TI81XX PSP U-Boot

DM81XX PSP U-Boot

Linux PSP

IMPORTANT

TI81xx refers to DM816x, DM814x and DM813X.

Read This First

This section show the major updates since last release, please review this section before any SW activities.

DM8168 & DM8148

- We have migrated from "legacy" mmc driver to a "generic" mmc driver onwards PSP_04.00.00.12/PSP_04.01.00.05 u-boot release. The new mmc driver has a different set of commands.This will break the existing boot script used. Please refer the U-boot user guide and update you scripts according to that.

Example

```
$mmc init must be replaced with $mmc rescan 0
```

The mmc channel number must be “0” in all commands instead of “1”

```
$fatls mmc 1 must be replaced with $fatls mmc 0
$fatload mmc 1 <address> <file> must be replaced with $fatls mmc 0 <address> <file>
```

- Quick-boot support from MMC/SD is added in this release. Please follow the steps in the user guide to build a quick-boot binaries. Refer DM81XX Quick Boot Measurements.

TI813X

U-Boot

In DM81xx the ROM code serves as the first stage bootloader. The 2nd stage bootloader is based on U-Boot [2]. It does the bare minimum configuration needed to boot the kernel and in the process relocates itself to DDR.

DM816X In case of NAND/SPI/SD boot mode, the ROM code loads the U-Boot image from the flash memory into OCMC RAM1. In case of NOR boot mode, the ROM code passes the control to the U-Boot image in NOR memory after some basic initialization.

Size of U-Boot (TEXT + BSS section) cannot exceed 255KB

Note

To avoid over-writing U-Boot when it is executing make sure that an address greater than 0x80800000 is used when downloading the kernel and/or filesystem images

DM814X/DM813X The ROM code loads the U-Boot-MIN image from the flash memory/SD card into OCMC RAM0. U-Boot-MIN does the basic configuration like setting up the clocks, initializing the necessary peripherals and then loads the U-Boot image from NAND,SPI/SD or UART.

Note

1. Please note the difference in the U-Boot prompt for the 2 stages. The 1st stage prompt is TI-MIN# while that for the 2nd stage is TI8148_EVM#. The prompt specified in the commands is meant to serve as an indication of which stage it should be executed from. Most of the commands mentioned in the doc are meant to be executed from the 2nd stage.

2. NOR boot has only two stages of bootloader. ROM code serves as the Ist stage bootloader and the u-boot serves as the second stage boot loader. U-boot min is not required here as the NOR is XIP and hence the complete u-boot binary can be placed in the NOR flash.

Two stage U-Boot design

This section gives an overview of the two stage U-Boot approach adopted for DM814x and DM385.

The size of the internal RAM in TI814x is 128KB out of which 18KB at the end is used by the ROM code. This placed a limit of 110KB on the size of the U-Boot binary which the ROM code can transfer to the internal RAM.

U-Boot also requires some space for the stack, heap and global data during executing and this region currently needs to be setup **before** the TEXT_BASE of U-Boot.

Since it is not possible to squeeze in all the functionality that is normally expected from U-Boot in < 110KB (after setting aside some space for stack, heap etc) a two stage approach has been adopted.

The first stage (or rather the 2nd stage if the ROM code is also considered) is built using a minimal configuration and has an embedded ENV. The purpose of this stage is to initialize the necessary peripherals, especially DDR, so that a full fledged U-Boot can be transferred to DDR from NAND/SPI/SD/UART/NOR and then control passed to it.

The boot command that the minimal U-Boot uses for fetching the larger U-Boot will not change for a particular boot mode as long as the same layout is used and is hence fixed. If any other behavior required then the minimal U-Boot stage can be interrupted and a different command issued.

The minimal U-Boot binary also has a hole at the top which is used as the space for stack, heap and global data. After some analysis the size of the hole has currently been set as 12KB.

Note

To avoid over-writing U-Boot when it is executing make sure that an address greater than 0x80800000 is used when downloading the kernel and/or filesystem images

U-Boot Support for DDR2/DDR3 Boards

DM814x/DM813x

The u-boot will automatically select the DDR2/DDR3 configurations at run-time. DM813x only supports DDR3 EVM's.

Automatic DDR2/DDR3 Selection

U-Boot will detect the PG version at run-time and will select DDR2/DDR3 automatically.We will assume that all PG1.0 Version uses DDR2 and PG2.1 will use DDR3.

Note

To use PG2.1 with DDR2 support, un-comment the macro "#define CONFIG_TI814X_EVM_DDR2" in file "include/configs/ti8148_evm.h" that will forcefully select DDR2 support.

DM816x

The pre-built U-Boot binaries that comes in the release package has been built for DDR3 based EVM and the configuration is for **DDR3@400MHz** I/O clk. The binaries MLO and u-boot.bin are meant for SD boot and u-boot.noxip.bin is for flashing to NAND. If you have a DDR3 based EVM and wish to use the pre-built U-Boot

image for flashing to NAND or if you want to use SD boot proceed to the Flashing U-Boot section

Modifying U-Boot for DDR2

This section described how to modify the U-Boot source to enable DDR2 support.

- Open the file *arch/arm/include/asm/arch-ti81xx/clocks_ti816x.h* and modify the #define as shown below

```
#define DDR_PLL_400     /* Values supported 400,531,675,796 */
```

- Open the file *include/configs/ti8168_evm.h* and make sure that the config option for DDR2 is not commented and that for DDR3 is commented out as shown below

```
[...]
//#define CONFIG_TI816X_EVM_DDR3                  /* Configure DDR3 in U-Boot */
#define CONFIG_TI816X_EVM_DDR2                /* Configure DDR2 in U-Boot */
[...]
```

Modifying U-Boot DDR3 frequency

This section described how to modify the U-Boot source to change the DDR3 frequency on DM816x EVMs. Currently the following are the supported frequencies on the DM816x EVM.

- 400MHz
- 531MHz
- 675MHz
- 796MHz

The default configuration is for DDR3@796MHz.

- Open the file *arch/arm/include/asm/arch-ti81xx/ddr_defs.h* and modify the number in the #define shown below to the required frequency.

```
#define CONFIG_TI8168_DDR3_796     /* Values supported 400,531,675,796 */
```

- Open the file *arch/arm/include/asm/arch-ti81xx/clocks_ti816x.h* and modify the number in the #define shown below to the required frequency. The number used here should be the same as the that in the previous step.

```
#define DDR_PLL_796     /* Values supported 400,531,675,796 */
```

- Open the file *include/configs/ti8168_evm.h* and make sure that the config option for DDR3 is not commented and that for DDR2 is commented out as shown below

```
[...]
#define CONFIG_TI816X_EVM_DDR3                  /* Configure DDR3 in U-Boot */
//#define CONFIG_TI816X_EVM_DDR2                /* Configure DDR2 in U-Boot */
[...]
```

Once these changes are done, recompile U-Boot as described in this User-Guide.

Building U-Boot

U-Boot Configurations

1st stage u-boot configurations

Boot Modes	Output File	TI816x	TI814x	TI813x(DM385)
NAND	**u-boot.min.nand**	single stage*	ti8148_evm_min_nand	dm385_evm_min_nand
SPI	**u-boot.min.spi**	single stage*	ti8148_evm_min_spi	dm385_evm_min_spi
NOR	NA	single stage*	Not supported	Not supported
SD	**u-boot.min.sd**	ti8168_evm_min_sd	ti8148_evm_min_sd	dm385_evm_min_sd
UART	**u-boot.min.uart**	Not supported	ti8148_evm_min_uart	dm385_evm_min_uart
EMAC	**u-boot.min.eth**	single stage*	ti8148_evm_min_eth	dm385_evm_min_eth

```
* Single stage u-boot support,Please refer 2nd stage configs
```

2nd stage u-boot configurations

Boot Modes	Output File	TI816x	TI814x	TI813x(DM385)
NAND	**u-boot.bin.noxip (TI816x)**	ti8168_evm_config_nand	ti8148_evm_config_nand	dm385_evm_config_nand
	u-boot.bin (TI814x/Ti813x)			
SPI	**u-boot.bin.noxip.spi**	ti8168_evm_config_spi	ti8148_evm_config_spi	dm385_evm_config_spi
NOR	**u-boot.bin**	ti8168_evm_config_nor	Not supported	Not supported
SD**	**u-boot.bin**	ti8168_evm_config_sd	ti8148_evm_config_sd	dm385_evm_config_sd
UART	NA	Not supported	use NAND/SPI/SD	use NAND/SPI/SD
EMAC	u-boot.in	use NAND/SPI/SD	use NAND/SPI/SD	use NAND/SPI/SD

```
** Environment variable in MMC/SD support and the build configurations is only applicable to
PSP_04.01.00.07/PSP_04.00.02.13 or higher Release.Older versions of u-boot must use NAND or SPI images for 2nd stage SD.
```

U-Boot supports saving environment variables to NAND, SPI, NOR or MMC. The media to be used for saving the ENV variables needs to be decided before building U-Boot. The default behavior is to place the ENV section of U-Boot (where the environment variables are stored) in the same media for which U-Boot was built.

Building 1st stage u-boot

The first stage u-boot images except SD boot are only applicable to TI814X series .

Different boot modes have different requirements for the U-Boot-MIN stage.

- When using memory boot (NAND/SPI) a header needs to be attached to the U-Boot binary indicating the the load address and the size of the image. SPI boot also requires endian conversion before flashing the image.
- When using peripheral boot (UART,EMAC) there can be no header as the load address is fixed.
- Header is not required in case of XIP (NOR)

For easy generation of the different images, different target names have been provided in the Makefile. The different target names take care of building the appropriate image (with or without the header and with or without endian conversion).

