGS_SECTORS
Sector and number of sectors to load kernel argument
parameters from when MMC is being used in raw mode
(for falcon mode)

CONFIG_SPL_FAT_SUPPORT
Support for fs/fat/libfat.o in SPL binary

CONFIG_SPL_FAT_LOAD_PAYLOAD_NAME
Filename to read to load U-Boot when reading from FAT

CONFIG_SPL_FAT_LOAD_KERNEL_NAME
Filename to read to load kernel uImage when reading
from FAT (for Falcon mode)

CONFIG_SPL_FAT_LOAD_ARGS_NAME
Filename to read to load kernel argument parameters
when reading from FAT (for Falcon mode)

CONFIG_SPL_MPC83XX_WAIT_FOR_NAND
Set this for NAND SPL on PPC mpc83xx targets, so that
start.S waits for the rest of the SPL to load before
continuing (the hardware starts execution after just
loading the first page rather than the full 4K).

CONFIG_SPL_SKIP_RELOCATE
Avoid SPL relocation

CONFIG_SPL_NAND_BASE
Include nand_base.c in the SPL.  Requires
CONFIG_SPL_NAND_DRIVERS.

CONFIG_SPL_NAND_DRIVERS
SPL uses normal NAND drivers, not minimal drivers.

CONFIG_SPL_NAND_ECC
Include standard software ECC in the SPL

CONFIG_SPL_NAND_SIMPLE
Support for NAND boot using simple NAND drivers that
expose the cmd_ctrl() interface.

CONFIG_SPL_MTD_SUPPORT
Support for the MTD subsystem within SPL.  Useful for
environment on NAND support within SPL.

CONFIG_SPL_MPC8XXX_INIT_DDR_SUPPORT
Set for the SPL on PPC mpc8xxx targets, support for
```

```
drivers/ddr/fsl/libddr.o in SPL binary.

CONFIG_SPL_COMMON_INIT_DDR
Set for common ddr init with serial presence detect in
SPL binary.

CONFIG_SYS_NAND_5_ADDR_CYCLE, CONFIG_SYS_NAND_PAGE_COUNT,
CONFIG_SYS_NAND_PAGE_SIZE, CONFIG_SYS_NAND_OOBSIZE,
CONFIG_SYS_NAND_BLOCK_SIZE, CONFIG_SYS_NAND_BAD_BLOCK_POS,
CONFIG_SYS_NAND_ECCPOS, CONFIG_SYS_NAND_ECCSIZE,
CONFIG_SYS_NAND_ECCBYTES
Defines the size and behavior of the NAND that SPL uses
to read U-Boot

CONFIG_SPL_NAND_BOOT
Add support NAND boot

CONFIG_SYS_NAND_U_BOOT_OFFS
Location in NAND to read U-Boot from

CONFIG_SYS_NAND_U_BOOT_DST
Location in memory to load U-Boot to

CONFIG_SYS_NAND_U_BOOT_SIZE
Size of image to load

CONFIG_SYS_NAND_U_BOOT_START
Entry point in loaded image to jump to

CONFIG_SYS_NAND_HW_ECC_OOBFIRST
Define this if you need to first read the OOB and then the
data. This is used for example on davinci plattforms.

CONFIG_SPL_OMAP3_ID_NAND
Support for an OMAP3-specific set of functions to return the
ID and MFR of the first attached NAND chip, if present.

CONFIG_SPL_SERIAL_SUPPORT
Support for drivers/serial/libserial.o in SPL binary

CONFIG_SPL_SPI_FLASH_SUPPORT
Support for drivers/mtd/spi/libspi_flash.o in SPL binary

CONFIG_SPL_SPI_SUPPORT
Support for drivers/spi/libspi.o in SPL binary

CONFIG_SPL_RAM_DEVICE
Support for running image already present in ram, in SPL binary

CONFIG_SPL_LIBGENERIC_SUPPORT
Support for lib/libgeneric.o in SPL binary

CONFIG_SPL_ENV_SUPPORT
Support for the environment operating in SPL binary

CONFIG_SPL_NET_SUPPORT
Support for the net/libnet.o in SPL binary.
It conflicts with SPL env from storage medium specified by
CONFIG_ENV_IS_xxx but CONFIG_ENV_IS_NOWHERE

CONFIG_SPL_PAD_TO
Image offset to which the SPL should be padded before appending
the SPL payload. By default, this is defined as
CONFIG_SPL_MAX_SIZE, or 0 if CONFIG_SPL_MAX_SIZE is undefined.
CONFIG_SPL_PAD_TO must be either 0, meaning to append the SPL
payload without any padding, or >= CONFIG_SPL_MAX_SIZE.
```

```
                CONFIG_SPL_TARGET
                Final target image containing SPL and payload.  Some SPLs
                use an arch-specific makefile fragment instead, for
                example if more than one image needs to be produced.

                CONFIG_FIT_SPL_PRINT
                Printing information about a FIT image adds quite a bit of
                code to SPL. So this is normally disabled in SPL. Use this
                option to re-enable it. This will affect the output of the
                bootm command when booting a FIT image.

- TPL framework
                CONFIG_TPL
                Enable building of TPL globally.

                CONFIG_TPL_PAD_TO
                Image offset to which the TPL should be padded before appending
                the TPL payload. By default, this is defined as
                CONFIG_SPL_MAX_SIZE, or 0 if CONFIG_SPL_MAX_SIZE is undefined.
                CONFIG_SPL_PAD_TO must be either 0, meaning to append the SPL
                payload without any padding, or >= CONFIG_SPL_MAX_SIZE.

Modem Support:
--------------

[so far only for SMDK2400 boards]

- Modem support enable:
                CONFIG_MODEM_SUPPORT

- RTS/CTS Flow control enable:
                CONFIG_HWFLOW

- Modem debug support:
                CONFIG_MODEM_SUPPORT_DEBUG

                Enables debugging stuff (char screen[1024], dbg())
                for modem support. Useful only with BDI2000.

- Interrupt support (PPC):

                There are common interrupt_init() and timer_interrupt()
                for all PPC archs. interrupt_init() calls interrupt_init_cpu()
                for CPU specific initialization. interrupt_init_cpu()
                should set decrementer_count to appropriate value. If
                CPU resets decrementer automatically after interrupt
                (ppc4xx) it should set decrementer_count to zero.
                timer_interrupt() calls timer_interrupt_cpu() for CPU
                specific handling. If board has watchdog / status_led
                / other_activity_monitor it works automatically from
                general timer_interrupt().

- General:

                In the target system modem support is enabled when a
                specific key (key combination) is pressed during
                power-on. Otherwise U-Boot will boot normally
                (autoboot). The key_pressed() function is called from
                board_init(). Currently key_pressed() is a dummy
                function, returning 1 and thus enabling modem
                initialization.

                If there are no modem init strings in the
                environment, U-Boot proceed to autoboot; the
                previous output (banner, info printfs) will be
                suppressed, though.
```

```
               See also: doc/README.Modem

Board initialization settings:
------------------------------

During Initialization u-boot calls a number of board specific functions
to allow the preparation of board specific prerequisites, e.g. pin setup
before drivers are initialized. To enable these callbacks the
following configuration macros have to be defined. Currently this is
architecture specific, so please check arch/your_architecture/lib/board.c
typically in board_init_f() and board_init_r().

- CONFIG_BOARD_EARLY_INIT_F: Call board_early_init_f()
- CONFIG_BOARD_EARLY_INIT_R: Call board_early_init_r()
- CONFIG_BOARD_LATE_INIT: Call board_late_init()
- CONFIG_BOARD_POSTCLK_INIT: Call board_postclk_init()

Configuration Settings:
-----------------------

- CONFIG_SYS_SUPPORT_64BIT_DATA: Defined automatically if compiled as 64-bit.
               Optionally it can be defined to support 64-bit memory commands.

- CONFIG_SYS_LONGHELP: Defined when you want long help messages included;
               undefine this when you're short of memory.

- CONFIG_SYS_HELP_CMD_WIDTH: Defined when you want to override the default
               width of the commands listed in the 'help' command output.

- CONFIG_SYS_PROMPT:    This is what U-Boot prints on the console to
               prompt for user input.

- CONFIG_SYS_CBSIZE:    Buffer size for input from the Console

- CONFIG_SYS_PBSIZE:    Buffer size for Console output

- CONFIG_SYS_MAXARGS:   max. Number of arguments accepted for monitor commands

- CONFIG_SYS_BARGSIZE: Buffer size for Boot Arguments which are passed to
               the application (usually a Linux kernel) when it is
               booted

- CONFIG_SYS_BAUDRATE_TABLE:
               List of legal baudrate settings for this board.

- CONFIG_SYS_CONSOLE_INFO_QUIET
               Suppress display of console information at boot.

- CONFIG_SYS_CONSOLE_IS_IN_ENV
               If the board specific function
                       extern int overwrite_console (void);
               returns 1, the stdin, stderr and stdout are switched to the
               serial port, else the settings in the environment are used.

- CONFIG_SYS_CONSOLE_OVERWRITE_ROUTINE
               Enable the call to overwrite_console().

- CONFIG_SYS_CONSOLE_ENV_OVERWRITE
               Enable overwrite of previous console environment settings.

- CONFIG_SYS_MEMTEST_START, CONFIG_SYS_MEMTEST_END:
               Begin and End addresses of the area used by the
               simple memory test.

- CONFIG_SYS_ALT_MEMTEST:
               Enable an alternate, more extensive memory test.
```

```
- CONFIG_SYS_MEMTEST_SCRATCH:
               Scratch address used by the alternate memory test
               You only need to set this if address zero isn't writeable

- CONFIG_SYS_MEM_TOP_HIDE (PPC only):
               If CONFIG_SYS_MEM_TOP_HIDE is defined in the board config header,
               this specified memory area will get subtracted from the top
               (end) of RAM and won't get "touched" at all by U-Boot. By
               fixing up gd->ram_size the Linux kernel should gets passed
               the now "corrected" memory size and won't touch it either.
               This should work for arch/ppc and arch/powerpc. Only Linux
               board ports in arch/powerpc with bootwrapper support that
               recalculate the memory size from the SDRAM controller setup
               will have to get fixed in Linux additionally.

               This option can be used as a workaround for the 440EPx/GRx
               CHIP 11 errata where the last 256 bytes in SDRAM shouldn't
               be touched.

               WARNING: Please make sure that this value is a multiple of
               the Linux page size (normally 4k). If this is not the case,
               then the end address of the Linux memory will be located at a
               non page size aligned address and this could cause major
               problems.

- CONFIG_SYS_LOADS_BAUD_CHANGE:
               Enable temporary baudrate change while serial download

- CONFIG_SYS_SDRAM_BASE:
               Physical start address of SDRAM. _Must_ be 0 here.

- CONFIG_SYS_MBIO_BASE:
               Physical start address of Motherboard I/O (if using a
               Cogent motherboard)

- CONFIG_SYS_FLASH_BASE:
               Physical start address of Flash memory.

- CONFIG_SYS_MONITOR_BASE:
               Physical start address of boot monitor code (set by
               make config files to be same as the text base address
               (CONFIG_SYS_TEXT_BASE) used when linking) - same as
               CONFIG_SYS_FLASH_BASE when booting from flash.

- CONFIG_SYS_MONITOR_LEN:
               Size of memory reserved for monitor code, used to
               determine _at_compile_time_ (!) if the environment is
               embedded within the U-Boot image, or in a separate
               flash sector.

- CONFIG_SYS_MALLOC_LEN:
               Size of DRAM reserved for malloc() use.

- CONFIG_SYS_BOOTM_LEN:
               Normally compressed uImages are limited to an
               uncompressed size of 8 MBytes. If this is not enough,
               you can define CONFIG_SYS_BOOTM_LEN in your board config file
               to adjust this setting to your needs.

- CONFIG_SYS_BOOTMAPSZ:
               Maximum size of memory mapped by the startup code of
               the Linux kernel; all data that must be processed by
               the Linux kernel (bd_info, boot arguments, FDT blob if
               used) must be put below this limit, unless "bootm_low"
               environment variable is defined and non-zero. In such case
               all data for the Linux kernel must be between "bootm_low"
               and "bootm_low" + CONFIG_SYS_BOOTMAPSZ.  The environment
```

```
                variable "bootm_mapsize" will override the value of
                CONFIG_SYS_BOOTMAPSZ.  If CONFIG_SYS_BOOTMAPSZ is undefined,
                then the value in "bootm_size" will be used instead.

- CONFIG_SYS_BOOT_RAMDISK_HIGH:
                Enable initrd_high functionality.  If defined then the
                initrd_high feature is enabled and the bootm ramdisk subcommand
                is enabled.

- CONFIG_SYS_BOOT_GET_CMDLINE:
                Enables allocating and saving kernel cmdline in space between
                "bootm_low" and "bootm_low" + BOOTMAPSZ.

- CONFIG_SYS_BOOT_GET_KBD:
                Enables allocating and saving a kernel copy of the bd_info in
                space between "bootm_low" and "bootm_low" + BOOTMAPSZ.

- CONFIG_SYS_MAX_FLASH_BANKS:
                Max number of Flash memory banks

- CONFIG_SYS_MAX_FLASH_SECT:
                Max number of sectors on a Flash chip

- CONFIG_SYS_FLASH_ERASE_TOUT:
                Timeout for Flash erase operations (in ms)

- CONFIG_SYS_FLASH_WRITE_TOUT:
                Timeout for Flash write operations (in ms)

- CONFIG_SYS_FLASH_LOCK_TOUT
                Timeout for Flash set sector lock bit operation (in ms)

- CONFIG_SYS_FLASH_UNLOCK_TOUT
                Timeout for Flash clear lock bits operation (in ms)

- CONFIG_SYS_FLASH_PROTECTION
                If defined, hardware flash sectors protection is used
                instead of U-Boot software protection.

