ConnectCore 6

Linux BSP in Digi Embedded Yocto 1.6

Reference Manual

Revision history—90001424

Revision	Date	Description
A	August, 2014	Initial release
B	November, 2014	Updated to General Availability Kit
C	September, 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

About the ConnectCore 6 Linux BSP

This guide describes the supported devices and interfaces on Linux BSP for the ConnectCore 6 platform in Digi Embedded Yocto 1.6.

Linux kernel device tree

The Flattened Device Tree (FDT, or simply DT) is a data structure for describing the hardware in a system. Rather than hard coding every detail of a device into the operating system, many aspects of the hardware can be described in a data structure that is passed to the operating system at boot time. The data structure itself 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 able to 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 the 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 id for each new board variant.
- Reduce the need to recompile the kernel. One kernel image with support for different hardware can be shipped and be run in different variants (each one with its own FDT describing the hardware which is really 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.

Platform device tree files

The DTS file for the **ccimx6sbc** platform can be found in the kernel source code tree under:

```
arch/ arm/boot/dts/imx6q-ccimx6sbc.dts.
```

This DTS file includes other DTS and DTSI files in the same path.

Devices and interfaces

ADC

The Dialog DA9063 PMIC provides a 10bit ADC with nine channels. Six of these channels measure different PMIC voltages and the internal temperature of the PMIC. The three remaining channels are analog inputs (0 ~2.5V).

The driver supports the reading of the following ADC channels:

```
# cd /sys/class/hwmon/hwmon0/device/
# cat temp1_input
36 <- temperature in Celsius
# cat in0_label in0_input
VSYS
4523 <- VSYS voltage in mV
# cat in1_label in1_input
ADCIN1
889 <- ADC input 1 voltage in mV
# cat in2_label in2_input
ADCIN2
938 <- ADC input 2 voltage in mV
# cat in3_label in3_input
ADCIN3
897 <- ADC input 3 voltage in mV
# cat in4_label in4_input
VBBAT
415 <- VBBAT voltage in mV
```

Bluetooth

The module assembles an Atheros wireless and Bluetooth chip connected to uSDHC1. The Bluetooth MAC address is taken 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 has created a Bluetooth entry node to pass the driver the MAC address (filled in by U-Boot) and the power down and disable GPIOs:

```
bluetooth {
    digi,pwrdown-gpios = <&gpio_extender 4 0>;
    digi,disable-gpios = <&gpio1 9 0>;
    /* U-Boot will fill in the MAC address here */
};
```

CAN Bus

The SoC has two Flexcan CAN ports. The CAN support is based on the SocketCAN stack. For more information about this project, documentation, and API, please refer to www.kernel.org/doc/Documentation/networking/can.txt.

The FlexCAN Device Tree binding is described at Documentation/devicetree/bindings/net/can/fsl-flexcan.txt

Information about programming the CAN socket interface is given in the kernel tree under www.kernel.org/doc/Documentation/networking/can.txt.

Each CAN port appears like a networking interface in the form **can*x*** where **x** is the port number. To configure the CAN, first set the bitrate using the ip tool. For example:

```
# ip link set can0 type can bitrate 500000
```

To enable a port execute this command (specifying the appropriate port number):

```
# ifconfig can0 up
```

A sample application called **can_test** is available and can be added to the rootfs by adding **dey-examples** to the EXTRA_IMAGE_FEATURES of your local.conf or by adding **dey-examples-can** to IMAGE_INSTALL_append. This sample application performs several operations on the CAN node, like sending and receiving messages. An additional CAN node (such as a CAN analyzer) is needed in the other end of the bus for the application to work.

For example, to send an 8-bit CAN message to node **can0** with ID **0x12** and the data pattern **0x65**:

```
# can_test -l 1 -b 8 -d can0 -i 0x12 -p 0x65 -m
```

And to receive a similar message:

```
# can_test -l 1 -b 8 -d can0 -i 0x12 -p 0x65
```

For more information, see the applications help with `can_test --help`.