Please note that the U-Boot-MIN stage uses a pre-defined command to load the 2nd stage and does not support commands such as dhcp, tftp, saveenv etc. All the commands are available only from the 2nd stage of U-Boot which the 1st stage U-Boot autoloads (if present).

```
$ make CROSS_COMPILE=arm-none-linux-gnueabi- ARCH=arm distclean
$ make CROSS_COMPILE=arm-none-linux-gnueabi- ARCH=arm <1st stage config from table *>
$ make CROSS_COMPILE=arm-none-linux-gnueabi- ARCH=arm u-boot.ti
```

- example ti8148_evm_min_nand , dm385_evm_min_sd

Building 2nd stage u-boot

NOR boot mode is only applicable for TI816x

The 2nd stage of U-Boot supports saving environment variables to NAND, SPI or SD Card. The media to be used stored for saving the ENV variables needs to be decided before building the 2nd stage of U-Boot.

```
$ make CROSS_COMPILE=arm-none-linux-gnueabi- ARCH=arm distclean
$ make CROSS_COMPILE=arm-none-linux-gnueabi- ARCH=arm <2nd stage config from the table*>
$ make CROSS_COMPILE=arm-none-linux-gnueabi- ARCH=arm u-boot.ti
```

- example ti8168_evm_config_nand , dm385_evm_config_sd

U-Boot quick-boot configurations for SD

Quick boot configurations

Boot Modes	Output File	TI816x	TI814x	TI813x(DM385)
1st stage SD	**u-boot.min.sd**	ti8168_evm_config_quick_mmc_min	ti8148_evm_config_quick_mmc_min	NA
2nd stage SD	**u-boot.bin**	ti8168_evm_config_quick_mmc	ti8148_evm_config_quick_mmc	NA

```
$ make CROSS_COMPILE=arm-none-linux-gnueabi- ARCH=arm distclean
$ make CROSS_COMPILE=arm-none-linux-gnueabi- ARCH=arm <1/2nd stage config from table>
$ make CROSS_COMPILE=arm-none-linux-gnueabi- ARCH=arm u-boot.ti
```

Quick-boot enable a faster booting using the SD card images . Quick boot removed the boot delay and suppress the console message from the u-boot. It uses a separate configuration file "include/configs/ti8168_config_quick_mmc.h" for building the quick-boot images.Please follow the steps to build 1st stage quick-boot images for mmc.

The 2nd stage quick-boot images for mmc is generated from the same quick-boot config file "include/configs/ti8168_config_quick_mmc.h". The u-boot image will load the kernel from the sd card.The env "quite" is passed by default from the u-boot . Normal messages about hardware detection at boot are suppressed ,only important and system critical kernel messages are printed to the console. It also avoid the boot delay by setting it to 0 .

This will generate binaries named **u-boot.noxip.bin** and **u-boot.bin**. Use u-boot.bin for boot from mmc.

Refer this DM81XX_Quick_Boot_Measurements for more information.

Serial port configuration

Connect a serial cable from the serial port of the EVM (there are two serial ports on the EVM, use the one which is next to the MMC/SD slot) to the COM port on either the Windows machine or Linux host depending on where you'll be running the serial terminal software.

For correct operation the serial terminal software should be configured with the following settings:

```
* Baud rate: 115,200
* Data bits: 8
* Parity: None
* Stop bits: 1
* Flow control: None
* Transmit delay: 0 msec/char, 100 msec/line
```

If Teraterm is being used ensure that the latest version (4.67) of Teraterm is installed. The implementation of the kermit protocol in Teraterm is not reliable in older versions. The latest version of Teraterm 4.67 can be downloaded from here [3]

EVM Switch Settings

	SW-1/SW-3					SW-2/SW-4	
Boot Mode	**BTM 4**	**BTM 3**	**BTM 2**	**BTM 1**	**BTM 0**	**NAND**	**SPI**
NAND	1	0	0	1	0	1	X
SPI	1	0	1	1	0	X	1
UART	1	0	0	0	0	X	X
SD	1	0	1	1	1	X	X
NOR	0	0	0	0	1	0	X
EMAC	0	0	1	0	0	X	X

EMAC ROM Phy Mode selection SW9---> BTM[9:8]

BTMODE[9:8]	PHY Mode
00b	MII/GMII
01b	RMII
10b	RGMII
11b	reserved

Note: SW-3&SW-4 for TI816X and SW-1&SW-2 for TI814X. (Other pins should be 0 i.e. OFF)
DM816X NOR Daughter Card (RevA) SW1---->SW1[3:0] ==> 0001 IODC (RevA) and other Daughter Cards (RevB) SW1---->SW1[3:0] ==> 0011

TI816X

The picture below captures the boot mode configuration switch SW3 on the DM8168 EVM. **NOTE** : The bootmode setting in this picture is for NAND boot.

NAND BOOT MODE SW-3[9:0] 0000010010 SW-4[1:0] 10 [NAND/SPI]

TI814x/DM813x

The picture below captures the boot mode configuration switch SW1 on the DM8148 EVM. **NOTE** : The bootmode setting in this picture is for NAND boot.

NAND BOOT MODE SW-1[11:0] 000000010010 SW-2[1:0] 10 [NAND/SPI]

Make sure that the EVM boot switch settings are set to the required boot mode and then power on the board.

Flashing U-Boot using CCS

DM816X You can flash u-boot onto NAND (for NAND boot), to NOR (NOR Boot) or to SPI (SPI boot).

Refer to Flashing to NAND Flash using CCS wiki page for instructions on how to flash the pre-built U-Boot binary (or the recompiled one) with the help of the NAND flash writer.

Refer to Flashing to NOR Flash using CCS wiki page for instructions on how to flash the pre-built U-Boot binary (or the recompiled one) with the help of the NOR flash writer.

Refer to Flashing to SPI Flash using CCS wiki page for instructions on how to flash the pre-built U-Boot binary (or the recompiled one) with the help of the SPI flash writer.

After flashing the image, configure the EVM switches as per the information provided in EVM switch settings section.

DM814x

PLEASE NOTE: BOTH THE STAGES OF U-BOOT NEED TO BE FLASHED ON THE SAME MEDIA

You can flash the **1st stage** of U-Boot onto NAND (for NAND boot) or to SPI (SPI Boot) using the Flashing Tools provided in the PSP releases.

Refer to Flashing to NAND Flash using CCS wiki page for instructions on how to flash the pre-built (or compiled) binary of the 1st stage for NAND boot i.e. u-boot.min.nand (or the recompiled one) with the help of the NAND flash writer.

Refer to Flashing to SPI Flash using CCS wiki page for instructions on how to flash the pre-built (or compiled) binary of the 1st stage for SPI boot i.e. u-boot.min.spi (or the recompiled one) with the help of the SPI flash writer.

Refer to Flashing to NOR Flash using CCS wiki page for instructions on how to flash the pre-built (or compiled) binary for NOR boot i.e. u-boot.bin (or the recompiled one) with the help of the NOR flash writer.

Note

NOR boot has only two stages of boot loader - ROM boot loader and u-boot. There is no u-boot min binary for NOR as in the case of other boot modes.

Once the 1st stage U-Boot has been flashed using the flashing tool

1. Load the 2nd stage U-Boot over UART as described here.
2. Flash the 2nd stage of U-Boot to either NAND using the steps over here or to SPI using the steps over here.
3. After flashing the 2 stages use the appropriate switch settings to boot from NAND or from SPI.

DM813X Not supported

Flashing U-Boot without CCS

PLEASE NOTE: BOTH THE STAGES OF U-BOOT NEED TO BE FLASHED ON THE SAME MEDIA

It is possible to flash U-Boot from a U-Boot binary already loaded into RAM. This technique can be used for flashing U-Boot in case CCS is not available.

A couple of ways in which this can be achieved is mentioned below.

DM816X

DM816X support SD Flash ans EMAC Flash. Please follow the steps.

SD-Flash-Method

This method involves booting from SD card and then flashing the U-Boot image to NAND or SPI as required.

1. Boot from the SD card using the steps described here
2. Flash the U-Boot binary either to NAND using the steps over here or to SPI using the steps over here.
3. After flashing use the appropriate switch settings to boot from Boot Pin.

EMAC-Flash-Method

This method involves using EMAC boot mode to get to the U-Boot prompt and then flashing the U-Boot image to NAND or SPI as required.

1. Boot over network using the steps described here.
2. Make sure the network configuration has been done as described over here.
3. Flash the U-Boot binary either to NAND using the steps over here or to SPI using the steps over here.
4. After flashing use the appropriate switch settings to boot from Boot Pin.

DM814x/DM813x

DM814X supports Flashing over UART and SD methods. Please follow the steps.

UART-Flash-Method

1. Boot over UART to load the 1st stage of U-Boot as described over here. Make sure that you interrupt the countdown in the 1st stage i.e. at the **TI-MIN#** prompt.
2. Load the 2nd stage of U-Boot over UART as described over here.
3. Flash the 1st stage of U-Boot to either NAND using the steps over here or to SPI using the steps over here.
4. Flash the 2nd stage of U-Boot to either NAND using the steps over here or to SPI using the steps over here.
5. After flashing the 2 stages use the appropriate switch settings to boot from NAND or from SPI.

SD-Flash-Method

Make sure the two binaries for NAND boot (u-boot.min.nand and u-boot.bin) or for SPI boot (u-boot.min.spi and u-boot.bin) which are required for flashing are present on the SD. Please refer to the section of building U-Boot to generate the images.

