DATA MODUL

eDM-SBC-iMX6-PPC
Linux BSP Manual

Revision: r-004
Date: 2019–09–02

Revision History

Revision	Date	Author(s)	Description
r-001	2019-02-14	Yazdani,Reyhaneh, Albert,Elmar	First release of Linux-BSP manual
r-002	2019-05-07	Yazdani,Reyhaneh	Reorder "source code repositories" section to make the procedure clear, add HW requirements of host, modified typo
r-003	2019-08-12	Albert,Elmar	Update host requirements for building
r-004	2019–09–02	Albert,Elmar	added chapter: Adding customer layer

Contents

Disclaimer

All drivers, software and tools available for download through any website owned by Data Modul AG (hereinafter Data Modul) are the copyrighted work either of Data Modul or of third party software vendors. By downloading and using such drivers, software and tools the User agrees to the respective licenses or other arrangements, if any, between Data Modul or third party vendor and User.

Drivers, software and tools downloaded from Data Modul websites are warranted, if at all, only to the extent agreed in license agreements. Except as warranted in those license agreements, Data Modul carries no additional warranty or liability obligations.

Data Modul general limitation of liability for software products and conditions of use:

1. Data Modul, as part of our product development process, develops software products (operating systems, images, device drivers, tools, manuals and documentation, hereinafter "Products") for purposes of developing, testing and demonstrating our hardware products. We make these Products available for download to the User community for free, unless otherwise stated. Data Modul does not represent or guarantee that free downloaded Products are error free or are suitable for use in any other User's or Manufacturer's devices or systems. Furthermore, Data Modul makes no representation or guarantee that downloaded Products will be maintained or updated by Data Modul or that future versions will be released.
2. Users who download and use Products from Data Modul websites are solely responsible for any potential damages to User's hardware or loss of data that results from the download or use of any such Products from Data Modul sites.
3. The User is solely responsible for warranting that the product meets User's requirements and expectations.
4. Before installation and usage of any Data Modul Products the User is obligated to read and strictly follow the manuals, installation guides and other documentation related to the operating systems, images, files, drivers, software and tools.
5. The User is solely responsible for adequate protection and backup of the data and equipment used in connection with any downloaded Products from Data Modul sites.
6. Data Modul makes no representations or warranties as to the truth, accuracy or completeness of any statements, information or materials concerning drivers, software and tools that are available on Data Modul websites.
7. Neither Data Modul nor any of our partners make any warranties of any kind, either express or implied, including, without limitation, warranties of title or implied warranties of merchantability or fitness for a particular purpose or non-infringement.
8. Any correspondence related to third party software products should be directed to the appropriate third party responsible for developing or distributing the software.
9. Data Modul makes no representations or warranties of any kind concerning the quality, safety or suitability of any third party software products
10. In the event that any paragraph of this disclaimer or any part thereof be declared invalid or unenforceable, the remainder of the disclaimer shall remain valid and in force.

1 Introduction

This document describes the features of the Data-Modul PPC Linux Board-Support-Package(BSP). It is possible that some features are not available on the requested board, and therefore some sections are not applicable.

The available releases are:

- Barebox version 2017.04
- Linux kernel version: 4.9
- Yocto 2.4 Rocko
- Qt 5.9
- Gstreamer 1.8.3
- Browser: QupZilla

As well, the system partition of the final image is presented in the table 2.

Table 2: System partitions

Partition name	Size	Access	Format
Boot	51M	rw	vfat
Root	2.5G	r	ext4
Overlay	700M	rw	ext4
Home	200M	rw	ext4

Note: The Overlay partition is defined to have write access on Root partition. This partition sits on top of Root partition. Thus all modifications go to the overlay partition and in case of any failure in boot procedure, it would be sufficient to unmount overlay partition.

2 Building the BSP

The PPC i.MX6 Linux BSP is built via Yocto. The Yocto project is not an embedded distribution. It creates a custom one for us.

The Yocto has a main engine called Bitbake. Bitbake is a highly efficient task execution system, which automates the process of creating a working Linux operating system. After the building, a directory of all compiled binary package is created. It has bootable Linux system disk image, kernel image, device tree binary file, ...

Moreover, an Application Developer SDK can be created which includes a cross-architecture compiler and tools. More information about the Yocto features is available at the Yocto Reference Manual.

2.1 Host requirements

The host development system must meet hardware requirements and specific version for serveral tools:

- Minimum 8GB of RAM
- Around 100GB free space on HDD

- Git 1.8.3.1 or greater
- tar 1.24 or greater
- Python 2.7.3 or greater not including Python 3.x
- Docker engine 17.05.0-ce or greater (optional)

It is recommended to build the BSP in an openembedded development docker container, which contains all necessary tools and libraries for building the BSP. The Dockerfile is available on DMO github. For installing **docker-engine CE** on the host system, please refer to Docker glossary for the various operating systems and follow 2.3 to setting up the docker container.

In case of not using the docker container, please refer to required software packages. In addition to these packages, make sure that *g++-multilib* is installed, which is needed for yocto-rocko and later.
Then, install the *repo* tool according to the Install Repo section in order to prepare the build host and do not follow sub-section 2.3. Moreover, make sure that the link **/bin/sh** points to **bash**:

```
$ ls -l /bin/ | grep sh
-rwxr-xr-x 1 root root 1037528 Mai 16 2017 bash*
-rwxr-xr-x 1 root root 154072 Feb 17 2016 dash*
lrwxrwxrwx 1 root root 21 Aug 10 15:46 ksh -> /etc/alternatives/ksh*
-rwxr-xr-x 1 root root 1554008 Jul 6 2017 ksh93*
lrwxrwxrwx 1 root root 4 Mai 16 2017 rbash -> bash*
lrwxrwxrwx 1 root root 5 Jul 6 2017 rksh -> ksh93*
lrwxrwxrwx 1 root root 5 Jul 6 2017 rksh93 -> ksh93*
lrwxrwxrwx 1 root root 9 Okt 23 2017 sh -> /bin/bash*
lrwxrwxrwx 1 root root 4 Dez 13 2016 sh.distrib -> dash*
lrwxrwxrwx 1 root root 7 Dez 13 2016 static-sh -> busybox*
$
```

2.2 Source code repositories

A collection of the BSP source code and Repo Manifest are available at DMO github repository.

The below list presents existing DMO Github repositories.

- https://github.com/data-modul/meta-dmo.git
- https://github.com/data-modul/dmo-manifest.git
- https://github.com/data-modul/initscripts.git
- https://github.com/data-modul/linux-imx6.git
- https://github.com/data-modul/barebox.git
- https://github.com/data-modul/meta-freescale.git (since Yocto 2.4 (Rocko))

*Note: If you already have a Yocto Project setup and only need PPC BSP layer, it would be enough to use **meta-dmo** layer.*

*Note: Pay attention to the branch which selected. **master** branch is **not** the latest one.*

2.3 Setting up the docker container

Firstly, it is required to build the docker image. For this, please use Ubuntu disribution. Thereafter, the docker image can be copied to any other Linux System by following the steps described in "docker save" and "docker load".

Then, the following steps describe how to setup the docker container for building the BSP:

- Create a directory for cloning the Dockerfile and change into it

```
$ mkdir -p /home/<user>/dockerfile-github
$ cd /home/<user>/dockerfile-github
```

- Clone the Dockerfile and build the docker image

```
$ git clone https://github.com/data-modul/oe-base.git
$ cd oe-base/
$ docker build -t image-name .
```

- Create *working-dir* and *build-dir* directories, which will be handed into the container later.
  These directories are used for storing the sources and build results on the host, while the actual build is done inside the container.

```
$ mkdir -p /home/<user>/working-dir
$ mkdir -p /home/<user>/build-dir
```

- Run the docker image (example)

```
$ docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
image-name latest 6a5bc5bdce25 2 minutes ago 862MB

$ docker run -d -t \
-v /home/<user>/working-dir:/working-dir \
-v /home/<user>/build-dir:/build-dir \
image-name:latest
```

- List docker containers

```
$ docker ps -a
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
912b5cd05511 image-name:latest "/usr/sbin/sshd␣-D" ...
```