Carrier board version number

Since the ConnectCore 6 is an SMD module, customers can design their own carrier boards. Carrier boards are often redesigned and it is useful for the software to be able to determine the version of the carrier board it is running on. This allows the user to have conditional code depending on the board's version (like enabling different GPIOs or using different scripts). Digi U-Boot has built-in support to program the carrier board version number on the OTP bits. Refer to the U-Boot Reference Manual for further information about programming the carrier board version number on the OTP bits.

Carrier board version on the Device Tree

U-Boot reads the carrier board version from the OTP bits and populates it on the Device Tree as a string property of the root node:

```
    / {
          digi,carrierboard,version = "1"
    };
```

The carrier board version number string is printed in decimal, and a zero is assumed to mean undefined version.

A kernel driver can query this Device Tree property to have conditional code based on the carrier board version number. For example:

```
{
    struct device_node *np = of_find_compatible_node(NULL, NULL,
                                                     "digi,ccimx6");
    const char *boardver_str;
    int board_version;
    if (!np)
        return -EPERM;
    if (of_property_read_string(np, "digi,carrierboard,version",
                                &boardver_str);
    board_version = atoi(boardver_str);
    /* Conditional code basing on carrier board version */
    switch(board_version) {
        case 0:
            /* code for undefined carrier board version */
            break;
        case 1:
            /* code for carrier board version 1 */
            break;
        case 2:
            /* code for carrier board version 2 */
            break;
        /* etc */
    }
}
```

Scripts or userspace applications can query the Device Tree property at the procfs:

```
# cat /proc/device-tree/digi,carrierboard,version
1
```

If U-Boot fails to read the carrier board version from the OTP bits, the OTP driver of the kernel at **drivers/char/fsl_otp.c**is a fallback that can read it and populate it on the Device Tree. The following constants in the driver define the location of the carrier board version on the OTP bits, which can be modified by the user:

```
  #define CONFIG_CARRIERBOARD_VERSION_BANK    4
  #define CONFIG_CARRIERBOARD_VERSION_WORD    6
  #define CONFIG_CARRIERBOARD_VERSION_MASK    0xf   /* 4 OTP bits */
  #define CONFIG_CARRIERBOARD_VERSION_OFFSET  0     /* lower 4 OTP bits */
```

For the SBC carrier board, Digi uses the lower 4 bits of the OTP General Purpose 1 register (GP1) which corresponds to Bank 4, Word 6 with a mask 0xf (four bits) and an offset of 0 (lower four bits). Not counting the value of 0 (undefined version), these four bits allow you to code up to 15 board versions.

You can use this layout for your own carrier board or a different one. If you change any of these values please make sure you do the same change in U-Boot source code.

eMMC

eMMC is attached to uSDHC4. The Device Tree lists this interface first so that Linux probes it always as **/dev/mmcblk0**.

Ethernet

The ConnectCore 6 supports both Gigabit and 10/100 Ethernet, although the ConnectCore 6 SBC only has a Gigabit PHY available.

The FEC driver Device Tree binding is described at Documentation/devicetree/bindings/net/fsl-fec.txt.

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

The Freescale i.MX6 CPU has a documented errata **ERR004512** whereby the maximum performance of Gigabit ENET is limited to 400Mbps (total for Tx and Rx).

Frequency scaling

CPU frequency scaling allows you to change the clock speed of the CPUs on the fly. The CPU uses less power when you reduce the clock speed. You can run the CPU at full speed when there is work to do, and reduce the clock rate when the system is idle.

The device tree binding for frequency scaling is described in Documentation/devicetree/bindings/cpufreq/cpufreq-imx6q.txt.