1. Boot using SD card to load both the 1st stage and the 2nd stage as described over here. Do not interrupt the countdown till the **TI8148_EVM#** prompt comes.
2. Flash the 1st stage of U-Boot to either NAND using the steps over here or to SPI using the steps over here.
3. Flash the 2nd stage of U-Boot to either NAND using the steps over here or to SPI using the steps over here.
4. After flashing the 2 stages use the appropriate switch settings to boot from NAND or from SPI.

U-Boot NAND Support

Before proceeding with any of the commands given in this section please make sure that NAND is enabled on the EVM. The switch for enabling NAND is shown over here.

This section gives an *overview* of the NAND support in U-Boot. It also describe how to store the kernel image, RAMDISK or the JFFS2/UBIFS filesystem to NAND so as to have a network-free boot right from powering on the board to getting the kernel up and running.

Overview

Micron Nand parts (page size 2KB, block size 128KB) are supported on DM81XX EVM platforms.

Note

- The following sub-sections illustrate the usage of NAND specific commands on DM816x, DM814x and DM813x EVMs.
- If u-boot is built with NAND support and if NAND is disabled in the EVM (or NOR is enabled) then u-boot will crash immediately after displaying the banner. Refer to EVM Switch Settings section for more info on enabling/disabling different boot devices.

NAND Layout

The NAND part on the EVM has been configured in the following manner. The addresses mentioned here are used in the subsequent NAND related commands.

DM816X

```
  +------------+->0x00000000-> U-Boot start
|            |
|            |
|            |
|            |
|            |-->0x0025FFFF-> U-Boot end
|            |-->0x00260000-> ENV start
|            |
|            |
|            |-->0x0027FFFF-> ENV end
|            |-->0x00280000-> Linux Kernel start
|            |
|            |
|            |
|            |
|            |-->0x006BFFFF-> Linux Kernel end
|            |-->0x006C0000-> Filesystem start
|            |
|            |
|            |
|            |
|            |
|            |
|            |
|            |
```

```
|             |
|             |-->0x0CEDFFFF-> Filesystem end
|             |-->0x0CEE0000-> Free start
|             |
|             |
|             |
+-------------+-->0x10000000-> NAND end (Free end)
```

DM814x/DM813x

```
 +-------------+->0x00000000-> U-Boot 1st stage start
|             |
|             |-->0x0001FFFF-> U-Boot 1st stage end
|             |-->0x00020000-> U-Boot 2nd stage start
|             |
|             |-->0x0025FFFF-> U-Boot 2nd stage end
|             |-->0x00260000-> ENV start
|             |
|             |
|             |-->0x0027FFFF-> ENV end
|             |-->0x00280000-> Linux Kernel start
|             |
|             |
|             |
|             |
|             |-->0x006BFFFF-> Linux Kernel end
|             |-->0x006C0000-> Filesystem start
|             |
|             |
|             |
|             |
|             |
|             |
|             |
|             |
|             |
|             |-->0x0CEDFFFF-> Filesystem end
|             |-->0x0CEE0000-> Free start
|             |
|             |
|             |
+-------------+-->0x10000000-> NAND end (Free end)
```

Writing to Nand

To write len bytes of data from a memory buffer located at addr to the NAND block offset:

```
TI8168_EVM# nand write <addr> <offset> <len>
```

If a bad block is encountered during the write operation, it is skipped and the write operation continues from next 'good' block.

For example, to write 0x40000 bytes from memory buffer at address 0x80000000 to NAND - starting at block 32 (offset 0x400000):

```
TI8168_EVM# nand write 0x80000000 0x400000 0x40000
```

Reading from Nand

To read len bytes of data from NAND block at a particular offset to the memory buffer in DDR located at addr:

```
TI8168_EVM# nand read <addr> <offset> <len>
```

If a bad block is encountered during the read operation, it is skipped and the read operation continues from next 'good' block.

For example, to read 0x40000 bytes from NAND - starting at block 32 (offset 0x400000) to memory buffer at address 0x80000000:

```
TI8168_EVM# nand read 0x80000000 0x400000 0x40000
```

Marking a bad block

Some of the blocks in the NAND may get corrupted over a period of time. In such cases you should explicitly mark such blocks as bad so that the image that you are writing to NAND does not end up getting corrupted.

To forcefully mark a block as bad:

```
TI8168_EVM # nand markbad <offset>
```

For example, to mark block 32 (assuming erase block size of 128Kbytes) as bad block - offset = blocknum * 128 * 1024:

```
TI8168_EVM# nand markbad 0x400000
```

Viewing bad blocks

To view the list of bad blocks:

```
TI8168_EVM# nand bad
```

Note
The user marked bad blocks can be viewed by using this command only after a reset.

Erasing Nand

To erase NAND blocks in a particular the address range or using block numbers:

```
TI8168_EVM# nand erase <stoffaddr> <len>
```

This commands skips bad blocks (both factory or user marked) encountered within the specified range.

For example, to erase blocks 32 through 34:

```
TI8168_EVM# nand erase 0x00400000 0x40000
```

NAND ECC algorithm selection

NAND flash memory, although cheap, suffers from problems like bit flipping which lead to data corruption. However by making use of some error correction coding (ECC) techniques it is possible to workaround this problem.

For the data stored in NAND flash, U-Boot supports following NAND ECC schemes

1. S/W ECC (Hamming code)
2. H/W ECC (Hamming code, BCH4, BCH8, BCH16)

NOTE: Current releases do not support BCH4 and BCH16.

BCH Flash OOB Layout

For any ECC scheme we need to add some extra data while writing so as to detect and correct (if possible) the errors introduced by the NAND part. In case of BCH scheme some bytes are needed to store the ECC related info.

The section of NAND memory where addition info like ECC data is stored is referred to as Out Of Band or OOB section.

The first 2 bytes are used for Bad block marker – 0xFFFF => Good block

The next 'N' bytes is used for BCH bytes

N = B * <Number of 512-byte sectors in a page>

```
B =  8 bytes per 512 byte sector in BCH4
B = 14 bytes per 512 byte sector in BCH8
B = 26 bytes per 512 byte sector in BCH16
```

So for a 2k page-size NAND flash with 64-byte OOB size, we will use BCH8. This will consume 2 + (14*4) = 58 bytes out of 64 bytes available.

The NAND flash part used in EVM does not have enough spare area to support BCH16.

ECC Schemes and their context of usage

ECC type	Usage
S/W ECC	Not used
H/W ECC - Hamming Code	Use this for flashing any image/binary which will be used by Linux. This scheme is also used for by the U-Boot ENV variables.
H/W ECC – BCH8	Only while flashing U-Boot from U-Boot. After flashing U-Boot revert back to hamming code h/w ecc

To select ECC algorithm for NAND:

```
TI8168_EVM# nandecc [sw | hw <hw_type>]
```

Usage:

```
   sw - Set software ECC for NAND
hw <hw_type> - Set hardware ECC for NAND
 <hw_type> - 0 for Hamming code
             1 for bch4
             2 for bch8
             3 for bch16
Currently we support only Software, Hamming Code and BCH8. We do not support BCH4 and BCH16
```

Transferring images to NAND via U-Boot

Note

When updating any partition in NAND, please erase the complete partition and not just the space needed by the image which will be transferred onto NAND

Make sure the EVM is connected to network. This is required to get Linux Image from a tftp server. If there is no DHCPserver in the network then use static ip. The section on U-Boot network configuration gives commands for doing this.

Make sure tftp server is running in a windows PC and the home directory for the tftp server has Linux kernel uImage. This windows PC should be accessible from the EVM.

Flashing U-Boot from U-Boot

DM816x

To flash u-boot image (u-boot.noxip.bin) to NAND execute the commands listed below:

```
TI8168_EVM# mw.b 0x81000000 0xFF 0x260000
TI8168_EVM# tftp 0x81000000 u-boot.noxip.bin
TI8168_EVM# nand erase 0x0 0x260000     <=== Erasing the whole partition before flashing the image
TI8168_EVM# nand write.i 0x81000000 0x0 0x260000
```

The above set to commands, fill in the section of memory starting from 0x81000000 with size 0x260000 with 0xFF. In DM816x EVM this memory region is a part of the DDR memory.

Next we download the U-Boot image (u-boot.noxip.bin) to the DDR memory.

To have persistent storage we then transfer the downloaded U-Boot image to NAND with the help of the NAND commands described earlier.

DM814x and DM813x

1st Stage:

```
TI8148_EVM# nand erase 0x0 0x20000   <=== Erasing the whole partition before flashing the image
TI8148_EVM# tftp 0x81000000 u-boot.min.nand
TI8148_EVM# nand write.i 0x81000000 0x0 0x20000
```

2nd Stage:

```
TI8148_EVM# nand erase 0x20000 0x240000   <=== Erasing the whole partition before flashing the image
TI8148_EVM# tftp 0x81000000 u-boot.bin
TI8148_EVM# nand write.i 0x81000000 0x0 0x240000
```

Flashing Linux Kernel from U-Boot

TFTP the kernel uImage to DDR.

```
TI8168_EVM# mw.b 0x81000000 0xFF 0x440000
TI8168_EVM# tftp 0x81000000 <kernel_image>
```

Now flash the kernel image to NAND at the appropriate offset (refer to NAND layout section for the offsets)

```
TI8168_EVM# nand erase 0x00280000 0x00440000     <=== Erasing the whole partition before flashing the image
TI8168_EVM# nand write 0x81000000 0x00280000 <image_size> (example, 0x200000)
```

Creating UBIFS File-System

Refer to UBIFS Support guide

Creating JFFS2 File-system

Since jffs2 is used only on flash devices, a standard Linux distribution does not have the tools to make a jffs2 file system.Refer MTD_Utilities wiki page for instructions on how to create JFFS2. mkfs.jffs2 is the tool needed to build a jffs2 file system and the source is in this code. All of the MTD code will be downloaded but only the code to build mkfs.jffs2 is built.

