TechNexion kernel 4.1.15 release notes

v1.11

TABLE OF CONTENTS

1. Overview

1.1. Contents

This release contains

- u-boot bootloader source code (version 2016.03)
- Linux kernel source code (version 4.1.15, based on NXP 2.0.0 version)
- WiFi Firmwares (for bcm4329, bcm4330 and bcm4339)
- Bluetooth firmwares (for bcm4329, bcm4330 and bcm4339)
- Broadcom tool for loading bluetooth firmware (including source code)
- Pre-built device trees, u-boot and kernel binaries for several configurations
- Cortex M4 Free RTOS source code and demo applications
- USB OTG / Type C software loader
- Release notes PDF (this document)

1.2. Supported products

CPU modules supported by this release include:

- **EDM1-CF-IMX6**
- **EDM1-IMX6**
- **EDM2-IMX6**
- **EDM1-CF-IMX6SX** (only the arm-v7 main core is supported)
- **EDM1-IMX6UL**
- **EDM1-IMX7D**
- **PICO-IMX6-SD**
- **PICO-IMX6-EMMC**

- **PICO-IMX6POP-SD**
- **PICO-IMX6POP-EMMC**
- **PICO-IMX6UL-EMMC**
- **PICO-IMX6UL-NAND**
- **PICO-IMX7-SD**
- **PICO-IMX7-EMMC**

and supported systems include

- **TC0700**
- **TC0710**
- **TC0750**
- **TC1000**
- **TEK3-IMX6**
- **TEK3-IMX6UL**
- **TEP1010-IMX6**
- **TEP1560-IMX6**
- **TEP0500-IMX6UL**
- **TEP0700-IMX6UL**
- **TEP0500-IMX7**
- **TEP0700-IMX7**

In addition to these, there are a number of example pre-configured baseboard-module combinations. These include:

- **EDM1-IMX6** + **EDM1-FAIRY**
- **EDM1-IMX6 + EDM1-GNOME**
- **EDM2-IMX6** + **EDM2-GREMLIN**
- **EDM1-CF-IMX6SX** + **EDM1-GOBLIN**
- **PICO-IMX6** + **PICO-DWARF**

- **PICO-IMX6** + **PICO-NYMPH**
- **PICO-IMX6POP-EMMC** + **PICO-DWARF**
- **PICO-IMX7D-SD** + **PICO-DWARF**
- **PICO-IMX7D-EMMC** + **PICO-DWARF**
- **PICO-IMX6UL + PICO-HOBBIT**
- **PICO-IMX6UL-NAND + PICO-HOBBIT**
- **PICO-IMX7D + PICO-PI**
- **PICO-IMX7D + PICO-HOBBIT**
- **PICO-IMX7D + PICO-NYMPH**

additionally this package also offers limited support for the 3rd party boards

- Wandboard (edm1 module + baseboard)
- Mimas (edm1 baseboard)
- INTEL Edison (pico baseboard)

2. Compiling and Deploying this BSP

This chapter describes how to compile the bootloader and kernel, and how to deploy them to boot the board.

These instructions assume that a linux command line build environment is used.

2.1. Setting up a build environment

In order to simplify the setup of a cross compiler, TechNexion provides a virtual machine of a Ubuntu Linux. Inside this VM image the cross compilers are already correctly set up. This virtual machine can be downloaded from TechNexion FTP.

Manual setup is slightly outside the scope of these release notes, but essentially three steps have to be taken on Linux computer:

- Download a cross compiler toolchain (linaro project often has recent gcc cross compilers),
- set environment variable `CROSS_COMPILE` to the compiler prefix (i.e arm-none-gnueabi),
- add the path to `$CROSS_COMPILE-gcc` to your `PATH` environment variable, and
- set `ARCH` variable to `arm`.

2.2. Preparing a bootable SD card

TechNexion products are usually bootable from SD card or from an on-board eMMC flash chip. Since these two work the same way in software (the differences are abstracted away by the very low level drivers), this description will guide how to compile the software and how to prepare a bootable SD card. The installation to eMMC should be done in a similar way (so that files are placed in the equivalent locations, but inside the eMMC flash).