- CONFIG_SYS_DIRECT_FLASH_TFTP:

                Enable TFTP transfers directly to flash memory;
                without this option such a download has to be
                performed in two steps: (1) download to RAM, and (2)
                copy from RAM to flash.

                The two-step approach is usually more reliable, since
                you can check if the download worked before you erase
                the flash, but in some situations (when system RAM is
                too limited to allow for a temporary copy of the
                downloaded image) this option may be very useful.

- CONFIG_SYS_FLASH_CFI:
                Define if the flash driver uses extra elements in the
                common flash structure for storing flash geometry.

- CONFIG_FLASH_CFI_DRIVER
                This option also enables the building of the cfi_flash driver
                in the drivers directory

- CONFIG_FLASH_CFI_MTD
                This option enables the building of the cfi_mtd driver
                in the drivers directory. The driver exports CFI flash
                to the MTD layer.

- CONFIG_SYS_FLASH_USE_BUFFER_WRITE
                Use buffered writes to flash.
```

```
- CONFIG_FLASH_SPANSION_S29WS_N
               s29ws-n MirrorBit flash has non-standard addresses for buffered
               write commands.

- CONFIG_SYS_FLASH_QUIET_TEST
               If this option is defined, the common CFI flash doesn't
               print it's warning upon not recognized FLASH banks. This
               is useful, if some of the configured banks are only
               optionally available.

- CONFIG_FLASH_SHOW_PROGRESS
               If defined (must be an integer), print out countdown
               digits and dots.  Recommended value: 45 (9..1) for 80
               column displays, 15 (3..1) for 40 column displays.

- CONFIG_FLASH_VERIFY
               If defined, the content of the flash (destination) is compared
               against the source after the write operation. An error message
               will be printed when the contents are not identical.
               Please note that this option is useless in nearly all cases,
               since such flash programming errors usually are detected earlier
               while unprotecting/erasing/programming. Please only enable
               this option if you really know what you are doing.

- CONFIG_SYS_RX_ETH_BUFFER:
               Defines the number of Ethernet receive buffers. On some
               Ethernet controllers it is recommended to set this value
               to 8 or even higher (EEPRO100 or 405 EMAC), since all
               buffers can be full shortly after enabling the interface
               on high Ethernet traffic.
               Defaults to 4 if not defined.

- CONFIG_ENV_MAX_ENTRIES

       Maximum number of entries in the hash table that is used
       internally to store the environment settings. The default
       setting is supposed to be generous and should work in most
       cases. This setting can be used to tune behaviour; see
       lib/hashtable.c for details.

- CONFIG_ENV_FLAGS_LIST_DEFAULT
- CONFIG_ENV_FLAGS_LIST_STATIC
       Enable validation of the values given to environment variables when
       calling env set.  Variables can be restricted to only decimal,
       hexadecimal, or boolean.  If CONFIG_CMD_NET is also defined,
       the variables can also be restricted to IP address or MAC address.

       The format of the list is:
               type_attribute = [s|d|x|b|i|m]
               access_atribute = [a|r|o|c]
               attributes = type_attribute[access_atribute]
               entry = variable_name[:attributes]
               list = entry[,list]

       The type attributes are:
               s - String (default)
               d - Decimal
               x - Hexadecimal
               b - Boolean ([1yYtT|0nNfF])
               i - IP address
               m - MAC address

       The access attributes are:
               a - Any (default)
               r - Read-only
               o - Write-once
```

```
                c - Change-default

        - CONFIG_ENV_FLAGS_LIST_DEFAULT
                Define this to a list (string) to define the ".flags"
                envirnoment variable in the default or embedded environment.

        - CONFIG_ENV_FLAGS_LIST_STATIC
                Define this to a list (string) to define validation that
                should be done if an entry is not found in the ".flags"
                environment variable.  To override a setting in the static
                list, simply add an entry for the same variable name to the
                ".flags" variable.

- CONFIG_ENV_ACCESS_IGNORE_FORCE
        If defined, don't allow the -f switch to env set override variable
        access flags.

- CONFIG_SYS_GENERIC_BOARD
        This selects the architecture-generic board system instead of the
        architecture-specific board files. It is intended to move boards
        to this new framework over time. Defining this will disable the
        arch/foo/lib/board.c file and use common/board_f.c and
        common/board_r.c instead. To use this option your architecture
        must support it (i.e. must define __HAVE_ARCH_GENERIC_BOARD in
        its config.mk file). If you find problems enabling this option on
        your board please report the problem and send patches!

- CONFIG_OMAP_PLATFORM_RESET_TIME_MAX_USEC (OMAP only)
        This is set by OMAP boards for the max time that reset should
        be asserted. See doc/README.omap-reset-time for details on how
        the value can be calulated on a given board.

The following definitions that deal with the placement and management
of environment data (variable area); in general, we support the
following configurations:

- CONFIG_BUILD_ENVCRC:

        Builds up envcrc with the target environment so that external utils
        may easily extract it and embed it in final U-Boot images.

- CONFIG_ENV_IS_IN_FLASH:

        Define this if the environment is in flash memory.

        a) The environment occupies one whole flash sector, which is
           "embedded" in the text segment with the U-Boot code. This
           happens usually with "bottom boot sector" or "top boot
           sector" type flash chips, which have several smaller
           sectors at the start or the end. For instance, such a
           layout can have sector sizes of 8, 2x4, 16, Nx32 kB. In
           such a case you would place the environment in one of the
           4 kB sectors - with U-Boot code before and after it. With
           "top boot sector" type flash chips, you would put the
           environment in one of the last sectors, leaving a gap
           between U-Boot and the environment.

        - CONFIG_ENV_OFFSET:

           Offset of environment data (variable area) to the
           beginning of flash memory; for instance, with bottom boot
           type flash chips the second sector can be used: the offset
           for this sector is given here.

           CONFIG_ENV_OFFSET is used relative to CONFIG_SYS_FLASH_BASE.

        - CONFIG_ENV_ADDR:
```

```
           This is just another way to specify the start address of
           the flash sector containing the environment (instead of
           CONFIG_ENV_OFFSET).

        - CONFIG_ENV_SECT_SIZE:

           Size of the sector containing the environment.


        b) Sometimes flash chips have few, equal sized, BIG sectors.
           In such a case you don't want to spend a whole sector for
           the environment.

        - CONFIG_ENV_SIZE:

           If you use this in combination with CONFIG_ENV_IS_IN_FLASH
           and CONFIG_ENV_SECT_SIZE, you can specify to use only a part
           of this flash sector for the environment. This saves
           memory for the RAM copy of the environment.

           It may also save flash memory if you decide to use this
           when your environment is "embedded" within U-Boot code,
           since then the remainder of the flash sector could be used
           for U-Boot code. It should be pointed out that this is
           STRONGLY DISCOURAGED from a robustness point of view:
           updating the environment in flash makes it always
           necessary to erase the WHOLE sector. If something goes
           wrong before the contents has been restored from a copy in
           RAM, your target system will be dead.

        - CONFIG_ENV_ADDR_REDUND
          CONFIG_ENV_SIZE_REDUND

           These settings describe a second storage area used to hold
           a redundant copy of the environment data, so that there is
           a valid backup copy in case there is a power failure during
           a "saveenv" operation.

BE CAREFUL! Any changes to the flash layout, and some changes to the
source code will make it necessary to adapt <board>/u-boot.lds*
accordingly!


- CONFIG_ENV_IS_IN_NVRAM:

        Define this if you have some non-volatile memory device
        (NVRAM, battery buffered SRAM) which you want to use for the
        environment.

        - CONFIG_ENV_ADDR:
        - CONFIG_ENV_SIZE:

          These two #defines are used to determine the memory area you
          want to use for environment. It is assumed that this memory
          can just be read and written to, without any special
          provision.

BE CAREFUL! The first access to the environment happens quite early
in U-Boot initalization (when we try to get the setting of for the
console baudrate). You *MUST* have mapped your NVRAM area then, or
U-Boot will hang.

Please note that even with NVRAM we still use a copy of the
environment in RAM: we could work on NVRAM directly, but we want to
keep settings there always unmodified except somebody uses "saveenv"
to save the current settings.
```

```
- CONFIG_ENV_IS_IN_EEPROM:

        Use this if you have an EEPROM or similar serial access
        device and a driver for it.

        - CONFIG_ENV_OFFSET:
        - CONFIG_ENV_SIZE:

          These two #defines specify the offset and size of the
          environment area within the total memory of your EEPROM.

        - CONFIG_SYS_I2C_EEPROM_ADDR:
          If defined, specified the chip address of the EEPROM device.
          The default address is zero.

        - CONFIG_SYS_EEPROM_PAGE_WRITE_BITS:
          If defined, the number of bits used to address bytes in a
          single page in the EEPROM device.  A 64 byte page, for example
          would require six bits.

        - CONFIG_SYS_EEPROM_PAGE_WRITE_DELAY_MS:
          If defined, the number of milliseconds to delay between
          page writes.  The default is zero milliseconds.

        - CONFIG_SYS_I2C_EEPROM_ADDR_LEN:
          The length in bytes of the EEPROM memory array address.  Note
          that this is NOT the chip address length!

        - CONFIG_SYS_I2C_EEPROM_ADDR_OVERFLOW:
          EEPROM chips that implement "address overflow" are ones
          like Catalyst 24WC04/08/16 which has 9/10/11 bits of
          address and the extra bits end up in the "chip address" bit
          slots. This makes a 24WC08 (1Kbyte) chip look like four 256
          byte chips.

          Note that we consider the length of the address field to
          still be one byte because the extra address bits are hidden
          in the chip address.

        - CONFIG_SYS_EEPROM_SIZE:
          The size in bytes of the EEPROM device.

        - CONFIG_ENV_EEPROM_IS_ON_I2C
          define this, if you have I2C and SPI activated, and your
          EEPROM, which holds the environment, is on the I2C bus.

        - CONFIG_I2C_ENV_EEPROM_BUS
          if you have an Environment on an EEPROM reached over
          I2C muxes, you can define here, how to reach this
          EEPROM. For example:

          #define CONFIG_I2C_ENV_EEPROM_BUS       1

          EEPROM which holds the environment, is reached over
          a pca9547 i2c mux with address 0x70, channel 3.

- CONFIG_ENV_IS_IN_DATAFLASH:

        Define this if you have a DataFlash memory device which you
        want to use for the environment.

        - CONFIG_ENV_OFFSET:
        - CONFIG_ENV_ADDR:
        - CONFIG_ENV_SIZE:
```

```
          These three #defines specify the offset and size of the
          environment area within the total memory of your DataFlash placed
          at the specified address.

- CONFIG_ENV_IS_IN_SPI_FLASH:

        Define this if you have a SPI Flash memory device which you
        want to use for the environment.

        - CONFIG_ENV_OFFSET:
        - CONFIG_ENV_SIZE:

          These two #defines specify the offset and size of the
          environment area within the SPI Flash. CONFIG_ENV_OFFSET must be
          aligned to an erase sector boundary.

        - CONFIG_ENV_SECT_SIZE:

          Define the SPI flash's sector size.

        - CONFIG_ENV_OFFSET_REDUND (optional):

          This setting describes a second storage area of CONFIG_ENV_SIZE
          size used to hold a redundant copy of the environment data, so
          that there is a valid backup copy in case there is a power failure
          during a "saveenv" operation. CONFIG_ENV_OFFSET_RENDUND must be
          aligned to an erase sector boundary.

        - CONFIG_ENV_SPI_BUS (optional):
        - CONFIG_ENV_SPI_CS (optional):

          Define the SPI bus and chip select. If not defined they will be 0.

        - CONFIG_ENV_SPI_MAX_HZ (optional):

          Define the SPI max work clock. If not defined then use 1MHz.

        - CONFIG_ENV_SPI_MODE (optional):

          Define the SPI work mode. If not defined then use SPI_MODE_3.

- CONFIG_ENV_IS_IN_REMOTE:

        Define this if you have a remote memory space which you
        want to use for the local device's environment.

        - CONFIG_ENV_ADDR:
        - CONFIG_ENV_SIZE:

          These two #defines specify the address and size of the
          environment area within the remote memory space. The
          local device can get the environment from remote memory
          space by SRIO or PCIE links.

BE CAREFUL! For some special cases, the local device can not use
"saveenv" command. For example, the local device will get the
environment stored in a remote NOR flash by SRIO or PCIE link,
but it can not erase, write this NOR flash by SRIO or PCIE interface.

- CONFIG_ENV_IS_IN_NAND:

        Define this if you have a NAND device which you want to use
        for the environment.

        - CONFIG_ENV_OFFSET:
        - CONFIG_ENV_SIZE:
```

```
          These two #defines specify the offset and size of the environment
          area within the first NAND device.  CONFIG_ENV_OFFSET must be
          aligned to an erase block boundary.

        - CONFIG_ENV_OFFSET_REDUND (optional):

          This setting describes a second storage area of CONFIG_ENV_SIZE
          size used to hold a redundant copy of the environment data, so
          that there is a valid backup copy in case there is a power failure
          during a "saveenv" operation.  CONFIG_ENV_OFFSET_RENDUND must be
          aligned to an erase block boundary.

        - CONFIG_ENV_RANGE (optional):

          Specifies the length of the region in which the environment
          can be written.  This should be a multiple of the NAND device's
          block size.  Specifying a range with more erase blocks than
          are needed to hold CONFIG_ENV_SIZE allows bad blocks within
          the range to be avoided.