With the mkfs.jffs2 utility built and in place it is now time to make the jffs2 file system from of the target directory. Change to the /home/user directory and enter the mkfs.jffs2 command below. Probably the most important argument to the utility is the erase block size. For the NAND part on the TI8168 EVM board the erase block is 128k Consult either u-boot, the kernel, or the data manual for the flash part used to find the erase block size.

```
  [root@localhost util]# cd /home/user
[root@localhost util]# mkfs.jffs2 -lqn —e 128 -r target -o /tftpboot/rd-jffs2.bin
```

By building the file in the /tftpboot directory, the step of copying it over is eliminated. The file must now be written into flash at a specific location. In u-boot, the flash file system will get flashed to the physical address 0x6c0000 and will be mounted by the kernel from /dev/mtdblock<partition-number>, partition-number starts from 0, refer flash layout for individual flash devices and partition creation order for exact number. Use 'cat /proc/partitions' at Linux prompt for the list of partitions.

After identifying the partition, unprotect the flash area where the file system will reside, erase that area. Download the rd-jffs2.bin file. Write it to flash. Modify bootargs to support a JFFS2 file system as root on /dev/mtdblock<partition-number>. Save the environment variables.

Flashing File-system from U-Boot

First TFTP the filesystem image (example - rootfs-base.jffs2 or ubi.img) to DDR.

```
TI8168_EVM# mw.b 0x81000000 0xFF 0x0C820000 <=== Required to get rid of "Empty Flash" JFFS2 warnings while booting.
TI8168_EVM# tftp 0x81000000 <filesystem_image>
```

Flash the file system image to NAND using the appropriate offsets

```
For UBIFS file system:
TI8168_EVM# nand erase 0x006C0000 0x0C820000  <=== Erasing the whole partition before flashing the image


For JFFS2 file system:
TI8168_EVM# nand erase clean 0x006C0000 0x0C820000      <=== Erasing the whole partition before flashing the image


TI8168_EVM# nand write 0x81000000 0x006C0000 <image_size> <=== Example, 0x01040000
NOTE: The image size should be upward aligned to NAND page size which is 2KiB (i.e. 0x800).
For example, if the image size is 0x19B8004 the size to be passed to the NAND write
command should be 0x19B8800.
```

```
  NOTE: The NAND page size alignment applies to any NAND write from u-boot and not only while flashing filesystem.
         Also, refer [[4]] for information on JFFS2 error/warning messages.
```

U-Boot NOR Support

This section gives an *overview* of the NOR support in U-Boot. It also describe how to store the kernel image, RAMDISK or the JFFS2 filesystem to NOR so as to have a network-free boot right from powering on the board to getting the kernel up and running.

Overview

Important

Please note that the U-Boot commands used in this section is applicable only from a U-Boot which is already running from NOR. Please refer to the Flashing Tools page for flashing the U-Boot image to NOR.

- This section details on the NOR support available in DM816x
- ROM Code always uses XIP for NOR boot.
- ROM Code initializes only 12 GPMC address lines (gpmc_a0 - gpmc_a11) hence it can access only first 4KBytes of the NOR.

NOR Pin Muxing Info for DM816x EVM

- Pin muxing for address lines gpmc_a0 to gpmc_a11 is not required (used in default mode)
- Pin muxing is done only for lines gpmc_a12 to gpmc_a25 using the following lines

```
gpmc_a12 -----> tim7_out
gpmc_a13 -----> uart1_ctsn
gpmc_a14 -----> uart1_rtsn
gpmc_a15 -----> uart2_rtsn
gpmc_a16 -----> uart2_ctsn
gpmc_a17 -----> uart0_rin
gpmc_a18 -----> uart0_dcdn
gpmc_a19 -----> uart0_dsrn
gpmc_a20 -----> uart0_dtrn
gpmc_a21 -----> spi_scs3
gpmc_a22 -----> spi_scs2
gpmc_a23 -----> gpo_io6
gpmc_a24 -----> tim6_out
gpmc_a25 -----> sc0_data
gpio0_20 -----> gpmc_a27
```

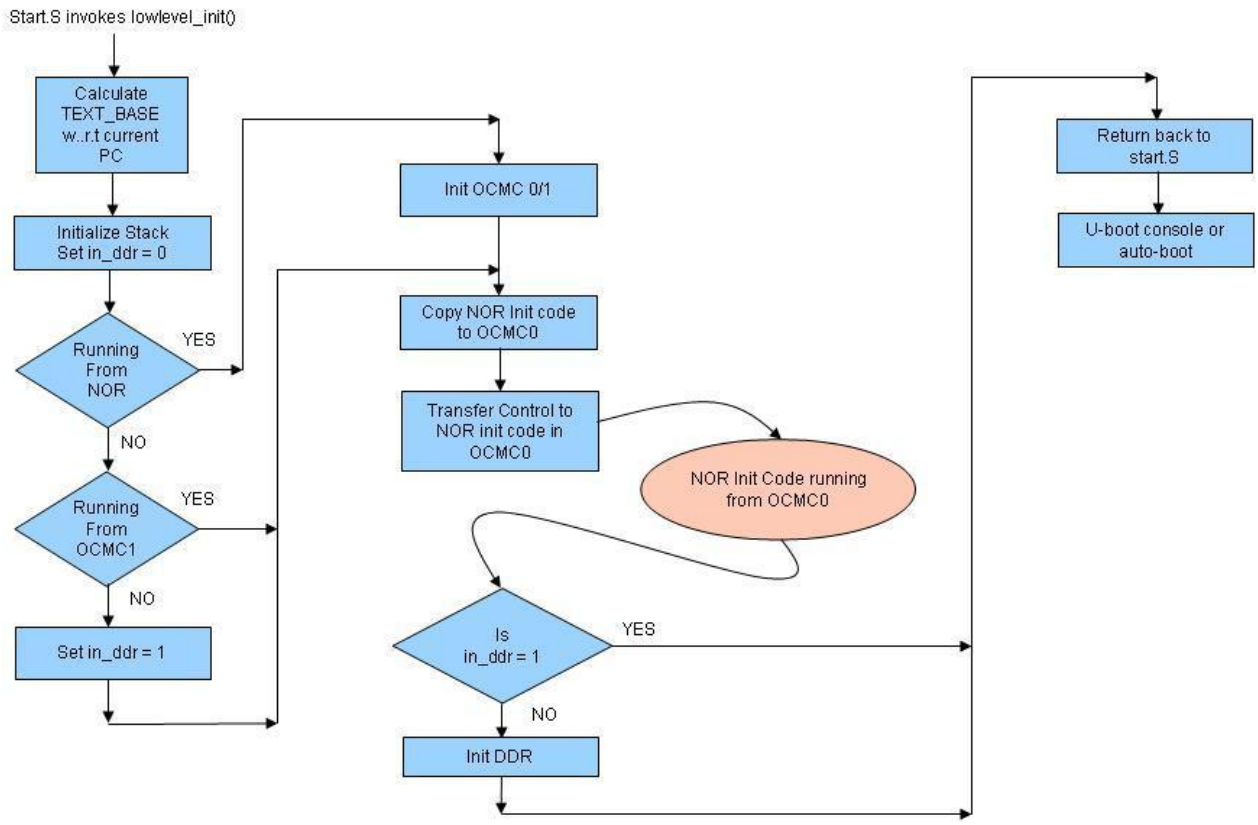
- gpmc_a27 should be configured as gpio0_20 and its output should be set low.
- For the above pad configuration we cannot use below mentioned modules
  - Smart Card 0
  - Advanced features of uart0/1/2
  - spi cs 2/3
  - timer 6/7 output

NOR Support

- The NOR boot logic is handled in **arch/arm/cpu/arm_cortexa8/ti81xx/lowlevel_init.S**
- Since ROM code does not initialize all 25 address lines required for accessing 64MBytes, u-boot should first initialize GPMC to continue with the XIP beyond 4KBytes.
- u-boot should also configure pin muxing for the remaining 13 address lines (gpmc_a12 - gpmc_a24) based on board schematics
- In case of NOR boot and XIP, we cannot initialize GPMC while running from NOR, hence we have to relocate the tiny GPMC init code to SRAM and then transfer control to SRAM and then come back to NOR XIP.
- Also, the relocatable GPMC init code should be placed in the first 4KBytes of the u-boot. This is achieved by adding lowlevel_init.o just below start.o in the u-boot linker script **board/ti8168_evm/u-boot.lds** as shown below.

```
SECTIONS
{
    . = 0x00000000;
    . = ALIGN(4);
    .text   :
    {
      arch/arm/cpu/arm_cortexa8/start.o  (.text)
      arch/arm/cpu/arm_cortexa8/ti81xx/lowlevel_init.o   (.text)
      *(.text)
    }
...
...
...
```

- Before relocating any code to SRAM we have to first enable the SRAM.
- All the above steps have to be done only if we are not already running from SRAM (or DDR)
- The diagram below explains the above discussed steps



Config Macros

```
#define CONFIG_SYS_FLASH_CFI
    #define CONFIG_MTD_DEVICE
    #define CONFIG_FLASH_CFI_DRIVER
    #define CONFIG_FLASH_CFI_MTD
    #define CONFIG_SYS_MAX_FLASH_SECT   512
    #define CONFIG_SYS_MAX_FLASH_BANKS  1
    #define CONFIG_ENV_IS_IN_FLASH      1
    #define CONFIG_SYS_FLASH_BASE       (0x08000000)
    #define CONFIG_SYS_MONITOR_BASE     CONFIG_SYS_FLASH_BASE
    #define NOR_SECT_SIZE               (128 * 1024)
    #define CONFIG_SYS_ENV_SECT_SIZE    (NOR_SECT_SIZE)
    #define CONFIG_ENV_OFFSET           (2 * NOR_SECT_SIZE)
    #define CONFIG_ENV_ADDR             (CONFIG_ENV_OFFSET)
```

- These macros assume spansion flash being used in the EVM. For any other flash part, the sector size macro (NOR_SECT_SIZE) has to be modified.