First prepare an SD card by partitioning it, leaving the first megabyte (or more) unpartitioned (most modern disk utilities do this by default). The first partition is called the boot partition and should be at least some 7-8MB in size. Larger is no problem, but smaller might be. Make it 16-32 MB if size is not a concern.

The rest of the SD card is to be used as Linux userland and/or swap. Partition accordingly to your needs. Then format the first partition as a FAT32 (also FAT16 or FAT12 should work - and be able to create smaller filesystems).

2.3. Preparing a NAND flash for boot

Booting a board with NAND Flash is a little more involved than a board with SD card or eMMC.

TBD

2.4. U-boot bootloader

This section outlines how to compile and install u-boot for your product..

2.4.1. Compiling u-boot

U-boot is the bootloader responsible for very low-level setup of the board. It initializes RAM and storage so a Linux kernel can be loaded and executed.

To compile u-boot you need to configure u-boot for your product. For your convenience, there are a number of pre-configured settings you can use (highly recommended).

U-boot (in general) is configured by the command

```
% make boardconfig
```

where *`boardconfig`* is the name of configuration file in `configs/` folder. The exact name depends on the target board/system and is detailed below.

To compile u-boot with pre-configured configurations first 'cd' into the folder with u-boot source code and then issue the commands:

For **EDM1-CF-IMX6**, **EDM1-IMX6**, **EDM2-IMX6** SoMs

```
% make edm-cf-imx6_defconfig

% make -j 4
```

For **TC0700**, **TC0710** or **TC0750** (note: it is possible, and even recommended to use the EDM1-CF-IMX6 configuration if a serial debug console is used)

```
% make edm-cf-imx6-no-console_defconfig

% make -j 4
```

For **EDM1-CF-IMX6SX**

```
% make edm1-cf-imx6sx_spl_defconfig

% make -j 4
```

For **EDM1-IMX7D**

```
% make edm1-imx7d_spl_defconfig

% make -j 4
```

For **EDM1-IMX6UL**

```
% make edm1-imx6ul-nand_defconfig

% make -j 4
```

For **PICO-IMX6-SD**, **PICO-IMX6-EMMC**, **PICO-IMX6POP-SD** or **PICO-IMX6POP-EMMC**

```
% make picosom-imx6_defconfig

% make -j 4
```

For **PICO-IMX6UL-QSPI** or **PICO-IMX6UL-EMMC**

```
% make picosom-imx6ul_spl_defconfig

% make -j 4
```

For **PICO-IMX6UL-NAND**

```
% make pico-imx6ul-nand_defconfig

% make -j 4
```

For **PICO-IMX6UL-NAND**

```
% make pico-imx6ul-nand_defconfig

% make -j 4
```

For **PICO-IMX7D-SD** or **PICO-IMX7D-EMMC**

```
% make pico-imx7d_spl_defconfig

% make -j 4
```

For **TEK3-IMX6**, **TEP1010-IMX6** or **TEP1560-IMX6**

```
% make tek-imx6_defconfig

% make -j 4
```

For **TEK3-IMX6UL**

```
% make tek3-imx6ul_defconfig

% make -j 4
```

For **TEP0500-IMX6UL** or **TEP0700-IMX6UL**

```
% make tep1-imx6ul_defconfig

% make -j 4
```

For **TEP0500-IMX7D** or **TEP0700-IMX7D**

```
% make tep1-imx7d_spl_defconfig

% make -j 4
```

The result from a succesful compile is for most products two files. One is named 'SPL' and the other 'u-boot.img'. These are both needed to boot the board.

2.4.2. Installing u-boot

U-boot needs to be installed in boot media, whether it is a bootable SD card or an eMMC flash chip. For most products, this is done in two steps, first install the SPL bootloader and then the main u-boot binary.

The first bootloader, SPL, is installed 1kB into the SD card, outside the partitioned area. This can be done with the 'dd' command:

```
# dd if=SPL of=/dev/sdX bs=1k seek=1 oflag=dsync
```

where /dev/sdX is the block device corresponding to the SD card. Note: you need to use the SD card device *itself*, not a partition inside the SD card. The device node name should not end with a number (that indicates it is a partition).

The second bootloader, `u-boot.img`, should be copied into the first FAT partition of your SD card.

2.5. Linux kernel

To compile a kernel, one has to configure the kernel, compile the kernel, compile the device tree and the kernel modules.