Frequency scaling is controlled through the SYS file system. The directory **/sys/devices/system/cpu/cpuN/cpufreq** contains the related files. You can read the file **scaling_available_governors** to see a list of the available governors. This list should correspond to the ones you selected in the kernel configuration menu. You can read the file **scaling_governor** to see which governor is currently selected, or write to that file to change the currently selected governor.

```
# cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_available_governors
conservative ondemand userspace powersave performance
# cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
performance
# echo powersave > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
# cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
powersave
```

You can read the file **cpuinfo_cur_freq** to see the current CPU clock speed. Normally, the governor automatically sets the CPU speed according to the system load. However, if you select the userspace governor, then you can set the CPU frequency directly by writing it to **cpuinfo_cur_freq**. The files **cpuinfo_min_freq** and **cpuinfo_max_freq** report the minimum and maximum CPU frequencies respectively.

```
# echo userspace > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
# cat /sys/devices/system/cpu/cpu0/cpufreq/cpuinfo_cur_freq
396000
# cat /sys/devices/system/cpu/cpu0/cpufreq/cpuinfo_max_freq
852000
# echo 792000 > /sys/devices/system/cpu/cpu0/cpufreq/scaling_setspeed
# cat /sys/devices/system/cpu/cpu0/cpufreq/cpuinfo_cur_freq
792000
```

GPIO

The CPU has seven GPIO ports, six with 32 GPIOs and one with 14 GPIOs. GPIOs are multiplexed with different functionalities of the chip. IOMUX of the pins is done at the device tree.

The GPIO Device Tree binding is described at Documentation/devicetree/bindings/gpio/fsl-imx-gpio.txt.

The GPIOs can be easily accessed from the sysfs. For information about how to manage the GPIOs from sysfs, refer to the Linux kernel documentation at https://www.kernel.org/doc/Documentation/gpio/gpio.txt.

The ConnectCore 6 SBC contains three user LEDs that can be used to test GPIOs.

	User LED1	User LED2	User LED 3
Port pin	GPIO2_2	GPIO2_3	GPIO2_4
Linux GPIO#	34	35	36

A sample application called **gpio_sysfs_test** is available and can be added to the **rootfs** by adding **dey-examples** to the EXTRA_IMAGE_FEATURES of your **local.conf** or by adding **dey-examples-gpio-sysfs** to IMAGE_INSTALL_append.

I2C

The CPU has three I2C ports. I2C2 is connected to the Dialog DA9063 PMIC and the Kinetis CPU and cannot be used for other peripherals.

I2C3 is connected to the HDMI, LCD touch screen, camera, audio codec, and routed to the SBC board connectors for additional peripheral connections.

I2C Device Tree binding is described at Documentation/devicetree/bindings/i2c/i2c-imx.txt.

Multimedia

Video playback

The i.MX6 processors includes a VPU for hardware video acceleration. Accelerated video playback is supported through the gstreamer framework. For a list of supported audio and video codecs and containers, please refer to Freescale's **i.MX6 multimedia feature matrix**, released as part of their multimedia framework. The status of the supported gstreamer plugins is specified on freescale's multimedia framework release notes.

Please note that special licensed codecs, as those for Microsoft video formats, are not included in the release. Access to these codecs need a special license agreement with the patent holders, after which the codecs can be provided by Freescale.

To reproduce a video file on a **dey-image-minimal** filesystem you can do:

```
    # gplay video.ext

or the equivalent:

    # gst-launch playbin2 uri=file:///video.ext
```

On **dey-image-graphical** there are SATO and QT graphical video playback applications.

You can also use **gst-launch** to customize a gstreamer test pipeline. Some examples follow:

Audio only playback

To reproduce through the main sound device (the sgtl5000 chipset on the SBC headphone output):

```
# gst-launch filesrc location=chieftains.mp3 ! mpegaudioparse ! beepdec !
alsasink
```

To reproduce through the HDMI audio with a connected HDMI display:

```
    # gst-launch filesrc location=chieftains.mp3 ! mpegaudioparse ! beepdec !
    alsasink card-name=imx-hdmi-soc
```

Video only playback

```
    # gst-launch filesrc location=video.ext typefind=true ! aiurdemux ! queue
    max-size-time=0 ! vpudec ! mfw_v4lsink
```