Spansion NOR parts are supported on DM816x platforms.

Note

- The following sub-sections illustrate the usage of NOR specific commands on DM8168EVM.
- If U-Boot is built with NOR support and if NOR is disabled in the EVM (or NAND is enabled) and if you try loading U-Boot through CCS then U-Boot will crash. Refer to EVM Switch Settings section for more info on enabling/disabling different boot devices.

NOR Layout

The NOR part on the EVM has been configured in the following manner. Please note the addresses mentioned here carefully as they are used in the subsequent NOR related commands.

```
-----------------+----> 0x08000000 (u-boot Start) Flash Start
|              |
|              |
|              |
|              |----> 0x0803FFFF (u-boot End)
|              |----> 0x08040000 (ENV Start)
|              |
|              |
|              |----> 0x0805FFFF (ENV End)
|              |----> 0x08060000 (Linux Start)
|              |
|              |
|              |
|              |
|              |----> 0x0845FFFF (Linux End)
|              |----> 0x08460000 (FS Start)
|              |
|              |
|              |
|              |
|              |
|              |
|              |
|              |----> 0x0B65FFFF (FS End)
|              |----> 0x0B660000 (Reserved Start)
|              |
|              |
-----------------+----> 0x0C000000 (Reserved End) Flash End
```

Writing to NOR

To write len bytes of data from a memory buffer located at addr to the NOR block offset (here <len> is always in 16-bit word count and not bytes):

```
TI8168_EVM# cp.w <src> <dest> <len>
```

For example, to write 0x40000 bytes (i.e. 0x20000 16-bit words) from memory buffer at address 0x81000000 to NOR - starting at offset 0x0B660000:

```
TI8168_EVM# cp.w 0x81000000 0x0B660000 0x20000
```

Reading from NOR

To read len bytes of data from NOR block at offset to memory buffer located at addr (here <len> is always in 16-bit word count and not bytes):

```
TI8168_EVM# cp.w <src> <dest> <len>
```

For example, to read 0x40000 bytes (i.e. 0x20000 16-bit words) from NOR - starting at offset 0x0B660000 to memory buffer at address 0x81000000:

```
TI8168_EVM# cp.w 0x0B660000 0x81000000 0x20000
```

Erasing NOR

To erase NOR in the address range (start and end should align to block size):

```
TI8168_EVM# erase <start> <end>
```

For example, to erase from 0x0B660000 to 0x0BFFFFFF:

```
TI8168_EVM# erase 0x0B660000 0x0BFFFFFF
```

Transferring images to NOR via U-Boot

Make sure the EVM is connected to network. This is required to get Linux Image from a tftp server. If there is no DHCPserver in the network then use static ip. The section on U-Boot network configuration gives the commands for doing this.

Make sure tftp server is running in a windows PC and the home directory for the tftp server has Linux kernel uImage. This windows PC should be accessible from board.

Flashing U-Boot from U-Boot

To flash u-boot image (u-boot.bin) to the NOR execute the commands listed below:

```
TI8168_EVM# mw.b 0x81000000 0xFF 0x100000
TI8168_EVM# tftp 0x81000000 u-boot.bin
TI8168_EVM# protect off 0x08000000 0x0803FFFF
TI8168_EVM# erase 0x08000000 0x0803FFFF
TI8168_EVM# cp.w 0x81000000 0x08000000 0x20000 (NOTE: 0x20000 is size of uboot.bin / 2)
```

Please note that the above steps can only be used from U-Boot already running out of NOR.

The above set of commands fill in the section of memory starting from 0x81000000 with size 0x100000 with 0xFF. In DM816x EVM this memory region is a part of the DDR memory. Next we download the U-Boot image to the DDR memory. To have persistent storage we then transfer the downloaded U-Boot image to NOR with the help of the NOR commands described earlier.

Flashing Linux Kernel from U-Boot

TFTP the kernel uImage to DDR.

```
TI8168_EVM# tftp 81000000 <kernel_image>
```

Flash the kernel image to NOR

```
TI8168_EVM# protect off 0x08060000  0x0845FFFF
TI8168_EVM# erase 0x08060000  0x0845FFFF
TI8168_EVM# cp.w 0x81000000 0x08060000 0xcb4da (NOTE: 0xcb4da is size of kernel_image / 2)
```

Creating JFFS2 File-system

Refer Creating JFFS2 File-system

Flashing File-system from U-Boot

TFTP filesystem image (example - rootfs-816x.jffs2) to DDR.

```
TI8168_EVM# tftp 81000000 <filesystem_image>
```

Flash the file system image to NOR

```
TI8168_EVM# protect off 0x08460000  0x0B65FFFF
TI8168_EVM# erase 0x08460000  0x0B65FFFF
TI8168_EVM# cp.w 0x81000000 0x08460000 0x800000 (NOTE: 0x800000 is size of filesystem / 2)
```

U-Boot SPI Support

Before proceeding with any of the commands given in this section please make sure that SPI is enabled on the EVM. The switch for enabling SPI is shown over here.

This section gives an *overview* of the SPI support in U-Boot. It also describe how to store the kernel image, RAMDISK or the JFFS2 filesystem to SPI flash so as to have a network-free boot right from powering on the board to getting the kernel up and running.

Overview

Winbond SPI Flash part W25X32 is supported on DM8168EVM. Refer http:/ / www. mail-archive. com/ u-boot@lists. denx. de/ msg35376. html [5] for information on supporting new Winbond SPI flashes. This mail thread discusses on adding support for W25Q64 SPI flash part.

Note

- The following sub-sections illustrate the usage of SPI specific commands on DM8168EVM.
- Refer to EVM Switch Settings section for more info on enabling/disabling different boot devices.

SPI Flash Layout

The SPI flash part on the EVM has been configured in the following manner. The addresses mentioned here are used in the subsequent SPI related commands.

DM816X

```
+-------------+-->0x00000000-> U-Boot start
|             |
|             |
|             |
```

```
|             |
|             |-->0x0003FFFF-> U-Boot end
|             |-->0x00040000-> ENV start
|             |
|             |
|             |-->0x00041FFF-> ENV end
|             |-->0x00042000-> Linux Kernel start
|             |
|             |
|             |
|             |
|             |-->0x002C1FFF-> Linux Kernel end
|             |-->0x002C2000-> Filesystem start
|             |
|             |
|             |
+---------------+-->0x00400000-> Filesystem end
```

DM814x/DM813x

```
+-------------+->0x00000000-> U-Boot 1st stage start
|             |
|             |-->0x0001FFFF-> U-Boot 1st stage end
|             |-->0x00020000-> U-Boot 2nd stage Start
|             |
|             |-->0x0005FFFF-> U-Boot 2nd stage end
|             |-->0x00060000-> ENV start
|             |
|             |
|             |-->0x00061FFF-> ENV end
|             |-->0x00062000-> Linux Kernel start
|             |
|             |
|             |
|             |
|             |-->0x002E1FFF-> Linux Kernel end
|             |-->0x002E2000-> Filesystem start
|             |
|             |
|             |
+---------------+-->0x00400000-> Filesystem end
```

NOTE :

- Type **sf** in u-boot prompt to get help on SPI flash commands
- Before using any SPI command, first the SPI flash has to be probed using the **sf probe 0** command. Here 0 is the SPI flash chip select number

Writing to SPI Flash

To write **len** bytes of data from a memory buffer located at **addr** to **offset** in SPI flash:

```
TI8168_EVM# sf write addr offset len
```

For example, to write 0x1000 bytes from memory buffer at address 0x80000000 to SPI flash - starting at offset 0x4000:

```
TI8168_EVM# sf write 0x80000000 0x4000 0x1000
```

Reading from SPI Flash

To read **len** bytes of data from SPI flash at a particular **offset** to the memory buffer located at **addr**:

```
TI8168_EVM# sf read addr offset len
```

For example, to read 0x1000 bytes from flash - starting at offset 0x4000 to memory buffer at address 0x80000000:

```
TI8168_EVM# sf write 0x80000000 0x4000 0x1000
```

Erasing SPI Flash

To erase **len** bytes from 'offset' in a SPI flash:

```
TI8168_EVM# sf erase offset len
```

For example, to erase 0x1000 bytes from offset 0x4000:

```
TI8168_EVM# sf erase 0x4000 0x1000
```

Transferring images to SPI flash via U-Boot

Make sure the EVM is connected to network. This is required to get Linux Image from a tftp server. If there is no DHCPserver in the network then use static ip. The section on U-Boot network configuration gives commands for doing this.

Make sure tftp server is running in a windows PC and the home directory for the tftp server has Linux kernel uImage. This windows PC should be accessible from the EVM.

Flashing U-Boot from U-Boot

To flash u-boot image (u-boot.noxip.bin.spi) to the Winbond SPI flash execute the commands listed below:

```
TI8168_EVM# mw.b 0x81000000 0xFF 0x100000
TI8168_EVM# tftp 0x81000000 u-boot.noxip.bin.spi
TI8168_EVM# sf probe 0:0
TI8168_EVM# sf erase 0x0 0x40000
TI8168_EVM# sf write 0x81000000 0x0 0x40000
```

The above set to commands, fill in the section of memory starting from 0x81000000 with size 0x100000 with 0xFF. In DM816x EVM this memory region is a part of the DDR memory.

Next we download the SPI U-Boot image to the DDR memory.

