Freescale Semiconductor
Application Note

Document Number:AN4814
Rev 0, 10/2013

Porting U-Boot and Linux to T4240 Systems

Contents

1 Introduction

This application note describes how to port U-Boot and Linux sources from Freescale's Software Development Kit (SDK) to a new T4240 platform.

This document covers the main areas within the U-Boot and Linux sources that a developer should be aware of to port these packages to a new T4240 platform. It also assumes the reader has at least a basic knowledge on U-Boot, Linux and git. Note that the code sources referenced in this document are based on SDK 1.4 which uses U-Boot 2013.01 and Linux kernel 3.8.13. For different U-Boot or kernel releases there may be slight changes to some of these references.

The SDK includes support for the T4240QDS, therefore T4240 device support is already included within the package. As such, the items discussed here focus on board level changes. For the purposes of this document, the example platform is the T4240-SDP, a custom T4240 reference design from Freescale.

2 Porting U-Boot

This section covers the process of obtaining the U-Boot source, modifying the source to support the new platform and building binary images that will execute on the platform.

2.1 Overview

U-Boot is a multi-functional open-source bootloader which allows a developer to load an operating system.

In addition to the bootstrapping functionality, U-Boot also supports other features such as device drivers, networking and file systems support.

U-Boot is free software released under the terms of the GNU General Public License (GPL). For the purposes of this document U-Boot will be responsible for initial board bring-up including DDR, PCIe and networking interfaces, and for booting the Linux operating system.

2.2 Obtaining the source code

The latest U-Boot source can be obtained from the repository at git://git.denx.de/u-boot.git and users can clone the tree from this location to port to their new platform using the command below.

```
git clone git://git.denx.de/u-boot.git
```

The method detailed below uses the U-Boot package available within the SDK as a starting point for the port. Information on how to extract/modify the U-Boot sources from the SDK is available on the infocenter website:

http://www.freescale.com/infocenter/index.jsp?topic=%2FQORIQSDK%2F2880375.html.

The main steps are as follows:

1. Install the latest SDK release
2. In the main install directory create a build directory for the T4240QDS, i.e.
   - `source ./fsl-setup-poky -m t4240qds`
3. From the <yocto_install_path>/build_t4240qds_release directory, extract the U-Boot source using the following command:
   - `bitbake -c patch u-boot`
4. To find the location of the extracted U-Boot source run the following command and look for the source ‘S=’ result
   - `bitbake -e u-boot | grep ^S`

   The output will look something like:
   - `S="/home/Projects/sdk-1.4/QorIQ-SDK-V1.4-20130625-yocto/build_t4240qds_release/tmp/work/t4240qds-fsl_networking-linux/u-boot/git-r33/git"`
5. cd to the source directory given by ‘S=’
6. Set <ARCH> and <CROSS_COMPILE> to the correct values, i.e.
   - export ARCH=powerpc
   - Using ‘export CROSS_COMPILE=’ set the path to the relevant cross compile tools within the /opt directory on your system
7. Check the build process by building the T4240QDS binary, i.e.
   - Run ‘make T4240QDS’
   - Check the u-boot.bin file is created in the top level U-Boot directory

The source code in the directory above can now be modified for the new platform. The steps in this application note assume U-Boot and the kernel are built manually, i.e. not using bitbake. If users prefer to use bitbake they should refer to the infocenter QorIQ instructions/FAQs for U-Boot and Linux.

2.3 Creating a new platform

Adding a new platform can be split into the following tasks:

- New board configuration files required to add overall support for the platform
- Modification of existing drivers to support new features/components used on the board

The steps below outline the process of completing both these tasks.