- Determine the IP-Address of intended docker container(some outputs shortened with ...)

```
$ docker inspect 912b5cd05511

...
            "NetworkSettings": {
                "Bridge": "",
                "SandboxID": "bfbc0c5b6ff03c2f5a346f2ee81c7799e2d7d38a␣...␣",
                "HairpinMode": false,
                "LinkLocalIPv6Address": "",
                "LinkLocalIPv6PrefixLen": 0,
                "Ports": {
                    "22/tcp": null
                },
```

```
            "SandboxKey": "/var/run/docker/netns/bfbc0c5b6ff0",
            "SecondaryIPAddresses": null,
            "SecondaryIPv6Addresses": null,
            "EndpointID": "bc019f82e0d4dcec0e8a036a68b324709eed1f0␣...␣",
            "Gateway": "172.17.0.1",
            "GlobalIPv6Address": "",
            "GlobalIPv6PrefixLen": 0,
            "IPAddress": "172.17.0.3",
            "IPPrefixLen": 16,
            "IPv6Gateway": "",
            "MacAddress": "02:42:ac:11:00:03",
            "Networks": {
                "bridge": {
                    "IPAMConfig": null,
                    "Links": null,
                    "Aliases": null,
                    "NetworkID": "77981fa87bdf2ae5bc4e8f2e99e0b8f70␣...␣",
                    "EndpointID": "bc019f82e0d4dcec0e8a036a68b32470␣...␣",
                    "Gateway": "172.17.0.1",
                    "IPAddress": "172.17.0.3",
                    "IPPrefixLen": 16,
                    "IPv6Gateway": "",
                    "GlobalIPv6Address": "",
                    "GlobalIPv6PrefixLen": 0,
                    "MacAddress": "02:42:ac:11:00:03"
                }
            }
        }
]
```


- Connect to the docker container via ssh through the docker internal network(example).
  user: oe
  password: foo

```
$ ssh -XA oe@172.17.0.3
The authenticity of host '172.17.0.3 (172.17.0.3)' can't be established.
ECDSA key fingerprint is SHA256:231IT11bVq1XdxGHr72P5+ghp8GHFFikKbmJ8dQVvyE.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '172.17.0.3' (ECDSA) to the list of known hosts.
oe@172.17.0.3's password:
Welcome to Ubuntu 16.04.5 LTS (GNU/Linux 4.15.0-38-generic x86_64)

 *Documentation: https://help.ubuntu.com
 *Management: https://landscape.canonical.com
 *Support: https://ubuntu.com/advantage
Last login: Tue Oct 23 12:37:23 2018 from 172.17.0.1
oe@912b5cd05511:~$
```

Now the container is ready for building the BSP.

2.4 Install Repo

Repo is a tool that provides a way to manage many Git repositories and makes it easier to work with them. For this matter, it is needed to have *manifest* file, that gives information to Repo to fetch which repositories. By doing so, Yocto Project build environment will be available and ready. If you use openombedded docker container, you do not need to install Repo, as it is already installed in the DMO dockerfile.

To install Repo tool on the host execute the following commands:

```
$ curl https://storage.googleapis.com/git-repo-downloads/repo > ~/bin/repo
$ chmod a+x ~/bin/repo
```

The installation status can be inspected by invoking help flag. If it is successfully installed, a usage message must be displayed.

```
$ repo --help
```

2.5 Initialize Repo client

Create *working-dir* and *build-dir* directories, which will be used for storing the sources and build results. If you have already applied the docker container, these directories are already created via the steps described in 2.3.

```
$ mkdir -p /home/<user>/working-dir
$ mkdir -p /home/<user>/build-dir
```

In case of using docker, do the following steps inside the docker, otherwise run them outside.

The next step is change into a *working-dir* directory.

```
$ cd working-dir
```

Next, initialize Repo. By the following command, repo knows where to find the manifest.

```
$ repo init -u git@github.com:data-modul/dmo-manifest.git -b {branch-name}
```

or

```
$ repo init -u git://github.com/data-modul/dmo-manifest.git -b {branch-name}
```

or

```
$ repo init -u https://github.com/data-modul/dmo-manifest.git -b {branch-name}
```

There is one released branch on github:

- yocto-rocko

After initialization, it exists only `.repo` in the current directory, which holds all repository information.

2.6 Fetch all the repositories

In the next step, the development tree needs to be synchronized and populated.

```
$ repo sync
```

This step takes time depending on the connection. By the end of the synchronization, a poky directory with some subdirectories are available in the *working-dir* folder.

2.7 Build images

For the first build, it is necessary to set the `TEMPLATECONF` environment variable to `meta-dmo/conf`.

```
$ cd working-dir
$ export TEMPLATECONF=meta-dmo/conf
$ source ./poky/oe-init-build-env build-dir
```

Note: If build-dir directory is not defined, the default path will be poky/build/.

Inside the *build-dir* directory, default configuration are copied. It is possible to modify the configuration based on your wish, in particular, *MACHINE* and *DISTRO* variables.

The different *distro* versions are available:

- dmo-distro to build an image as a minimal bootable version
- dmo-distro-qt to build an image, which includes qt5, where X11 is disabled.
- dmo-distro-X11 to build a full image. containing X11 and qt5.

For any modification, edit *conf/local.conf* file.

Note: It is essential to agree the end-user license agreement(EULA) of the fsl-arm/freescale layer inside local.conf file. For this matter, inside the build-dir directory:

```
$ vi conf/local.conf
```

*Then, uncomment **#ACCEPT_FSL_EULA = "1"** line by removing **#** from the begining. Then, save and close the file.*

After preparation, start building your own image. This process takes several source code from different sources and compiles them. Therefore, the size of the storage must be large enough(ca. 80 GB). Moreover, it would be possible to take some hours.

```
$ bitbake dmo-image
```

Note: It my happen, that the following error message will arise:

```
Please use a locale setting which supports utf-8.
Python can't change the filesystem locale after loading so we need a utf-8 ...
```

*If so, language settings need to be reconfigured to **en_US.UTF-8**.*

```
$ sudo dpkg-reconfigure locales
$ export LANG="en_US.UTF-8"
```

If the build process ended up successfully, below files will be created in *build-dir/tmp/deploy/images/{machine-name}/*:

- Linux system disk image(sdcard2)
- Root filesystem
- Kernel binary
- Device trees binaries
- Bootloader binary

In case of unsuccessful process when building on the host, please check missing software packages.

2.8 Boot target

The board is shipped only with programmed bootloader and Linux should be programmed on it via user for the first use. Via Linux system disk image(sdcard2), the storage of the target, eMMC, SD-card or SATA, can be programmed.

For updating and programming the board, use SWU mechanism, which is explained in the subsection 31.1.

2.9 Build toolchain

For building software development tools, toolchain, it is enough to bitbake it.

```
$ bitbake meta-toolchain
```

And for the qt5 version:

```
$ bitbake meta-toolchain-qt5
```

After the building is finished, inside build directory, the install script is placed. Via this script, the SDK will be installed on the target directory of the host.

Note: For executing the install script, please open a new shell or exit the docker container, otherwise the toolchain will be installed inside the container and cannot be used on the host.

```
$ cd build-dir/tmp/deploy/sdk
$ ls -l

poky-glibc-x86_64-meta-toolchain-qt-cortexa9hf-neon-toolchain-2.2.1.host.manifest
poky-glibc-x86_64-meta-toolchain-qt-cortexa9hf-neon-toolchain-2.2.1.sh
poky-glibc-x86_64-meta-toolchain-qt-cortexa9hf-neon-toolchain-2.2.1.target.
manifest

$ ./poky-glibc-x86_64-meta-toolchain-qt-cortexa9hf-neon-toolchain-2.2.1.sh
```

In order to use the toolchain for building customer applications, the build-environment needs to be setup by sourcing the environment-setup-script, e.g.

```
$ . /opt/poky/2.2.1/environment-setup-cortexa9hf-neon-poky-linux-gnueabi
```

2.10 Build packages

Besides toolchain, it is also possible to build tools and utilities for the target (inside/outside the docker container).

- Generate list of potential tools and utilities with the version information

  ```
  $ bitbake -s > list.txt
  ```

- Based on the list, select a package to build. e.g. minicom

  ```
  $ bitbake minicom
  ```

- Update list of package index

  ```
  $ bitbake package-index
  ```

After building the package, it is the time to install that package on the target.

Final image provides **opkg** as a package management system to use for installation and reinstallation of packages.

Note: Do not install everything just because it would be possible.
Note: Before uninstalling a package, make sure that no other program needs it any more.

- Access Internet connection or a Network-connection between host and target
- Configure opkg configuration file in target by adding repository information

```
$ vim /etc/opkg/base-feeds.conf
```

As an example:

```
src/gz pl_all http://10.1.1.1/rocko-ppc-dl-qt/all
src/gz pl_cortexa9hf-neon http://10.1.1.1/rocko-ppc-dl-qt/
                                        cortexa9hf-neon
src/gz pl_cortexa9hf-neon-mx6qdl http://10.1.1.1/rocko-ppc-dl-qt/cortexa9hf
                                        -neon-mx6qdl
src/gz pl_imx6dl_dmo_ppc http://10.1.1.1/rocko-ppc-dl-qt/imx6dl_dmo
                                        _ppc
```

The third column shows the connections between host and target and where target should search for the package in the host.