This section describes how to do these steps for several product combinations.

2.5.1. Configuring the kernel

Like u-boot, the kernel also needs to be configured. There is a "one size fits all" configuration ("tn_imx_defconfig") for all TechNexion products.

It is recommended that any kernel development starsts with the configurations and device trees present for the sample configurations.

To use the general configuration (for all TechNexion products), first change folder ('cd') where the linux kernel source code resides, and then configure the kernel with:

```
% make tn_imx_defconfig
```

This should be enough as a starting point.

2.5.2. Modifying the kernel config

This section describes how to modify the kernel configuration if needed. It is not a necessary step for most users.

If the pre-defined configuration nedds to be fine tuned (like additional drivers to be added), it can ne done using the command:

```
% make menuconfig
```

This will present a text based menu system where it is possible to navigate through the available configuration options. For instance, to enable an additional driver, browse down the "Drivers" menu and then press space once a suitable driver candidate is found. It is recommended to select more drivers than necessary as a first step to enable a hardware device.

2.5.3. Compiling the kernel and device trees

Depending on your product (system or SoM+baseboard combination) the commands to compile an appropriate device tree are given below. For SoM and baseboard combinations not listed below, it often suffices to use a device tree that is intended for the same SoM but with a different baseboard.

The kernel and the accompanying modules are compiled with the command:

```
% make -j4 zImage modules
```

The device tree(s) are compiled with the command

```
% make device_tree_name.dtb
```

where the device_tree_name can be found in the table below.

Note: IMX6S (single core solo CPU) uses the **IMX6DL** dual lite device tree

Product	**device tree name**
EDM1-IMX6DL + **EDM1-FAIRY**	imx6dl-edm1-cf-pmic_fairy.dtb
EDM1-IMX6Q + **EDM1-FAIRY**	imx6q-edm1-cf-pmic_fairy.dtb
EDM1-IMX6QP + **EDM1-FAIRY**	imx6qp-edm1-cf-pmic_fairy.dtb
EDM1-IMX6DL + **EDM1-TOUCAN**	imx6dl-edm1-cf-pmic_tc0700.dtb
EDM1-IMX6Q + **EDM1-TOUCAN**	imx6q-edm1-cf-pmic_tc0700.dtb
EDM1-IMX6QP + **EDM1-TOUCAN**	imx6qp-edm1-cf-pmic_tc0700.dtb

EDM1-IMX6DL + **EDM1-TOUCAN1000**	imx6dl-edm1-cf-pmic_tc1000.dtb
EDM1-IMX6Q + **EDM1-TOUCAN1000**	imx6q-edm1-cf-pmic_tc1000.dtb
EDM1-IMX6QP + **EDM1-TOUCAN1000**	imx6qp-edm1-cf-pmic_tc1000.dtb
EDM1-CF-IMX6DL + **EDM1-FAIRY**	imx6dl-edm1-cf_fairy.dtb
EDM1-CF-IMX6Q + **EDM1-FAIRY**	imx6q-edm1-cf_fairy.dtb
EDM1-CF-IMX6DL + **EDM1-TOUCAN**	imx6dl-edm1-cf_tc0700.dtb
EDM1-CF-IMX6Q + **EDM1-TOUCAN**	imx6q-edm1-cf_tc0700.dtb
EDM1-IMX6SX + **EDM1-GOBLIN**	imx6sx-edm1-cf.dtb
EDM1-IMX6SX (M4 enabled)+ **EDM1-GOBLIN**	imx6sx-edm1-cf-m4.dtb
EDM1-IMX6DL + **EDM1-FAIRY**	imx6ul-edm1-nand_fairy.dtb
EDM1-IMX6DL+ **EDM1-GNOME**	imx6ul-edm1-nand_gnome.dtb
EDM1-IMX6DL + **EDM1-GOBLIN**	imx6ul-edm1-nand_goblin.dtb
EDM1-IMX7D + **EDM1-GOBLIN**	imx7d-edm1_goblin.dtb
EDM2-IMX6DL + **EDM2-ELF**	imx6dl-edm2-cf-pmic_elf.dtb
EDM2-IMX6Q + **EDM2-ELF**	imx6q-edm2-cf-pmic_elf.dtb
EDM2-IMX6DL + **EDM2-GREMLIN**	imx6dl-edm2-cf-pmic_gremlin.dtb
EDM2-IMX6Q + **EDM2-GREMLIN**	imx6q-edm2-cf-pmic_gremlin.dtb
PICO-IMX6UL + **PICO-HOBBIT**	imx6ul-pico-hobbit.dtb
PICO-IMX6UL + **PICO-HOBBIT**	imx6ul-pico-nand-hobbit.dtb
PICO-IMX6UL + **PICO-HOBBIT**	imx6ul-pico-nand_hobbit.dtb
PICO-IMX6DL + **PICO-DWARF**	imx6dl-pico_dwarf.dtb
PICO-IMX6DQ + **PICO-DWARF**	imx6q-pico_dwarf.dtb
PICO-IMX6UL + **PICO-DWARF**	imx6ul-pico_dwarf.dtb
PICO-IMX6ULL + **PICO-DWARF**	imx6ull-pico_dwarf.dtb
PICO-IMX7D + **PICO-DWARF**	imx7d-pico_dwarf.dtb
PICO-IMX6DL + **PICO-HOBBIT**	imx6dl-pico_hobbit.dtb
PICO-IMX6Q + **PICO-HOBBIT**	imx6q-pico_hobbit.dtb
PICO-IMX6UL + **PICO-HOBBIT**	imx6ul-pico_hobbit.dtb