To have persistent storage we then transfer the downloaded U-Boot image to SPI flash with the help of the SPI flash commands described earlier.

Flashing Linux Kernel from U-Boot

TFTP the kernel uImage to DDR.

```
TI8168_EVM# tftp 81000000 <kernel_image>
```

Now flash the kernel image to flash at the appropriate offset (refer to SPI layout section for the offsets)

```
TI8168_EVM# sf probe 0
TI8168_EVM# sf erase 0x42000 0x280000
TI8168_EVM# sf write 0x81000000 0x42000 0x280000
```

Creating JFFS2 File-system

NOTE: JFFS2 cannot be supported on the SPI flash available on DM8168 EVM due to erase size restrictions. The erase sector size of W25X32 is 4KiB but JFFS2 requires it be minimum 8KiB. Hence JFFS2 filesystem cannot be used on this flash part. For help on JFFS2 filesystem creation (for other SPI flash parts) refer Creating JFFS2 File-system

Flashing File-system from U-Boot

```
TI8168_EVM# tftp 81000000 <filesystem_image>
TI8168_EVM#
```

Flash the file system image to flash using the appropriate offsets

```
TI8168_EVM# sf probe 0
TI8168_EVM# sf erase 0x2c2000 0x13E000
TI8168_EVM# sf write 0x81000000 0x2c2000 0x13E000
TI8168_EVM#
```

U-Boot UART support

Note: UART Flash is only supported on DM814X.

This section describes how to use UART boot mode using TeraTerm.

Ensure that you have the latest version 4.67 of the Teraterm installed. The kermit protocol implementation in Teraterm is not reliable in older versions. The latest version of Teraterm 4.67 can be downloaded from here [3]

Boot Over UART

Note
The release package does not contain the binary for UART boot. Please follow the steps mentioned #U-Boot_UART_support for UART boot for compiling u-boot.min.uart

1. Switch ON EVM with switch settings for UART boot. When “CCCC” characters appear on TeraTerm window, from the File Menu select Transfer --> XMODEM --> Send (1K mode)
2. Select “u-boot.min.uart” for the transfer
3. Press "Reset button" to initiate image download
4. After image is successfully downloaded, the ROM code will boot it.
5. Hit enter and get to u-boot prompt “TI-MIN#”

```
TI_MIN#
```

Loading 2nd stage over UART

- From the U-Boot-MIN prompt

```
TI_MIN# loadb 0x81000000
```

- From TeraTerm Menu click “File -> Transfer -> Kermit -> Send”.
- Select the 2nd stage u-boot image “u-boot.bin” built earlier and click “OPEN” button
- Wait for download to complete

```
TI_MIN# go 0x81000000
```

- You should get 2nd Stage u-boot prompt

```
TI8148_EVM#
```

Flashing images to NAND

Before proceeding with any of the commands given in this section please make sure that NAND is enabled on the EVM. The switch for enabling NAND is shown over here.

DM814x and DM813x

After the 2nd stage prompt **TI8148_EVM#** comes up, the images for the 1st stage and 2nd stage can be flashed to NAND for persistent storage.

Before proceeding with flashing of the images please make sure that NAND is enabled on the EVM.

Flashing 1st stage to NAND from 2nd stage

Flash U-Boot-MIN for NAND (**u-boot.min.nand**) to NAND by executing the following commands:

```
TI8148_EVM# mw.b 0x81000000 0xFF 0x20000
TI8148_EVM# loadb 0x81000000
```

- From TeraTerm Menu click “File -> Transfer -> Kermit -> Send”.
- Select the 1st stage u-boot image “**u-boot.min.nand**” and click “OPEN” button
- Wait for download to complete and then run following commands in u-boot prompt

```
TI8148_EVM# nand erase 0x0 0x20000
TI8148_EVM# nand write.i 0x81000000 0x0 0x20000
```

If no error messages are displayed the 1st stage of NAND boot has been successfully transferred to NAND.

Flashing 2nd stage to NAND from 2nd stage

Flash the 2nd stage U-Boot (**u-boot.bin**) to NAND by executing the following commands:

```
TI8148_EVM# mw.b 0x81000000 0xFF 0x60000
TI8148_EVM# loadb 0x81000000
```

- From TeraTerm Menu click “File -> Transfer -> Kermit -> Send”.
- Select the 2nd stage u-boot image “**u-boot.bin**” and click “OPEN” button
- Wait for download to complete and then run following commands in u-boot prompt

```
TI8148_EVM# nand erase 0x20000 0x60000
TI8148_EVM# nand write.i 0x81000000 0x20000 0x60000
```

If no error messages are displayed the 2nd stage of NAND boot has been successfully transferred to NAND.

Flashing images to SPI

Before proceeding with any of the commands given in this section please make sure that SPI is enabled on the EVM. The switch for enabling SPI is shown over here.

After the 2nd stage prompt TI8148_EVM# comes up the images for the 1st stage and 2nd stage can be flashed to SPI for persistent storage.

Flashing 1st stage to SPI from 2nd stage

Flash U-Boot-MIN for SPI (**u-boot.min.spi**) to SPI flash by executing the following commands:

```
TI8148_EVM# mw.b 0x81000000 0xFF 0x20000
TI8148_EVM# loadb 0x81000000
```

- From TeraTerm Menu click “File -> Transfer -> Kermit -> Send”.
- Select the 1st stage u-boot image “**u-boot.min.spi**” and click “OPEN” button
- Wait for download to complete and then run following commands in u-boot prompt

```
TI8148_EVM# sf probe 0:0
TI8148_EVM# sf erase 0x0 0x20000
TI8148_EVM# sf write 0x81000000 0x0 0x20000
```

If no error messages are displayed then the 1st stage for SPI boot is successfully flashed to SPI.

Flashing 2nd stage to SPI from 2nd stage

Flash 2nd stage u-boot image (**u-boot.bin**) to SPI flash by executing the following commands:

```
TI8148_EVM# mw.b 0x81000000 0xFF 0x40000
TI8148_EVM# loadb 0x81000000
```

- From TeraTerm Menu click “File -> Transfer -> Kermit -> Send”.
- Select the 1st stage u-boot image “**u-boot.bin**” and click “OPEN” button
- Wait for download to complete and then run following commands in u-boot prompt

```
TI8148_EVM# sf probe 0:0
TI8148_EVM# sf erase 0x20000 0x40000
TI8148_EVM# sf write 0x81000000 0x20000 0x40000
```

If no error messages are displayed then the 2nd stage for SPI boot is successfully flashed to SPI.

U-Boot SD (Secured Digital card) Support

This section gives an overview of the SD (Secured Digital Card) support in U-Boot

Overview

SD (Secured Digital Card) support is available through HSMMC controller.

SD support is available by default in all U-Boot configs. SD card can be access in all other boot modes (NAND/SPI/NOR).

Read and execute application from SD card

List files on a FAT32 formated SD card

```
TI8168_EVM# mmc rescan 0
TI8168_EVM# fatls mmc 0
```

Booting kernel image from the SD card

```
TI8168_EVM# mmc rescan 0
TI8168_EVM# fatload mmc 0 0x81000000 uImage
TI8168_EVM# bootm 0x81000000
```

Booting application (ex. u-boot.bin)image from the SD card

```
TI8168_EVM# mmc rescan 0
TI8168_EVM# fatload mmc 0 0x81000000 u-boot.bin
TI8168_EVM# go 0x81000000
```

Note: For the PSP_04.00.00.12 and older release use the following mmc commands

```
$mmc rescan 0 must be replaced with $mmc init
$fatload mmc 0 <addr> <file> must be replaced with $fatload mmc 1 <addr> <file>
```

Setting Up Boot Environment on SD Card

This section describes steps to be followed to create a standalone power-on bootable system on SD card.

Prerequisites Ensure that following is available:

- A Linux host with fdisk, sfdisk, mkfs.ext3 and mkfs.vfat utilities is available
- Copy images MLO, u-boot.bin, uImage, nfs.tar.gz and mksd-ti816x.sh from release package to a directory on this Linux machine. For subsequent description, we will assume these files are copied to /home/ti816x. Please refer Package Contents section in DM816x User Guide for location of these files.
- Empty SD card (at least 256MB, preferably 4GB SDHC)
- A SD memory card reader/programmer to copy files from Linux Host

```
Note: The SD Boot has some specific restriction about the format of the 1st partition and copying the MLO image.
So it is recommended that the supplied script mksd-ti816x.sh is used for creating partitions and copying files.
```

Steps

- Connect the SD memory card using Memory Card reader to the Linux Host
- Note the name allotted for this device. We assume the device is named as "/dev/sdd"
- Navigate to the /home/ti816x directory where all the mentioned files are copied
- Ensure that the script mksd-ti816x.sh has executable permissions
- This scripts expects arguments in the following format

```
./mksd-ti816x.sh <sd-device-name> <sd-1st-stage-bootloader> <kernel-uImage> <tar-gzipped-filesystem-directory>
```

- **Note** that the user will require root/sudo permissions
- In our example, we run the following command

```
sudo ./mksd-ti816x.sh /dev/sdd MLO u-boot.bin uImage nfs.tar.gz
```

- You will be asked about data getting overwritten, confirm it and the files along with filesystem will be copied to SD card
- Note that this script will create two primary partitions:
  - 1st partition is formatted as FAT32 containing MLO, u-boot.bin, uImage files
  - 2nd partition is formatted as ext3 where the filesystem is extracted in root

Boot using SD card

Once the SD card has been setup as described in the previous section make sure the switch setting are set for SD boot mode and then plug in the SD card in the MMC/SD card slot on the EVM.