Audio and video playback

```
    # gst-launch filesrc location=video.ext typefind=true ! aiurdemux
    name=demux demux. ! queue max-size-buffers=0 max-size-time=0 ! vpudec !
```

```
  mfw_v4lsink demux. ! queue max-size-buffers=0 max-size-time=0 ! beepdec !
alsasink
```

HTTP streaming

You can stream from an HTTP server as follows:

```
  # gplay http://myserver.com/video.ext
  gst-launch playbin2 uri=http://myserver.com/video.ext
```

Or customize the whole pipeline:

```
  # gst-launch souphttpsrc location=http://myserver.com/video.ext ! typefind !
  aiurdemux name=demux demux. ! queue max-size-buffers=0 max-size-time=0 ! vpudec
!
  mfw_v4lsink demux. ! queue max-size-buffers=0 max-size-time=0 ! beepdec !
alsasink
```

Camera

The current release supports a single camera input device that can be connected to either the MIPI CSI2 interface or the CSI0 or CSI1 interface.

The device tree binding for MIPI CSI2 is described in Documentation/devicetree/bindings/fsl,mipi-csi2.txt, and the one for the CSI in Documentation/devicetree/bindings/fsl,csi-v4l2-capture.txt.

Access to the camera is provided through the gstreamer framework. Some pipeline examples can be found next. Unfortunately there are no graphical camera applications available at the moment.

Camera preview

The **v4l2_preview_test** application displays an overlay or preview of the image captured by the camera on a video interface. This preview is different from the capture mode in that it does not use the CPU so images displayed in this way cannot be captured. It has a help output as follows:

```
  # v4l2_preview_test -?
  V4L2 preview test application.
  Usage: v4l_preview_test -d </dev/video0> -o </dev/fb0> -h <height> -w <width> -
t <top> -l <left> [-r] [-v] [-?]
       -d V4L2 capture device, by default /dev/video0
       -o V4L2 output (framebuffer) device, by default /dev/fb0
       -h Video height in pixels
       -w Video width in pixels
       -t Video top offset in pixels
       -l Video left offset in pixels
       -x Non destructive overlay (do not overwrite framebuffer)
         Setting '-x' discards the output device option '-o' and
         uses the overlay background framebuffer.
       -r RGB565 (default is UYVY)
       -u YUYV (default is UYVY)
       -v Verbose mode
       -? This help
```

Video capture

The v4l2_preview_test application will use the UYVY color format by default. If your sensor works with another color format it also supports YUYV and RGB565.

For example, the ov5642 CSI camera uses the YUYV color format, so the following command would work:

```
# v4l2_preview_test  -d /dev/video1 -u
```

To capture from a video device and display it on a display:

```
# gst-launch mfw_v4lsrc device=/dev/videoN ! mfw_v4lsink
```

Video capture encoding

To capture from a video device and encode it on a file with the matroska container:

```
  # gst-launch mfw_v4lsrc device=/dev/videoN ! queue ! vpuenc ! matroskamux !
  filesink location=test.mkv sync=false
```

Refer to the Linux Multimedia Framework User Guide available from the Freescale BSP documentation for the i.MX6 SoC for further details.

One-time programmable (OTP) bits

The i.MX6 CPU contains several one-time programmable bits (also known as e-fuses).

The OTP bits can be read and programmed through the sysfs, under /sys/fsl_otp.

WARNING! Programming the OTP bits is an irreversible operation that could potentially brick your module. Please don't program the OTP bits unless you are sure of what you are doing.

Power management

The ConnectCore 6 supports several low power modes.

Suspend to RAM

On this mode, the system is suspended and the CPU is in WFI (Wait For Interruption) mode. To enter suspend mode, press the power button between 2 and 4 seconds, or issue the **suspend** command on the terminal.

On the ConnectCore 6 SBC the power button is always enabled to wake up the system. Other possible wake up sources are the CPU's GPIOs and the RTCs.