PICO-IMX6ULL + **PICO-HOBBIT**	imx6ull-pico_hobbit.dtb
PICO-IMX7D + **PICO-HOBBIT**	imx7d-pico_hobbit.dtb
PICO-IMX6DL + PICO-NYMPH	imx6dl-pico_nymph.dtb
PICO-IMX6Q + **PICO-NYMPH**	imx6q-pico_nymph.dtb
PICO-IMX6UL + **PICO-NYMPH**	imx6ul-pico_nymph.dtb
PICO-IMX7D + **PICO-NYMPH**	imx7d-pico_nymph.dtb
PICO-IMX7D + **PICO-PI**	imx7d-pico_pi.dtb
PICO-IMX7D (M4 enabled) + **PICO-PI**	imx7d-pico_pi-m4.dtb
PICO-IMX6DL + **PICO-PI**	imx6dl-pico_pi.dtb
PICO-IMX6Q + **PICO-PI**	imx6q-pico_pi.dtb
PICO-IMX6UL + **PICO-PI**	imx6ul-pico_pi.dtb
PICO-IMX6ULL + **PICO-PI**	imx6ull-pico_pi.dtb
TEK3-IMX6DL	imx6dl-tek3.dtb
TEK3-IMX6Q	imx6q-tek3.dtb
TEK3-IMX6UL	imx6ul-tek3.dtb
TEP0X00-IMX6UL	imx6ul-tep1.dtb
TEP0X00-IMX7D	imx7d-tep1.dtb
TEP1010-IMX6DL	imx6dl-tep5.dtb
TEP1010-IMX6Q	imx6q-tep5.dtb
WANDBOARD-IMX6DL rev. B1	imx6dl-wandboard-revb1.dtb
WANDBOARD-IMX6Q rev. B1	imx6q-wandboard-revb1.dtb
WANDBOARD-IMX6DL rev. C1	imx6dl-wandboard-revc1.dtb
WANDBOARD-IMX6DL rev. C1	imx6q-wandboard-revc1.dtb
WANDBOARD-IMX6DL rev. D1	imx6dl-wandboard-revd1.dtb
WANDBOARD-IMX6Q rev. D1	imx6q-wandboard-revd1.dtb
WANDBOARD-IMX6QP rev. D1	imx6qp-wandboard-revd1.dtb

For **TC0700** and **TC0750** the steps are the same as for **EDM1-CF-IMX6** and **EDM1-FAIRY** (see above), but needs a small manual change to enable correct audio (the board inside **TC0700** and **TC0750**, the EDM1-TOUCAN, uses line out audio, while the **EDM1-FAIRY** uses headphone out).

Edit `arch/arm/boot/dts/imx6qdl-edm1-cf.dtsi`, and change the line

```
                "Headphone Jack", "HP_OUT";
```

to

```
                "Line Out Jack", "HP_OUT";
```

and then compile the device tree.