        - CONFIG_ENV_OFFSET_OOB (optional):

          Enables support for dynamically retrieving the offset of the
          environment from block zero's out-of-band data.  The
          "nand env.oob" command can be used to record this offset.
          Currently, CONFIG_ENV_OFFSET_REDUND is not supported when
          using CONFIG_ENV_OFFSET_OOB.

- CONFIG_NAND_ENV_DST

        Defines address in RAM to which the nand_spl code should copy the
        environment. If redundant environment is used, it will be copied to
        CONFIG_NAND_ENV_DST + CONFIG_ENV_SIZE.

- CONFIG_ENV_IS_IN_UBI:

        Define this if you have an UBI volume that you want to use for the
        environment.  This has the benefit of wear-leveling the environment
        accesses, which is important on NAND.

        - CONFIG_ENV_UBI_PART:

          Define this to a string that is the mtd partition containing the UBI.

        - CONFIG_ENV_UBI_VOLUME:

          Define this to the name of the volume that you want to store the
          environment in.

        - CONFIG_ENV_UBI_VOLUME_REDUND:

          Define this to the name of another volume to store a second copy of
          the environment in.  This will enable redundant environments in UBI.
          It is assumed that both volumes are in the same MTD partition.

        - CONFIG_UBI_SILENCE_MSG
        - CONFIG_UBIFS_SILENCE_MSG

          You will probably want to define these to avoid a really noisy system
          when storing the env in UBI.

- CONFIG_ENV_IS_IN_FAT:
       Define this if you want to use the FAT file system for the environment.

       - FAT_ENV_INTERFACE:

         Define this to a string that is the name of the block device.
```

```
       - FAT_ENV_DEV_AND_PART:

         Define this to a string to specify the partition of the device. It can
         be as following:

           "D:P", "D:0", "D", "D:" or "D:auto" (D, P are integers. And P >= 1)
               - "D:P": device D partition P. Error occurs if device D has no
                        partition table.
               - "D:0": device D.
               - "D" or "D:": device D partition 1 if device D has partition
                              table, or the whole device D if has no partition
                              table.
               - "D:auto": first partition in device D with bootable flag set.
                           If none, first valid paratition in device D. If no
                           partition table then means device D.

       - FAT_ENV_FILE:

         It's a string of the FAT file name. This file use to store the
         envrionment.

       - CONFIG_FAT_WRITE:
         This should be defined. Otherwise it cannot save the envrionment file.

- CONFIG_ENV_IS_IN_MMC:

       Define this if you have an MMC device which you want to use for the
       environment.

       - CONFIG_SYS_MMC_ENV_DEV:

         Specifies which MMC device the environment is stored in.

       - CONFIG_SYS_MMC_ENV_PART (optional):

         Specifies which MMC partition the environment is stored in. If not
         set, defaults to partition 0, the user area. Common values might be
         1 (first MMC boot partition), 2 (second MMC boot partition).

       - CONFIG_ENV_OFFSET:
       - CONFIG_ENV_SIZE:

         These two #defines specify the offset and size of the environment
         area within the specified MMC device.

         If offset is positive (the usual case), it is treated as relative to
         the start of the MMC partition. If offset is negative, it is treated
         as relative to the end of the MMC partition. This can be useful if
         your board may be fitted with different MMC devices, which have
         different sizes for the MMC partitions, and you always want the
         environment placed at the very end of the partition, to leave the
         maximum possible space before it, to store other data.

         These two values are in units of bytes, but must be aligned to an
         MMC sector boundary.

       - CONFIG_ENV_OFFSET_REDUND (optional):

         Specifies a second storage area, of CONFIG_ENV_SIZE size, used to
         hold a redundant copy of the environment data. This provides a
         valid backup copy in case the other copy is corrupted, e.g. due
         to a power failure during a "saveenv" operation.

         This value may also be positive or negative; this is handled in the
         same way as CONFIG_ENV_OFFSET.
```

```
          This value is also in units of bytes, but must also be aligned to
          an MMC sector boundary.

        - CONFIG_ENV_SIZE_REDUND (optional):

          This value need not be set, even when CONFIG_ENV_OFFSET_REDUND is
          set. If this value is set, it must be set to the same value as
          CONFIG_ENV_SIZE.

- CONFIG_SYS_SPI_INIT_OFFSET

        Defines offset to the initial SPI buffer area in DPRAM. The
        area is used at an early stage (ROM part) if the environment
        is configured to reside in the SPI EEPROM: We need a 520 byte
        scratch DPRAM area. It is used between the two initialization
        calls (spi_init_f() and spi_init_r()). A value of 0xB00 seems
        to be a good choice since it makes it far enough from the
        start of the data area as well as from the stack pointer.

Please note that the environment is read-only until the monitor
has been relocated to RAM and a RAM copy of the environment has been
created; also, when using EEPROM you will have to use getenv_f()
until then to read environment variables.

The environment is protected by a CRC32 checksum. Before the monitor
is relocated into RAM, as a result of a bad CRC you will be working
with the compiled-in default environment - *silently*!!! [This is
necessary, because the first environment variable we need is the
"baudrate" setting for the console - if we have a bad CRC, we don't
have any device yet where we could complain.]

Note: once the monitor has been relocated, then it will complain if
the default environment is used; a new CRC is computed as soon as you
use the "saveenv" command to store a valid environment.

- CONFIG_SYS_FAULT_ECHO_LINK_DOWN:
                Echo the inverted Ethernet link state to the fault LED.

                Note: If this option is active, then CONFIG_SYS_FAULT_MII_ADDR
                      also needs to be defined.

- CONFIG_SYS_FAULT_MII_ADDR:
                MII address of the PHY to check for the Ethernet link state.

- CONFIG_NS16550_MIN_FUNCTIONS:
                Define this if you desire to only have use of the NS16550_init
                and NS16550_putc functions for the serial driver located at
                drivers/serial/ns16550.c.  This option is useful for saving
                space for already greatly restricted images, including but not
                limited to NAND_SPL configurations.

- CONFIG_DISPLAY_BOARDINFO
                Display information about the board that U-Boot is running on
                when U-Boot starts up. The board function checkboard() is called
                to do this.

- CONFIG_DISPLAY_BOARDINFO_LATE
                Similar to the previous option, but display this information
                later, once stdio is running and output goes to the LCD, if
                present.

Low Level (hardware related) configuration options:
---------------------------------------------------

- CONFIG_SYS_CACHELINE_SIZE:
                Cache Line Size of the CPU.
```

```
- CONFIG_SYS_DEFAULT_IMMR:
                Default address of the IMMR after system reset.

                Needed on some 8260 systems (MPC8260ADS, PQ2FADS-ZU,
                and RPXsuper) to be able to adjust the position of
                the IMMR register after a reset.

- CONFIG_SYS_CCSRBAR_DEFAULT:
                Default (power-on reset) physical address of CCSR on Freescale
                PowerPC SOCs.

- CONFIG_SYS_CCSRBAR:
                Virtual address of CCSR.  On a 32-bit build, this is typically
                the same value as CONFIG_SYS_CCSRBAR_DEFAULT.

                CONFIG_SYS_DEFAULT_IMMR must also be set to this value,
                for cross-platform code that uses that macro instead.

- CONFIG_SYS_CCSRBAR_PHYS:
                Physical address of CCSR.  CCSR can be relocated to a new
                physical address, if desired.  In this case, this macro should
                be set to that address.  Otherwise, it should be set to the
                same value as CONFIG_SYS_CCSRBAR_DEFAULT.  For example, CCSR
                is typically relocated on 36-bit builds.  It is recommended
                that this macro be defined via the _HIGH and _LOW macros:

                #define CONFIG_SYS_CCSRBAR_PHYS ((CONFIG_SYS_CCSRBAR_PHYS_HIGH
                        * 1ull) << 32 | CONFIG_SYS_CCSRBAR_PHYS_LOW)

- CONFIG_SYS_CCSRBAR_PHYS_HIGH:
                Bits 33-36 of CONFIG_SYS_CCSRBAR_PHYS.  This value is typically
                either 0 (32-bit build) or 0xF (36-bit build).  This macro is
                used in assembly code, so it must not contain typecasts or
                integer size suffixes (e.g. "ULL").

- CONFIG_SYS_CCSRBAR_PHYS_LOW:
                Lower 32-bits of CONFIG_SYS_CCSRBAR_PHYS.  This macro is
                used in assembly code, so it must not contain typecasts or
                integer size suffixes (e.g. "ULL").

- CONFIG_SYS_CCSR_DO_NOT_RELOCATE:
                If this macro is defined, then CONFIG_SYS_CCSRBAR_PHYS will be
                forced to a value that ensures that CCSR is not relocated.

- Floppy Disk Support:
                CONFIG_SYS_FDC_DRIVE_NUMBER

                the default drive number (default value 0)

                CONFIG_SYS_ISA_IO_STRIDE

                defines the spacing between FDC chipset registers
                (default value 1)

                CONFIG_SYS_ISA_IO_OFFSET

                defines the offset of register from address. It
                depends on which part of the data bus is connected to
                the FDC chipset. (default value 0)

                If CONFIG_SYS_ISA_IO_STRIDE CONFIG_SYS_ISA_IO_OFFSET and
                CONFIG_SYS_FDC_DRIVE_NUMBER are undefined, they take their
                default value.

                if CONFIG_SYS_FDC_HW_INIT is defined, then the function
                fdc_hw_init() is called at the beginning of the FDC
                setup. fdc_hw_init() must be provided by the board
```

```
                source code. It is used to make hardware dependant
                initializations.

- CONFIG_IDE_AHB:
                Most IDE controllers were designed to be connected with PCI
                interface. Only few of them were designed for AHB interface.
                When software is doing ATA command and data transfer to
                IDE devices through IDE-AHB controller, some additional
                registers accessing to these kind of IDE-AHB controller
                is requierd.

- CONFIG_SYS_IMMR:      Physical address of the Internal Memory.
                DO NOT CHANGE unless you know exactly what you're
                doing! (11-4) [MPC8xx/82xx systems only]

- CONFIG_SYS_INIT_RAM_ADDR:

                Start address of memory area that can be used for
                initial data and stack; please note that this must be
                writable memory that is working WITHOUT special
                initialization, i. e. you CANNOT use normal RAM which
                will become available only after programming the
                memory controller and running certain initialization
                sequences.

                U-Boot uses the following memory types:
                - MPC8xx and MPC8260: IMMR (internal memory of the CPU)
                - MPC824X: data cache
                - PPC4xx:  data cache

- CONFIG_SYS_GBL_DATA_OFFSET:

                Offset of the initial data structure in the memory
                area defined by CONFIG_SYS_INIT_RAM_ADDR. Usually
                CONFIG_SYS_GBL_DATA_OFFSET is chosen such that the initial
                data is located at the end of the available space
                (sometimes written as (CONFIG_SYS_INIT_RAM_SIZE -
                CONFIG_SYS_INIT_DATA_SIZE), and the initial stack is just
                below that area (growing from (CONFIG_SYS_INIT_RAM_ADDR +
                CONFIG_SYS_GBL_DATA_OFFSET) downward.

        Note:
                On the MPC824X (or other systems that use the data
                cache for initial memory) the address chosen for
                CONFIG_SYS_INIT_RAM_ADDR is basically arbitrary - it must
                point to an otherwise UNUSED address space between
                the top of RAM and the start of the PCI space.

- CONFIG_SYS_SIUMCR:    SIU Module Configuration (11-6)

- CONFIG_SYS_SYPCR:     System Protection Control (11-9)

- CONFIG_SYS_TBSCR:     Time Base Status and Control (11-26)

- CONFIG_SYS_PISCR:     Periodic Interrupt Status and Control (11-31)

- CONFIG_SYS_PLPRCR:    PLL, Low-Power, and Reset Control Register (15-30)

- CONFIG_SYS_SCCR:      System Clock and reset Control Register (15-27)

- CONFIG_SYS_OR_TIMING_SDRAM:
                SDRAM timing

- CONFIG_SYS_MAMR_PTA:
                periodic timer for refresh

- CONFIG_SYS_DER:       Debug Event Register (37-47)
```

```
- FLASH_BASE0_PRELIM, FLASH_BASE1_PRELIM, CONFIG_SYS_REMAP_OR_AM,
  CONFIG_SYS_PRELIM_OR_AM, CONFIG_SYS_OR_TIMING_FLASH, CONFIG_SYS_OR0_REMAP,
  CONFIG_SYS_OR0_PRELIM, CONFIG_SYS_BR0_PRELIM, CONFIG_SYS_OR1_REMAP, CONFIG_SYS_OR1_PRELIM,
  CONFIG_SYS_BR1_PRELIM:
               Memory Controller Definitions: BR0/1 and OR0/1 (FLASH)

- SDRAM_BASE2_PRELIM, SDRAM_BASE3_PRELIM, SDRAM_MAX_SIZE,
  CONFIG_SYS_OR_TIMING_SDRAM, CONFIG_SYS_OR2_PRELIM, CONFIG_SYS_BR2_PRELIM,
  CONFIG_SYS_OR3_PRELIM, CONFIG_SYS_BR3_PRELIM:
               Memory Controller Definitions: BR2/3 and OR2/3 (SDRAM)

- CONFIG_SYS_MAMR_PTA, CONFIG_SYS_MPTPR_2BK_4K, CONFIG_SYS_MPTPR_1BK_4K, CONFIG_SYS_MPTPR_2BK_8K,
  CONFIG_SYS_MPTPR_1BK_8K, CONFIG_SYS_MAMR_8COL, CONFIG_SYS_MAMR_9COL:
               Machine Mode Register and Memory Periodic Timer
               Prescaler definitions (SDRAM timing)

- CONFIG_SYS_I2C_UCODE_PATCH, CONFIG_SYS_I2C_DPMEM_OFFSET [0x1FC0]:
               enable I2C microcode relocation patch (MPC8xx);
               define relocation offset in DPRAM [DSP2]

- CONFIG_SYS_SMC_UCODE_PATCH, CONFIG_SYS_SMC_DPMEM_OFFSET [0x1FC0]:
               enable SMC microcode relocation patch (MPC8xx);
               define relocation offset in DPRAM [SMC1]

- CONFIG_SYS_SPI_UCODE_PATCH, CONFIG_SYS_SPI_DPMEM_OFFSET [0x1FC0]:
               enable SPI microcode relocation patch (MPC8xx);
               define relocation offset in DPRAM [SCC4]

- CONFIG_SYS_USE_OSCCLK:
               Use OSCM clock mode on MBX8xx board. Be careful,
               wrong setting might damage your board. Read
               doc/README.MBX before setting this variable!