- Refresh the package index

```
$ opkg update
```

- Install new software package:

```
$ opkg install new-package
```

2.11 Starting fresh

If the build breaks, a fresh build of a package or the system would help.

- Clean a package:

```
$ bitbake -c cleansstate package-name
```

- Re-download package:

```
$ bitbake -c cleanall package-name
```

- Remove everything except download:

```
$ rm -rf build-dir/tmp
```

- Remove everything:

```
$ rm -rf build-dir
```

2.12 Build Linux kernel from source code

Besides building PPC-BSP completely, it is possible to build Linux kernel individually from the source code. The source of the Linux kernel is available at Data Modul Github.

The below steps explain how download and build the kernel. This should be done on the host, not inside the docker container.

- Create a directory and change to it

```
$ mkdir -p /home/usrname/linux-arm
$ cd /home/usrname/linux-arm
```

- Clone and check out Linux source code

```
$ git clone https://github.com/data-modul/linux-imx6.git
$ cd linux-imx6/
$ git branch -a
*imx6-ppc/release-4.9.99
  remotes/origin/HEAD -> origin/imx6-ppc/release-4.9.99
  remotes/origin/imx6-ppc/release-4.4.57
  remotes/origin/imx6-ppc/release-4.9.99
```

  As shown above, there are many branches. Please checkout the proper branch.

- Setup a build directory

```
$ mkdir -p /home/username/linux-arm-build
$ cd /home/username/linux-arm-build
```

- Configure the build directory with the installed toolchain(Section 2.9 explains how build and install the toolchain)

```
$ . /opt/poky/2.4.3/environment-setup-cortexa9hf-neon-poky-linux-gnueabi
```

- Start configuring and building the kernel

```
$ make -C /home/usr-name/linux-arm/linux-imx6/ O=$PWD dmo-imx6-ppc_defconfig
$ make -j4
```

The build results are placed in the build directory under the paths described in table 3:

Table 3: Kernel and Device-trees

Path	Type	Description
arch/arm/boot/zImage	kernel	-
arch/arm/boot/dts/imx6dl-dmo-ppc-HDMI.dtb	dtb	solo/duallight, HDMI output
arch/arm/boot/dts/imx6dl-dmo-ppc-chimei-g121I1_12018381.dtb	dtb	solo/duallight, DMO-LVDS-12inch output
arch/arm/boot/dts/imx6dl-dmo-ppc-chimei-g101ICE_12018182.dtb	dtb	solo/duallight, DMO-LVDS-10inch output
arch/arm/boot/dts/imx6dl-dmo-ppc-chimei-g070y2_12018614.dtb	dtb	solo/duallight, DMO-LVDS-7inch output
arch/arm/boot/dts/imx6q-dmo-ppc-HDMI.dtb	dtb	dual/quad, HDMI output
arch/arm/boot/dts/imx6q-dmo-ppc-chimei-g121I1_12018381.dtb	dtb	dual/quad, DMO-LVDS-12inch output
arch/arm/boot/dts/imx6q-dmo-ppc-chimei-g101ICE_12018182.dtb	dtb	dual/quad, DMO-LVDS-10inch output
arch/arm/boot/dts/imx6q-dmo-ppc-chimei-g070y2_12018614.dtb	dtb	dual/quad, DMO-LVDS-7inch output

2.13 Build Barebox from source code

Besides building PPC-BSP completely, it is possible to build Barebox individually from the source code. The source of the Barebox is available at Data Modul Github.

The below steps explain how download and build Barebox.

- Create a directory and change to it

```
$ mkdir -p /home/usrname/barebox-arm
$ cd /home/usrname/barebox-arm
```

- Clone and check out Barebox source code

```
$ git clone https://github.com/data-modul/barebox-imx6.git
$ cd barebox-imx6
```

Checkout the the corresponding branch regarding PPC-Barebox source code

```
$ git co -b local-barebox remotes/origin/dmo-release-2017.04
```

- Setup a build directory

```
$ mkdir -p /home/username/barebox-arm-build
$ cd /home/username/barebox-arm-build
```

- Configure the build directory with the installed toolchain (Section 2.9 explains how build and install the toolchain)

```
$ . /opt/poky/2.4.3/environment-setup-cortexa9hf-neon-poky-linux-gnueabi
$ unset LDFLAGS
```

- Start configuring and building Barebolx

```
$ make -C /home/usr-name/barebox-arm/barebox-imx6/ O=$PWD dmo-imx6-ppc_defconfig
$ make -j4
```

The build results is placed in the build directory under the **images/barebox-dmo-ppc.img** path.

2.14 Adding the customer layer

How to add/create own layers into yocto is described in The Yocto Project Development Manual, chapter 5.1.

Customers can commision Data Modul AG to setup a customer layer for there dedicated need. In such a case, Data Modul will deliver the customer layer separately, e.g. as a tar-file, not via its GitHub. The following part descibes how to add the customer layer to the Data Modul BSP.

Assuming the BSP was downloaded already according to chapters 2.5 and 2.6.

- Extract the archive meta-customer-xxx.bz2 to the poky folder inside the working-dir:

```
$ cd working-dir/poky
$ tar -xjf <path-to-tar-file>/meta-customer-xxx.bz2
$ ls -l
```

You should now see the meta-customer-xxx subdirectory in addition to the already available directories.

- To use the customer layer for building an image, follow the next steps inside the working-dir:

```
$ export TEMPLATECONF=meta-customer-xxx/conf
$ source ./poky/oe-init-build-env build-dir
```

- Building the customer image:

  After the previous steps, it was changed into the build-dir and the customer configuration is in place.

  Please note: It is essential to agree the end-user license agreement(EULA) of the fsl-arm/freescale layer inside local.conf file. For this, uncomment the line **#ACCEPT_FSL_EULA = "1"** in the *build-dir/conf/local.conf* file (see also 2.7).

  Now build the image:

```
$ bitbake dmo-image-xxx
```

  The build results can be found in *build-dir/tmp/deploy/images/{machine-name}/* as described in 2.7.

2.15 Getting informed for new releases

To get informed when a new version is released on GitHub, simply subscribe the *releases.atom* feeds of the repositories:

https://github.com/data-modul/meta-dmo/releases.atom
https://github.com/data-modul/linux-imx6/releases.atom
https://github.com/data-modul/barebox-imx6/releases.atom

Add these addresses to the RSS-Feeds in the mail-client and you will be informed about new releases by E-mail.

3 CPU Frequency

The Linux kernel provides a preferred interface in sysfs filesystem to support **Dynamic Voltage and Frequency Scaling(DVFS)**. It means kernel uses this interface to scale frequency to save power consumption. This interface is represented in the */sys/devices/system/cpu* directory. For each CPU core, there will be a subdirectory. Thus, user can set min/max frequency and a governor.

- Get list of available scaling frequencies

```
$ cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_available_frequencies
```

- Get maximum and minimum frequency

```
$ cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_max_freq
$ cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_min_freq
```

- Set new min/max frequency corresponding to the available scaling frequencies list.

```
$ echo 792000 > /sys/devices/system/cpu/cpu0/cpufreq/scaling_max_freq
```

- Get current CPU frequency

```
$ cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_cur_freq
```

Besides the frequency, a governor must be specified. The governor decides what speed the processor shall run within the boundaries. One governor is the **user space** governor. This one allows the user to decide what specific speed the processor shall run at.

- Get list of available governors

  $ **cat** /sys/devices/system/cpu/cpu0/cpufreq/scaling_available_governors

- Get current governor type

  $ **cat** /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor

- Set governor

  $ **echo** performance > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor

Explanation about each type of the governor is available at Linux cpu-freq document.

4 PWM

The PPC i.MX6 board defines two PWM channels: PWM1_OUT and PWM2_OUT.

PWM1 is used for display backlight brightness. The Linux handles backlight by specific driver, which is explained in the section 5.

Another one is the PWM2 functional output of the PWM. A modulated signal of the block is observed at this pin. It can be viewed as a clock signal whose period and duty cycle can be varied with different settings of the cycle of 50%.

PWM2 is a generic PWM channel, which can be handled by Linux with generic PWM interface. The interface is accessible via */sys/class/pwm/* from user space.