2.5.4. Installing the linux kernel, device trees and modules

Install your device tree by copying arch/arm/boot/zImage and all .dtb files in arch/arm/boot/dts/ to the first (FAT) partition of your SD card.

```
    # cp arch/arm/boot/zImage arch/arm/boot/dts/*.dtb /mnt/boot/
```

assuming your FAT partition on the SD card is mounted at /mnt/boot.

Kernel modules are installed with

```
    # make ARCH=arm modules_install INSTALL_MOD_PATH=/path/to/rootfs
```

where */path/to/rootfs* is where your Linux root filesystem is mounted (or located). The `modules_install` make rule installs the modules in `/lib/modules/...` folder.

2.5.5. TechNexion device tree architecture

The device trees used by TechNexion modular systems are usually composed of three parts (at least). There usually are the following fragments:

- CPU type specific device tree, provided by NXP (i.e. `imx7d.dtsi`)
- CPU module specific device tree (i.e. `imx7d-pico.dtsi`)
- Baseboard device tree (i.e. `baseboard_pico-pi.dtsi`)
- A glue device tree including the above (i.e. `imx7d-pico_pi.dts`)

For a custom made baseboard, it could suffice to make a device tree and glue dts using TechNexion device trees as template.

2.6. Installing WIFI and bluetooth firmware

TechNexion products are equipped with different WiFi models depending on product. The older versions of **EDM1-CF-IMX6**, **EDM2-CF-IMX6**, **TC0700** and **TC0750** are equipped with a Broadcom 4330 design, and the **PICO-IMX6**, **PICO-IMX6POP, PICO-IMX7D, PICO-IMX7D**, **EDM1-CF-IMX6SX**, **EDM1-IMX6**, **EDM2-IMX6** and **PICO-IMX6UL** are equipped with a Broadcom 4339 design.

Note that the **TEK3-IMX6**, **TEK3-IMX6UL**, **TEP1010-IMX6**, **TEP1560-IMX6**, **TEP0500-IMX7**, **TEP0700-IMX7**, **TEP0500-IMX6UL**, and **TEP0700-IMX6UL** has no WiFi, and many SoMs comes in versions with or without a WiFi. Check your model!

The wifi driver kernel modules (named brcmfmac.ko and brcmutil.ko for the bcm4330, and bcmdhd.ko for the bcm4339) should be installed into the `/lib/firmware/brcm/` directory in your Linux root filesystem.

The firmware files needed

a) for bcm4330 (**EDM1-CF-IMX6**, **EDM2-CF-IMX6**, **TC0700, TC0750**) are:

- brcmfmac4330-sdio.txt
- brcmfmac4330-sdio.bin
- bcm4330.hcd

b) for bcm4339 (**EDM1-CF-IMX6SX**, **PICO-IMX6-SD**, **PICO-IMX6POP-SD**, **PICO-IMX6-EMMC**, **PICO-IMX6POP-EMMC**, **PICO-IMX7D-SD, PICO-IMX7D-EMMC** and **PICO-IMX6UL**) are:

- fw_bcmdhd.bin
- bcmdhd.cal
- Type_ZP.hcd

3.0 Product Usage

3.1. Serial debug console

A common way to access the unit is by using the serial debug feature of ARM boards. Consult the documentation for your board on how to attach a serial cable to your PC, and start a terminal emulator (such as Minicom, Teraterm or Putty).

The terminal settings should be set to 115200 baud, no stop bits, 8 data bits, one parity bit and no flow control.

3.2. Enabling WiFi

This section describes how to manually enable and verify WiFi from the Linux command line.

3.2.1 Loading driver and firmware

For bcm4330 devices (**EDM1-CF-IMX6**, **EDM2-CF-IMX6**, **EDM1-CF-IMX6SX**, **TC0700** and **TC0750**) the driver used is the Broadcom Fullmac driver, and the kernel modules needed to enable wifi are

`brcmfmac.ko` and `brcmutil.ko`. Use the command:

```
# modprobe brcmfmac
```

to load the kernel module. The firmware is loaded when the module is inserted.