2.3.1 Step 1: Update boards.cfg file

At the top level in U-Boot the boards.cfg file lists all supported platforms, for example:

```
T4240QDS      powerpc      mpc85xx      t4qds      Freescale      -      T4240QDS:PPC_T4240
```

The fields above define the Target, ARCH, CPU, Board name, Vendor, SoC and Options for the board. The Target is the name used during the build process to create binaries for a specific platform. A new entry is therefore required in this file for the new platform, for example:

```
T4240SDP      powerpc      mpc85xx      t4sdp      Freescale      -      T4240SDP:PPC_T4240
```

2.3.2 Step 2: Create a board configuration file

The include/configs directory contains board configuration files where the user should define which interfaces are supported (e.g. CONFIG_SATA1 to enable SATA controller 1), the memory map of the system (e.g. CONFIG_SYS_DDR_SDRAM_BASE defines the DDR base address), other options such as supported U-Boot commands (e.g. CONFIG_CMD_I2C) and the U-Boot environment variables (e.g. bootargs).

For a new development platform, in this case the T4240-SDP, developers could create T4240SDP.h and t4sdp.h files within the configs directory based on those already available for the T4240QDS. These files can then be modified to meet the requirements of the new board.

2.3.3 Step 3: Create board specific directory and files

The board/freescale directory contains directories for supported platforms, e.g. t4qds. A new T4240 platform would require a directory at this level and the files listed below.

As with the configuration file users can start with the T4240QDS files and modify them to meet their requirements.

- t4sdp.c Board specific peripheral set-up. Includes functions to print clock and RCW details at start-up.
- ddr.c Board specific DDR initialization functions. Includes DDR controller settings such as clock adjust and write leveling control.

Part of this initialization handles enabling interleaving based on the requirements passed in from the U-Boot environment variable hwconfig. For example: If the user has the hwconfig setting shown below, the populate_memctl_options() function in arch/powerpc/cpu/mpc8xxx/ddr/options.c will take these inputs and ensure the DDR initialization completes with the relevant interleaving options in place. The example below enables both controller and bank based interleaving. If interleaving should be disabled these entries within hwconfig should be removed.

```
hwconfig=fsl_ddr:ctlr_intlv=3way_4KB,bank_intlv=auto;
```

- eth.c Board specific Ethernet initialization functions including setting up the MDIO bus for each of the FMan interfaces.

The PHY addresses used in the board_eth_init() function are defined in the board configuration file discussed in Step 2: Create a board configuration file , for example:.

```
#define RGMII_PORT1_PHY_ADDR 0x5
```

- pci.c Functions for configuring the PCIe interface.

pci_of_setup() is called in ft_board_setup() within t4sdp.c if CONFIG_PCI is defined in the board configuration file covered in Step 2: Create a board configuration file.

- law.c Defines local access windows for interfaces such as flash memory based on the memory map of the system.

If users copy the T4240QDS file as a starting point it is easy to add/remove/modify entries in law_table[]. For example, to change the size of the flash LAW from 256MB to 128MB users should change the entry below from:

```
SET_LAW(CONFIG_SYS_FLASH_BASE_PHYS, LAW_SIZE_256M, LAW_TRGT_IF_IFC)
```

to

```
SET_LAW(CONFIG_SYS_FLASH_BASE_PHYS, LAW_SIZE_128M, LAW_TRGT_IF_IFC)
```

All valid size options can be found in arch/powerpc/include/asm/fsl_law.h. The LAW target options can also be found in this header file. If the base address of a LAW needs to be changed, for example CONFIG_SYS_FLASH_BASE_PHYS, the relevant parameter in the board configuration files covered in Step 2: Create a board configuration file should be changed to the correct value for the new platform.

- tlb.c Defines translation lookaside buffers to handle memory management in regions such as DDR, CCSR space, flash and PCIe.

As with all files in this section the best approach is to start with the T4240QDS file and modify as required for the new platform.

- Makefile Makefile is used to compile files listed above.

2.3.4 Step 4: Updating the device memory map (optional)

The device memory map is defined in arch/powerpc/include/asm/immap_85xx.h.

There are, however, some registers within the device that may not be currently exposed to users for use within board configuration code. If the user requires access to these registers the memory map can be updated to accommodate this, e.g. the code below defines the offset required for the serdes 4 block.

```
#define CONFIG_SYS_FSL_CORENET_SERDES4_OFFSET    0xED000

…

#define CONFIG_SYS_FSL_CORENET_SERDES4_ADDR \

(CONFIG_SYS_IMMR + CONFIG_SYS_FSL_CORENET_SERDES4_OFFSET)
```

Once defined the identifier can be used within board initialization code to read/write registers within this block.