By default, PWM2 signal is disabled and instead of PWM, a GPIO signal is assigned to the corresponding pad. Thus, if PWM2 is needed, it should be enabled in the device tree.

```
pinctrl_pwm2: pwm2grp {
  fsl,pins = <
    MX6QDL_PAD_DISP0_DAT9_PWM2_OUT    0x0001B0B0
  >;
};

&pwm2 {
  pinctrl-names = "default";
  pinctrl-0 = <&pinctrl_pwm2>;
  status = "disabled";
};
```

Table 4: PWMs list

PWM name	Sysfs Path	Description
PWM1	/sys/class/backlight/backlight/	Used for display backlight brightness
PWM2	/sys/class/pwm/pwmchip1/ -	

Moreover, to use PWM2 in a user space, pwm should be exported at first:

```
$ cd /sys/class/pwm/pwmchip1/
$ echo 0 > export
```

Then, the pwm channel can be configured by assigning values to its *period*, *duty_cycle* and *enable* properties:

```
$ cd /sys/class/pwm/pwmchip1/
$ echo value1_nanosec > pwm0/period
$ echo value2_nanosec > pwm0/duty_cycle
$ echo 1 > enable
```

5 Backlight

A PWM channel is defined to handle display backlight brightness. By using a Linux kernel driver, this PWM channel is available to user-space in a different look than other PWM channels.

The device tree allows to define brightness level as an array. The range should start at 0. The actual brightness level will be modelled based on this array. 0 means 0% duty cycle and the last value shows a 100% duty cycle.

The default brightness value is an index of the **brightness-level** array entries.

```
backlight {
  brightness-levels = <0 1 10 20 30 40 50 60 70 75 80 90 100>;
  default-brightness-level = <7>;
};
```

The display backlight can be controlled using the sysfs kernel interface. The sysfs files are located in */sys/class/backlight/* directory.

- Check backlight system-ID. Usually it is */sys/class/backlight/backlight*, but it is possible to change from boot to boot.

  ```
  $ ls -al /sys/class/backlight/

  backlight -> ../../devices/soc0/backlight/backlight/backlight
  ```

- Get maximal brightness

  ```
  $ cd /sys/class/pwm/pwmchip1/
  $ echo 0 > export
  ```

  The valid brightness value is defined from 0 to max_brightness.

- Get current value of brightness

  ```
  $ cat /sys/class/backlight/backlight/max_brightness
  ```

- Set brightness level

  ```
  $ echo 5 > /sys/class/backlight/backlight/brightness
  ```

  Setting brightness to 0 makes the backlight off.

6 RTC

RTC clock is a hardware clock, which keeps the time even the system is in power-off mode, in case the battery exists.

On the board, RTC is located on second I2C bus. Its driver and proper register are defined in device tree, under I2C node.

When the system boots, the hardware clock is read. Then system clock, which is defined by Linux kernel, is set to the hardware clock. At shutdown, the system clock get stored into the hardware clock.

The RTC should be set at least one time, otherwise it cannot be read.

- Boot the board with the RTC

```
$ dmesg |grep rtc

rtc-m41t80 1-0068: rtc core: registered m41t62 as rtc0
rtc-m41t80 1-0068: setting system clock to 2017-10-16 06:06:27 UTC(1508133987)
```

- Check the current hardware clock

```
$ cat /proc/driver/rtc
rtc_time : 06:16:35
rtc_date : 2017-10-16
24hr : yes
```

or:

```
$ hwclock -r
```

- To set hardware clock to system clock, it is required to set system clock at first and then set hardware clock.

```
$ date --set="20171102␣08:15"
$hwclock -w
```

- Set system clock from hardware clock

```
$ hwclock -s
```

7 Watchdog

The PPC i.MX6 module provides two hardware watchdogs:

- A hardware watchdog on the Freescale i.MX6 processor
- A built-in hardware watchdog

Both watchdogs can be used to restart the system if after a specific time, an application crashes. It should take into account, that only one watchdog can be used at a time. Thus, when one watchdog is enabled, the other one should be disabled.

The driver of each watchdog, creates one character device under /dev/watchdog/. Through Linux Watchdog API, user can handle watchdog: open, trigger and close it.

7.1 i.MX6 Freescale Watchdog

The driver of the Freescale hardware watchdog is called *imx2-wdt*. In */sys/class/watchdog/* the list of detected watchdogs with corresponding node in devicetree is presented.

```
$ cd /sys/class/watchdog
$ ls -l

watchdog0->../../devices/soc0/soc/2000000.aips-bus/20bc000.wdog/watchdog/watchdog0
```

```
wdog1: wdog@020bc000 {
  compatible = "fsl,imx6q-wdt", "fsl,imx21-wdt";
  reg = <0x020bc000 0x4000>;
  interrupts = <0 80 IRQ_TYPE_LEVEL_HIGH>;
  clocks = <&clks IMX6QDL_CLK_DUMMY>;
};
```

The device regarding to i.MX6 watchdog is specified with *20bc000.wdog*. By default, the watchdog timeout is 60 seconds.

7.2 Built-in Hardware Watchdog

The built-in hardware watchdog is controlled through GPIO-lines. One GPIO-line is used to kick the watchdog, which is controlled inside the driver(The driver is called *wdt-gpio*). Another GPIO-line is required to enable watchdog. This GPIO-line (GPIO 37) should be controlled by user in an application. It means that using only the Linux API in order to work with watchdog is not sufficient and watchdog must be enabled via GPIO. The explanation will come in subsection 7.3.

The built-in watchdog uses window-watchdog method for safety critical applications. The watchdog window is set by an external capacitor in the module. The processor needs to trigger the watchdog within this window to avoid resetting the system.

In */sys/class/watchdog* the detected built-in watchdog is listed.

```
$ cd /sys/class/watchdog
$ ls -l

watchdog1 -> ../../devices/soc0/watchdog/watchdog/watchdog1
```

```
watchdog-gpio {
  compatible = "linux,wdt-gpio";
  gpios = <&gpio1 2 GPIO_ACTIVE_HIGH>;
  hw_algo = "level";
};
```

The device regarding to built-in watchdog is specified with *watchdog-gpio*. In this type, timeout is defined as a window. The window is:
100 ms < timeout < 1400 ms

7.3 C example

In the following, a simple C program shows how to handle watchdog. By end of the program, the watchdog will be closed and therefore no system reset. However, in general usage of watchdog, after the program is killed or stopped, watchdog should not be closed and therefore system will be reset.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <linux/watchdog.h>
```

```
#include <signal.h>

#define SYSFS_GPIO_DIR "/sys/class/gpio"
#define DEV_WATCHDOG   "/dev/watchdog1"

static int fd, fd_gpio;

static void term(int sig)
{
  /*****************************************************************/
  /* This part related only to built-in hardware watchdog       */
  /* the GPIO 37 is a GPIO line to disable built-in HW watchdog */
  /*****************************************************************/

  /* watchdog gpio pin disable */
  fd_gpio = open(SYSFS_GPIO_DIR "/gpio37/value", O_WRONLY);
  if( fd_gpio < 0 )
    return;
  write(fd_gpio, "0", 2);
  close(fd_gpio);

  fd_gpio = open(SYSFS_GPIO_DIR "/unexport", O_WRONLY);
  if( fd_gpio < 0 )
    return;
  write(fd_gpio, "37", 3);
  close(fd_gpio);

  /*****************************************************************/
  /* End of part related only to built-in hardware watchdog   */
  /*****************************************************************/

  /* watchdog closed */
  int n = write(fd, "V", sizeof("V"));
  close(fd);

  exit(0);
}

int main(int argc, char *argv[])
{
  int ret = 0;
  int dummy;
  unsigned int kicktime = 0;

  signal(SIGINT, term);

  /*****************************/
  /* Set the timeout, in ms     */
  /* It should be inside window */
  /*****************************/
  kicktime = 1000;

  /* watchdog open */
  fd = open(DEV_WATCHDOG, O_WRONLY);
```

```
  if (fd == -1)
    exit(EXIT_FAILURE);

  /*****************************************************************/
  /* This part related only to built-in hardware watchdog      */
  /* the GPIO 37 is a GPIO line to enable built-in HW watchdog */
  /*****************************************************************/

  /* watchdog gpio pin enable */
  fd_gpio = open(SYSFS_GPIO_DIR "/export", O_WRONLY);
  if( fd_gpio < 0 )
    return fd_gpio;
  write(fd_gpio, "37", 3);
  close(fd_gpio);

  fd_gpio = open(SYSFS_GPIO_DIR "/gpio37/direction", O_WRONLY);
  if( fd_gpio < 0 )
    return fd_gpio;
  write(fd_gpio, "high", 5);
  close(fd_gpio);

  /*****************************************************************/
  /* End of part related only to built-in hardware watchdog   */
  /*****************************************************************/

  /* watchdog kicking */
  while (1) {
    ret = ioctl(fd, WDIOC_KEEPALIVE, &dummy);
    /* If kicking failed, watchdog will not be closed, system reset */
    if (ret)
      break;

    usleep(kicktime * 1000);
  }

  return 0;
}
```

8 SD-Card

8.1 User-space

SD-card is supported as a block device and during system boot, it will appear in a form of MMC, due to using MMC subsystem.