- CONFIG_SYS_CPM_POST_WORD_ADDR: (MPC8xx, MPC8260 only)
               Offset of the bootmode word in DPRAM used by post
               (Power On Self Tests). This definition overrides
               #define'd default value in commproc.h resp.
               cpm_8260.h.

- CONFIG_SYS_PCI_SLV_MEM_LOCAL, CONFIG_SYS_PCI_SLV_MEM_BUS, CONFIG_SYS_PICMR0_MASK_ATTRIB,
  CONFIG_SYS_PCI_MSTR0_LOCAL, CONFIG_SYS_PCIMSK0_MASK, CONFIG_SYS_PCI_MSTR1_LOCAL,
  CONFIG_SYS_PCIMSK1_MASK, CONFIG_SYS_PCI_MSTR_MEM_LOCAL, CONFIG_SYS_PCI_MSTR_MEM_BUS,
  CONFIG_SYS_CPU_PCI_MEM_START, CONFIG_SYS_PCI_MSTR_MEM_SIZE, CONFIG_SYS_POCMR0_MASK_ATTRIB,
  CONFIG_SYS_PCI_MSTR_MEMIO_LOCAL, CONFIG_SYS_PCI_MSTR_MEMIO_BUS, CPU_PCI_MEMIO_START,
  CONFIG_SYS_PCI_MSTR_MEMIO_SIZE, CONFIG_SYS_POCMR1_MASK_ATTRIB, CONFIG_SYS_PCI_MSTR_IO_LOCAL,
  CONFIG_SYS_PCI_MSTR_IO_BUS, CONFIG_SYS_CPU_PCI_IO_START, CONFIG_SYS_PCI_MSTR_IO_SIZE,
  CONFIG_SYS_POCMR2_MASK_ATTRIB: (MPC826x only)
               Overrides the default PCI memory map in arch/powerpc/cpu/mpc8260/pci.c if set.

- CONFIG_PCI_DISABLE_PCIE:
               Disable PCI-Express on systems where it is supported but not
               required.

- CONFIG_PCI_ENUM_ONLY
               Only scan through and get the devices on the busses.
               Don't do any setup work, presumably because someone or
               something has already done it, and we don't need to do it
               a second time.  Useful for platforms that are pre-booted
               by coreboot or similar.

- CONFIG_PCI_INDIRECT_BRIDGE:
               Enable support for indirect PCI bridges.

- CONFIG_SYS_SRIO:
               Chip has SRIO or not
```

```
- CONFIG_SRIO1:
               Board has SRIO 1 port available

- CONFIG_SRIO2:
               Board has SRIO 2 port available

- CONFIG_SRIO_PCIE_BOOT_MASTER
               Board can support master function for Boot from SRIO and PCIE

- CONFIG_SYS_SRIOn_MEM_VIRT:
               Virtual Address of SRIO port 'n' memory region

- CONFIG_SYS_SRIOn_MEM_PHYS:
               Physical Address of SRIO port 'n' memory region

- CONFIG_SYS_SRIOn_MEM_SIZE:
               Size of SRIO port 'n' memory region

- CONFIG_SYS_NAND_BUSWIDTH_16BIT
               Defined to tell the NAND controller that the NAND chip is using
               a 16 bit bus.
               Not all NAND drivers use this symbol.
               Example of drivers that use it:
               - drivers/mtd/nand/ndfc.c
               - drivers/mtd/nand/mxc_nand.c

- CONFIG_SYS_NDFC_EBC0_CFG
               Sets the EBC0_CFG register for the NDFC. If not defined
               a default value will be used.

- CONFIG_SPD_EEPROM
               Get DDR timing information from an I2C EEPROM. Common
               with pluggable memory modules such as SODIMMs

  SPD_EEPROM_ADDRESS
               I2C address of the SPD EEPROM

- CONFIG_SYS_SPD_BUS_NUM
               If SPD EEPROM is on an I2C bus other than the first
               one, specify here. Note that the value must resolve
               to something your driver can deal with.

- CONFIG_SYS_DDR_RAW_TIMING
               Get DDR timing information from other than SPD. Common with
               soldered DDR chips onboard without SPD. DDR raw timing
               parameters are extracted from datasheet and hard-coded into
               header files or board specific files.

- CONFIG_FSL_DDR_INTERACTIVE
               Enable interactive DDR debugging. See doc/README.fsl-ddr.

- CONFIG_SYS_83XX_DDR_USES_CS0
               Only for 83xx systems. If specified, then DDR should
               be configured using CS0 and CS1 instead of CS2 and CS3.

- CONFIG_ETHER_ON_FEC[12]
               Define to enable FEC[12] on a 8xx series processor.

- CONFIG_FEC[12]_PHY
               Define to the hardcoded PHY address which corresponds
               to the given FEC; i. e.
                       #define CONFIG_FEC1_PHY 4
               means that the PHY with address 4 is connected to FEC1

               When set to -1, means to probe for first available.

- CONFIG_FEC[12]_PHY_NORXERR
```

```
                The PHY does not have a RXERR line (RMII only).
                (so program the FEC to ignore it).

- CONFIG_RMII
                Enable RMII mode for all FECs.
                Note that this is a global option, we can't
                have one FEC in standard MII mode and another in RMII mode.

- CONFIG_CRC32_VERIFY
                Add a verify option to the crc32 command.
                The syntax is:

                => crc32 -v <address> <count> <crc32>

                Where address/count indicate a memory area
                and crc32 is the correct crc32 which the
                area should have.

- CONFIG_LOOPW
                Add the "loopw" memory command. This only takes effect if
                the memory commands are activated globally (CONFIG_CMD_MEM).

- CONFIG_MX_CYCLIC
                Add the "mdc" and "mwc" memory commands. These are cyclic
                "md/mw" commands.
                Examples:

                => mdc.b 10 4 500
                This command will print 4 bytes (10,11,12,13) each 500 ms.

                => mwc.l 100 12345678 10
                This command will write 12345678 to address 100 all 10 ms.

                This only takes effect if the memory commands are activated
                globally (CONFIG_CMD_MEM).

- CONFIG_SKIP_LOWLEVEL_INIT
                [ARM, NDS32, MIPS only] If this variable is defined, then certain
                low level initializations (like setting up the memory
                controller) are omitted and/or U-Boot does not
                relocate itself into RAM.

                Normally this variable MUST NOT be defined. The only
                exception is when U-Boot is loaded (to RAM) by some
                other boot loader or by a debugger which performs
                these initializations itself.

- CONFIG_SPL_BUILD
                Modifies the behaviour of start.S when compiling a loader
                that is executed before the actual U-Boot. E.g. when
                compiling a NAND SPL.

- CONFIG_TPL_BUILD
                Modifies the behaviour of start.S  when compiling a loader
                that is executed after the SPL and before the actual U-Boot.
                It is loaded by the SPL.

- CONFIG_SYS_MPC85XX_NO_RESETVEC
                Only for 85xx systems. If this variable is specified, the section
                .resetvec is not kept and the section .bootpg is placed in the
                previous 4k of the .text section.

- CONFIG_ARCH_MAP_SYSMEM
                Generally U-Boot (and in particular the md command) uses
                effective address. It is therefore not necessary to regard
                U-Boot address as virtual addresses that need to be translated
                to physical addresses. However, sandbox requires this, since
```

```
                it maintains its own little RAM buffer which contains all
                addressable memory. This option causes some memory accesses
                to be mapped through map_sysmem() / unmap_sysmem().

- CONFIG_USE_ARCH_MEMCPY
  CONFIG_USE_ARCH_MEMSET
                If these options are used a optimized version of memcpy/memset will
                be used if available. These functions may be faster under some
                conditions but may increase the binary size.

- CONFIG_X86_RESET_VECTOR
                If defined, the x86 reset vector code is included. This is not
                needed when U-Boot is running from Coreboot.

- CONFIG_SYS_MPUCLK
                Defines the MPU clock speed (in MHz).

                NOTE : currently only supported on AM335x platforms.

- CONFIG_SPL_AM33XX_ENABLE_RTC32K_OSC:
                Enables the RTC32K OSC on AM33xx based plattforms

- CONFIG_SYS_NAND_NO_SUBPAGE_WRITE
                Option to disable subpage write in NAND driver
                driver that uses this:
                drivers/mtd/nand/davinci_nand.c

Freescale QE/FMAN Firmware Support:
-----------------------------------

The Freescale QUICCEngine (QE) and Frame Manager (FMAN) both support the
loading of "firmware", which is encoded in the QE firmware binary format.
This firmware often needs to be loaded during U-Boot booting, so macros
are used to identify the storage device (NOR flash, SPI, etc) and the address
within that device.

- CONFIG_SYS_FMAN_FW_ADDR
        The address in the storage device where the FMAN microcode is located.  The
        meaning of this address depends on which CONFIG_SYS_QE_FW_IN_xxx macro
        is also specified.

- CONFIG_SYS_QE_FW_ADDR
        The address in the storage device where the QE microcode is located.  The
        meaning of this address depends on which CONFIG_SYS_QE_FW_IN_xxx macro
        is also specified.

- CONFIG_SYS_QE_FMAN_FW_LENGTH
        The maximum possible size of the firmware.  The firmware binary format
        has a field that specifies the actual size of the firmware, but it
        might not be possible to read any part of the firmware unless some
        local storage is allocated to hold the entire firmware first.

- CONFIG_SYS_QE_FMAN_FW_IN_NOR
        Specifies that QE/FMAN firmware is located in NOR flash, mapped as
        normal addressable memory via the LBC.  CONFIG_SYS_FMAN_FW_ADDR is the
        virtual address in NOR flash.

- CONFIG_SYS_QE_FMAN_FW_IN_NAND
        Specifies that QE/FMAN firmware is located in NAND flash.
        CONFIG_SYS_FMAN_FW_ADDR is the offset within NAND flash.

- CONFIG_SYS_QE_FMAN_FW_IN_MMC
        Specifies that QE/FMAN firmware is located on the primary SD/MMC
        device.  CONFIG_SYS_FMAN_FW_ADDR is the byte offset on that device.

- CONFIG_SYS_QE_FMAN_FW_IN_SPIFLASH
        Specifies that QE/FMAN firmware is located on the primary SPI
```

```
        device.  CONFIG_SYS_FMAN_FW_ADDR is the byte offset on that device.

- CONFIG_SYS_QE_FMAN_FW_IN_REMOTE
        Specifies that QE/FMAN firmware is located in the remote (master)
        memory space.   CONFIG_SYS_FMAN_FW_ADDR is a virtual address which
        can be mapped from slave TLB->slave LAW->slave SRIO or PCIE outbound
        window->master inbound window->master LAW->the ucode address in
        master's memory space.

Freescale Layerscape Management Complex Firmware Support:
---------------------------------------------------------
The Freescale Layerscape Management Complex (MC) supports the loading of
"firmware".
This firmware often needs to be loaded during U-Boot booting, so macros
are used to identify the storage device (NOR flash, SPI, etc) and the address
within that device.

- CONFIG_FSL_MC_ENET
        Enable the MC driver for Layerscape SoCs.

- CONFIG_SYS_LS_MC_FW_ADDR
        The address in the storage device where the firmware is located.  The
        meaning of this address depends on which CONFIG_SYS_LS_MC_FW_IN_xxx macro
        is also specified.

- CONFIG_SYS_LS_MC_FW_LENGTH
        The maximum possible size of the firmware.  The firmware binary format
        has a field that specifies the actual size of the firmware, but it
        might not be possible to read any part of the firmware unless some
        local storage is allocated to hold the entire firmware first.

- CONFIG_SYS_LS_MC_FW_IN_NOR
        Specifies that MC firmware is located in NOR flash, mapped as
        normal addressable memory via the LBC. CONFIG_SYS_LS_MC_FW_ADDR is the
        virtual address in NOR flash.

Building the Software:
======================

Building U-Boot has been tested in several native build environments
and in many different cross environments. Of course we cannot support
all possibly existing versions of cross development tools in all
(potentially obsolete) versions. In case of tool chain problems we
recommend to use the ELDK (see http://www.denx.de/wiki/DULG/ELDK)
which is extensively used to build and test U-Boot.

If you are not using a native environment, it is assumed that you
have GNU cross compiling tools available in your path. In this case,
you must set the environment variable CROSS_COMPILE in your shell.
Note that no changes to the Makefile or any other source files are
necessary. For example using the ELDK on a 4xx CPU, please enter:

        $ CROSS_COMPILE=ppc_4xx-
        $ export CROSS_COMPILE

Note: If you wish to generate Windows versions of the utilities in
      the tools directory you can use the MinGW toolchain
      (http://www.mingw.org).  Set your HOST tools to the MinGW
      toolchain and execute 'make tools'.  For example:

       $ make HOSTCC=i586-mingw32msvc-gcc HOSTSTRIP=i586-mingw32msvc-strip tools

      Binaries such as tools/mkimage.exe will be created which can
      be executed on computers running Windows.