When the EVM is powered on the 1st stage will autoload the 2nd stage from SD. Interrupt the countdown in the 2nd stage if kernel boot is not required. You should see the **TI8168_EVM#** prompt on the console.

Flashing DM816X U-Boot to NAND using SD boot

Before proceeding with any of the commands given in this section please make sure that NAND is enabled on the EVM. The switch for enabling NAND on TI8168 EVM is SW4.

Copy the U-Boot image **u-boot.noxip.bin** built for NAND as described [#U-Boot_for_NAND_boot here] in the FAT partition on the SD card. (The release package contains a pre-built image u-boot.noxip.bin which can be used for this purpose)

Once the second stage of SD boot comes up use the following commands to flash to NAND

```
TI8168_EVM# mmc rescan 0
TI8168_EVM# fatload mmc 0 0x81000000 u-boot.noxip.bin
TI8168_EVM# nand erase 0x0 0x1c0000
TI8168_EVM# nand write.i 0x81000000 0x0 0x1c0000
```

After this the EVM switch settings can be changed to [#NAND_boot NAND boot mode] if boot out of NAND is required.

Flashing DM816X U-Boot to SPI using SD boot

Before proceeding with any of the commands given in this section please make sure that SPI is enabled on the EVM. The switch for enabling SPI on TI8168 EVM is SW4.

Copy the U-Boot image **u-boot.noxip.bin.spi** built for SPI as described U-Boot_for_SPI_boot [6] in the FAT partition on the SD card.

Once the second stage of SD boot comes up use the following commands to flash to SPI

```
TI8168_EVM# mmc rescan 0
TI8168_EVM# mw.b 0x81000000 0xFF 0x100000
TI8168_EVM# fatload mmc 0 0x81000000 u-boot.noxip.bin.spi
TI8168_EVM# sf probe 0:0
TI8168_EVM# sf erase 0x0 0x40000
TI8168_EVM# sf write 0x81000000 0x0 0x40000
```

After this the EVM switch settings can be changed to SPI_boot_mode [7] if boot out of SPI is required.

Flashing images to DM814x & DM813x NAND in SD boot

Before proceeding with any of the commands given in this section please make sure that NAND is enabled on the EVM. The switch for enabling NAND is shown over here.

Copy the U-Boot image **u-boot.min.nand** built for 1st stage NAND and **u-boot.bin** built for NAND in the FAT partition on the SD card.

After the 2nd stage prompt **TI8148_EVM#** comes up, the images for the 1st stage and 2nd stage can be flashed to NAND for persistent storage.

Flashing 1st stage to NAND from 2nd stage in SD boot

Flash U-Boot-MIN for NAND (**u-boot.min.nand**) to NAND by executing the following commands:

```
TI8148_EVM# mmc rescan 0
TI8148_EVM# mw.b 0x81000000 0xFF 0x20000
TI8148_EVM$ fatload mmc 0 0x81000000 u-boot.min.nand
TI8148_EVM# nand erase 0x0 0x20000
TI8148_EVM# nand write.i 0x81000000 0x0 0x20000
```

If no error messages are displayed the 1st stage of NAND boot has been successfully transferred to NAND.

Flashing 2nd stage to NAND from 2nd stage in SD boot

Flash the 2nd stage U-Boot (**u-boot.bin**) to NAND by executing the following commands:

```
TI8148_EVM# mmc rescan 0
TI8148_EVM# mw.b 0x81000000 0xFF 0x20000
TI8148_EVM$ fatload mmc 0 0x81000000 u-boot.bin
TI8148_EVM# nand erase 0x20000 0x60000
TI8148_EVM# nand write.i 0x81000000 0x20000 0x60000
```

If no error messages are displayed the 2nd stage of NAND boot has been successfully transferred to NAND.

Flashing images to DM814X SPI in SD boot

Before proceeding with any of the commands given in this section please make sure that SPI is enabled on the EVM. The switch for enabling SPI is shown over here.

Copy the U-Boot image **u-boot.min.spi** built for 1st stage SPI as described**here** and **u-boot.bin** built for SPI as described **here** in the FAT partition on the SD card.

After the 2nd stage prompt TI8148_EVM# comes up the images for the 1st stage and 2nd stage can be flashed to SPI for persistent storage.

Flashing DM814X 1st stage to SPI from 2nd stage in SD boot

Flash U-Boot-MIN for SPI (**u-boot.min.spi**) to SPI flash by executing the following commands:

```
TI8148_EVM# mmc rescan 0
TI8148_EVM# mw.b 0x81000000 0xFF 0x20000
TI8148_EVM# fatload mmc 0 0x81000000 u-boot.min.spi
TI8148_EVM# sf probe 0:0
TI8148_EVM# sf erase 0x0 0x20000
TI8148_EVM# sf write 0x81000000 0x0 0x20000
```

If no error messages are displayed then the 1st stage for SPI boot is successfully flashed to SPI.

Flashing DM814X 2nd stage to SPI from 2nd stage in SD boot

Flash 2nd stage u-boot image (**u-boot.bin**) to SPI flash by executing the following commands:

```
TI8148_EVM# mmc rescan 0
TI8148_EVM# mw.b 0x81000000 0xFF 0x40000
TI8148_EVM# fatload mmc 0 0x81000000 u-boot.bin
TI8148_EVM# sf probe 0:0
TI8148_EVM# sf erase 0x20000 0x40000
TI8148_EVM# sf write 0x81000000 0x20000 0x40000
```

If no error messages are displayed then the 2nd stage for SPI boot is successfully flashed to SPI.

Note: For the PSP_04.00.00.12 and older release use the following mmc commands

```
$mmc rescan 0 must be replaced with $mmc init
$fatload mmc 0 <addr> <file> must be replaced with $fatload mmc 1 <addr> <file>
```

ENV on SD card using a script

U-Boot environment variables can be modified using U-Boot scripts. The scripts can be used to modify and even over-ride the various parameters like bootargs, TFTP serverip etc.

Generation of the scripts is done with the help of the *mkimage* tool which can be found under the tools directory of the U-Boot source. The *mkimage* binary gets generated whenever any U-Boot image is built.

Once the *mkimage* binary generated, create a text file named *boot.txt* with the U-Boot commands that would normally be executed manually at the U-Boot prompt.

Example text file named *boot.txt*

DM816X

```
setenv bootargs 'console=ttyO2,115200n8 root=/dev/mmcblk0p2 mem=128M rootwait'
setenv bootcmd 'mmc rescan 0; fatload mmc 0 0x81000000 uImage; bootm 0x81000000'
boot
```

DM814x/DM813x

```
setenv bootargs 'console=ttyO0,115200n8 root=/dev/mmcblk0p2 mem=128M rootwait'
setenv bootcmd 'mmc rescan 0; fatload mmc 0 0x81000000 uImage; bootm 0x81000000'
boot
```

Note: For the PSP_04.00.00.12/PSP_04.00.00.5 and older release use the following

DM816X

```
setenv bootargs 'console=ttyO2,115200n8 root=/dev/mmcblk0p2 mem=128M rootwait'
setenv bootcmd 'mmc init; fatload mmc 1 0x81000000 uImage; bootm 0x81000000'
boot
```

DM814x/DM813x

```
setenv bootargs 'console=ttyO0,115200n8 root=/dev/mmcblk0p2 mem=128M rootwait'
setenv bootcmd 'mmc init; fatload mmc 1 0x81000000 uImage; bootm 0x81000000'
boot
```

Now use the following command to generate the script named *boot.scr*

```
mkimage -A arm -O linux -T script -C none -n TI_script -d boot.txt boot.scr
```

The file *boot.scr* can now be executed from U-Boot using the source command to execute all the commands present in it. Eg: TFTP the file to some location in DDR and then use source command in the following manner (assumption: network setting has already been done using the steps mentioned in this User Guide)

```
TI8168_EVM# tftp 0x81000000 boot.scr
TI8168_EVM# source 0x81000000
```

The script can even be placed in the SD card and then bootcmd set to autorun this script if present. This is the default behavior of the SD boot images built from the PSP release.

EMAC Boot

This section gives an overview of the EMAC Boot support in U-Boot

TI816x

EMAC boot support can be referred in EMAC Boot

TI814x/TI813x

EMAC boot support can be referred in EMAC Boot

U-Boot Network configuration

In order to download the Linux kernel image from the TFTP server and for mounting NFS the network settings in U-Boot need to be configured.

Note: Uboot supports only Ethernet on Port 1 (J14), Port 2 (J27) is not supported in current release.

Please note that the following commands are being run from the 2nd stage U-Boot prompt on the serial console.

```
TI8168_EVM# setenv autoload no
TI8168_EVM# setenv bootfile uImage
```

The above two commands configure U-Boot not to autoboot the kernel and then set the name of the kernel image to be downloaded over the network.

When booting for the first time, U-Boot tries to fetch the MAC address from the env space. If it returns empty, it will look for MAC address from the eFuse registers in the Control module space and set the "ethaddr" variable in the env appropriately.

The ethaddr can also be set using the setenv/saveenv commands. In such cases the user-set MAC address will take effect on subsequent reboot only.

To set a different MAC address use the following command

```
TI8168_EVM# set ethaddr <random MAC address eg- 08:11:23:32:12:77>
```

Note
When setting a MAC address please ensure that the LSB of the 1st byte if not 1 i.e. when setting the MAC address: **y** in x**y**:ab:cd:ef:gh:jk has to be an even number. For more info this refer to the wiki page http://en.wikipedia.org/wiki/MAC_address

In case a static ip is not available run the dhcp command to obtain the ip address from the DHCP server on the network to which the EVM is connected.

```
TI8168_EVM# setenv serverip <tftp server in your network>
TI8168_EVM# dhcp
```

```
TI8168_EVM# saveenv
```

In case a static ip is available run the following commands

```
TI8168_EVM# setenv ipaddr <your static ip>
TI8168_EVM# saveenv
```

This completes the network configuration in U-Boot.