```
$ dmesg |grep mmc1

[ 2.543033] sdhci-esdhc-imx 2198000.usdhc: No vqmmc regulator found
[ 2.597708] mmc1: SDHCI controller on 2198000.usdhc [2198000.usdhc] using ADMA
```

If SD-card is inserted, kernel will scan it and for each partition create a device node in /dev. Devices, which are corresponded to partitions, are named by using the appendix p{x}, e.g. mmcblk1p1 for the first partition of the SD-card.

```
$ dmesg |grep mmc1
[ 2.543033] sdhci-esdhc-imx 2198000.usdhc: No vqmmc regulator found
[ 2.597708] mmc1: SDHCI controller on 2198000.usdhc [2198000.usdhc] using ADMA
[ 3.324894] mmc1: new high speed SDHC card at address 0007
[ 3.331983] mmcblk1: mmc1:0007 SD16G 14.5 GiB
[ 3.347420] mmcblk1: p1 p2 p3 p4
```

If SD-card is not formatted, it does not have partition, thus /dev/mmcblk1 must be formatted at first in order to be operational.

```
$ fdisk /dev/mmcblk1
```

The partitions can be formatted in any type of file system and mounted or unmounted with proper commands.

```
$ mkfs.ext4 /dev/mmcblkip1
```

The first partition of SD-card is formatted with ext4 file system.

```
$ mount /dev/mmcblk1p1 /run/media/mmcblk1p1
$ umount /run/media/mmcblk1p1
```

Mount command returns the mounting status:

```
$ mount |grep mmc

/dev/mmcblk1p4 on /run/media/mmcblk1p4 type ext4 (rw,relatime,data=ordered)
/dev/mmcblk1p3 on /run/media/mmcblk1p3 type ext4 (rw,relatime,data=ordered)
/dev/mmcblk0p3 on /run/media/mmcblk0p3 type ext4 (rw,relatime,data=ordered)
/dev/mmcblk0p4 on /run/media/mmcblk0p4 type ext4 (rw,relatime,data=ordered)
```

8.2 Barebox

In addition to the user-space, SD-card is available in bootloader, Barebox. After stopping autoboot by pressing a key in Barebox, SD-card must be probed.

```
$ mmc2.probe=1
```

Partitions and their corresponding size will be returned by

```
$ devinfo

[...]
          `-- 2198000.usdhc
              `-- mmc2
                  `-- 0x00000000-0x39d3fffff ( 14.5 GiB): /dev/mmc2
                  `-- 0x00100000-0x033fffff ( 51 MiB): /dev/mmc2.0
                  `-- 0x03400400-0x9f8003ff ( 2.4 GiB): /dev/mmc2.1
                  `-- 0x9f800800-0xcb4007ff ( 700 MiB): /dev/mmc2.2
                  `-- 0xcb400c00-0xd7c007ff ( 200 MiB): /dev/mmc2.3

[...]
```

9 EMMC

9.1 User-space

The eMMC, which is also represented as a block device, is detected with */dev/mmcblk0* in Linux user-space.

EMMC can be recognized by its special hardware partitions *mmcblk0boot0*, *mmcblk0boot1* and *mmcblk0rpmb*. *boot0* and *boot1* can be used to store a bootloader. In PPC i.MX6 board, SPI flash is used to store a bootloader and do not use eMMC boot partitions to host it.

The user partitions of the device are named as the same as the device, while the partition number p{x} is appended at end, e.g. */dev/mmcblk0p1*.

```
$dmesg |grep mmc0

[ 2.123322] sdhci-esdhc-imx 219c000.usdhc: No vqmmc regulator found
[ 2.181514] mmc0: SDHCI controller on 219c000.usdhc [219c000.usdhc] using ADMA
[ 2.334693] mmc0: MAN_BKOPS_EN bit is not set
[ 2.885162] mmc0: new DDR MMC card at address 0001
[ 2.892441] mmcblk0: mmc0:0001 004G60 3.69 GiB
[ 2.897520] mmcblk0boot0: mmc0:0001 004G60 partition 1 2.00 MiB
[ 2.904666] mmcblk0boot1: mmc0:0001 004G60 partition 2 2.00 MiB
[ 2.913538] mmcblk0rpmb: mmc0:0001 004G60 partition 3 512 KiB
[ 2.932478] mmcblk0: p1 p2 p3 p4
```

The eMMC block device file can be accessed by applications as normal file using the open/read/write system calls.

It is possible to transfer the contents of the SD Card to the eMMC. For this process, the system should boot from SD-card(mmcblk1:SD-card, mmcblk0:eMMC) and makes sure that the eMMC is unmounted.

```
$ umount /run/media/mmcblk0p
$ dd if=/dev/mmcblk1 of=/dev/mmcblk0 conv=fdatasync bs=4K count=1048576
```

This takes some time. After reboot, the eMMC partitions should be mounted in */run/media*.

```
$ mount |grep mmcblk0

/dev/mmcblk0p4 on /home type ext4 (rw,relatime,data=ordered)
/dev/mmcblk0p3 on /run/media/mmcblk0p3 type ext4 (rw,relatime,data=ordered)
/dev/mmcblk0p2 on /run/media/mmcblk0p2 type ext4 (rw,relatime,data=ordered)
/dev/mmcblk0p1 on /run/media/mmcblk0p1 type vfat (rw,relatime,gid=6)
```

The contents of the partitions, e.g. third partition, can be inspected using the following commands:

```
$ cd /run/media/mmcblk0p3
$ ls -l
```

9.2 Barebox

In the Barebox, to retrieve eMMC device information, issue the following command after stopping autoboot.

```
$ mmc3.probe=1
$ devinfo
```

The following information should be found by devinfo:

```
[...]

          `-- 219c000.usdhc
              `-- mmc3
                  `-- 0x00000000-0xebffffff ( 3.7 GiB): /dev/mmc3
                  `-- 0x00100000-0x033fffff ( 51 MiB): /dev/mmc3.0
                  `-- 0x03400400-0x9f8003ff ( 2.4 GiB): /dev/mmc3.1
                  `-- 0x9f800800-0xcb4007ff ( 700 MiB): /dev/mmc3.2
```

```
                `-- 0xcb400c00-0xd7c007ff ( 200 MiB): /dev/mmc3.3

[...]
```

In order to boot operating system from eMMC, in Barebox, after stopping autoboot by pressing any key, use command *boot mmc3* to boot from eMMC.

10 SATA

10.1 User-space

The SATA, which is detected with */dev/sd_*, can be connected to the board as mSATA via pci subsystem, or eSATA with SATA-pcie adapter.

Note: There is only one pcie connector and it is possible to have eSATA device pluged in or mSATA at the same time.

The user partitions of the device are named as the same as the device, while the partition number is appended at end, e.g. */dev/sda1*.

```
$ls -la /dev/sda*

brw-rw---- 1 root disk 8, 0 Mar 14 15:00 /dev/sda
brw-rw---- 1 root disk 8, 1 Mar 14 15:00 /dev/sda1
brw-rw---- 1 root disk 8, 2 Mar 14 15:00 /dev/sda2
brw-rw---- 1 root disk 8, 3 Mar 14 15:00 /dev/sda3
brw-rw---- 1 root disk 8, 4 Mar 14 15:00 /dev/sda4
```

After reboot, the SATA partitions should be mounted in */run/media*.

```
$ mount |grep sda

/dev/sda2 on / type ext4 (rw,relatime,data=ordered)
/dev/sda1 on /run/media/sda1 type vfat (rw,relatime,shortname=mixed,errors=remount-ro)
/dev/sda3 on /run/media/sda3 type ext4 (rw,relatime,data=ordered)
/dev/sda4 on /run/media/sda4 type ext4 (rw,relatime,data=ordered)
```

The contents of the partitions, e.g. third partition, can be inspected using the following commands:

```
$ cd /run/media/sda3
$ ls -l
```

10.2 Barebox

In the Barebox, to retrieve SATA device information, issue the following command after stopping autoboot.