U-Boot is intended to be simple to build. After installing the
sources you must configure U-Boot for one specific board type. This
```

```
is done by typing:

        make NAME_config

where "NAME_config" is the name of one of the existing configu-
rations; see boards.cfg for supported names.

Note: for some board special configuration names may exist; check if
      additional information is available from the board vendor; for
      instance, the TQM823L systems are available without (standard)
      or with LCD support. You can select such additional "features"
      when choosing the configuration, i. e.

      make TQM823L_config
        - will configure for a plain TQM823L, i. e. no LCD support

      make TQM823L_LCD_config
        - will configure for a TQM823L with U-Boot console on LCD

      etc.


Finally, type "make all", and you should get some working U-Boot
images ready for download to / installation on your system:

- "u-boot.bin" is a raw binary image
- "u-boot" is an image in ELF binary format
- "u-boot.srec" is in Motorola S-Record format

By default the build is performed locally and the objects are saved
in the source directory. One of the two methods can be used to change
this behavior and build U-Boot to some external directory:

1. Add O= to the make command line invocations:

        make O=/tmp/build distclean
        make O=/tmp/build NAME_config
        make O=/tmp/build all

2. Set environment variable BUILD_DIR to point to the desired location:

        export BUILD_DIR=/tmp/build
        make distclean
        make NAME_config
        make all

Note that the command line "O=" setting overrides the BUILD_DIR environment
variable.


Please be aware that the Makefiles assume you are using GNU make, so
for instance on NetBSD you might need to use "gmake" instead of
native "make".


If the system board that you have is not listed, then you will need
to port U-Boot to your hardware platform. To do this, follow these
steps:

1.  Add a new configuration option for your board to the toplevel
    "boards.cfg" file, using the existing entries as examples.
    Follow the instructions there to keep the boards in order.
2.  Create a new directory to hold your board specific code. Add any
    files you need. In your board directory, you will need at least
    the "Makefile", a "<board>.c", "flash.c" and "u-boot.lds".
3.  Create a new configuration file "include/configs/<board>.h" for
    your board
```

```
3.  If you're porting U-Boot to a new CPU, then also create a new
    directory to hold your CPU specific code. Add any files you need.
4.  Run "make <board>_config" with your new name.
5.  Type "make", and you should get a working "u-boot.srec" file
    to be installed on your target system.
6.  Debug and solve any problems that might arise.
    [Of course, this last step is much harder than it sounds.]


Testing of U-Boot Modifications, Ports to New Hardware, etc.:
==============================================================

If you have modified U-Boot sources (for instance added a new board
or support for new devices, a new CPU, etc.) you are expected to
provide feedback to the other developers. The feedback normally takes
the form of a "patch", i. e. a context diff against a certain (latest
official or latest in the git repository) version of U-Boot sources.

But before you submit such a patch, please verify that your modifi-
cation did not break existing code. At least make sure that *ALL* of
the supported boards compile WITHOUT ANY compiler warnings. To do so,
just run the "MAKEALL" script, which will configure and build U-Boot
for ALL supported system. Be warned, this will take a while. You can
select which (cross) compiler to use by passing a `CROSS_COMPILE'
environment variable to the script, i. e. to use the ELDK cross tools
you can type

        CROSS_COMPILE=ppc_8xx- MAKEALL

or to build on a native PowerPC system you can type

        CROSS_COMPILE=' ' MAKEALL

When using the MAKEALL script, the default behaviour is to build
U-Boot in the source directory. This location can be changed by
setting the BUILD_DIR environment variable. Also, for each target
built, the MAKEALL script saves two log files (<target>.ERR and
<target>.MAKEALL) in the <source dir>/LOG directory. This default
location can be changed by setting the MAKEALL_LOGDIR environment
variable. For example:

        export BUILD_DIR=/tmp/build
        export MAKEALL_LOGDIR=/tmp/log
        CROSS_COMPILE=ppc_8xx- MAKEALL

With the above settings build objects are saved in the /tmp/build,
log files are saved in the /tmp/log and the source tree remains clean
during the whole build process.


See also "U-Boot Porting Guide" below.


Monitor Commands - Overview:
============================

go      - start application at address 'addr'
run     - run commands in an environment variable
bootm   - boot application image from memory
bootp   - boot image via network using BootP/TFTP protocol
bootz   - boot zImage from memory
tftpboot- boot image via network using TFTP protocol
               and env variables "ipaddr" and "serverip"
               (and eventually "gatewayip")
tftpput - upload a file via network using TFTP protocol
rarpboot- boot image via network using RARP/TFTP protocol
diskboot- boot from IDE devicebootd   - boot default, i.e., run 'bootcmd'
```

```
loads   - load S-Record file over serial line
loadb   - load binary file over serial line (kermit mode)
md      - memory display
mm      - memory modify (auto-incrementing)
nm      - memory modify (constant address)
mw      - memory write (fill)
cp      - memory copy
cmp     - memory compare
crc32   - checksum calculation
i2c     - I2C sub-system
sspi    - SPI utility commands
base    - print or set address offset
printenv- print environment variables
setenv  - set environment variables
saveenv - save environment variables to persistent storage
protect - enable or disable FLASH write protection
erase   - erase FLASH memory
flinfo  - print FLASH memory information
nand    - NAND memory operations (see doc/README.nand)
bdinfo  - print Board Info structure
iminfo  - print header information for application image
coninfo - print console devices and informations
ide     - IDE sub-system
loop    - infinite loop on address range
loopw   - infinite write loop on address range
mtest   - simple RAM test
icache  - enable or disable instruction cache
dcache  - enable or disable data cache
reset   - Perform RESET of the CPU
echo    - echo args to console
version - print monitor version
help    - print online help
?       - alias for 'help'


Monitor Commands - Detailed Description:
========================================

TODO.

For now: just type "help <command>".


Environment Variables:
======================

U-Boot supports user configuration using Environment Variables which
can be made persistent by saving to Flash memory.

Environment Variables are set using "setenv", printed using
"printenv", and saved to Flash using "saveenv". Using "setenv"
without a value can be used to delete a variable from the
environment. As long as you don't save the environment you are
working with an in-memory copy. In case the Flash area containing the
environment is erased by accident, a default environment is provided.

Some configuration options can be set using Environment Variables.

List of environment variables (most likely not complete):

  baudrate      - see CONFIG_BAUDRATE

  bootdelay     - see CONFIG_BOOTDELAY

  bootcmd       - see CONFIG_BOOTCOMMAND

  bootargs      - Boot arguments when booting an RTOS image
```

```
bootfile      - Name of the image to load with TFTP

bootm_low     - Memory range available for image processing in the bootm
                command can be restricted. This variable is given as
                a hexadecimal number and defines lowest address allowed
                for use by the bootm command. See also "bootm_size"
                environment variable. Address defined by "bootm_low" is
                also the base of the initial memory mapping for the Linux
                kernel -- see the description of CONFIG_SYS_BOOTMAPSZ and
                bootm_mapsize.

bootm_mapsize - Size of the initial memory mapping for the Linux kernel.
                This variable is given as a hexadecimal number and it
                defines the size of the memory region starting at base
                address bootm_low that is accessible by the Linux kernel
                during early boot.  If unset, CONFIG_SYS_BOOTMAPSZ is used
                as the default value if it is defined, and bootm_size is
                used otherwise.

bootm_size    - Memory range available for image processing in the bootm
                command can be restricted. This variable is given as
                a hexadecimal number and defines the size of the region
                allowed for use by the bootm command. See also "bootm_low"
                environment variable.

updatefile    - Location of the software update file on a TFTP server, used
                by the automatic software update feature. Please refer to
                documentation in doc/README.update for more details.

autoload      - if set to "no" (any string beginning with 'n'),
                "bootp" will just load perform a lookup of the
                configuration from the BOOTP server, but not try to
                load any image using TFTP

autostart     - if set to "yes", an image loaded using the "bootp",
                "rarpboot", "tftpboot" or "diskboot" commands will
                be automatically started (by internally calling
                "bootm")

                If set to "no", a standalone image passed to the
                "bootm" command will be copied to the load address
                (and eventually uncompressed), but NOT be started.
                This can be used to load and uncompress arbitrary
                data.

fdt_high      - if set this restricts the maximum address that the
                flattened device tree will be copied into upon boot.
                For example, if you have a system with 1 GB memory
                at physical address 0x10000000, while Linux kernel
                only recognizes the first 704 MB as low memory, you
                may need to set fdt_high as 0x3C000000 to have the
                device tree blob be copied to the maximum address
                of the 704 MB low memory, so that Linux kernel can
                access it during the boot procedure.

                If this is set to the special value 0xFFFFFFFF then
                the fdt will not be copied at all on boot.  For this
                to work it must reside in writable memory, have
                sufficient padding on the end of it for u-boot to
                add the information it needs into it, and the memory
                must be accessible by the kernel.

fdtcontroladdr- if set this is the address of the control flattened
                device tree used by U-Boot when CONFIG_OF_CONTROL is
                defined.
```

```
i2cfast       - (PPC405GP|PPC405EP only)
                if set to 'y' configures Linux I2C driver for fast
                mode (400kHZ). This environment variable is used in
                initialization code. So, for changes to be effective
                it must be saved and board must be reset.

initrd_high   - restrict positioning of initrd images:
                If this variable is not set, initrd images will be
                copied to the highest possible address in RAM; this
                is usually what you want since it allows for
                maximum initrd size. If for some reason you want to
                make sure that the initrd image is loaded below the
                CONFIG_SYS_BOOTMAPSZ limit, you can set this environment
                variable to a value of "no" or "off" or "0".
                Alternatively, you can set it to a maximum upper
                address to use (U-Boot will still check that it
                does not overwrite the U-Boot stack and data).

                For instance, when you have a system with 16 MB
                RAM, and want to reserve 4 MB from use by Linux,
                you can do this by adding "mem=12M" to the value of
                the "bootargs" variable. However, now you must make
                sure that the initrd image is placed in the first
                12 MB as well - this can be done with

                setenv initrd_high 00c00000

                If you set initrd_high to 0xFFFFFFFF, this is an
                indication to U-Boot that all addresses are legal
                for the Linux kernel, including addresses in flash
                memory. In this case U-Boot will NOT COPY the
                ramdisk at all. This may be useful to reduce the
                boot time on your system, but requires that this
                feature is supported by your Linux kernel.

ipaddr        - IP address; needed for tftpboot command

loadaddr      - Default load address for commands like "bootp",
                "rarpboot", "tftpboot", "loadb" or "diskboot"

loads_echo    - see CONFIG_LOADS_ECHO

serverip      - TFTP server IP address; needed for tftpboot command

bootretry     - see CONFIG_BOOT_RETRY_TIME

bootdelaykey  - see CONFIG_AUTOBOOT_DELAY_STR

bootstopkey   - see CONFIG_AUTOBOOT_STOP_STR

ethprime      - controls which interface is used first.

ethact        - controls which interface is currently active.
                For example you can do the following

                => setenv ethact FEC
                => ping 192.168.0.1 # traffic sent on FEC
                => setenv ethact SCC
                => ping 10.0.0.1 # traffic sent on SCC

ethrotate     - When set to "no" U-Boot does not go through all
                available network interfaces.
                It just stays at the currently selected interface.

netretry      - When set to "no" each network operation will
                either succeed or fail without retrying.
                When set to "once" the network operation will
```

```
                  fail when all the available network interfaces
                  are tried once without success.
                  Useful on scripts which control the retry operation
                  themselves.

  npe_ucode     - set load address for the NPE microcode

  silent_linux  - If set then linux will be told to boot silently, by
                  changing the console to be empty. If "yes" it will be
                  made silent. If "no" it will not be made silent. If
                  unset, then it will be made silent if the U-Boot console
                  is silent.

  tftpsrcport   - If this is set, the value is used for TFTP's
                  UDP source port.

  tftpdstport   - If this is set, the value is used for TFTP's UDP
                  destination port instead of the Well Know Port 69.

  tftpblocksize - Block size to use for TFTP transfers; if not set,
                  we use the TFTP server's default block size

  tftptimeout   - Retransmission timeout for TFTP packets (in milli-
                  seconds, minimum value is 1000 = 1 second). Defines
                  when a packet is considered to be lost so it has to
                  be retransmitted. The default is 5000 = 5 seconds.
                  Lowering this value may make downloads succeed
                  faster in networks with high packet loss rates or
                  with unreliable TFTP servers.

  vlan          - When set to a value < 4095 the traffic over
                  Ethernet is encapsulated/received over 802.1q
                  VLAN tagged frames.

The following image location variables contain the location of images
used in booting. The "Image" column gives the role of the image and is
not an environment variable name. The other columns are environment
variable names. "File Name" gives the name of the file on a TFTP
server, "RAM Address" gives the location in RAM the image will be
loaded to, and "Flash Location" gives the image's address in NOR
flash or offset in NAND flash.

*Note* - these variables don't have to be defined for all boards, some
boards currenlty use other variables for these purposes, and some
boards use these variables for other purposes.

Image               File Name        RAM Address       Flash Location
-----               ---------        -----------       --------------
u-boot              u-boot           u-boot_addr_r     u-boot_addr
Linux kernel        bootfile         kernel_addr_r     kernel_addr
device tree blob    fdtfile          fdt_addr_r        fdt_addr
ramdisk             ramdiskfile      ramdisk_addr_r    ramdisk_addr

The following environment variables may be used and automatically
updated by the network boot commands ("bootp" and "rarpboot"),
depending the information provided by your boot server:

  bootfile      - see above
  dnsip         - IP address of your Domain Name Server
  dnsip2        - IP address of your secondary Domain Name Server
  gatewayip     - IP address of the Gateway (Router) to use
  hostname      - Target hostname
  ipaddr        - see above
  netmask       - Subnet Mask
  rootpath      - Pathname of the root filesystem on the NFS server
  serverip      - see above
```

```
There are two special Environment Variables:

  serial#       - contains hardware identification information such
                  as type string and/or serial number
  ethaddr       - Ethernet address

These variables can be set only once (usually during manufacturing of
the board). U-Boot refuses to delete or overwrite these variables
once they have been set once.


