Digi® Embedded Yocto ConnectCard

U-Boot and Linux BSP

User Guide

Revision history—90001389

Revision	Date	Description
A	July, 2014	Initial release
B	November, 2014	Added updates for U-Boot v2013.01 updates.
C	April, 2017	Updated branding and made editorial enhancements.

Trademarks and copyright

Disclaimers

Warranty

To view product warranty information, go to the following website:

www.digi.com/howtobuy/terms

Send comments

Documentation feedback: To provide feedback on this document, send your comments to techcomm@digi.com.

Customer support

Digi Technical Support: Digi offers multiple technical support plans and service packages to help our customers get the most out of their Digi product. For information on Technical Support plans and pricing, contact us at +1 952.912.3444 or visit us at www.digi.com/support.

Contents

Flattened Device Tree in U-Boot and BSP design

The Flattened Device Tree (FDT, or simply DT) is a data structure for describing the hardware in a system. Instead of hard coding every detail of a device into the operating system, data structure describes many aspects of the hardware that are passed to the operating system at boot time.

The data structure is a simple tree of named nodes and properties. Nodes contain properties and child nodes. Properties are simple name-value pairs. The structure can hold any kind of data. The format is expressive and can describe most board design aspects including:

- The number and type of CPUs
- Base addresses and size of RAM
- Buses and bridges
- Peripheral device connections
- Interrupt controllers and IRQ line connections

Advantages

- Ship one FDT image per machine (a few kB) instead of one kernel image per machine (several MB).
- Reduce or eliminate effort needed to write machine support code (such as **arch/arm/mach-***). Most board specific code changes are constrained to FDT file and device drivers.
- No need to allocate a new global ARM machine identity number for each new board variant.
- Reduce the need to recompile the kernel. One kernel image with support for different hardware can be shipped and run in different variants (each one with its own FDT describing the hardware that is available).
- Expressive format to describe related board variants without allocating new machine numbers or new ATAGs.
- U-Boot firmware can inspect and modify an FDT image before booting.

Formats

- *.dts: This is a Device Tree file in plain text (human readable).
- *.dtsi: This is like a DTS include file (a plain text file to be included by a DTS file).
- *.dtb: This is a Device Tree Blob, a binary representation of a Device Tree, once compiled with the Device Tree compiler.

U-Boot v2013.01

U-Boot v2013.01 allows booting recent Linux kernels that support Device Tree. This manual does not cover U-Boot. For more information on U-Boot, see the *U-Boot v2009 Reference Manual*, part number 90001129 at www.digi.com/support. It describes the differences introduced by the need for Device Tree in new kernels. It also describes the differences between the new v2013.01 and the old v2009.08 versions.

Note U-Boot v2009.08 can still boot a v3.10 kernel. A special kernel image with appended Device Tree at the end is needed for that.

Supported platforms

The Digi supported platforms in U-Boot v2013.01 are:

- ConnectCard for i.MX28 (ccardimx28js)
- ConnectPort X2 (cpx2)

U-Boot commands

U-Boot has a command called **fdt** (with many subcommands) to manage Device Tree Blobs.

```
# help fdt
fdt - flattened device tree utility commands
Usage:
fdt addr <addr> [<length>] - Set the fdt location to <addr>
fdt boardsetup - Do board-specific set up
fdt move <fdt> <newaddr> <length> - Copy the fdt to <addr> and make it
active
fdt resize - Resize fdt to size + padding to 4k addr
fdt print <path> [<prop>] - Recursive print starting at <path>
fdt list <path> [<prop>] - Print one level starting at <path>
fdt get value <var> <path> <prop> - Get <property> and store in <var>
fdt get name <var> <path> <index> - Get name of node <index> and store
in <var>
fdt get addr <var> <path> <prop> - Get start address of <property> and
store in
<var>
fdt get size <var> <path> [<prop>] - Get size of [<property>] or num
nodes and st
ore in <var>
fdt set <path> <prop> [<val>] - Set <property> [to <val>]
fdt mknode <path> <node> - Create a new node after <path>
fdt rm <path> [<prop>] - Delete the node or <property>
fdt header - Display header info
fdt bootcpu <id> - Set boot cpuid
fdt memory <addr> <size> - Add/Update memory node
fdt rsvmem print - Show current mem reserves
fdt rsvmem add <addr> <size> - Add a mem reserve
fdt rsvmem delete <index> - Delete a mem reserves
fdt chosen [<start> <end>] - Add/update the /chosen branch in the tree
<start>/<end> - initrd start/end addrNote: Dereference aliases by
omitting the leading
'/', e.g. fdt print ethernet0.
```

The main subcommand is:

```
fdt addr <addr> [<length>]
```

The **addr** subcommand tells U-Boot the RAM address where the Device Tree is located. It must have been previously copied to RAM. This step is required previous to any other **fdt** subcommand. If the optional parameter **<length>** is passed (in bytes -hex-), this establishes the length limit of the Device Tree, which means the Device Tree is unable to grow beyond this limit. Usually, it is convenient to pass a number a little larger than the actual size of the DTB, so we can manually append additional nodes from U-Boot, if needed.

Other commands allow you to:

- Print the Device Tree (in human-readable form), or the contents of a certain node (if a path is provided).