For example, to enable a GPIO, 36 on the example below, to wake up the system you can do:

```
  # echo -n 36 > /sys/class/gpio/export
  # echo in > /sys/class/gpio/gpio36/direction
  # echo 1 > /sys/class/gpio/gpio36/active_low
  # echo rising > /sys/class/gpio/gpio36/edge
  # echo enabled > /sys/class/gpio/gpio36/device/power/wakeup
```

The following example demonstrates how to configure a 60-second RTC alarm to wake up the system:

```
  # echo enabled > /sys/class/rtc/rtc0/device/power/wakeup
  # echo +60 > /sys/class/rtc/rtc0/wakealarm
```

Power off

Press the button for 4 to 9 seconds to power off the system, or issue the **poweroff** command on the terminal.

Real time clock (RTC)

The Dialog DA9063 PMIC provides a Real Time Clock (RTC) circuit with alarm function. To preserve the date and time during power off, the RTC must be powered externally through a battery.

DA9030 PMIC's binding is described at Documentation/devicetree/bindings/mfd/da9063.txt.

For information about RTC control in Linux, refer to the kernel documentation at Documentation/rtc.txt.

A sample application called **rtc_test** is available and can be added to the **rootfs** by adding **dey-examples** to the EXTRA_IMAGE_FEATURES of your **local.conf** or by adding **dey-examples-rtc** to **IMAGE_INSTALL_append**.

SATA

The device tree binding for SATA is described in Documentation/devicetree/bindings/ata/fsl-sata.txt.

Serial ATA disks are detected by the kernel during the boot process and assigned a device node in the form sdx. Partitions in the drive appear as nodes in the form sdxN, being N the partition number.

Serial ATA Disks support hot-plugging, assuming that the proper power cables are provided. Adaptors from legacy Molex type of connectors to SATA connectors will not provide safe hotplugging capabilities.

SD/SDIO/MMC controller

The CPU has four uSDHC controllers:

- uSDHC1 is internally connected to the Atheros wireless chip.
- uSDHC2 is available at the module and connected in the development board to a micro SD socket.
- uSDCH3 is available at the module but it is also internally connected to the Atheros chip Bluetooth UART in modules with Bluetooth support. Note that in such modules this controller cannot be used.
- uSDHC4 is internally connected to the eMMC.

MMC binding is described at Documentation/devicetree/bindings/mmc/mmc.txt.

Serial port

The CPU has five UARTs. UART1 is a full modem whereas the other four UARTS (2..5) are only four wires. The console on the ConnectCore 6 SBC is connected to UART4.

UART2 is internally connected to the Atheros chip Bluetooth UART in modules with Bluetooth support. Note that in such modules this UART cannot be used.

Please refer to the hardware reference manual of your board to determine available ports and multiplexed functionality.

The driver supports RS-232 and RS-485 half-duplex modes. UART binding is described at Documentation/devicetree/bindings/tty/serial/fsl-imx-uart.txt.

For RS-485 half-duplex, a three wire IOMUX pinctrl configuration must be selected in the Device Tree (RX/TX/CTS). Please notice that pin CTS of the i.MX6 CPU is an output and is used as RTS line to control the flow of the RS-485 drivers in half-duplex communication.

Additional Device Tree binding documentation for RS-485 can be found at Documentation/devicetree/bindings/serial/rs485.txt.

The standard serial programming API applies to the serial ports. For information about serial programming, see the Serial Programming HOWTO document or the Serial Programming Guide for POSIX Operating Systems document.

Serial Peripheral Interface (SPI)

The CPU has five SPI controllers. Refer to the hardware reference manual of your board to determine available ports and multiplexed functionality.

ECSPI2 communicates with the Kinetis CPU (on module variants with Kinetis) and is routed to the development board SPI connector.

SPI binding is described at Documentation/devicetree/bindings/spi/fsl-imx-spi.txt.

A sample application called spi_test is available to use with spidev driver to test the port in loopback mode by connecting MISO and MOSI lines. The application can be added to the rootfs by adding **dey-examples** to the EXTRA_IMAGE_FEATURES of your local.conf or by adding **dey-examples-spidev** to IMAGE_INSTALL_append.