Further special Environment Variables:

  ver           - Contains the U-Boot version string as printed
                  with the "version" command. This variable is
                  readonly (see CONFIG_VERSION_VARIABLE).


Please note that changes to some configuration parameters may take
only effect after the next boot (yes, that's just like Windoze :-).


Callback functions for environment variables:
---------------------------------------------

For some environment variables, the behavior of u-boot needs to change
when their values are changed.  This functionailty allows functions to
be associated with arbitrary variables.  On creation, overwrite, or
deletion, the callback will provide the opportunity for some side
effect to happen or for the change to be rejected.

The callbacks are named and associated with a function using the
U_BOOT_ENV_CALLBACK macro in your board or driver code.

These callbacks are associated with variables in one of two ways.  The
static list can be added to by defining CONFIG_ENV_CALLBACK_LIST_STATIC
in the board configuration to a string that defines a list of
associations.  The list must be in the following format:

        entry = variable_name[:callback_name]
        list = entry[,list]

If the callback name is not specified, then the callback is deleted.
Spaces are also allowed anywhere in the list.

Callbacks can also be associated by defining the ".callbacks" variable
with the same list format above.  Any association in ".callbacks" will
override any association in the static list. You can define
CONFIG_ENV_CALLBACK_LIST_DEFAULT to a list (string) to define the
".callbacks" envirnoment variable in the default or embedded environment.


Command Line Parsing:
=====================

There are two different command line parsers available with U-Boot:
the old "simple" one, and the much more powerful "hush" shell:

Old, simple command line parser:
--------------------------------

- supports environment variables (through setenv / saveenv commands)
- several commands on one line, separated by ';'
- variable substitution using "... ${name} ..." syntax
- special characters ('$', ';') can be escaped by prefixing with '\',
  for example:
```

```
        setenv bootcmd bootm \${address}
- You can also escape text by enclosing in single apostrophes, for example:
        setenv addip 'setenv bootargs $bootargs ip=$ipaddr:$serverip:$gatewayip:$netmask:$hostname::off'

Hush shell:
-----------

- similar to Bourne shell, with control structures like
  if...then...else...fi, for...do...done; while...do...done,
  until...do...done, ...
- supports environment ("global") variables (through setenv / saveenv
  commands) and local shell variables (through standard shell syntax
  "name=value"); only environment variables can be used with "run"
  command

General rules:
--------------

(1) If a command line (or an environment variable executed by a "run"
    command) contains several commands separated by semicolon, and
    one of these commands fails, then the remaining commands will be
    executed anyway.

(2) If you execute several variables with one call to run (i. e.
    calling run with a list of variables as arguments), any failing
    command will cause "run" to terminate, i. e. the remaining
    variables are not executed.

Note for Redundant Ethernet Interfaces:
=======================================

Some boards come with redundant Ethernet interfaces; U-Boot supports
such configurations and is capable of automatic selection of a
"working" interface when needed. MAC assignment works as follows:

Network interfaces are numbered eth0, eth1, eth2, ... Corresponding
MAC addresses can be stored in the environment as "ethaddr" (=>eth0),
"eth1addr" (=>eth1), "eth2addr", ...

If the network interface stores some valid MAC address (for instance
in SROM), this is used as default address if there is NO correspon-
ding setting in the environment; if the corresponding environment
variable is set, this overrides the settings in the card; that means:

o If the SROM has a valid MAC address, and there is no address in the
  environment, the SROM's address is used.

o If there is no valid address in the SROM, and a definition in the
  environment exists, then the value from the environment variable is
  used.

o If both the SROM and the environment contain a MAC address, and
  both addresses are the same, this MAC address is used.

o If both the SROM and the environment contain a MAC address, and the
  addresses differ, the value from the environment is used and a
  warning is printed.

o If neither SROM nor the environment contain a MAC address, an error
  is raised.

If Ethernet drivers implement the 'write_hwaddr' function, valid MAC addresses
will be programmed into hardware as part of the initialization process.  This
may be skipped by setting the appropriate 'ethmacskip' environment variable.
The naming convention is as follows:
"ethmacskip" (=>eth0), "eth1macskip" (=>eth1) etc.
```

```
Image Formats:
==============

U-Boot is capable of booting (and performing other auxiliary operations on)
images in two formats:

New uImage format (FIT)
-----------------------

Flexible and powerful format based on Flattened Image Tree -- FIT (similar
to Flattened Device Tree). It allows the use of images with multiple
components (several kernels, ramdisks, etc.), with contents protected by
SHA1, MD5 or CRC32. More details are found in the doc/uImage.FIT directory.


Old uImage format
-----------------

Old image format is based on binary files which can be basically anything,
preceded by a special header; see the definitions in include/image.h for
details; basically, the header defines the following image properties:

* Target Operating System (Provisions for OpenBSD, NetBSD, FreeBSD,
  4.4BSD, Linux, SVR4, Esix, Solaris, Irix, SCO, Dell, NCR, VxWorks,
  LynxOS, pSOS, QNX, RTEMS, INTEGRITY;
  Currently supported: Linux, NetBSD, VxWorks, QNX, RTEMS, LynxOS,
  INTEGRITY).
* Target CPU Architecture (Provisions for Alpha, ARM, AVR32, Intel x86,
  IA64, MIPS, NDS32, Nios II, PowerPC, IBM S390, SuperH, Sparc, Sparc 64 Bit;
  Currently supported: ARM, AVR32, Intel x86, MIPS, NDS32, Nios II, PowerPC).
* Compression Type (uncompressed, gzip, bzip2)
* Load Address
* Entry Point
* Image Name
* Image Timestamp

The header is marked by a special Magic Number, and both the header
and the data portions of the image are secured against corruption by
CRC32 checksums.


Linux Support:
==============

Although U-Boot should support any OS or standalone application
easily, the main focus has always been on Linux during the design of
U-Boot.

U-Boot includes many features that so far have been part of some
special "boot loader" code within the Linux kernel. Also, any
"initrd" images to be used are no longer part of one big Linux image;
instead, kernel and "initrd" are separate images. This implementation
serves several purposes:

- the same features can be used for other OS or standalone
  applications (for instance: using compressed images to reduce the
  Flash memory footprint)

- it becomes much easier to port new Linux kernel versions because
  lots of low-level, hardware dependent stuff are done by U-Boot

- the same Linux kernel image can now be used with different "initrd"
  images; of course this also means that different kernel images can
  be run with the same "initrd". This makes testing easier (you don't
  have to build a new "zImage.initrd" Linux image when you just
  change a file in your "initrd"). Also, a field-upgrade of the
  software is easier now.
```

```
Linux HOWTO:
============

Porting Linux to U-Boot based systems:
---------------------------------------

U-Boot cannot save you from doing all the necessary modifications to
configure the Linux device drivers for use with your target hardware
(no, we don't intend to provide a full virtual machine interface to
Linux :-).

But now you can ignore ALL boot loader code (in arch/powerpc/mbxboot).

Just make sure your machine specific header file (for instance
include/asm-ppc/tqm8xx.h) includes the same definition of the Board
Information structure as we define in include/asm-<arch>/u-boot.h,
and make sure that your definition of IMAP_ADDR uses the same value
as your U-Boot configuration in CONFIG_SYS_IMMR.

Note that U-Boot now has a driver model, a unified model for drivers.
If you are adding a new driver, plumb it into driver model. If there
is no uclass available, you are encouraged to create one. See
doc/driver-model.


Configuring the Linux kernel:
-----------------------------

No specific requirements for U-Boot. Make sure you have some root
device (initial ramdisk, NFS) for your target system.


Building a Linux Image:
-----------------------

With U-Boot, "normal" build targets like "zImage" or "bzImage" are
not used. If you use recent kernel source, a new build target
"uImage" will exist which automatically builds an image usable by
U-Boot. Most older kernels also have support for a "pImage" target,
which was introduced for our predecessor project PPCBoot and uses a
100% compatible format.

Example:

        make TQM850L_config
        make oldconfig
        make dep
        make uImage

The "uImage" build target uses a special tool (in 'tools/mkimage') to
encapsulate a compressed Linux kernel image with header  information,
CRC32 checksum etc. for use with U-Boot. This is what we are doing:

* build a standard "vmlinux" kernel image (in ELF binary format):

* convert the kernel into a raw binary image:

        ${CROSS_COMPILE}-objcopy -O binary \
                                 -R .note -R .comment \
                                 -S vmlinux linux.bin

* compress the binary image:

        gzip -9 linux.bin
```

```
* package compressed binary image for U-Boot:

        mkimage -A ppc -O linux -T kernel -C gzip \
                -a 0 -e 0 -n "Linux Kernel Image" \
                -d linux.bin.gz uImage


The "mkimage" tool can also be used to create ramdisk images for use
with U-Boot, either separated from the Linux kernel image, or
combined into one file. "mkimage" encapsulates the images with a 64
byte header containing information about target architecture,
operating system, image type, compression method, entry points, time
stamp, CRC32 checksums, etc.

"mkimage" can be called in two ways: to verify existing images and
print the header information, or to build new images.

In the first form (with "-l" option) mkimage lists the information
contained in the header of an existing U-Boot image; this includes
checksum verification:

        tools/mkimage -l image
          -l ==> list image header information

The second form (with "-d" option) is used to build a U-Boot image
from a "data file" which is used as image payload:

        tools/mkimage -A arch -O os -T type -C comp -a addr -e ep \
                      -n name -d data_file image
          -A ==> set architecture to 'arch'
          -O ==> set operating system to 'os'
          -T ==> set image type to 'type'
          -C ==> set compression type 'comp'
          -a ==> set load address to 'addr' (hex)
          -e ==> set entry point to 'ep' (hex)
          -n ==> set image name to 'name'
          -d ==> use image data from 'datafile'

Right now, all Linux kernels for PowerPC systems use the same load
address (0x00000000), but the entry point address depends on the
kernel version:

- 2.2.x kernels have the entry point at 0x0000000C,
- 2.3.x and later kernels have the entry point at 0x00000000.

So a typical call to build a U-Boot image would read:

        -> tools/mkimage -n '2.4.4 kernel for TQM850L' \
        > -A ppc -O linux -T kernel -C gzip -a 0 -e 0 \
        > -d /opt/elsk/ppc_8xx/usr/src/linux-2.4.4/arch/powerpc/coffboot/vmlinux.gz \
        > examples/uImage.TQM850L
        Image Name:   2.4.4 kernel for TQM850L
        Created:      Wed Jul 19 02:34:59 2000
        Image Type:   PowerPC Linux Kernel Image (gzip compressed)
        Data Size:    335725 Bytes = 327.86 kB = 0.32 MB
        Load Address: 0x00000000
        Entry Point:  0x00000000

To verify the contents of the image (or check for corruption):

        -> tools/mkimage -l examples/uImage.TQM850L
        Image Name:   2.4.4 kernel for TQM850L
        Created:      Wed Jul 19 02:34:59 2000
        Image Type:   PowerPC Linux Kernel Image (gzip compressed)
        Data Size:    335725 Bytes = 327.86 kB = 0.32 MB
        Load Address: 0x00000000
        Entry Point:  0x00000000
```

```
NOTE: for embedded systems where boot time is critical you can trade
speed for memory and install an UNCOMPRESSED image instead: this
needs more space in Flash, but boots much faster since it does not
need to be uncompressed:

        -> gunzip /opt/elsk/ppc_8xx/usr/src/linux-2.4.4/arch/powerpc/coffboot/vmlinux.gz
        -> tools/mkimage -n '2.4.4 kernel for TQM850L' \
        > -A ppc -O linux -T kernel -C none -a 0 -e 0 \
        > -d /opt/elsk/ppc_8xx/usr/src/linux-2.4.4/arch/powerpc/coffboot/vmlinux \
        > examples/uImage.TQM850L-uncompressed
        Image Name:   2.4.4 kernel for TQM850L
        Created:      Wed Jul 19 02:34:59 2000
        Image Type:   PowerPC Linux Kernel Image (uncompressed)
        Data Size:    792160 Bytes = 773.59 kB = 0.76 MB
        Load Address: 0x00000000
        Entry Point:  0x00000000


Similar you can build U-Boot images from a 'ramdisk.image.gz' file
when your kernel is intended to use an initial ramdisk:

        -> tools/mkimage -n 'Simple Ramdisk Image' \
        > -A ppc -O linux -T ramdisk -C gzip \
        > -d /LinuxPPC/images/SIMPLE-ramdisk.image.gz examples/simple-initrd
        Image Name:   Simple Ramdisk Image
        Created:      Wed Jan 12 14:01:50 2000
        Image Type:   PowerPC Linux RAMDisk Image (gzip compressed)
        Data Size:    566530 Bytes = 553.25 kB = 0.54 MB
        Load Address: 0x00000000
        Entry Point:  0x00000000

The "dumpimage" is a tool to disassemble images built by mkimage. Its "-i"
option performs the converse operation of the mkimage's second form (the "-d"
option). Given an image built by mkimage, the dumpimage extracts a "data file"
from the image:

        tools/dumpimage -i image -p position data_file
          -i ==> extract from the 'image' a specific 'data_file', \
           indexed by 'position'


Installing a Linux Image:
-------------------------

To downloading a U-Boot image over the serial (console) interface,
you must convert the image to S-Record format:

        objcopy -I binary -O srec examples/image examples/image.srec

The 'objcopy' does not understand the information in the U-Boot
image header, so the resulting S-Record file will be relative to
address 0x00000000. To load it to a given address, you need to
specify the target address as 'offset' parameter with the 'loads'
command.