- Create or modify properties and nodes.
- Read property values into environment variables, etc.

Environment variables

The following dynamic variables related to Device Tree are generated:

Variable	Description
fdtimg	Filename of platform's Device Tree Blob, such as, `imx28-ccardimx28js.dtb`.
fdtaddr	Address in RAM where to load the Device Tree Blob during **dboot** command.

Flash partition for Device Tree

When booting from flash, the Device Tree Blob needs a partition of its own. This only needs to be a few kilobytes, but for security reasons (bad blocks prevention) a four-erase block area is reserved, just like for NVRAM partition. The default partition table now has a layout like this:

```
Nr | Name         | Start    | Size    | Type                 | FS    | Flags
----------------------------------------------------------------------------
------------
0  | Bstrm-U-Boot | 0        | 3 MiB   | Bootstream           |       | fixed
1  | NVRAM        | 3 MiB    | 512 KiB | NVRAM                |       | fixed
2  | FDT          | 3584 KiB | 512 KiB | Device Tree          |       |
3  | Kernel       | 4 MiB    | 5 MiB   | Linux/Android-Kernel |       |
4  | RootFS       | 9 MiB    | 115 MiB | Filesystem           | UBIFS | rootfs
5  | UserFS       | 124 MiB  | 132 MiB | Filesystem           | UBIFS |
```

This is a raw partition (no filesystem). Similar to other key flash partitions, the new FDT partition has a reserved short name **fdt** that can be used for updating the Device Tree with the **update** command:

```
# update fdt tftp
```

Direct booting with dboot

The **dboot** command syntax now supports Device Tree.

The filename of the Device Tree blob now becomes an optional parameter of the **dboot** syntax (if not provided, the filename from variable **fdtimg** will be used instead).

If booting from RAM, the address where the DTB is located is also an optional parameter (if not provided, the address from variable **fdtaddr** will be used instead).

```
# help dboot
dboot - Digi modules boot commands

Usage:
dboot <os> [source] [extra-args...]
Description: Boots <os> via <source>
Arguments:
    - os: a partition name or one of the reserved names:
          linux|android
    - [source]: tftp (default)|flash|nfs|usb|mmc|hsmmc|sata|ram
    - [extra-args]: extra arguments depending on 'source'

    source=tftp|nfs -> [filename] [fdtfilename]
```

```
        - filename: kernel file to transfer (required if using a partition name)
        - fdtfilename: DTB file to transfer

    source=usb|mmc|hsmmc|sata -> [device:part filesystem] [filename] [rootfspart]
[fdtfilename]
        - device:part: number of device and partition
        - filesystem: fat|vfat|ext2|ext3
        - filename: kernel file to transfer
        - rootfspart: root parameter to pass to the kernel command line.
                      If omitted uses the one at variable
    usb_rpart|mmc_rpart|sata_rpart.
        - fdtfilename: DTB file to transfer

        source=ram -> [image_address] [initrd_address] [initrd_max_size] [fdt_
address]
        - image_address: address of image in RAM
        - initrd_address: address of initrd image (default: loadaddr_initrd)
        - initrd_max_size: max. allowed ramdisk size (in kB) to pass to the
kernel
          (default: kernel default)
        - fdt_address: address of DTB image in RAM
```

Other differences in v2013.01 U-Boot

Integrated bootlets code

Integrated bootlets is a Secondary Program Loader (SPL) that configures the power rails and the SDRAM controller of the i.MX28 SoC before running U-Boot. In the past, the bootlets code was external to U-Boot. Now the code is integrated into U-Boot source tree and built together with the rest of the boot loader.

Support for conditional scripting

Conditional statements if-then/else are now supported and can be used to do complex scripts. The only condition supported is the result (success or error) of executing a command. Syntax is:

```
if <command>; then
     <command>;
else
     <command>;
fi;
```

These conditional statements can be nested. The semi-colon characters must be escaped with a backslash.

Expressions evaluation

The new **setexpr** command allows computing mathematical and logical operations on a pair of values:

```
# help setexpr
etexpr - set environment variable as the result of eval expression

Usage:
setexpr [.b, .w, .l] name [*]value1 <op> [*]value2
       - set environment variable 'name' to the result of the evaluated expression
       specified by <op>. <op> can be &, |, ^, +, -, *, /, %
```

```
    size argument is only meaningful if value1 and/or value2 are
    memory addresses (*)
setexpr[.b, .w, .l] name *value
       - load a memory address into a variable
```

Note Some of the **<op>** characters must be escaped with a backslash. See the following example.

```
  # printenv val
  val=5
  # setexpr val 4 \| 1
  # printenv val
  val=5
  # setexpr val 4 \& 1
  # printenv val
  val=0
```

Note The abbreviated form set no longer maps to the **setenv** command, because U-Boot can't decide whether set refers to the **setenv** or **setexpr** command.

Dual Boot

The Dual Boot mechanism is not supported in the v2013.01 U-Boot.

U-Boot Environment and NVRAM

The U-Boot environment area has been enlarged from 8K to 16K. Migration of NVRAM and U-Boot environment area is automatic from a v2009.08 U-Boot to v2013.01. You only need to do a **saveenv** to save NVRAM after upgrading U-Boot.