2.3.5 Step 5: Adding new errata workarounds (optional)

If the user requires errata workarounds that may not be currently supported in the SDK the following files can be modified.

NOTE

The actual implementation of the errata workaround is not defined here since that will be unique to each errata.

- arch/powerpc/cpu/mpc85xx/cmd_errata.c

Once running on the platform U-Boot supports an ‘errata’ command which will list all the errata workarounds implemented in the particular build. To ensure any new errata workarounds are included when the command is executed users should update cmd_errata.c to include a new entry in the format shown below.

```
#ifdef CONFIG_SYS_FSL_ERRATUM_A005977
            if (IS_SVR_REV(svr, 1, 0))
            puts("Work-around for Erratum A005977 enabled\n");
            #endif
```

- arch/powerpc/include/asm/config_mpc85xx.h

The config_mpc85xx.h file includes SoC specific defines for Freescale MPC85xx and QorIQ processors. This includes values such as the maximum number of CPUs (CONFIG_MAX_CPUS) and the number of DDR controllers (CONFIG_NUM_DDR_CONTROLLERS). Also defined for each processor are the errata workarounds that should be implemented, e.g.

```
 #define CONFIG_SYS_FSL_ERRATUM_A005977
```

Therefore if new workarounds are being added a #define should be added for each to ensure all workaround code is controlled at this level.

2.3.6 Step 6: Updating Ethernet PHY drivers (optional)

If the Ethernet PHY used is not already supported within U-Boot the PHY driver code can be updated to add support for the new device. For example, if the user wishes to add a new Broadcom PHY they should follow the example below shown for the BCM54616S PHY.

- In drivers/net/phy/broadcom.c create a new structure in the format below. In this case the config, startup and shutdown functions are reuse of already existing routines but if the device has different programming requirements these can be changed to call new functions.

```
Static struct phy_driver BCM54616S_driver = {
                .name = “Broadcom BCM54616S”,
                .uid = 0x3625d10,
                .mask = 0xffffff0,
                .features = PHY_GBIT_FEATURES,
                .config =&bcm5461_config,
                .startup = &bcm54xx_startup,
                .shutdown = &genphy_shutdown,
            };
```

- In drivers/net/phy/broadcom.c ensure the new PHY is registered by adding a phy_register() call, for example:

```
phy_register(&BCM54616S_driver);
```

For other PHY vendors users should refer to the drivers/net/phy options where other files are available such as vitesse.c.

2.4 Building U-Boot images

To build the U-Boot binary for the newly created platform users should open a terminal window at the top level U-Boot directory and run the **make target** command, for example:

make T4240SDP

The u-boot.bin file will be created within the top level directory and can then be flashed to the board for testing.

3 Porting Linux

This section covers the process of obtaining the Linux sources, modifying the source to support the new platform and building binary images that will execute on the platform.

3.1 Overview

Linux as an operating system in the embedded space is increasing in popularity and provides a stable, scalable solution that is updated frequently to include the latest device drivers, etc for embedded system components.

In a system the Linux operating system must be capable of bringing up the correct network interfaces and other system peripherals such as SATA and PCIe.

3.2 Obtaining the source code

The main Linux repository is at

```
git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git
```

Users can clone the tree from this location to port their new platform using the command below.

```
git clone git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git
```

The method detailed below uses the Freescale SDK package and will use the Linux kernel sources available within the SDK as a starting point for the port.

Information on how to extract/modify the kernel sources from the SDK is available on the infocenter website: http://www.freescale.com/infocenter/index.jsp?topic=%2FQORIQSDK%2F2880375.html.