The bcm4339 devices (like **PICO-IMX6-SD**, **PICO-IMX6-EMMC**, **PICO-IMX7D-SD, PICO-IMX7D-EMMC** and **PICO-IMX6UL**) uses the Broadcom DHD driver. The kernel module needed is `bcmdhd.ko` and inserted with

```
# modprobe bcmdhd
```

The firmware is loaded when the module is inserted.

3.2.2 Verifying WiFi operation

First load driver module, and then check that a wireless interface has appeared (issue the command on the board, in a linux console, perhaps in debug console)

```
# ifconfig -a
```

If a wlan* interface is listed (lets call it wlan*X*), the firmware has been loaded succesfully. Now bring the interface up and perform a scan for networks:

```
# ifconfig wlanX up

# iwlist wlanX scan
```

A list of wireless networks and their parameters should be displayed,

3.3. Enabling bluetooth

This section describes how to load bluetooth firmware and how to verify bluetooth is working.

3.3.1. Loading bluetooth driver and firmware

The bluetooth part of the wifi chip requires a userland tool to load the firmware, This tool is named `brcm_patchram_plus` and is included as both a precompiled arm hardfloat binary and as source code.

The bluetooth is attached over a serial interface, and while the commands look to load the firmware look similar, there are minor differences in the UART used.

To load the firmware for bluetooth, use the `brcm_patchram_plus` as described for your product, and then proceed with the steps at the end of the section to verify bluetooth functionality.

For **EDM1-CF-IMX6**, **EDM2-CF-IMX6**, **TC0700** or **TC0750**:

```
    # brcm_patchram_plus -d --patchram /lib/firmware/brcm/bcm4330.hcd
--baudrate 3000000 --no2bytes --tosleep=2000 --enable_hci /dev/ttymxc2
&
```

This command takes a few seconds (less than ten) to execute.

For **EDM1-CF-IMX6SX**,

```
    # brcm_patchram_plus -d --timeout=6.0 --patchram
/lib/firmware/brcm/bcm4330.hcd --baudrate 3000000 --no2bytes
--tosleep=2000 --enable_hci /dev/ttymxc5 &
```

for **PICO-IMX6-SD**, **PICO-IMX6-EMMC**, **PICO-IMX6POP-SD**, **PICO-IMX6POP-EMMC**,

```
    # brcm_patchram_plus -d --timeout=6.0 --patchram
/lib/firmware/brcm/Type_ZP.hcd --baudrate 3000000 --no2bytes
--tosleep=2000 --enable_hci /dev/ttymxc1 &
```

for **PICO-IMX7D-SD**, **PICO-IMX7D-EMMC** and

```
    # brcm_patchram_plus -d --timeout=6.0 --patchram
/lib/firmware/brcm/Type_ZP.hcd --baudrate 3000000 --no2bytes
--tosleep=2000 --enable_hci /dev/ttymxc3 &
```

for **PICO-IMX6UL-QSPI**

```
    # brcm_patchram_plus -d --timeout=6.0 --patchram
/lib/firmware/brcm/Type_ZP.hcd --baudrate 3000000 --no2bytes
--tosleep=2000 --enable_hci /dev/ttymxc4 &
```

3.3.2. Verify Bluetooth functionality

Easiest way to verify Bluetooth is to scan for nearby bluetooth devices. Make sure your bluetooth device is broadcasting (usually by pressing a "connect" button or the like). Boot with Linux and on the command line (over debug console) issue the commands

```
    # hciconfig hci0 up

    # hcitool -i hci0 scan
```

and a list of nearby bluetooth devices should appear.

3.4. Cortex M4 core and FreeRTOS

Some iMX-series CPUs contain a low power Cortex M4 MCU in addition to the main ARM core. The M4 core is present in

- **PICO-IMX7**
- **EDM1-IMX7**
- **EDM1-CF-IMX6SX**

The Cortex M4 microcontroller lacks an MMU, and can therefore not run a full Linux kernel. Instead they are limited to run Free RTOS (Real Time Operating System).

For TechNexion boards the BSP source code release also contains a Free RTOS demo with full source code.

3.4.1 Compiling a demo application

The source code contains a number of demo applications and a very brief document on how to compile them. This section also describes how to compile the hello world application.

This example assumes that the toolchain at

https://launchpad.net/gcc-arm-embedded/4.9/4.9-2015-q3-update/download/gcc-arm-none-eabi-4_9-2015q3-20150921-linux.tar.bz2

is used.