Support for old kernels

Old kernels (without Device Tree) can still be booted by the bootloader. However, if using the **dboot** command, be aware of the following limitations:

- **Booting from TFTP**: The v2013.01 U-Boot expects you to provide a DTB filename to download from TFTP. If not provided, it will try to download the filename stored in the **fdtimg** variable. If the file does not exist in the TFTP server, the **dboot** command assumes the kernel does not need a Device Tree.
- **Booting from Flash**: If an FDT partition exists in the flash partition table, the **dboot** command assumes you are booting a kernel that requires Device Tree. If you want to boot an old kernel that does not need Device Tree you must remove the FDT partition from the flash partition table.

- **Command line arguments**: The **dboot** command automatically generates a **bootargs** variable that carries the kernel command line arguments, depending on the selected boot media. The v2013.01 U-Boot generates variables (like console variable) with default values for a v3.10 kernel. Old kernels may require you to pass different arguments.
  For **ccardimx28**:
  - The default value for the console variable is **ttyAMA0**,**115200** for the v3.10 kernel, but it should be **ttyAM0**,**115200** for a v2.6.35 kernel.
  - The v2009.08 U-Boot passes the argument **gpmi** to the command line (needed to probe the NAND driver), but v2013.01 does not, because this is not required for the v2013.01 NAND driver. To have a v2009 kernel properly probe the NAND driver you must supply the **gpmi** parameter (for example using the **std_bootarg** variable).
  - The **mtdparts** argument must start with the NAND driver name, which is **gpmi-nfc-nand** for a v2.6.35 kernel but **gpmi-nand** for a v3.10.

UBIFS root file system

The UBIFS file system offers better performance and wear leveling capabilities than JFFS2. It is the default file system for the root file system partition.

To update a UBIFS partition you must attach the UBI volume to an MTD partition and then update the volume with a UBIFS image. For example, to update an unmounted UserFS partition (/dev/mtd5) you would do:

```
# ubiattach /dev/ubi_ctrl -m 5
# ubiupdatevol /dev/ubi1_0 <UBIFS_image>
```

You can then mount the volume with:

```
# mount -t ubifs /dev/ubi1_0 <mount_point>
```

Note To update a UBI volume, the MTD partition must be unmounted, so it is not possible to update the RootFS partition if you are running from Flash. Use U-Boot update command or boot from NFS to update the RootFS partition.

Migrating U-Boot from v2009.08 to v2013.01

NVRAM update

After updating the U-Boot partition with a v2013.01 U-Boot image, the first boot complains that the U-Boot environment area size has changed and has been automatically resized.

```
  U-Boot 2013.01 - dub-2.0.0.4 (Oct 17 2013 - 13:14:53) - GCC 4.4.6
  for ConnectCard for i.MX28
  CPU: Freescale i.MX28 rev1.2 at 454 MHz
  BOOT: NAND, 1V8
  I2C: ready
  DRAM: 256 MiB
  NAND: 256 MiB
  MMC: MXS MMC: 0
  *** Warning - U-Boot ENV size in NVRAM is 0x2000 but should be
0x4000. Resizing .
  ..
  In: serial
  Out: serial
  Err: serial
  Net: FEC0 [PRIME]
  Autoscript from TFTP... [not available]
  Hit any key to stop autoboot: 0
```

To avoid the warning on future boot cycles, save the environment with **saveenv**.

Flash partition table

A new partition is required to hold the Device Tree. You can reset the partition table to the new defaults for Linux (**flpart** --> **r** --> **l**) or Android (**flpart** --> **r** --> **a**), or create an FDT partition manually. You must also update the new partition with your Device Tree Blob file.

Note Downgrading from a v2013.01 U-Boot to a v2009.08 is not supported, and U-Boot complains of an invalid **NVRAM** in this scenario. If you must downgrade U-Boot, first print the contents of the Environment (with **printenv**) and **NVRAM** (with **intnvram printall**) so you can manually restore them after the downgrade.

Device Tree guide to ConnectCard for i.MX28

As explained in the introduction to the Device Tree, the kernel now rarely needs to be recompiled to include or exclude support for a certain piece of hardware. The description of your hardware is now created in the Device Tree file, which is a small text file that compiles faster into a Device Tree Blob (DTB).

This document explains how to modify the Device Tree to enable/disable each of the hardware interfaces available in the ConnectCard for i.MX28 platform.

Platform Device Tree

The Device Tree configuration for the ConnectCard for i.MX28 is located in two places:

- Hardware fragments: **arch/arm/boot/dts/digi/ccardimx28/*.dtsi**
- Board DTS: **arch/arm/boot/dts/imx28-ccardimx28js.dts**
  - The hardware fragments are small fragments of Device Tree that configure a specific hardware interface or feature.
  - The board Device Tree defines the ConnectCard for i.MX28 JumpStart kit hardware by including different hardware fragments.

Configuring the platform Device Tree

To include a hardware interface or feature from the board Device Tree, edit the file **arch/arm/boot/dts/imx28_ccardimx28js.dts** and uncomment the selected fragment. This adds it to the final Device Tree.

To exclude it, comment the selected fragment, so it does not appear in the final Device Tree.