The main steps are as follows:

1. Install the latest SDK release
2. In the main install directory create a build directory for the T4240QDS, i.e.
   - `source ./fsl-setup-poky -m t4240qds`
3. From the **<yocto_install_path>/build_t4240qds_release** directory extract the kernel source using the following command:
   - `bitbake -c patch virtual/kernel`
4. To find the location of the extracted kernel source run the following command and look for the source **S=** result
   - `bitbake -e virtual/kernel | grep ^S`

   The output will look something like:
   - `S="/home/Projects/sdk-1.4/QorIQ-SDK-V1.4-20130625-yocto/build_t4240qds_release/tmp/work/t4240qds-fsl_networking-linux/linux-qoriq-sdk/3.8-r14.3/git"`
5. **cd** to the source directory given by **S=**
6. Set <ARCH> and <CROSS_COMPILE> to the correct values:
   - `export ARCH=powerpc`
   - Using **export CROSS_COMPILE=** set the path to the relevant cross compile tools within the **/opt** directory on your system
7. Check the build process by building the T4240QDS binary, i.e.
   - Run **make 85xx/e6500rev1_defconfig**
   - Run **make uImage**
   - Check the uImage file is created within the **arch/powerpc/boot** directory of Linux

The source code in the directory above can now be modified for the new platform. The steps in this application note assume the kernel is built manually, i.e. not using bitbake. If users prefer to use bitbake they should refer to the infocenter QorIQ instructions/FAQs for Linux.

3.3 Creating a new platform

As with U-Boot adding support for a new platform involves creating new files specific to the board and updating existing kernel code to support features on the board that may not already be part of the kernel.

The steps below outline the process of completing both these tasks.

3.3.1 Step 1: Create a new board initialization file

The **arch/powerpc/platforms/85xx** directory contains initialisation files for supported platforms.

In this example a developer could create a **t4240_sdp.c** file which includes set-up such as interrupt and power saving functions as shown below. As most of this file is generic across all T4240 platforms it is best to start with a known working example such as the T4240QDS and modify as required.

```
define_machine(t4240_sdp) {
            .name            = "T4240 SDP",
            .probe            = t4240_sdp_probe,
            .setup_arch        = corenet_ds_setup_arch,
            .init_IRQ        = corenet_ds_pic_init,
            #ifdef CONFIG_PCI
            .pcibios_fixup_bus    = fsl_pcibios_fixup_bus,
            #endif
            /* coreint doesn't play nice with lazy EE, use legacy mpic for now */
            #ifdef CONFIG_PPC64
            .get_irq        = mpic_get_irq,
            #else
            .get_irq        = mpic_get_coreint_irq,
            #endif
            .restart        = fsl_rstcr_restart,
            .calibrate_decr        = generic_calibrate_decr,
            .progress        = udbg_progress,
            #ifdef CONFIG_PPC64
            .power_save        = book3e_idle,
            #else
            .power_save        = e500_idle,
            #endif
            .init_early        = corenet_ds_init_early,
            };
```

3.3.2 Step 2: Update Kconfig and Makefile

To provide build support for the new platform the user must make sure Kconfig and Makefile within the **arch/powerpc/platforms/85xx** directory have the relevant updates as shown below.

- In Kconfig add an entry for the new platform, e.g.

```
config T4240_SDP
            bool "Freescale T4240 SDP"
            select DEFAULT_UIMAGE
            select E500
            select PPC_E500MC
            select PHYS_64BIT
            select SWIOTLB
```

```
        select ARCH_REQUIRE_GPIOLIB
        select GPIO_MPC8XXX
        select PPC_EPAPR_HV_PIC
        select HAS_FSL_QBMAN
        help
```

This option enables support for the T4240 SDP board

- Update the Makefile to include the build option for the new platform, e.g.

```
obj-$(CONFIG_T4240_SDP)    += t4240_sdp.o corenet_ds.o
```

NOTE

The first file name specified for the board should match the board initialization file created in Step 1: Create a new board initialization file

3.3.3 Step 3: Create Kernel configuration file

The kernel configuration settings are held in a **.config** file located in the top level directory.

The content of this file is updated during the build process when the user specifies a configuration file for the board.

The board specific kernel options are defined in a **target_defconfig** file located in the **arch/powerpc/configs/85xx** directory. The user should create a new file such as, **t4240sdp_defconfig**. In the example, T4240-SDP board the platform support section of this file would contain an entry for **CONFIG_T4240_SDP=y**.

3.3.4 Step 4: Create a device tree

The device tree is a data structure for describing the hardware. Linux is passed this information at boot time and it therefore avoids hardware details being hard coded within the operating system.