Sound

The module can output sound through an external audio chip SGTL5000 on the SBC or through the HDMI interface. The available cards can be listed with the following:

```
# aplay -L
null
    Discard all samples (playback) or generate zero samples (capture)
default:CARD=sgtl5000audio
    sgtl5000-audio,
    Default Audio Device
sysdefault:CARD=sgtl5000audio
    sgtl5000-audio,
    Default Audio Devicedefault:CARD=imxhdmisoc
    imx-hdmi-soc,
    Default Audio Device
sysdefault:CARD=imxhdmisoc
    imx-hdmi-soc,
    Default Audio Device
```

To change the default device, please refer to ALSA documentation available at www.alsa-project.org/main/index.php/Asoundrc.

The following configuration change will use the HDMI sound card as default:

```
# echo "defaults.pcm.!card 1" > /etc/asound.conf && sync
```

The sound driver can be accessed using the ALSA API. ALSA utilities package also offers user space applications:

- **aplay**: for playback
- **arecord**: for recording

- **alsactl**: for configuration
- **amixer**: for specific control setup

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

To enable one of the configuration files above, execute the alsactl application. For example, for enabling recording the input-stream over the line-in, execute:

```
# alsactl restore -f /var/lib/alsa/asound.inline.state
```

Thermal management

The ConnectCore 6 has a runtime thermal management feature that will monitor the system temperature and react to high temperature situations.

The system defines two temperature thresholds below the die's nominal maximum temperature:

- **Critical**, nominal minus 5° C
- **Passive**, nominal minus 20° C

These thresholds can be modified through the sysfs at **/sys/class/thermal/thermal_zone0/**. Upon reaching the passive threshold, the thermal management system starts corrective cooling actions and the CPU and GPU frequency scaling is activated. GPU frequency scaling can have adverse effects on graphical systems (from simple slow renderization to complete video freeze). The driver will reduce the GPU clock frequency by applying a divisor of **gpu3DMinClock/64**. The value of **gpu3DMinClock** can be controlled via a driver parameter with values ranging between 1 (lowest frequency) and 64 (highest frequency). The default value of 0 equates to 1 (lowest frequency). To change it, do:

```
# echo 3 > /sys/module/galcore/parameters/gpu3DMinClock
```

If the critical threshold is reached, the system will power off.

USB

The CPU has four USB controllers. The default IOMUX exposes USB_OTG and USB_H1.

USB Device Tree bindings are described at Documentation/devicetree/bindings/usb/ci13xxx-imx.txt and Documentation/devicetree/bindings/usb/mxs-phy.txt.

USB device

USB_OTG port can work as a USB device.

Serial gadget

To load the serial gadget:

```
  # modprobe configfs
  # modprobe libcomposite
```

```
# modprobe usb_f_acm
# modprobe u_serial
```

The serial gadget exposes a TTY style serial line interface, usable with **minicom** and similar tools. Most Linux hosts can talk to this using the generic usb-serial driver. The latest versions of this driver implement the CDC ACM class. This driver works with the MS-Windows usbser.sys driver, the Linux cdc-acm driver, and many other USB Host systems. The kernel has a detailed documentation file at www.kernel.org/doc/Documentation/usb/gadget_serial.txt with information on how to set up this driver with both Windows and Linux systems. Follow the instructions in this file for exposing your target as a serial port to the USB host.

Ethernet gadget

By loading the Ethernet gadget the target enumerates to the host computer as an Ethernet device, using the usbnet driver on Linux hosts or Microsoft's RNDIS driver on Windows hosts.

To load the Ethernet gadget:

```
# modprobe configfs
# modprobe libcomposite
# modprobe g_ether
```

This command will create an Ethernet interface in the target called **usb0** and will assign random MAC addresses to the target and the host.

We need to give this new network interface **usb0** an IP address. For example:

```
# ifconfig usb0 192.168.44.30 netmask 255.255.255.0
```

On a host computer, the **usbnet** module must be loaded so that the device is recognized:

```
# sudo modprobe usbnet
# ifconfig usb0 192.168.44.1 netmask 255.255.255.0
```

Now the target can be accessed via the USB cable as if it was an Ethernet port. You can do a ping or open a telnet session from the host to the target or vice versa.