```
$ ata0.probe=1
$ devinfo
```

The following information should be found by devinfo:

```
[...]
          `-- 2200000.sata
           `-- ata0
            `-- 0x00000000-0xdf99e5fff ( 55.9 GiB): /dev/ata0
            `-- 0x00100000-0x033fffff ( 51 MiB): /dev/ata0.0
            `-- 0x03400400-0x9f8003ff ( 2.4 GiB): /dev/ata0.1
```

```
          `-- 0x9f800800-0xcb4007ff ( 700 MiB): /dev/ata0.2
          `-- 0xcb400c00-0xd7c007ff ( 200 MiB): /dev/ata0.3

[...]
```

In order to boot operating system from SATA, in Barebox, after stopping autoboot by pressing any key, use command *boot ata0* to boot from SATA.

11 SPI (Serial Peripheral Interface Bus)

The eDM-SBC-i.MX6-PPC provides two SPI interfaces.

Table 5: SPI interfaces list

Device tree	Linux kernel devices
ECSPI1	spi0.0, spi0.1
ECSPI3	spi2.0

The SPI bus cannot be accessed directly from user space. It could be accessed via a SPI device driver.

11.1 ECSPI1

On the standard BSP, on ECSPI1 no device is defined by default. Although it would be possible to use *spidev* to have access to it.

Spidev is a sample device driver that provides read and write access to the SPI bus through */dev*. Currently this sample device is disabled, because it is no more supported in device tree. In order to use it, the status field in device tree should be set to *okay*.

Note: Using spidev node will cause a warning to prevent users writing such device trees

```
&ecspi1 {
  /* The spidev node is not available by default since it is no more
   * supported in device trees. To have a dummy spi device, the following
   * nodes should be enabled( status = "okay" ) and in kernel config
   * CONFIG_SPI_SPIDEV=y
   */

  spidev@0 {
    reg = <0>;
    compatible = "spidev";
    spi-max-frequency = <1000000>;
    status = "disabled";
    };

    spidev@1 {
      reg = <1>;
      compatible = "spidev";
      spi-max-frequency = <1000000>;
      status = "okay";
    };
};
```

The sample *spidev* device creates device nodes */dev/spidev0.0* and */dev/spidev0.1*, which each one corresponds to the SPI bus chip select.

Fo reading from and writing to the SPI bus, an application is required, which uses the spidev interface. The source of the sample application can be found in *Documentation/spi/spidev_test.c* of the Linux kernel. This *spidev_test* is a simple utility used to test SPI functionality through *spidev* device.

For the existing source code, MISO and MOSI lines of the SPI bus should be short- circuited to create a loopback, otherwise it is required to modify source in order to write to a hardware device. The following shows how use the utility:

```
$ spidev_test -D /dev/spidev0.0 -v

spi mode: 0x0
bits per word: 8
max speed: 500000 Hz (500 KHz)
TX | FF FF FF FF FF FF 40 00 00 00 00 95 FF FF FF FF FF FF FF FF FF FF FF FF
     FF FF FF FF FF FF F0 0D |
RX | FF FF FF FF FF FF 40 00 00 00 00 95 FF FF FF FF FF FF FF FF FF FF FF FF
     FF FF FF FF FF FF F0 0D |
```

11.2 ECSPI3: SPI NOR Flash

SPI NOR flash in eDM-SBC-i.MX6-PPC connects to *ecspi3* interface.

```
&ecspi3 {
  flash: m25p80@0 {
    compatible = "m25p80";
    spi-max-frequency = <40000000>;
    reg = <0>;

    #address-cells = <1>;
    #size-cells = <1>;
  };
};
```

Currently, it is used to host Barebox as a bootloader. Therefore, two partitions are defined on SPI NOR flash: *barebox* and *barebox-environment*.

```
partition@0 {
  label = "barebox";
  reg = <0x0 0x80000>;
};

partition@1 {
  label = "barebox-environment";
  reg = <0x80000 0x20000>;
};
```

11.2.1 User space

The NOR-Flash block device is represented as */dev/mtdX*. If the NOR-Flash is detected, these messages should appear during boot.

```
[ 1.741114] m25p80 spi2.0: found s25fl164k, expected m25p80
[ 1.746719] m25p80 spi2.0: s25fl164k (8192 Kbytes)
```

and in case of flashing Barebox on SPI, extra messages about MTD partitions:

```
[ 1.751637] 2 ofpart partitions found on MTD device spi2.0
[ 1.757135] Creating 2 MTD partitions on "spi2.0":
[ 1.761983] 0x000000000000-0x000000080000 : "barebox"
[ 1.769402] 0x000000080000-0x0000000a0000 : "barebox-environment"
```

SPI NOR Flash would be accessed via the mtd-utils e.g. nandwrite, nanddump.

- To get detailed information about a MTD device

```
$ mtdinfo
Count of MTD devices: 2
Present MTD devices: mtd0, mtd1
Sysfs interface supported: yes
```

- To write Barebox image into the first partition of the NOR flash

```
$ flashcp barebox.img /dev/mtd0
```

- To check flash devices

```
$ cat /proc/mtd
```

- To dump flash device in length of 1000 bytes

```
$ hexdump -C -n 1000 /dev/mtd0
```

11.2.2 Barebox

In Barebox boot, the NOR-Flash should appear in the Barebox log:

```
m25p80 m25p80@00: s25fl164k (8192 Kbytes)
```

If Barebox is not flashed into the SPI-Flash, it will be possible to program SPI via USB-stick. The USB-stick should be FAT or ext4 format. By assuming that there is only one partition on the stick, the following commands will write Barebox into SPI flash.

```
$ usb
$ mkdir /mnt/usb
$ mount /dev/disk0.0 /mnt/usb
$ barebox_update -t spiflash -y /mnt/usb/barebox.imgroot

update succeed
```

12 USB Host Controller

The USB Controller in the PPC i.MX6 module consists of two USB controller cores: one On-The-Go(OTG) controller core and the other one, host-only controller core.

The Host-only controller core can operate in Host mode only.

The PPC i.MX6 BSP supports mass storage devices, mouse and keyboard. For other USB devices, the corresponding device driver must be set in Linux configuration.

```
&usbh1 {
  pinctrl-names = "default";
  vbus-supply = <&reg_sys_3v3>;
  dr_mode = "host";
  status = "okay";
};
```

In Linux USB device is auto detected and the USB-sticks are auto mounted under /run/media.

Moreover, each USB mass storage device gets a unique ID, which is available in */dev/disk/by-id/*

13 USBOTG

The OTG controller core can operate in HOST mode and Device mode. In current setting, the USBOTG is configured in host-mode. Thus when an USB mass storage device is attached to the USBOTG port, it will be detected as */dev/sda* in kernel.

```
&usbotg {
  pinctrl-names = "default";
  pinctrl-0 = <&pinctrl_usbotg>;
  vbus-supply = <&reg_sys_4v2>;
  dr_mode = "host";
  status = "okay";
};
```

14 PCIe

The PPC i.MX6 provides a MINI PCI Express SMT Socket 52 Pin, connected to the respective PCIe pins of the i.MX6.

The i.MX6 provides the PCIe reference clock on CLKx_P/N, which is a LVDS port. According to the check list of the hardware design, provided by NXP community, this clock does not match PCIe reference clock specification and does not pass PCIe GEN2 compliance test.

The PPC i.MX6 is prepared to support an external clock generator, which feeds the i.MX6 on CLKx_P/N (switched to input) and the mPCIe socket(REFCLK- and REFCLK+).

By default, PPC i.MX6 is using the internal clock generator.

On customer request however, we can provide special board variants supporting the external clock generator, if there will be any complains or issues regarding clock integrity.

In such cases, the external clock generator will be enabled permanently by an external pulldown resistor on the CLKREQ# signal, normally provided by the inserted PCIe module.

For controlling the PCIe port with external clock generator, the following parts are prepared in the device tree:

```
&iomuxc {

  ...

  pinctrl_pcie: pciegrp {
```

```
    fsl,pins = <
      MX6QDL_PAD_EIM_D19__GPIO3_IO19    0x1b0b0 /* pwr en */
      MX6QDL_PAD_CSI0_DATA_EN__GPIO5_IO20 0x1b0b0 /* wakeup */
      MX6QDL_PAD_GPIO_17__GPIO7_IO12    0x130b0 /* reset */
      MX6QDL_PAD_GPIO_19__GPIO4_IO05    0x1b0b0 /* dis */
      MX6QDL_PAD_EIM_CS0__GPIO2_IO23    0x4001b0b0 /* PCIE_CLKREQ#  */
    >;
  };

        ...
};

clocks {

        ...