Note Be aware of the conflicts between different multiplexed interfaces in the ConnectCard for i.MX28. Not all hardware can coexist at the same time. For more information on hardware conflicts, see Interfaces. You can also refer to the *Hardware Reference Manual*.

Compiling a Device Tree

The platform DTS is compiled by Digi Embedded Yocto into a DTB file (Device Tree Blob). This is the file that must be programmed to the **fdt** flash partition, so U-Boot can read it and pass it to the kernel.

Creating a custom platform DTS

To create a new hardware platform:

1. Start by copying the JumpStart board DTS file to a new file named after the product/board.
2. Modify this Device Tree file according to the hardware requirements.
3. Include the new file into the **arch/arm/boot/dts/Makefile**. It must be similar to the JumpStart board DTS to be compiled. You can also modify the hardware DTSI fragments at **arch/arm/boot/dts/digi/ccardimx28** or add new fragments.

Interfaces

The following sections list the interfaces and their Device Tree entries. Each interface is listed with the path to its fragment DTS file.

To enable an interface, uncomment the include path in the Board DTS file **arch/arm/boot/dts/imx28-ccardimx28js.dts**.

Additionally, if the interface has conflicts with other interfaces, you will be warned to disable them.

Note Apart from the software configurations, refer to the hardware reference manual of the board, for specific hardware configurations needed to enable each interface, such as jumpers and micro-switches.

ADC

Include	/include"digi/ccardimx28/cccardimx28_lradc.dtsi:"
Conflicts	Different LRADC channels conflict with different interfaces. The Device Tree entry for the LRADC, however, may remain enabled to use those channels that do not have conflicts. ■ LRADC0..3 conflict with SSP1 ■ LRADC2..5 conflict with touch screen ■ LRADC4 conflicts with: ■ AUART1_CTS: ■ Ethernet 0 link LED ■ LRADC5 conflicts with: ■ AUART1_RTS: ■ Ethernet 0 activity LED ■ LRADC6 conflicts with sound

Bluetooth

The MAC address for the Bluetooth interface is acquired from the U-Boot environment variable **btaddr**, which is populated by U-Boot on the Device Tree before booting Linux.

There is no generic Device Tree binding for the Bluetooth interface. Digi created a bluetooth entry node that passes the MAC address (populated by U-Boot) to the driver. The Bluetooth interface requires enabling AUART0. See AUART0 where the Bluetooth interface is connected.

CAN

Includes	`/include/ "digi/ccardimx28/ccardimx28_can0.dtsi"`
	`/include/ "digi/ccardimx28/ccardimx28_can1.dtsi"`

Conflicts	**CAN1 conflicts with:**
	■ **AUART3:**
	`/include/ "digi/ccardimx28/ccardimx28_auart3_2wires.dtsi"`
	or
	`/include/ "digi/ccardimx28/ccardimx28_auart3_4wires.dtsi"`
	■ **Ethernet1 LEDs**
	`/include/ "digi/ccardimx28/ccardimx28_ethernet1_leds.dtsi"`
	which must be disabled.

CPU Frequency scaling

Include	`/include/ "digi/ccardimx28/ccardimx28_cpufreq.dtsi"`

Ethernet

Includes	`/include/ "digi/ccardimx28/ccardimx28_ethernet0.dtsi"`
	`/include/ "digi/ccardimx28/ccardimx28_ethernet1.dtsi"`
	Modules with only one Ethernet interface should disable ethernet1.

Conflicts	**Ethernet 1 conflicts with USB1, which must be disabled:** `/include/ "digi/ccardimx28/ccardimx28_usb1.dtsi"`

Ethernet LEDs

Includes	`/include/ "digi/ccardimx28/ccardimx28_ethernet0_leds.dtsi"`
	`/include/ "digi/ccardimx28/ccardimx28_ethernet1_leds.dtsi"`

Conflicts	Ethernet 0 link LED conflicts with:
	▪ **AUART1_CTS:**
	`/include/ "digi/ccardimx28/ccardimx28_auart1_4wires.dtsi"`
	▪ **LRADC4/touch-screen:**
	`/include/ "digi/ccardimx28/ ccardimx28_lradc_touchscreen.dtsi"`
	which must be disabled.
	Ethernet0 activity LED conflicts with:
	▪ **AUART1_RTS:**
	`/include/ "digi/ccardimx28/ccardimx28_auart1_4wires.dtsi"`
	▪ **LRADC5/touch-screen:**
	`/include/ "digi/ccardimx28/ ccardimx28_lradc_touchscreen.dtsi" which must be disabled.`
	Ethernet1 link LED conflicts with:
	▪ **AUART3_TX:**
	`/include/ "digi/ccardimx28/ccardimx28_auart3_2wires.dtsi" or`
	`/include/ "digi/ccardimx28/ccardimx28_auart3_4wires.dtsi"`
	▪ **CAN1_TX:**
	`/include/ "digi/ccardimx28/ccardimx28_can1.dtsi"`
	which must be disabled.
	Ethernet1 activity LED conflicts with:
	▪ **AUART3_RX:**
	`/include/ "digi/ccardimx28/ccardimx28_auart3_2wires.dtsi" or`
	`/include/ "digi/ccardimx28/ccardimx28_auart3_4wires.dtsi"`
	▪ **CAN1_RX:**
	`/include/ "digi/ccardimx28/ccardimx28_can1.dtsi"`
	which must be disabled.