File-backed mass storage gadget

This gadget implements the USB Mass Storage class, appearing to the host as a SCSI disk drive. Use a file or block device as a backing store for the drive.

To load the file-backed storage gadget:

```
# modprobe configfs
# modprobe libcomposite
# modprobe q_mass-_storage file=<filename>
```

For more information, read the kernel documentation at Documentation/usb/mass-storage.txt.

USB host

USB_H1 port works as USB host. USB_OTG can also work as USB host.

Video

Video can be output through the HDMI interface or two LVDS ports.

Device Tree bindings for IPU, framebuffer and LCD display are described at Documentation/devicetree/bindings/fb/fsl_ipuv3_fb.txt.

The current software release only supports a single display. The default device tree configuration enables the HDMI display on mxcfb0, mapping to fb0 for the background plane and fb1 for the overlay plane.

The default device tree configuration can be overwritten through the **video** kernel command line argument passed by the U-Boot bootloader. For example, to enable the LVDS0 display on mxcfb0 instead of HDMI you would append the following the kernel command line:

```
video=mxcfb0:dev=ldb,LDB-HSD101PFW2,if=RGB666 video=mxcfb1:dev=hdmi,1920x1080M@60
```

In U-Boot you can append strings to the **bootargs** variable by using the variable **extra_bootargs**:

```
=> setenv extra_bootargs video=mxcfb0:dev=ldb,LDB-HSD101PFW2,if=RGB666
video=mxcfb1:dev=hdmi,1920x1080M@60
```

Backlight

The Dialog DA9063 PMIC provides three PWM outputs, two of which are used for LCD backlight control.

DA9063 PMIC's GPIO binding is described at devicetree/bindings/gpio/gpio-da9063.txt.

For information about backlight control in Linux please refer to the kernel documentation at: https://www.kernel.org/doc/Documentation/ABI/testing/sysfs-class-led.

Watchdog

Device Tree binding for watchdog is described at Documentation/devicetree/bindings/watchdog/fsl-imx-wdt.txt.

A sample application called **wd_test** is available and can be added to the rootfs by adding **dey-examples** to the EXTRA_IMAGE_FEATURES of your **local.conf** or by adding **dey-examples-watchdog** to IMAGE_INSTALL_append.

The watchdog test application sets the watchdog timeout value and refreshes the watchdog timer every second during the test time. After the test time is over, the watchdog is not refreshed anymore and the driver executes a reset.

For further information about the watchdog interface, refer to the Linux kernel documentation at: www.kernel.org/doc/Documentation/watchdog/.

Wireless

The module assembles an Atheros wireless and Bluetooth chip connected to uSDHC1.

The wireless MAC address is taken from the 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:

```
  wireless {
       digi,pwrdown-gpios = <&gpio_extender 3 0>;
       /* U-Boot will fill in the MAC address here */
      };
```

The Atheros wireless chip does not support wireless adhoc mode.

For WiFi direct mode, the Atheros wireless driver needs to be loaded with the **ath6kl_p2p=1** parameter.

You can do this at runtime with:

```
# echo 1 > /sys/module/ath6kl_sdio/parameters/ath6kl_p2p
```

U-Boot environment

U-Boot environment can be accessed from a Linux user space using the **fw_printenv** and **fw_setenv** tools.

Config file **/etc/fw_env.config** determines the device, start offset, and size of the environment and its redundant copy. The default config file points to the U-Boot environment stored in the eMMC.

If booting from a U-Boot in an external micro SD card, the U-Boot environment is stored at the micro SD card, and the config file must be changed to point to that block device instead.

Known issues and limitations

Hob

The Hob image builder does not work with i.MX6 platforms that use the meta-fsl-arm layer on the Yocto 1.6 release. As Hob is not actively maintained and is being replaced by toaster, it is recommended to use the latter instead.