Before compile set the environment variable ARMGCC_DIR to point at the location of the toolchain, for instance by:

```
% export ARMGCC_DIR=/opt/gcc-arm-none-eabi-4_9-2015q3/
```

The demo application folder `examples/imx7d_pico_m4/demo_apps/hello_world/`

contains the source code, but the building is done from the armgcc folder by:

```
% cd examples/imx7d_pico_m4/demo_apps/hello_world/armgcc/

% ./build_release.sh
```

The result should be a ELF binary, `hello_world.elf`, placed in the `armgcc/release/` folder.

Similarily to building, the "make clean" equivalent is to excute the `build_clean.sh` script in the same `armgcc` folder.

3.4.2. Deploying a demo application

To test a Free RTOS application, prepare a bootable SD card (or other boot method) as described earlier. Place the Free RTOS ELF image in the same folder and partition as the u-boot image.

Also place the m4 enabled device tree in the same folder as the non-m4 device tree.

When booting, u-boot will attempt to load an M4 firmware specified with the u-boot variable `m4image`. If the load fails, the normal device tree blob will be used. If the m4image is succesfully loaded, the m4 enabled device tree will be used.

4.0 Customization

This section describes some common customizations.

4.1. Display settings

There are two types of display changes: to change the output device (LCD display, HDMI monitor, LVDS panel etc) or change the display settings (resolution, timings).

The output device is controlled by u-boot, and the settings are partially in u-boot but sometimes also in the kernel device tree.

4.1.1. Change output device

To change output device, enter your u-boot (use debug console, and press a key to stop the boot process). A prompt appears. Give commands:

```
    => setenv display_autodetect off
```

and then set your output device with (example commands)

for HDMI (where available):

```
    => setenv displayinfo
'video=mxcfb0:dev=hdmi,1920x1080M@60,if=RGB24'
```

for LVDS (4 lanes, 24 bit interface):

```
    => setenv displayinfo
'video=mxcfb0:dev=ldb,1024x600@60,if=RGB24,bpp=32'
```

for LCD/TTL:

```
    => setenv displayinfo
'video=mxcfb0:dev=lcd,800x480@60,if=RGB24,bpp=32'
```

or for a dual LVDS + HDMI display:

```
    => setenv displayinfo
'video=mxcfb0:dev=ldb,1024x600@60,if=RGB24,bpp=32
video=mxcfb1:dev=hdmi,1280x720M@60,if=RGB24'
```

To test the changes (without saving), issue the

```
    => boot
```

command.

To save settings, use the

```
=> saveenv
```

command before (re)booting.

4.1.2. Change display parameters

The examples above have resolutions given as CVT timings. These do not always work (even when the panel supports CVT timings). Especially LVDS and LCD panels requires modification of the device tree to set detailed display settings.

The devicetree changes are to be done in kernel source tree, in the `.dts` (or `.dtsi`) file for your product, located in `arch/arm/boot/dts` folder.

In the dts ("device tree source") file, there should be a display section, similar to:

```
display-timings {

    native-mode = <&timing0>;

    timing0: lcd_panel {

         clock-frequency = <51000000>;

         hactive = <1024>;

         vactive = <600>;

         hback-porch = <90>;

         hfront-porch = <90>;

         hsync-len = <130>;

         vback-porch = <5>;

         vfront-porch = <5>;

         vsync-len = <25>;

         de-active = <1>;

         hsync-active = <1>;

         vsync-active = <1>;

         pixelclk-active = <1>;
```

```
        };
    };
```

Here, the timings of your panel needs to be filled in. These can (hopefully) be found in the datasheet for the panel. To give a quick guide to the commonly used parameters:

- `clock` is the pixel clock in Hz
- `hactive`, `vactive` is the resolution of the display (in pixels).
- `hback-porch`, `hfront-porch`, `hsync-len` are the horizontal timings (in pixel clocks).
- `vback-porch`, `vfront-porch`, `vsync-len` are the vertical timings (also in pixel clocks)
- `de-active` is a `1` or `0` value (boolean) whether to use DE mode or not (for LVDS panels)
- `hsync-active` and `vsync-active` are also `1` or `0` values indicating positive or negative edge for the sync signals
- `pixelclk-active` is also `1` or `0` indicating whether to use positive or negative edge for the pixel clock.