Example: install the image to address 0x40100000 (which on the
TQM8xxL is in the first Flash bank):

        => erase 40100000 401FFFFF

        .......... done
        Erased 8 sectors

        => loads 40100000
        ## Ready for S-Record download ...
        ~>examples/image.srec
```

```
        1 2 3 4 5 6 7 8 9 10 11 12 13 ...
        ...
        15989 15990 15991 15992
        [file transfer complete]
        [connected]
        ## Start Addr = 0x00000000


You can check the success of the download using the 'iminfo' command;
this includes a checksum verification so you can be sure no data
corruption happened:

        => imi 40100000

        ## Checking Image at 40100000 ...
           Image Name:   2.2.13 for initrd on TQM850L
           Image Type:   PowerPC Linux Kernel Image (gzip compressed)
           Data Size:    335725 Bytes = 327 kB = 0 MB
           Load Address: 00000000
           Entry Point:  0000000c
           Verifying Checksum ... OK


Boot Linux:
-----------

The "bootm" command is used to boot an application that is stored in
memory (RAM or Flash). In case of a Linux kernel image, the contents
of the "bootargs" environment variable is passed to the kernel as
parameters. You can check and modify this variable using the
"printenv" and "setenv" commands:


        => printenv bootargs
        bootargs=root=/dev/ram

        => setenv bootargs root=/dev/nfs rw nfsroot=10.0.0.2:/LinuxPPC nfsaddrs=10.0.0.99:10.0.0.2

        => printenv bootargs
        bootargs=root=/dev/nfs rw nfsroot=10.0.0.2:/LinuxPPC nfsaddrs=10.0.0.99:10.0.0.2

        => bootm 40020000
        ## Booting Linux kernel at 40020000 ...
           Image Name:   2.2.13 for NFS on TQM850L
           Image Type:   PowerPC Linux Kernel Image (gzip compressed)
           Data Size:    381681 Bytes = 372 kB = 0 MB
           Load Address: 00000000
           Entry Point:  0000000c
           Verifying Checksum ... OK
           Uncompressing Kernel Image ... OK
        Linux version 2.2.13 (wd@denx.local.net) (gcc version 2.95.2 19991024 (release)) #1 Wed Jul 19
02:35:17 MEST 2000
        Boot arguments: root=/dev/nfs rw nfsroot=10.0.0.2:/LinuxPPC nfsaddrs=10.0.0.99:10.0.0.2
        time_init: decrementer frequency = 187500000/60
        Calibrating delay loop... 49.77 BogoMIPS
        Memory: 15208k available (700k kernel code, 444k data, 32k init) [c0000000,c1000000]
        ...

If you want to boot a Linux kernel with initial RAM disk, you pass
the memory addresses of both the kernel and the initrd image (PPBCOOT
format!) to the "bootm" command:

        => imi 40100000 40200000

        ## Checking Image at 40100000 ...
           Image Name:   2.2.13 for initrd on TQM850L
           Image Type:   PowerPC Linux Kernel Image (gzip compressed)
```

```
           Data Size:    335725 Bytes = 327 kB = 0 MB
           Load Address: 00000000
           Entry Point:  0000000c
           Verifying Checksum ... OK

        ## Checking Image at 40200000 ...
           Image Name:   Simple Ramdisk Image
           Image Type:   PowerPC Linux RAMDisk Image (gzip compressed)
           Data Size:    566530 Bytes = 553 kB = 0 MB
           Load Address: 00000000
           Entry Point:  00000000
           Verifying Checksum ... OK

        => bootm 40100000 40200000
        ## Booting Linux kernel at 40100000 ...
           Image Name:   2.2.13 for initrd on TQM850L
           Image Type:   PowerPC Linux Kernel Image (gzip compressed)
           Data Size:    335725 Bytes = 327 kB = 0 MB
           Load Address: 00000000
           Entry Point:  0000000c
           Verifying Checksum ... OK
           Uncompressing Kernel Image ... OK
        ## Loading RAMDisk Image at 40200000 ...
           Image Name:   Simple Ramdisk Image
           Image Type:   PowerPC Linux RAMDisk Image (gzip compressed)
           Data Size:    566530 Bytes = 553 kB = 0 MB
           Load Address: 00000000
           Entry Point:  00000000
           Verifying Checksum ... OK
           Loading Ramdisk ... OK
        Linux version 2.2.13 (wd@denx.local.net) (gcc version 2.95.2 19991024 (release)) #1 Wed Jul 19
02:32:08 MEST 2000
        Boot arguments: root=/dev/ram
        time_init: decrementer frequency = 187500000/60
        Calibrating delay loop... 49.77 BogoMIPS
        ...
        RAMDISK: Compressed image found at block 0
        VFS: Mounted root (ext2 filesystem).

        bash#

Boot Linux and pass a flat device tree:
-----------

First, U-Boot must be compiled with the appropriate defines. See the section
titled "Linux Kernel Interface" above for a more in depth explanation. The
following is an example of how to start a kernel and pass an updated
flat device tree:

=> print oftaddr
oftaddr=0x300000
=> print oft
oft=oftrees/mpc8540ads.dtb
=> tftp $oftaddr $oft
Speed: 1000, full duplex
Using TSEC0 device
TFTP from server 192.168.1.1; our IP address is 192.168.1.101
Filename 'oftrees/mpc8540ads.dtb'.
Load address: 0x300000
Loading: #
done
Bytes transferred = 4106 (100a hex)
=> tftp $loadaddr $bootfile
Speed: 1000, full duplex
Using TSEC0 device
TFTP from server 192.168.1.1; our IP address is 192.168.1.2
Filename 'uImage'.
```

```
Load address: 0x200000
Loading:############
done
Bytes transferred = 1029407 (fb51f hex)
=> print loadaddr
loadaddr=200000
=> print oftaddr
oftaddr=0x300000
=> bootm $loadaddr - $oftaddr
## Booting image at 00200000 ...
   Image Name:   Linux-2.6.17-dirty
   Image Type:   PowerPC Linux Kernel Image (gzip compressed)
   Data Size:    1029343 Bytes = 1005.2 kB
   Load Address: 00000000
   Entry Point:  00000000
   Verifying Checksum ... OK
   Uncompressing Kernel Image ... OK
Booting using flat device tree at 0x300000
Using MPC85xx ADS machine description
Memory CAM mapping: CAM0=256Mb, CAM1=256Mb, CAM2=0Mb residual: 0Mb
[snip]


More About U-Boot Image Types:
------------------------------

U-Boot supports the following image types:

   "Standalone Programs" are directly runnable in the environment
        provided by U-Boot; it is expected that (if they behave
        well) you can continue to work in U-Boot after return from
        the Standalone Program.
   "OS Kernel Images" are usually images of some Embedded OS which
        will take over control completely. Usually these programs
        will install their own set of exception handlers, device
        drivers, set up the MMU, etc. - this means, that you cannot
        expect to re-enter U-Boot except by resetting the CPU.
   "RAMDisk Images" are more or less just data blocks, and their
        parameters (address, size) are passed to an OS kernel that is
        being started.
   "Multi-File Images" contain several images, typically an OS
        (Linux) kernel image and one or more data images like
        RAMDisks. This construct is useful for instance when you want
        to boot over the network using BOOTP etc., where the boot
        server provides just a single image file, but you want to get
        for instance an OS kernel and a RAMDisk image.

        "Multi-File Images" start with a list of image sizes, each
        image size (in bytes) specified by an "uint32_t" in network
        byte order. This list is terminated by an "(uint32_t)0".
        Immediately after the terminating 0 follow the images, one by
        one, all aligned on "uint32_t" boundaries (size rounded up to
        a multiple of 4 bytes).

   "Firmware Images" are binary images containing firmware (like
        U-Boot or FPGA images) which usually will be programmed to
        flash memory.

   "Script files" are command sequences that will be executed by
        U-Boot's command interpreter; this feature is especially
        useful when you configure U-Boot to use a real shell (hush)
        as command interpreter.

Booting the Linux zImage:
-------------------------

On some platforms, it's possible to boot Linux zImage. This is done
```

```
using the "bootz" command. The syntax of "bootz" command is the same
as the syntax of "bootm" command.

Note, defining the CONFIG_SUPPORT_RAW_INITRD allows user to supply
kernel with raw initrd images. The syntax is slightly different, the
address of the initrd must be augmented by it's size, in the following
format: "<initrd addres>:<initrd size>".


Standalone HOWTO:
=================

One of the features of U-Boot is that you can dynamically load and
run "standalone" applications, which can use some resources of
U-Boot like console I/O functions or interrupt services.

Two simple examples are included with the sources:

"Hello World" Demo:
-------------------

'examples/hello_world.c' contains a small "Hello World" Demo
application; it is automatically compiled when you build U-Boot.
It's configured to run at address 0x00040004, so you can play with it
like that:

        => loads
        ## Ready for S-Record download ...
        ~>examples/hello_world.srec
        1 2 3 4 5 6 7 8 9 10 11 ...
        [file transfer complete]
        [connected]
        ## Start Addr = 0x00040004

        => go 40004 Hello World! This is a test.
        ## Starting application at 0x00040004 ...
        Hello World
        argc = 7
        argv[0] = "40004"
        argv[1] = "Hello"
        argv[2] = "World!"
        argv[3] = "This"
        argv[4] = "is"
        argv[5] = "a"
        argv[6] = "test."
        argv[7] = "<NULL>"
        Hit any key to exit ...

        ## Application terminated, rc = 0x0

Another example, which demonstrates how to register a CPM interrupt
handler with the U-Boot code, can be found in 'examples/timer.c'.
Here, a CPM timer is set up to generate an interrupt every second.
The interrupt service routine is trivial, just printing a '.'
character, but this is just a demo program. The application can be
controlled by the following keys:

        ? - print current values og the CPM Timer registers
        b - enable interrupts and start timer
        e - stop timer and disable interrupts
        q - quit application

        => loads
        ## Ready for S-Record download ...
        ~>examples/timer.srec
        1 2 3 4 5 6 7 8 9 10 11 ...
        [file transfer complete]
```

```
        [connected]
        ## Start Addr = 0x00040004

        => go 40004
        ## Starting application at 0x00040004 ...
        TIMERS=0xfff00980
        Using timer 1
          tgcr @ 0xfff00980, tmr @ 0xfff00990, trr @ 0xfff00994, tcr @ 0xfff00998, tcn @ 0xfff0099c, ter
@ 0xfff009b0

Hit 'b':
        [q, b, e, ?] Set interval 1000000 us
        Enabling timer
Hit '?':
        [q, b, e, ?] ........
        tgcr=0x1, tmr=0xff1c, trr=0x3d09, tcr=0x0, tcn=0xef6, ter=0x0
Hit '?':
        [q, b, e, ?] .
        tgcr=0x1, tmr=0xff1c, trr=0x3d09, tcr=0x0, tcn=0x2ad4, ter=0x0
Hit '?':
        [q, b, e, ?] .
        tgcr=0x1, tmr=0xff1c, trr=0x3d09, tcr=0x0, tcn=0x1efc, ter=0x0
Hit '?':
        [q, b, e, ?] .
        tgcr=0x1, tmr=0xff1c, trr=0x3d09, tcr=0x0, tcn=0x169d, ter=0x0
Hit 'e':
        [q, b, e, ?] ...Stopping timer
Hit 'q':
        [q, b, e, ?] ## Application terminated, rc = 0x0


Minicom warning:
================

Over time, many people have reported problems when trying to use the
"minicom" terminal emulation program for serial download. I (wd)
consider minicom to be broken, and recommend not to use it. Under
Unix, I recommend to use C-Kermit for general purpose use (and
especially for kermit binary protocol download ("loadb" command), and
use "cu" for S-Record download ("loads" command).  See
http://www.denx.de/wiki/view/DULG/SystemSetup#Section_4.3.
for help with kermit.


Nevertheless, if you absolutely want to use it try adding this
configuration to your "File transfer protocols" section:

           Name    Program                      Name U/D FullScr IO-Red. Multi
        X  kermit  /usr/bin/kermit -i -l %l -s   Y    U    Y       N      N
        Y  kermit  /usr/bin/kermit -i -l %l -r   N    D    Y       N      N


NetBSD Notes:
=============

Starting at version 0.9.2, U-Boot supports NetBSD both as host
(build U-Boot) and target system (boots NetBSD/mpc8xx).

Building requires a cross environment; it is known to work on
NetBSD/i386 with the cross-powerpc-netbsd-1.3 package (you will also
need gmake since the Makefiles are not compatible with BSD make).
Note that the cross-powerpc package does not install include files;
attempting to build U-Boot will fail because <machine/ansi.h> is
missing.  This file has to be installed and patched manually:

        # cd /usr/pkg/cross/powerpc-netbsd/include
        # mkdir powerpc
```

```
        # ln -s powerpc machine
        # cp /usr/src/sys/arch/powerpc/include/ansi.h powerpc/ansi.h
        # ${EDIT} powerpc/ansi.h        ## must remove __va_list, _BSD_VA_LIST

Native builds *don't* work due to incompatibilities between native
and U-Boot include files.

Booting assumes that (the first part of) the image booted is a
stage-2 loader which in turn loads and then invokes the kernel
proper. Loader sources will eventually appear in the NetBSD source
tree (probably in sys/arc/mpc8xx/stand/u-boot_stage2/); in the
meantime, see ftp://ftp.denx.de/pub/u-boot/ppcboot_stage2.tar.gz


Implementation Internals:
=========================

The following is not intended to be a complete description of every
implementation detail. However, it should help to understand the
inner workings of U-Boot and make it easier to port it to custom
hardware.