Flash

Include	`include/ "digi/ccardimx28/ccardimx28_nand.dtsi"`

GPIO

No special Device Tree entries are required to use the GPIOs via the standard **gpiolib sysfs** interface.

Graphics

Include	`/include/ "digi/ccardimx28/ ccardimx28_display_ lq70y3dg3b.dtsi"`

Conflicts	**LCD backlight conflicts with:**
	■ **AUART1:**
	`/include/ "digi/ccardimx28/ ccardimx28_auart1_ 2wires.dtsi" or`
	`/include/ "digi/ccardimx28/ ccardimx28_auart1_ 4wires.dtsi"`
	■ **User LED2 (no device tree entry)**
	which must be removed.

I2C

I2C0

Include	`/include/ "digi/ccardimx28/ccardimx28_i2c0.dtsi"`
Conflicts	**I2C0 conflicts with DUART which must be disabled:** `/include/ "digi/ccardimx28/ccardimx28_duart_i2c0.dtsi"`

I2C1

Include	`/include/ "digi/ccardimx28/ccardimx28_i2c1.dtsi"`

Power management

Include	`/include/ "digi/ccardimx28/ccardimx28_power.dtsi"`

Serial

AUART0

Connected to Bluetooth on modules with Atheros wireless chip.

Include	`/include/ "digi/ccardimx28/ccardimx28_auart0_bluetooth.dtsi"` **AUART0 is internally connected to the Bluetooth chip in the module (where available). Must be disabled in modules without Bluetooth.**

AUART1

Include	`/include/ "digi/ccardimx28/ccardimx28_auart1_2wires.dtsi"`
	`/include/ "digi/ccardimx28/ccardimx28_auart1_4wires.dtsi`

Conflicts	**AUART1 conflicts with:** ■ User LED1 and LED2 (no device tree entry) ■ LCD backlight/PWM0: `/include/ "digi/ccardimx28/ccardimx28_display_ lq70y3dg3b.dtsi"` **which must be disabled.**

AUART2

Include	`/include/ "digi/ccardimx28/ccardimx28_auart2_2wires.dtsi"` `/include/ "digi/ccardimx28/ccardimx28_auart2_4wires.dtsi"`
Conflicts	**AUART2 CTS conflicts with:** ■ **1-wire interface (if available in module):** `/include/ "digi/ccardimx28/ccardimx28_onewire_i2c1.dtsi"` **which must be disabled.**

AUART3

Include	`/include/ "digi/ccardimx28/ccardimx28_auart3_2wires.dtsi"` `/include/ "digi/ccardimx28/ccardimx28_auart3_4wires.dtsi"`

Conflicts	**AUART3 conflicts with:**
	 ■ CAN1
	 `/include/ "digi/ccardimx28/ccardimx28_can1.dtsi"`
	 ■ Ethernet1 LEDs
	 `/include/ "digi/ccardimx28/ccardimx28_ethernet1_leds.dtsi"`
	 which must be disabled.
	 ■ AUART3 CTS/RTS conflict with SSP0:
	 `/include/ "digi/ccardimx28/ccardimx28_ssp0_mmc.dtsi" or`
	 `/include/ "digi/ccardimx28/ccardimx28_ssp0_spi.dtsi" or`
	 `/include/ "digi/ccardimx28/ccardimx28_ssp0_spi_gpio.dtsi"`
	 which must be disabled.

AUART4

Include	`/include/ "digi/ccardimx28/ccardimx28_auart4_2wires.dtsi"`
	`/include/ "digi/ccardimx28/ccardimx28_auart4_4wires.dtsi"`

Conflicts	**AUART4 conflicts with:**
	■ **- SSP3**
	`/include/ "digi/ccardimx28/ccardimx28_ssp3_spi.dtsi" or`
	`/include/ "digi/ccardimx28/ccardimx28_ssp3_spi_gpio.dtsi"`
	■ **- Sound:**
	`/include/ "digi/ccardimx28/ccardimx28_sound_i2c1.dtsi"`
	which must be disabled.

DUART

Include	`/include/ "digi/ccardimx28/ccardimx28_duart_i2c0.dtsi"`
Conflicts	**DUART conflicts with I2C0 which must be disabled:** `/include/ "digi/ccardimx28/ccardimx28_i2c0.dtsi"`

Sound

Include	/include/ "digi/ccardimx28/ccardimx28_sound_i2c1.dtsi"

Conflicts	**Audio conflicts with:**
	▪ **- SSP3**
	`/include/ "digi/ccardimx28/ccardimx28_ssp3_spi.dtsi"` or
	`/include/ "digi/ccardimx28/ccardimx28_ssp3_spi_gpio.dtsi"`
	▪ **- AUART3:**
	`/include/ "digi/ccardimx28/ccardimx28_auart3_2wires.dtsi"`
	or
	`/include/ "digi/ccardimx28/ccardimx28_auart3_4wires.dtsi"`
	▪ **- LRADC6**
	which must be disabled.