Many panels' datasheets do not give complete timings, but list for example only numbers for the horizontal total, vertical total and resolution. In this case a practical approach is to distribute the difference between the horizontal resolution and horizontal total among the horizontal back porch, front porch and sync length. A heuristic guess of the values is to assign 50-60% as sync length, and 20-25% of the total as front and back porch each.

Example: a panel has 1024x768 resolution and 1104 horizontal total. The difference between the visible width and the total is 80 dotclocks. A reasonable guess is to have horizontal front porch, back porch and sync lengths as 20, 20 and 40 respectively. The total of horizontal resolution and sync lengths (1024+40+20+20) equals the horizontal total (1104).

A similar division can be done for the vertical porches and sync length.

Recompile the device tree by

```
% make file.dtb
```

and copy it to your boot partition, replacing the existing `dtb` (device tree blob) file.

4.2. Boot splash screen

The boot splash image is built into u-boot, and does not completely initialize the graphics subsystem. It is recommended to use a small image with black borders for best results.

To replace the default splash image with *image.jpeg*, follow the steps below on a ubuntu PC:

Install the needed graphics packages (often already installed) by:

```
# apt-get install netpbm imagemagick
```

Convert the image to 8-bit BMP (with max 128 colors), for instance with:

```
% convert image.jpeg -colors 128 image.ppm
```

and then overwrite technexion.bmp in u-boot source code folder with

```
% cat image.ppm | ppmtobmp -bpp 8 > tools/logos/technexion.bmp
```

Then recompile u-boot (the name of the file embedded into the u-boot binary depends on the name of the folder where the boards resign, see board/ folder).

5.0. Troubleshooting

This section covers some common problems.

Also be noted that you can update your software to the latest published 3.14.52 version by issueing the command

```
    % git pull --rebase
```

in either the u-boot or kernel folder. This will retrieve the latest patches from TechNexion git at https://www.github.com/TechNexion

5.1. *It* is not working!

Sad to hear that. Whatever you are trying to do, try some of the pre-compiled software images provided by TechNexion and see if *It* is working there.

Then do your changes (like replace u-boot or kernel) on the pre-compiled image, and see if *it* is still working or not.

Doing only one change at a time is a good way to find the source of your problems.

5.2. Debug console problems

5.2.1. There is nothing on the console!

Check:

- Your cable is connected to the right place
- That you have a null modem / gender changer. Try changing to the other kind.
- Your baudrate is 115200 baud (see section 3.1)
- You are using the right serial port on your PC, right?

5.2.2. There is still nothing on the console on my TC0700

By default debug console is disabled on **TC0700**, to avoid interference with equipment connected to the external UART port.

To enable the debug console first try attaching a Prolific or FTDI USB to serial adapters to a USB host port before powering up the board. A few selected USB seria models might provide a serial debug console that way.

If this does not work or is not an option, compile a u-boot for **EDM1-CF-IMX6** and use that on your **TC0700**. The debug console for **TC0700** is enabled there.

5.2.3. Garbage on the console

"Garbage" on the console is a typical sign of baudrate problems. Check your settings.

If the console became garbled during usage, chances are that something (like viewing binary data -- more specifically character "`\016`") has triggered the 8th bit on mode.

The easiest way to fix this is by either starting a new terminal or (if possible) by issuing the shell command `reset` in a linux terminal (even if typed blindly on a garbled console). Also, displaying the character "`\017`" (like with the shell command `printf "\017"`) will turn the 8th bit off again.

5.3. Wifi and bluetooth problems

This section contains solutions to a few common wifi and bluetooth problems.

5.3.1. No wlan*X* interface?

Usually this is due to firmwares not loaded. Check your firmwares are in `/lib/firmware/brcm`. Sometimes this can be due to chip not powered up properly (for products with gpio controlled wifi power) or an rfkill driver blocking the wifi.

5.3.2. No nearby wireless networks?

You do have an antenna connected?

Also be reminded that an antenna for 2.4GHz and 5GHz antenna are different.

5.3.3. No bluetooth devices?

The devices nearby should sometimes be entered into pairing/connect mode.

Also Bluetooth is a short-range network, so while debugging have your bluetooth device next to the board.