Bindings are used within the device tree to define typical hardware features such as data busses and peripheral devices. Users should refer to the **Documentation/devicetree/bindings** directory for a breakdown of the available bindings and examples of each case.

When porting to a new T4240 platform the best approach would be to start from the T4240QDS device tree (t4240qds.dts) and modify to match the hardware characteristics of the new platform. For example, the T4240QDS device tree has Integrated Flash Controller (IFC) entries for various options such as NAND flash that are not used on the T4240-SDP. The IFC entry can therefore be simplified to the settings below where only NOR flash is required. The ranges field is where the user sets the size, in this case 0x8000000 for 128MB and the start address, e.g. 0xFE8000000. Note that in this case a 36 bit address map is assumed so the user must make sure this is taken into account when defining the ranges.

```
ifc: localbus@ffe124000 {
        reg = <0xf 0xfe124000 0 0x2000>;
        ranges = <0 0 0xf 0xe8000000 0x08000000>;

        nor@0,0 {
                #address-cells = <1>;
                #size-cells = <1>;
                compatible = "cfi-flash";
                reg = <0x0 0x0 0x8000000>;

                bank-width = <2>;
                device-width = <1>;
        };
};
```

Developers should ensure that the U-Boot addressing for LAWs and TLBs is inline with the addressing defined in the device tree.

3.3.5 Step 5: Update Ethernet PHY driver (optional)

As with U-Boot if the Ethernet PHY used on the T4240 platform is not already supported within Linux the PHY driver code can be updated to add support for the new device. E.g. if the user wishes to add a new Broadcom PHY they should follow the example below shown for the BCM54616S PHY.

- In **drivers/net/phy/broadcom.c** update broadcom_drivers[] to include an entry for the new PHY. In this case the **config_init, etc** functions are reuse of already existing routines but if the device has different programming requirements these can be changed to call new functions.

```
.phy_id         = PHY_ID_BCM54616,
                                .phy_id_mask    = 0xfffffff0,
                                .name         = "Broadcom BCM54616",
                                .features     = PHY_GBIT_FEATURES |
                                 SUPPORTED_Pause | SUPPORTED_Asym_Pause,
                                .flags         = PHY_HAS_MAGICANEG | PHY_HAS_INTERRUPT,
                                .config_init    = bcm54xx_config_init,
                                .config_aneg    = genphy_config_aneg,
                                .read_status    = genphy_read_status,
                                .ack_interrupt    = bcm54xx_ack_interrupt,
                                .config_intr    = bcm54xx_config_intr,
                                .driver         = { .owner = THIS_MODULE },
```

  The final step in broadcom.c is to add an entry for the PHY to the broadcom_tbl structure, like this:

```
{ PHY_ID_BCM54616, 0xfffffff0 }
```

- In **include/linux/brcmphy.h** add a **#define** for the new PHY to specify what the PHY identifier register values are (this value should be provided in the PHY datasheet).

```
#define PHY_ID_BCM54616    0x03625d12
```

NOTE

Users should ensure the device drivers section in the kernel config file (Step 3: Create Kernel configuration file) includes the correct PHY option for the board, e.g. **CONFIG_BROADCOM_PHY=y**.

3.4 Building Linux images

To build the Linux binary for the newly created platform users should open a terminal window at the top level Linux directory and run the following commands:

- **make config**, like this:

```
make 85xx/t4240sdp_defconfig
```

  This step reads the kernel configuration options from the specified file and these are stored in the **.config** file at the top level directory. If the user wishes add/remove options the **make menuconfig** command can be used at this stage to bring up a user interface showing all available options.
- **make uImage**
- **make t4240sdp.dtb**

The kernel binary (uImage) file and the device tree binary (e.g. t4240sdp.dtb) will be created within the **arch/powerpc/boot** directory. These can then be flashed to the new platform and tested using the U-Boot build created in Porting U-Boot.

4 Revision history

This table summarizes revisions to this document.

Table 1. Revision history

Revision	Date	Description
0	10/2013	Initial public release.

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number AN4814
Revision 0, 10/2013