SSP

SSP0

Include (MMC)	`/include/ "digi/ccardimx28/ccardimx28_ssp0_mmc.dtsi"`
Include (SPI)	`/include/ "digi/ccardimx28/ccardimx28_ssp0_spi.dtsi"`
Include (SPI bit banging)	`/include/ "digi/ccardimx28/ccardimx28_ssp0_spi_gpio.dtsi"`

SSP1

Include (SPI)	`/include/ "digi/ccardimx28/ccardimx28_ssp1_spi.dtsi"`
Include (SPI bit banging)	`/include/ "digi/ccardimx28/ccardimx28_ssp1_spi_gpio.dtsi"`
Conflicts	**SSP1 conflicts with LRADC0..3 and touch screen** `/include/ "digi/ccardimx28/ccardimx28_lradc_touchscreen.dtsi"` **which must be disabled.**

SSP2 (MMC connected to WiFi chip in module)

Include	`/include/ "digi/ccardimx28/ccardimx28_ssp2_mmc_wifi.dtsi"` **Modules without WiFi chip should disable this.**

SSP3

Include (SPI)	`/include/ "digi/ccardimx28/ccardimx28_ssp3_spi.dtsi"`
Include (SPI bit banging)	`/include/ "digi/ccardimx28/ccardimx28_ssp3_spi_gpio.dtsi"`

Conflicts	**SSP3 conflicts with:**
	■ **- AUART4:**
	`/include/ "digi/ccardimx28/ccardimx28_auart4_2wires.dtsi"`
	`or`
	`/include/ "digi/ccardimx28/ccardimx28_auart4_4wires.dtsi"`
	■ **- Audio:**
	`/include/ "digi/ccardimx28/ccardimx28_sound_i2c1.dtsi"`
	which must be disabled.

Touchscreen

Include	`/include/ "fdigi/ccardimx28/ccardimx28_lradc_touchscreen.dtsi"`

Conflicts	**The touchscreen uses channels LRADC2..5**
	
	LRADC0..3 conflict with SSP1 which must be disabled:
	
	`/include/ "digi/ccardimx28/ccardimx28_ssp1_spi.dtsi" and`
	`/include/ "digi/ccardimx28/ccardimx28_ssp1_spi_gpio.dtsi"`
	LRADC2..5 conflict with touch screen.
	
	LRADC4 conflicts with:
	■ **- AUART1_CTS:**
	`/include/ "digi/ccardimx28/ccardimx28_auart1_4wires.dtsi"`
	■ **- Ethernet 0 link LED**
	`/include/ "digi/ccardimx28/ccardimx28_ethernet0_leds.dtsi"`
	which must be disabled.
	LRADC5 conflicts with:
	■ **- AUART1_RTS:**
	`/include/ "digi/ccardimx28/ccardimx28_auart1_4wires.dtsi"`
	■ **- Ethernet 0 activity LED**
	`/include/ "digi/ccardimx28/ccardimx28_ethernet0_leds.dtsi"`
	which must be disabled.

USB

USB0

Include	`/include/ "digi/ccardimx28/ccardimx28_usb0.dtsi"`

USB1 (only available on modules with one Ethernet)

Include	`/include/ "digi/ccardimx28/ccardimx28_usb1.dtsi"`
Conflicts	**USB1 (if available in module) conflicts with Ethernet1:** `/include/ "digi/ccardimx28/ccardimx28_ethernet1.dtsi"` **which must be disabled.**

Wireless

The MAC address is taken from U-Boot environment variable **wlanaddr** which is populated by U-Boot on the Device Tree before booting Linux.

There is no generic Device Tree binding for the Wireless interface. Digi has created a wireless entry node to pass the driver the MAC address (filled-in by U-Boot) and the power down GPIO. Apart from that, the Wireless requires the enabling of SSP2. Refer to SSP2 to which the wireless interface is connected.

1-wire

Include	`/include/ "digi/ccardimx28/ccardimx28_onewire_i2c1.dtsi"`
Conflicts	**1-wire (if available in module) conflicts with AUART2_CTS :** `/include/ "digi/ccardimx28/ccardimx28_auart2_4wires.dtsi"` **which must be disabled.**

Devices and interfaces of ConnectCard for i.MX28 in kernel v3.10

This chapter complements the official Digi Embedded Linux documentation (available on any Digi Embedded Linux installation) with the differences the version v3.10 kernel brings with respect to previous BSP kernel v2.6.35.

Device Tree

The default kernel image supports all devices and interfaces present on the module and the JSK development board. Bootup is controlled from the Device Tree through the status property of each driver node, if the drivers are or are not probed during kernel.

To ease the enabling/disabling of devices, different DTSI files have been placed at **arch/arm/boot/dts/digi/ccardimx28**. They include sentences inserted in the platform DTS file **arch/arm/boot/dts/imx28-ccardimx28js.dts**.

You will hardly ever need to rebuild the kernel to enable/disable a specific interface of the platform.

Devices and interfaces

ADC

Loading the module

If enabled as a loadable module, load the driver with:

```
# modprobe mxs-lradc.ko
```

Identifying the ADC device in the Linux system

The LRADC driver is an IIO (industrial I/O) device and is exposed to the user space via the sysfs. For more information, refer to the kernel documentation at **Documentation/ABI/testing/sysfs-bus-iio** and at **drivers/staging/iio/Documentation/**.