  anaclk1 {
    compatible = "fixed-clock";
    reg = <0>;
    #clock-cells = <0>;
    clock-frequency = <100000000>;  /* 100MHz */
  };
};

soc {

        ...

  pcie: pcie@0x01000000 {
    compatible = "fsl,imx6q-pcie", "snps,dw-pcie";
    reg = <0x01ffc000 0x04000>,
          <0x01f00000 0x80000>;
    reg-names = "dbi", "config";
    #address-cells = <3>;
    #size-cells = <2>;
    device_type = "pci";
    ranges = <0x00000800 0 0x01f00000 0x01f00000 0 0x00080000 /* configuration space */
        0x81000000 0 0          0x01f80000 0 0x00010000 /* downstream I/O */
        0x82000000 0 0x01000000 0x01000000 0 0x00f00000>; /* non-prefetchable memory */
    num-lanes = <1>;
    interrupts = <GIC_SPI 120 IRQ_TYPE_LEVEL_HIGH>;
    interrupt-names = "msi";
    #interrupt-cells = <1>;
    interrupt-map-mask = <0 0 0 0x7>;
    interrupt-map = <0 0 0 1 &gpc GIC_SPI 123 IRQ_TYPE_LEVEL_HIGH>,
                    <0 0 0 2 &gpc GIC_SPI 122 IRQ_TYPE_LEVEL_HIGH>,
                    <0 0 0 3 &gpc GIC_SPI 121 IRQ_TYPE_LEVEL_HIGH>,
                    <0 0 0 4 &gpc GIC_SPI 120 IRQ_TYPE_LEVEL_HIGH>;
    clocks = <&clks IMX6QDL_CLK_PCIE_AXI>,
       <&clks IMX6QDL_CLK_LVDS1_GATE>,
       <&clks IMX6QDL_CLK_PCIE_REF_125M>;
    clock-names = "pcie", "pcie_bus", "pcie_phy";
    status = "disabled";
  };
```

```
60
61
62 &clks {
63   assigned-clocks = ...,
64       <&clks IMX6QDL_PLL6_BYPASS_SRC>,
65       <&clks IMX6QDL_PLL6_BYPASS>;
66   assigned-clock-parents = ...,
67      <&clks IMX6QDL_CLK_LVDS1_IN>,
68      <&clks IMX6QDL_PLL6_BYPASS_SRC>;
69   assigned-clock-rates = <100000000>, <100000000>;
70 };
71
72 &pcie {
73   pinctrl-names = "default";
74   pinctrl-0 = <&pinctrl_pcie>;
75   reset-gpio = <&gpio7 12 GPIO_ACTIVE_HIGH>;
76   clkreq-gpio = <&gpio2 23 GPIO_ACTIVE_LOW>;
77   clocks = <&clks IMX6QDL_CLK_PCIE_AXI>,
78            <&clks IMX6QDL_CLK_LVDS1_IN>,
79            <&clks IMX6QDL_CLK_SATA_REF_100M>,
80            <&clks IMX6QDL_PLL6_BYPASS>,
81          <&clks IMX6QDL_PLL6_BYPASS_SRC>;
82   clock-names = "pcie", "pcie_bus", "pcie_phy", "pcie_ext", "pcie_ext_src";
83   ext_osc = <1>;
84   fsl,max-link-speed = <1>;
85   status = "okay";
86 };
```

GPIO02_23 (= GPIO55 = input) is used reading the CLKREQ# signal during PCIe-probing. In case of using an external clock generator, PLL6 will be bypassed according to this content.

In all other cases, i.MX6-internal clocks will be used.

14.1 Detecting a PCIe-module

After connecting a device via PCIe slot and booting up the system, the device should be detected. It is assumed that the respective driver is properly configured in the kernel.

The following log shows the detection of the PCI bridge and an Intel Wireless Mini PCI Express card.

```
$ lspci -v

00:00.0 PCI bridge: Synopsys, Inc. Device abcd (rev 01) (prog-if 00 [Normal decode])
        Flags: bus master, fast devsel, latency 0, IRQ 304
        Memory at 01000000 (32-bit, non-prefetchable) [size=1M]
        Bus: primary=00, secondary=01, subordinate=01, sec-latency=0
        I/O behind bridge: None
        Memory behind bridge: 01100000-011fffff [size=1M]
        Prefetchable memory behind bridge: None
        [virtual] Expansion ROM at 01200000 [disabled] [size=64K]
        Capabilities: [40] Power Management version 3
        Capabilities: [50] MSI: Enable- Count=1/1 Maskable+ 64bit+
        Capabilities: [70] Express Root Port (Slot-), MSI 00
        Capabilities: [100] Advanced Error Reporting
        Capabilities: [140] Virtual Channel
```

```
        Kernel driver in use: pcieport

01:00.0 Network controller: Intel Corporation Wireless 3160 (rev cb)
        Subsystem: Intel Corporation Dual Band Wireless-AC 3160
        Flags: bus master, fast devsel, latency 0, IRQ 304
        Memory at 01100000 (64-bit, non-prefetchable) [size=8K]
        Capabilities: [c8] Power Management version 3
        Capabilities: [d0] MSI: Enable- Count=1/1 Maskable- 64bit+
        Capabilities: [40] Express Endpoint, MSI 00
        Capabilities: [100] Advanced Error Reporting
        Capabilities: [140] Device Serial Number e4-02-9b-ff-ff-c6-42-fb
        Capabilities: [14c] Latency Tolerance Reporting
        Capabilities: [154] Vendor Specific Information: ID=cafe Rev=1 Len=014 <?>
        Kernel driver in use: iwlwifi
```

15 GPIO

PPC i.MX6 board can provide a set of dedicated GPIO pins for user IO-usage. Before use any GPIO, be sure that the pin is muxed as a GPIO in device tree.

```
[...]
pinctrl_hog: hoggrp {
  fsl,pins = <
    /* hwid 0 */
    MX6QDL_PAD_SD2_CMD__GPIO1_IO11    0x1b0b0
    /* hwid 1 */
    MX6QDL_PAD_SD2_CLK__GPIO1_IO10    0x1b0b0
    /* hwid 2 */
    MX6QDL_PAD_SD2_DAT0__GPIO1_IO15   0x1b0b0
    /* hwid 3 */
    MX6QDL_PAD_SD2_DAT1__GPIO1_IO14   0x1b0b0
    /* hwid 4 */
    MX6QDL_PAD_SD2_DAT2__GPIO1_IO13   0x1b0b0
    /* hwid 5 */
    MX6QDL_PAD_SD2_DAT3__GPIO1_IO12   0x1b0b0
    /* hwid 6 */
    MX6QDL_PAD_SD1_CMD__GPIO1_IO18    0x1b0b0
    /* hwid 7 */
    MX6QDL_PAD_SD1_CLK__GPIO1_IO20    0x1b0b0
    /* dram id 0 */
    MX6QDL_PAD_EIM_D18__GPIO3_IO18    0x0001B0B0
    /* dram id 1 */
    MX6QDL_PAD_EIM_D17__GPIO3_IO17    0x1b0b0
    /* jog dial ina */
    MX6QDL_PAD_CSI0_DAT16__GPIO6_IO02 0x0001B0B0
    /* jog dial inb */
    MX6QDL_PAD_CSI0_DAT17__GPIO6_IO03 0x0001B0B0
    /* jog dial push in */
    MX6QDL_PAD_DISP0_DAT9__GPIO4_IO30 0x0001B0B0
  >;
[...]
```

GPIO pins are presented in 7 banks. Each bank, except the last one, comes with 32 GPIOs. The 7th GPIO bank has 14 GPIOs. Therefore, in Linux user-space, the following gpiochips, as representation of each bank,

are present.

```
$ cd /sys/class/gpio
$ ls -l

total 0
export
gpiochip0 -> ../../devices/soc0/soc/2000000.aips-bus/209c000.gpio/gpio/gpiochip0
gpiochip128 -> ../../devices/soc0/soc/2000000.aips-bus/20ac000.gpio/gpio/gpiochip128
gpiochip160 -> ../../devices/soc0/soc/2000000.aips-bus/20b0000.gpio/gpio/gpiochip160
gpiochip192 -> ../../devices/soc0/soc/2000000.aips-bus/20b4000.gpio/gpio/gpiochip192
gpiochip32 -> ../../devices/soc0/soc/2000000.aips-bus/20a0000.gpio/gpio/gpiochip32
gpiochip64 -> ../../devices/soc0/soc/2000000.aips-bus/20a4000.gpio/gpio/gpiochip64
gpiochip96 -> ../../devices/soc0/soc/2000000.aips-bus/20a8000.gpio/gpio/gpiochip96
unexport
```

The GPIOs are named as: GPIOx_IOyy, where x stands for bank number(x=1 . . . 7) and yy is used as GPIO number within each bank(yy=00 . . . 31, except for x=7: yy=00 . . . 13).

Linux uses only one number to identify each GPIO. Thus, the following formula can be used to figure out the proper number:

```
GPIO# = ((x - 1) * 32) + yy
```

To access GPIOs from Linux user-space, sysfs API can be used(/sys/class/gpio). As a requirement, the desired GPIO should be exported at first. The below command will add *gpio160* to the GPIO list.