U-Boot Environment Variables

After completing the network configuration and flashing the kernel image and filesystems to flash you need to set some other parameters which are essential for booting the kernel.

Environment Settings for Ramdisk

In case you are using a RAMDISK as the Linux filesystem

DM816x

```
TI8168_EVM# setenv bootargs 'console=ttyO2,115200n8 mem=256M earlyprintk root=/dev/ram rw initrd=0x82000000,32MB'
```

DM814x/DM813x

```
TI8148_EVM# setenv bootargs 'console=ttyO0,115200n8 mem=256M earlyprintk root=/dev/ram rw initrd=0x82000000,32MB'
```

In case of NAND boot (0x170000 => size of Linux kernel uImage, 0x00320000 => size of Filesystem)

```
TI8168_EVM# setenv bootcmd 'nand read 0x81000000 0x280000 0x170000;nand read 0x82000000 0x6C0000 0x320000;bootm 0x81000000'
```

In case of NOR boot (0x00320000 => size of Filesystem)

```
TI8168_EVM# setenv bootcmd 'cp.w 0x8460000 0x82000000 0x320000;bootm 0x08060000'
```

In case of SPI boot (0x280000 => size of Linux kernel uImage, 0x13E000 => size of Filesystem)

```
TI8168_EVM# setenv bootcmd 'sf probe 0; sf read 0x81000000 0x42000 0x280000; sf read 0x82000000 0x2C2000 0x13E000;bootm 0x81000000'
```

In case of SD boot

```
TI8168_EVM# setenv bootcmd 'mmc init;fatload mmc 1 0x80009000 uImage;bootm 0x80009000'
```

If auto boot is required then prefix 'dhcp' and 'run addip' to the above 'bootcmd'

```
TI8168_EVM# setenv bootcmd 'dhcp;run addip;${bootcmd}'
```

NOTE: The sizes of images mentioned in the above commands have to be modified based on the actual image size. Also, it should be aligned to sector size of the flash device used.

Finally

```
TI8168_EVM# saveenv
```

Environment Settings for JFFS2 Filesystem

- U-Boot environment variable **bootargs** has information on arguments to be passed to Linux kernel. The **bootargs** format for JFFS2 root filesystem is,

DM816x

```
console=ttyO2,115200n8 root=/dev/mtdblock<partion_id> [ro|rw] rootfstype=jffs2 mem=100M earlyprintk
```

DM814x/DM813x

```
console=ttyO0,115200n8 root=/dev/mtdblock<partion_id> [ro|rw] rootfstype=jffs2 mem=100M earlyprintk
```

The value of **<partition_id>** depends on memory device which holds the rootfs. The below list gives values for different scenarios and it also assumes that only that respective device is enabled on the board,

```
 rootfs on NAND ==> partion_id should be 3
rootfs on NOR ==> partion_id should be 3
rootfs on SPI ==> partion_id should be 3
 NOTE: If the kernel is compiled with both SPI and NAND support then /dev/mtdblock7 should be used for NAND.
   If the kernel is compiled with both SPI and NOR support then /dev/mtdblock7 should be used for SPI.
```

- Below examples set **bootargs** variable from u-boot prompt for different memory devices (in case of SPI, mtdblock3 should be used).

In case if you want readonly filesystem,

DM816x

```
TI8168_EVM# setenv bootargs 'console=ttyO2,115200n8 root=/dev/mtdblock3 ro rootfstype=jffs2 mem=100M earlyprintk'
```

DM814x/DM813x TI8148_EVM# setenv bootargs 'console=ttyO0,115200n8 root=/dev/mtdblock3 ro rootfstype=jffs2 mem=100M earlyprintk'

In case if you want read/write filesystem,

DM816x

```
TI8168_EVM# setenv bootargs 'console=ttyO2,115200n8 root=/dev/mtdblock3 rw rootfstype=jffs2 mem=100M earlyprintk'
```

DM814x/DM813x

```
TI8148_EVM# setenv bootargs 'console=ttyO0,115200n8 root=/dev/mtdblock3 rw rootfstype=jffs2 mem=100M earlyprintk'
```

NOTE: If the kernel is compiled with both SPI and NAND support then /dev/mtdblock7 should be used for NAND

In case of NAND boot (0x170000 => size of Linux kernel uImage)

```
TI8168_EVM# setenv bootcmd 'nand read 0x81000000 0x00280000 0x170000;bootm 0x81000000'
```

In case of NOR boot

```
TI8168_EVM# setenv bootcmd 'bootm 0x08060000'
```

In case of SPI boot (0x280000 => size of Linux kernel uImage)

```
TI8168_EVM# setenv bootcmd 'sf probe 0; sf read 0x81000000 0x42000 0x280000;bootm 0x81000000'
```

NOTE: The sizes of images mentioned in the above commands have to be modified based on the actual image size. Also, it should be aligned to sector size of the flash device used.

Environment Settings for NFS Filesystem

In case you want to have the kernel use the same ip as that assigned to U-Boot

```
TI8168_EVM# print ethaddr                          <-- Check if MAC address is assigned and is unique

TI8168_EVM# setenv ethaddr <unique-MAC-address>    <-- Set only if not present already, format uv:yy:zz:aa:bb:cc

TI8168_EVM# setenv bootcmd 'dhcp;run addip; tftp 81000000 uImage;bootm'

TI8168_EVM# setenv hostname <unique-hostname>

TI8168_EVM# setenv addip 'setenv bootargs ${bootargs} ip=${ipaddr}:${nfsserver}:${gatewayip}:${netmask}:${hostname}:eth0:off'

TI8168_EVM# setenv autoload no

TI8168_EVM# setenv nfsserver <NFS server-ip>         <-- Make sure the same NFS server IP is used below

TI8168_EVM# setenv bootargs 'console=ttyO2,115200n8 root=/dev/nfs nfsroot=<NFS server-ip>:<NFS share>,nolock rw mem=128M'

TI8168_EVM# setenv serverip <tftp-server-ip>
```

In case you want to use dhcp for getting an IP when the kernel boots up

```
TI8168_EVM# print ethaddr                         <-- Check if MAC address is assigned and is unique

TI8168_EVM# setenv ethaddr <unique-MAC-address>    <-- Set only if not present already, format uv:yy:zz:aa:bb:cc

TI8168_EVM# setenv bootcmd 'dhcp;tftp 81000000 uImage;bootm'

TI8168_EVM# setenv autoload no

TI8168_EVM# setenv nfsserver <NFS server-ip>         <-- Make sure the same NFS server IP is used below

TI8168_EVM# setenv bootargs 'console=ttyO2,115200n8 root=/dev/nfs nfsroot=<NFS server-ip>:<NFS share>,nolock rw mem=128M ip=dhcp'

TI8168_EVM# setenv serverip <tftp-server-ip>
```

Finally

```
TI8168_EVM# saveenv
```

Booting the kernel

In case everything went well you should now be able to boot the kernel from U-Boot using the following command.

TI8168_EVM# boot

References

[1] http://creativecommons.org/licenses/by-sa/3.0/
[2] http://www.denx.de/wiki/U-Boot/WebHome
[3] http://logmett.com/index.php?/download/tera-term-467.html
[4] http://www.linux-mtd.infradead.org/archive/tech/faq.html
[5] http://www.mail-archive.com/u-boot@lists.denx.de/msg35376.html
[6] http://processors.wiki.ti.com/index.php?title=DM816x_C6A816x_AM389x_PSP_U-Boot#U-Boot_for_SPI_boot
[7] http://processors.wiki.ti.com/index.php?title=DM816x_C6A816x_AM389x_PSP_U-Boot#SPI_boot_mode

Article Sources and Contributors

TI81XX PSP U-Boot *Source*: http://processors.wiki.ti.com/index.php?oldid=96265 *Contributors*: Deepu.raj, Mugunthanvnm, Parth.saxena

Image Sources, Licenses and Contributors

Image:TIBanner.png *Source*: http://processors.wiki.ti.com/index.php?title=File:TIBanner.png *License*: unknown *Contributors*: Nsnehaprabha

Image:U-boot nor-boot gpmc-init.jpg *Source*: http://processors.wiki.ti.com/index.php?title=File:U-boot_nor-boot_gpmc-init.jpg *License*: unknown *Contributors*: Vaibhav

License

b. Subject to the above terms and conditions, the license granted here is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, Licensor reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.

8. Miscellaneous

a. Each time You Distribute or Publicly Perform the Work or a Collection, the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted to You under this License.
b. Each time You Distribute or Publicly Perform an Adaptation, Licensor offers to the recipient a license to the original Work on the same terms and conditions as the license granted to You under this License.
c. If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this agreement, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.
d. No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.
e. This License constitutes the entire agreement between the parties with respect to the Work licensed here. There are no understandings, agreements or representations with respect to the Work not specified here. Licensor shall not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Licensor and You.
f. The rights granted under, and the subject matter referenced, in this License were drafted utilizing the terminology of the Berne Convention for the Protection of Literary and Artistic Works (as amended on September 28, 1979), the Rome Convention of 1961, the WIPO Copyright Treaty of 1996, the WIPO Performances and Phonograms Treaty of 1996 and the Universal Copyright Convention (as revised on July 24, 1971). These rights and subject matter take effect in the relevant jurisdiction in which the License terms are sought to be enforced according to the corresponding provisions of the implementation of those treaty provisions in the applicable national law. If the standard suite of rights granted under applicable copyright law includes additional rights not granted under this License, such additional rights are deemed to be included in the License; this License is not intended to restrict the license of any rights under applicable law.