Sysfs entries appear for each LRADC channel at **/sys/bus/iio/devices/iio:device0/** (which is a link to **/sys/devices/80000000.apb/80040000.apbx/80050000.lradc/iio:device0/**).

Managing the ADC device from user space

Each LRADC channel has three descriptors:

- **in_voltageX_raw**: Shows the decimal value of the last raw sample (this is read-only).
- **in_voltageX_scale**: Shows the current scale of the channel. This is the value you must multiply the raw value by, to get the voltage of the channel in mV (this is read/write).
- **in_voltageX_scale_available**: Shows the available scales you can select for this channel (this is read-only).

For example, to read the raw value of channel 0:

```
# cat in_voltage0_raw
2788
```

To read the current selected scale for channel 0:

```
# cat in_voltage0_scale
0.451660156
```

To guess the voltage measured in channel 0 we need to multiply the raw value by the scale factor:

2788 * 0.451660156 = **1259 mV**

To see the available scales of the channel:

```
# cat in_voltage0_scale_available
0.451660156 0.903320310
```

This means there are two scales available (the second one basically enables the optional divider by two of the LRADC channel).

To change the scale:

```
# echo 0.903320310 > in_voltage0_scale
```

If we read the raw value again, it reads half the original value, so when you multiply by the new scale, you get the same voltage.

You can input up to to 3.7V to the LRADC channels with the divider enabled.

Battery

Battery is not currently supported in the v3.10 kernel.

CAN bus

Using the CAN bus

The CAN driver is built using the SocketCAN networking stack. The kernel documentation at **Documentation/networking/can.txt** states that you must use the NETLINK interface to get/set CAN devices properties.

You can list the parameters that can be modified with the following command, if your file system contains the **iproute2** suite.

```
# ip link set can0 type can help
Usage: ip link set DEVICE type can
    [ bitrate BITRATE [ sample-point SAMPLE-POINT] ] |
    [ tq TQ prop-seg PROP_SEG phase-seg1 PHASE-SEG1
    phase-seg2 PHASE-SEG2 [ sjw SJW ] ]

    [ loopback { on | off } ]
    [ listen-only { on | off } ]
    [ triple-sampling { on | off } ]
    [ one-shot { on | off } ]
    [ berr-reporting { on | off } ]

    [ restart-ms TIME-MS ]
    [ restart ]

    Where: BITRATE := { 1..1000000 }
           SAMPLE-POINT := { 0.000..0.999 }
           TQ := { NUMBER }
           PROP-SEG := { 1..8 }
           PHASE-SEG1 := { 1..8 }
           PHASE-SEG2 := { 1..8 }
           SJW := { 1..4 }
           RESTART-MS := { 0 | NUMBER }
```

The CAN bus doesn’t have a bitrate set up by default, so you must configure the bitrate using this command before attempting communication.

Ethernet

In previous versions, the network parameters (IP address, netmask, and gateway) were configured in U-Boot. Then they were saved in the **NVRAM** and passed to the kernel via the command line or directly retrieved by the user space from the **NVRAM**.

In the new v3.10 BSP, network settings must be configured in the standard Linux way by saving them in the configuration file **/etc/network/interfaces**. The MAC address is acquired from the U-Boot environment variable **ethaddr** which is populated by U-Boot on the Device Tree before booting Linux.

GPIO

Managing GPIO pins from user space

The standard **gpiolib** API provides GPIO support. Use **sysfs** to provide access to GPIOs from the user space.

Refer to the kernel documentation at **Documentation/gpio.txt** for information about using the GPIO **sysfs** interface.

The old **gpio** external module driver has been deprecated.

Graphics interface

Backlight support

The number and value of backlight levels is defined on the Device Tree as integers with values between 0 and 255:

```
backlight {
    compatible = "pwm-backlight";
    pwms = <&pwm0 0 15000 1>; /* inverted duty */
    brightness-levels = <0 2 4 8 16 32 54 76 98
                       120 142 164 186 208 230 255>;
    default-brightness-level = <13>;
    status = "okay";
};
```

The default brightness level is defined in the Device Tree as well, as an index to the brightness levels list (starting from 0). Similarly, the brightness of the LCD backlight can be controlled from user space through the **sysfs** using indexes to the levels list.

For a detailed list of the available backlight controls at the sysfs, refer to the Linux kernel documentation at: **Documentation/ABI/stable/sysfs-class-backlight**.

Display selection

The display selection and definition of the display timings and parameters is done in the Device Tree:

```
apb@80000000 {
    apbh@80000000 {
        lcdif@80030000 {
            display = <&lq70y3dg3b>;
            status = "okay";
            lq70y3dg: display@0 {
                bits-per-pixel = <32>;
                bus-width = <18>;
                display-timings {
                    timing0: timing0 {
                        clock-frequency = <20000000>;
                        hactive = <800>;
                        vactive = <480>;
                        hback-porch = <129>;
                        hfront-porch = <110>;
                        vback-porch = <33>;
```

```
                            vfront-porch = <5>;
                            hsync-len = <5>;
                            vsync-len = <2>;
                            hsync-active = <0>;
                            vsync-active = <0>;
                            de-active = <1>;
                            pixelclk-active = <0>;
                        };
                    };
                };
            };
        };
    };
```

The **video** command in U-Boot has been removed. Additional displays can be added to the device Tree at build-time. For example, you can use the U-Boot **fdt** command to modify the device tree on the fly for display selection.