Initial Stack, Global Data:
---------------------------

The implementation of U-Boot is complicated by the fact that U-Boot
starts running out of ROM (flash memory), usually without access to
system RAM (because the memory controller is not initialized yet).
This means that we don't have writable Data or BSS segments, and BSS
is not initialized as zero. To be able to get a C environment working
at all, we have to allocate at least a minimal stack. Implementation
options for this are defined and restricted by the CPU used: Some CPU
models provide on-chip memory (like the IMMR area on MPC8xx and
MPC826x processors), on others (parts of) the data cache can be
locked as (mis-) used as memory, etc.

        Chris Hallinan posted a good summary of these issues to the
        U-Boot mailing list:

        Subject: RE: [U-Boot-Users] RE: More On Memory Bank x (nothingness)?
        From: "Chris Hallinan" <clh@net1plus.com>
        Date: Mon, 10 Feb 2003 16:43:46 -0500 (22:43 MET)
        ...

        Correct me if I'm wrong, folks, but the way I understand it
        is this: Using DCACHE as initial RAM for Stack, etc, does not
        require any physical RAM backing up the cache. The cleverness
        is that the cache is being used as a temporary supply of
        necessary storage before the SDRAM controller is setup. It's
        beyond the scope of this list to explain the details, but you
        can see how this works by studying the cache architecture and
        operation in the architecture and processor-specific manuals.

        OCM is On Chip Memory, which I believe the 405GP has 4K. It
        is another option for the system designer to use as an
        initial stack/RAM area prior to SDRAM being available. Either
        option should work for you. Using CS 4 should be fine if your
        board designers haven't used it for something that would
        cause you grief during the initial boot! It is frequently not
        used.

        CONFIG_SYS_INIT_RAM_ADDR should be somewhere that won't interfere
        with your processor/board/system design. The default value
        you will find in any recent u-boot distribution in
        walnut.h should work for you. I'd set it to a value larger
        than your SDRAM module. If you have a 64MB SDRAM module, set
```

```
        it above 400_0000. Just make sure your board has no resources
        that are supposed to respond to that address! That code in
        start.S has been around a while and should work as is when
        you get the config right.

        -Chris Hallinan
        DS4.COM, Inc.

It is essential to remember this, since it has some impact on the C
code for the initialization procedures:

* Initialized global data (data segment) is read-only. Do not attempt
  to write it.

* Do not use any uninitialized global data (or implicitely initialized
  as zero data - BSS segment) at all - this is undefined, initiali-
  zation is performed later (when relocating to RAM).

* Stack space is very limited. Avoid big data buffers or things like
  that.

Having only the stack as writable memory limits means we cannot use
normal global data to share information beween the code. But it
turned out that the implementation of U-Boot can be greatly
simplified by making a global data structure (gd_t) available to all
functions. We could pass a pointer to this data as argument to _all_
functions, but this would bloat the code. Instead we use a feature of
the GCC compiler (Global Register Variables) to share the data: we
place a pointer (gd) to the global data into a register which we
reserve for this purpose.

When choosing a register for such a purpose we are restricted by the
relevant  (E)ABI  specifications for the current architecture, and by
GCC's implementation.

For PowerPC, the following registers have specific use:
        R1:     stack pointer
        R2:     reserved for system use
        R3-R4:  parameter passing and return values
        R5-R10: parameter passing
        R13:    small data area pointer
        R30:    GOT pointer
        R31:    frame pointer

        (U-Boot also uses R12 as internal GOT pointer. r12
        is a volatile register so r12 needs to be reset when
        going back and forth between asm and C)

    ==> U-Boot will use R2 to hold a pointer to the global data

    Note: on PPC, we could use a static initializer (since the
    address of the global data structure is known at compile time),
    but it turned out that reserving a register results in somewhat
    smaller code - although the code savings are not that big (on
    average for all boards 752 bytes for the whole U-Boot image,
    624 text + 127 data).

On Blackfin, the normal C ABI (except for P3) is followed as documented here:
        http://docs.blackfin.uclinux.org/doku.php?id=application_binary_interface

    ==> U-Boot will use P3 to hold a pointer to the global data

On ARM, the following registers are used:

        R0:     function argument word/integer result
        R1-R3:  function argument word
        R9:     platform specific
```

```
        R10:    stack limit (used only if stack checking is enabled)
        R11:    argument (frame) pointer
        R12:    temporary workspace
        R13:    stack pointer
        R14:    link register
        R15:    program counter

    ==> U-Boot will use R9 to hold a pointer to the global data

    Note: on ARM, only R_ARM_RELATIVE relocations are supported.

On Nios II, the ABI is documented here:
        http://www.altera.com/literature/hb/nios2/n2cpu_nii51016.pdf

    ==> U-Boot will use gp to hold a pointer to the global data

    Note: on Nios II, we give "-G0" option to gcc and don't use gp
    to access small data sections, so gp is free.

On NDS32, the following registers are used:

        R0-R1:  argument/return
        R2-R5:  argument
        R15:    temporary register for assembler
        R16:    trampoline register
        R28:    frame pointer (FP)
        R29:    global pointer (GP)
        R30:    link register (LP)
        R31:    stack pointer (SP)
        PC:     program counter (PC)

    ==> U-Boot will use R10 to hold a pointer to the global data

NOTE: DECLARE_GLOBAL_DATA_PTR must be used with file-global scope,
or current versions of GCC may "optimize" the code too much.

Memory Management:
------------------

U-Boot runs in system state and uses physical addresses, i.e. the
MMU is not used either for address mapping nor for memory protection.

The available memory is mapped to fixed addresses using the memory
controller. In this process, a contiguous block is formed for each
memory type (Flash, SDRAM, SRAM), even when it consists of several
physical memory banks.

U-Boot is installed in the first 128 kB of the first Flash bank (on
TQM8xxL modules this is the range 0x40000000 ... 0x4001FFFF). After
booting and sizing and initializing DRAM, the code relocates itself
to the upper end of DRAM. Immediately below the U-Boot code some
memory is reserved for use by malloc() [see CONFIG_SYS_MALLOC_LEN
configuration setting]. Below that, a structure with global Board
Info data is placed, followed by the stack (growing downward).

Additionally, some exception handler code is copied to the low 8 kB
of DRAM (0x00000000 ... 0x00001FFF).

So a typical memory configuration with 16 MB of DRAM could look like
this:

        0x0000 0000     Exception Vector code
              :
        0x0000 1FFF
        0x0000 2000     Free for Application Use
              :
              :
```

```
              :
              :
        0x00FB FF20     Monitor Stack (Growing downward)
        0x00FB FFAC     Board Info Data and permanent copy of global data
        0x00FC 0000     Malloc Arena
              :
        0x00FD FFFF
        0x00FE 0000     RAM Copy of Monitor Code
        ...             eventually: LCD or video framebuffer
        ...             eventually: pRAM (Protected RAM - unchanged by reset)
        0x00FF FFFF     [End of RAM]


System Initialization:
----------------------

In the reset configuration, U-Boot starts at the reset entry point
(on most PowerPC systems at address 0x00000100). Because of the reset
configuration for CS0# this is a mirror of the onboard Flash memory.
To be able to re-map memory U-Boot then jumps to its link address.
To be able to implement the initialization code in C, a (small!)
initial stack is set up in the internal Dual Ported RAM (in case CPUs
which provide such a feature like MPC8xx or MPC8260), or in a locked
part of the data cache. After that, U-Boot initializes the CPU core,
the caches and the SIU.

Next, all (potentially) available memory banks are mapped using a
preliminary mapping. For example, we put them on 512 MB boundaries
(multiples of 0x20000000: SDRAM on 0x00000000 and 0x20000000, Flash
on 0x40000000 and 0x60000000, SRAM on 0x80000000). Then UPM A is
programmed for SDRAM access. Using the temporary configuration, a
simple memory test is run that determines the size of the SDRAM
banks.

When there is more than one SDRAM bank, and the banks are of
different size, the largest is mapped first. For equal size, the first
bank (CS2#) is mapped first. The first mapping is always for address
0x00000000, with any additional banks following immediately to create
contiguous memory starting from 0.

Then, the monitor installs itself at the upper end of the SDRAM area
and allocates memory for use by malloc() and for the global Board
Info data; also, the exception vector code is copied to the low RAM
pages, and the final stack is set up.

Only after this relocation will you have a "normal" C environment;
until that you are restricted in several ways, mostly because you are
running from ROM, and because the code will have to be relocated to a
new address in RAM.


U-Boot Porting Guide:
----------------------

[Based on messages by Jerry Van Baren in the U-Boot-Users mailing
list, October 2002]


int main(int argc, char *argv[])
{
        sighandler_t no_more_time;

        signal(SIGALRM, no_more_time);
        alarm(PROJECT_DEADLINE - toSec (3 * WEEK));

        if (available_money > available_manpower) {
```

```
                Pay consultant to port U-Boot;
                return 0;
        }

        Download latest U-Boot source;

        Subscribe to u-boot mailing list;

        if (clueless)
                email("Hi, I am new to U-Boot, how do I get started?");

        while (learning) {
                Read the README file in the top level directory;
                Read http://www.denx.de/twiki/bin/view/DULG/Manual;
                Read applicable doc/*.README;
                Read the source, Luke;
                /* find . -name "*.[chS]" | xargs grep -i <keyword> */
        }

        if (available_money > toLocalCurrency ($2500))
                Buy a BDI3000;
        else
                Add a lot of aggravation and time;

        if (a similar board exists) {   /* hopefully... */
                cp -a board/<similar> board/<myboard>
                cp include/configs/<similar>.h include/configs/<myboard>.h
        } else {
                Create your own board support subdirectory;
                Create your own board include/configs/<myboard>.h file;
        }
        Edit new board/<myboard> files
        Edit new include/configs/<myboard>.h

        while (!accepted) {
                while (!running) {
                        do {
                                Add / modify source code;
                        } until (compiles);
                        Debug;
                        if (clueless)
                                email("Hi, I am having problems...");
                }
                Send patch file to the U-Boot email list;
                if (reasonable critiques)
                        Incorporate improvements from email list code review;
                else
                        Defend code as written;
        }

        return 0;
}

void no_more_time (int sig)
{
      hire_a_guru();
}


Coding Standards:
-----------------

All contributions to U-Boot should conform to the Linux kernel
coding style; see the file "Documentation/CodingStyle" and the script
"scripts/Lindent" in your Linux kernel source directory.

Source files originating from a different project (for example the
```

```
MTD subsystem) are generally exempt from these guidelines and are not
reformated to ease subsequent migration to newer versions of those
sources.

Please note that U-Boot is implemented in C (and to some small parts in
Assembler); no C++ is used, so please do not use C++ style comments (//)
in your code.

Please also stick to the following formatting rules:
- remove any trailing white space
- use TAB characters for indentation and vertical alignment, not spaces
- make sure NOT to use DOS '\r\n' line feeds
- do not add more than 2 consecutive empty lines to source files
- do not add trailing empty lines to source files

Submissions which do not conform to the standards may be returned
with a request to reformat the changes.


Submitting Patches:
-------------------

Since the number of patches for U-Boot is growing, we need to
establish some rules. Submissions which do not conform to these rules
may be rejected, even when they contain important and valuable stuff.

Please see http://www.denx.de/wiki/U-Boot/Patches for details.

Patches shall be sent to the u-boot mailing list <u-boot@lists.denx.de>;
see http://lists.denx.de/mailman/listinfo/u-boot

When you send a patch, please include the following information with
it:

* For bug fixes: a description of the bug and how your patch fixes
  this bug. Please try to include a way of demonstrating that the
  patch actually fixes something.

* For new features: a description of the feature and your
  implementation.

* A CHANGELOG entry as plaintext (separate from the patch)

* For major contributions, your entry to the CREDITS file

* When you add support for a new board, don't forget to add a
  maintainer e-mail address to the boards.cfg file, too.

* If your patch adds new configuration options, don't forget to
  document these in the README file.

* The patch itself. If you are using git (which is *strongly*
  recommended) you can easily generate the patch using the
  "git format-patch". If you then use "git send-email" to send it to
  the U-Boot mailing list, you will avoid most of the common problems
  with some other mail clients.

  If you cannot use git, use "diff -purN OLD NEW". If your version of
  diff does not support these options, then get the latest version of
  GNU diff.

  The current directory when running this command shall be the parent
  directory of the U-Boot source tree (i. e. please make sure that
  your patch includes sufficient directory information for the
  affected files).

  We prefer patches as plain text. MIME attachments are discouraged,
```

```
  and compressed attachments must not be used.

* If one logical set of modifications affects or creates several
  files, all these changes shall be submitted in a SINGLE patch file.

* Changesets that contain different, unrelated modifications shall be
  submitted as SEPARATE patches, one patch per changeset.


Notes:

* Before sending the patch, run the MAKEALL script on your patched
  source tree and make sure that no errors or warnings are reported
  for any of the boards.

* Keep your modifications to the necessary minimum: A patch
  containing several unrelated changes or arbitrary reformats will be
  returned with a request to re-formatting / split it.

* If you modify existing code, make sure that your new code does not
  add to the memory footprint of the code ;-) Small is beautiful!
  When adding new features, these should compile conditionally only
  (using #ifdef), and the resulting code with the new feature
  disabled must not need more memory than the old code without your
  modification.

* Remember that there is a size limit of 100 kB per message on the
  u-boot mailing list. Bigger patches will be moderated. If they are
  reasonable and not too big, they will be acknowledged. But patches
  bigger than the size limit should be avoided.
```