I2C bus

You must declare I2C slave devices in the Device Tree as children of the I2C bus node they are connected to and with their I2C slave address. As an example, look at the SGTL5000 audio chip that's connected to I2C1 on the JumpStart board of ConnectCard for i.MX28:

```
/ {
      apb@80000000 {
          apbx@80040000 {
             i2c1: i2c@8005a000 {
                 sgtl5000:codec@0a {
                    compatible = "fsl,sgtl5000";
                   reg = <0x0a>
                    VDDA-supply = <&reg_3p3v>;
                    VDDIO-supply = <&reg_3p3v>;
                 };
             };
          };
      };
};
```

Power Management

You must write **enabled** or **disabled** to the device's power wakeup descriptor to enable or disable an interrupt as wakeup.

GPIOs as wakeup interrupts

All GPIO pins on the same port share one unique interrupt. Enabling the common interrupt as wakeup source means that any GPIO pin of that port that is configured as interrupt will be able to wake the system up.

To configure a GPIO pin as interrupt, you must set its edge descriptor to either falling or rising (for example GPIO107 which corresponds to User Button 1):

```
echo falling > /sys/class/gpio/gpio107/edge
```

The wakeup descriptor in the **sysfs** can be reached:

- At the specific GPIO (for example 107 which corresponds to User Button 1):

```
echo enabled > /sys/class/gpio/gpio107/device/power/wakeup
```

- At the GPIO port (for example gpio107 belongs to port 3):

```
echo enabled > /sys/class/gpio/gpiochip96/device/power/wakeup
```

PWM

Handling of PWM outputs from the user space is not supported in kernel v3.10.

Serial port

Serial ports names

In kernel v3.10 the serial ports have the following names:

- Debug UART: **/dev/ttyAMA0**
- Application UARTs: /**dev/ttyAPPx** (where x goes from 0 to 4)

SPI port

You must declare SPI slave devices in the Device Tree as children of the SPI bus node they are connected to. This is how an example of the SPIDEV test device would look like, on SPI3 bit banging bus:

```
  / {
        spi3 {
          spidev@0 {
                compatible = "spidev";
                reg = <0>;
                spi-max-frequency = <4000000>;
                status = "okay";
            };
        };
    };
```

Sound

Using the sound interface

The sound card is detected by the kernel as sound device index 0 with two sub-devices:

- **hw:0,0** for playback
- **hw:0,1** for recording

Several predefined configuration files are stored at**/var/lib/alsa/**:

- **asound.inline_play.state**: for recording from LINE-IN and playback
- **asound.inline.state**: for recording from LINE-IN only (no playback)
- **asound.micro_play.state**: for recording from MIC and playback
- **asound.micro.state**: for recording from MIC only (no playback)
- **asound.play.state**: for playback only

ALSA controls

You can use **amixer** to tell you which are the available simple controls:

```
root@ccardimx28js:~# amixer scontrols
Simple mixer control 'Headphone',0
Simple mixer control 'Headphone Mux',0
Simple mixer control 'Headphone Playback ZC',0
Simple mixer control 'PCM',0
Simple mixer control 'Mic',0
Simple mixer control 'Capture',0
Simple mixer control 'Capture Attenuate Switch (-6dB)',0
Simple mixer control 'Capture Mux',0
Simple mixer control 'Capture ZC',0
```

For example, you can increase the playback volume either of the following code lines:

```
# amixer set Headphone 125
```

```
# amixer set PCM 150
```

Recording

For recording with **arecord** ALSA application, you must:

- Specify the device to use with the **-D**, **--device=NAME** parameter.
- Set the number of channels with the **-c**, **--channels=#** parameter to 2.
- Specify the rate with the **-r**, **--rate=#** parameter.

```
  # arecord -M --duration 8 -D hw:0,1 -f cd --rate=44100 --channels=2
sound.wav
```

Touch screen

If using a SATO graphics system, the touch screen can be calibrated with the **xinput_calibrator** application. If the application can't be launched from the graphic IDE, you must launch it from the command line with:

```
# export DISPLAY=:0
# xinput_calibrator
```

If using a resistive touch screen like LQ70Y3DG3B that has a lot of jitter, the calibration may fail if the acquired samples move away too much from a certain threshold. You may need to repeat the calibration process until it completes successfully.

USB Device

USB gadget support

The following modules must be loaded before a gadget driver can be loaded.

```
# modprobe configs
# modprobe libcomposite
```

Serial gadget

To load the serial gadget:

```
# modprobe u_serial
# modprobe usb_f_acm (default) OR usb_f_obex OR usb_f_serial
# modprobe g_serial
```

Ethernet gadget

To load the Ethernet gadget:

```
  # modprobe g_ether
```

File-backed storage gadget

To load the file-backed storage gadget:

```
  # modprobe g_mass_storage
```

Wireless

In previous versions, the network parameters IP address, netmask, and gateway were configured in U-Boot. Then they were, saved in the **NVRAM** and passed to the kernel via the command line, or directly retrieved by the user space from the **NVRAM**.

You must configure network settings in the standard Linux way in the new v3.10 BSP and save them in the **/etc/network/interfaces** configuration file.