```
$ echo 160 > /sys/class/gpio/export
```

Inside gpio160 directory, *direction* and *value* files allow to control GPIO.

```
$ cd /sys/class/gpio/gpio160
$ ls -l

total 0
active_low
device -> ../../../20b0000.gpio
direction
edge
power
subsystem -> ../../../../../../../class/gpio
uevent
value
```

- Set gpio160 as output/input

  ```
  $ echo out > /sys/class/gpio/gpio160/direction
  $ echo int > /sys/class/gpio/gpio160/direction
  ```

- Get gpio160 value

  ```
  $ cat /sys/class/gpio/gpio160/value
  ```

- Set gpio160 value to 0/1

  ```
  $ echo 0 > /sys/class/gpio/gpio160/value
  $ echo 1 > /sys/class/gpio/gpio160/value
  ```

When a direction of the GPIOs is defined as *input*, the value cannot be changed. Only read operation is defined in this mode. To write a new value, the direction should be changed to *output* firstly.

- To directly change gpio160 to output and set its value to 1

```
$ echo high > /sys/class/gpio/gpio160/direction
```

- To directly change gpio160 to output and set its value to 0

```
$ echo low > /sys/class/gpio/gpio160/direction
```

Besides above activities, it is possible to configure GPIO as an interrupt source.

- Configure GPIO edge to rising/falling/both

```
$ echo in > /sys/class/gpio/gpio160/direction
$ echo rising > /sys/class/gpio/gpio160/edge
```

For a quick view, the exported GPIOs over sysfs can be checked using the below commands:

```
$ cat /sys/kernel/debug/gpio

GPIOs 0-31, platform/209c000.gpio, 209c000.gpio:
 gpio-2 ( |watchdog-gpio ) out hi
 gpio-5 ( |enable ) out hi

GPIOs 32-63, platform/20a0000.gpio, 20a0000.gpio:
 gpio-32 ( |cd ) in hi
 gpio-33 ( |wp ) in lo

GPIOs 64-95, platform/20a4000.gpio, 20a4000.gpio:
 gpio-83 ( |stmpe_vcc ) out hi

GPIOs 96-127, platform/20a8000.gpio, 20a8000.gpio:
 gpio-105 ( |spi_imx ) in lo
 gpio-120 ( |spi_imx ) out hi

GPIOs 128-159, platform/20ac000.gpio, 20ac000.gpio:
 gpio-137 ( |spi_imx ) in lo

GPIOs 160-191, platform/20b0000.gpio, 20b0000.gpio:

GPIOs 192-223, platform/20b4000.gpio, 20b4000.gpio:
 gpio-204 ( |PCIe reset ) out hi
```

More information about how these GPIOS can be used, is described in the kernel GPIO documentation manual.

16 I2C

The PPC i.MX6 provides 3 I2C busses. Most modules, such as PMIC, audio codec and RTC use I2C bus. Depending on the request, additional I2C devices as brightness or temperature sensor can apply on I2C bus too.

```
&i2c1 {
  pinctrl-names = "default";
  pinctrl-0 = <&pinctrl_i2c1>;
  status = "okay";
  [...]
};

&i2c2 {
  pinctrl-names = "default";
  pinctrl-0 = <&pinctrl_i2c2>;
  status = "okay"
  [...]

  pmic: pfuze100@08 {
    compatible = "fsl,pfuze100";
    reg = <0x08>;
    [...]
  };

  codec: sgtl5000@0a {
    compatible = "fsl,sgtl5000";
    reg = <0x0a>;
    [...]
  };

  rtc: m41t62@86 {
    compatible = "stm,m41t62";
    reg = <0x68>;
  };
};

&i2c3 {
  pinctrl-names = "default";
  pinctrl-0 = <&pinctrl_i2c3>;
  status = "okay";
  [...]
};
```

The table 6 shows I2C bus assignment for the default I2C bus devices, which are located on the board.

Table 6: I2C buses

I2C address	I2C Bus device	I2C Bus number
0xD0	RTC Controller	I2C2
xxxx	LVDS Display	I2C2
0x14	Audio Codec	I2C2
0x10	PMIC	I2C2
var.	mPCIE Connector	I2C3

To access I2C devices in user-space, most kernel drivers provide relevent methods to get/set data from/to that device, but in general, user can do operations through low-level I2C access.

- Check I2C busses

```
$ ls -al /dev/i2c*

dev/i2c-0
dev/i2c-1
dev/i2c-2
```

- Check detection of each device on corresponding bus

```
$ i2cdetect -y 1

     0 1 2 3 4 5 6 7 8 9 a b c d e f
00: ----------UU --UU ----------
10: --------------------------------
20: --------------------------------
30: --------------------------------
40: --------------------------------
50: --------------------------------
60: ---------------UU --------------
70: ---------------
```

As displayed above, three devices are detected on I2C1. The I2C addresses in *Bus assignment* table are represented as 8-bit address. Because I2C bus drivers support 7-bit address, 8-bit addresses should be shifted one bit right. The 7-bit address are equal to the *reg* label in i2c node of device tree.

Note: UU stands for a device which is used by a kernel driver and usually cannot be touched from user-space.

- I2C tools to have low-level access
  - i2cdump : examine I2C registers
  - i2cget : read from I2C chip registers
  - i2cset : set I2C registers

For more information related to access i2c devices in application, check kernel I2C document.

17 UART

The PPC i.MX6 provides maximum five uart units. Depends on the requirement, it is possible that uart3 is configured to be used as can1.

```
pinctrl_uart1: uart1grp {
  fsl,pins = <
    MX6QDL_PAD_CSI0_DAT11__UART1_RX_DATA  0x1b0b0
    MX6QDL_PAD_CSI0_DAT10__UART1_TX_DATA  0x1b0b0
  >;
};

pinctrl_uart2: uart2grp {
  fsl,pins = <
    MX6QDL_PAD_EIM_D26__UART2_TX_DATA 0x0001B0B0
    MX6QDL_PAD_EIM_D27__UART2_RX_DATA 0x0001B0B0
    MX6QDL_PAD_EIM_D28__UART2_CTS_B   0x0001B0B0
    MX6QDL_PAD_EIM_D28__UART2_RTS_B   0x0001B0B0
  >;
```

```
};

pinctrl_uart3: uart3grp {
  fsl,pins = <
    MX6QDL_PAD_EIM_D25__UART3_RX_DATA 0x1b0b0
    MX6QDL_PAD_EIM_D24__UART3_TX_DATA 0x1b0b0
  >;
};

pinctrl_uart4: uart4grp {
  fsl,pins = <
    MX6QDL_PAD_CSI0_DAT12__UART4_TX_DATA  0x0001B0B0
    MX6QDL_PAD_CSI0_DAT13__UART4_RX_DATA  0x0001B0B0
  >;
};

pinctrl_uart5: uart5grp {
  fsl,pins = <
    MX6QDL_PAD_CSI0_DAT14__UART5_TX_DATA  0x0001B0B0
    MX6QDL_PAD_CSI0_DAT15__UART5_RX_DATA  0x0001B0B0
  >;
};
```

The uart ports which are defined in a device tree file, are identified by Linux kernel as tty devices under /dev.

```
$ dmesg |grep tty

2020000.serial: ttymxc0 at MMIO 0x2020000(irq = 25, base_baud = 5000000) is a IMX
console [ttymxc0] enabled
21e8000.serial: ttymxc1 at MMIO 0x21e8000(irq = 294, base_baud = 5000000) is a IMX
21ec000.serial: ttymxc2 at MMIO 0x21ec000(irq = 295, base_baud = 5000000) is a IMX
21f0000.serial: ttymxc3 at MMIO 0x21f0000(irq = 296, base_baud = 5000000) is a IMX
21f4000.serial: ttymxc4 at MMIO 0x21f4000(irq = 297, base_baud = 5000000) is a IMX
```

Note: ttymxc0 is used by the Linux as a debug console.
Note: uart2 is RS485 uart.
Note: uart3 is enabled, when CAN1 bus is disabled.

Table 7: The list of the UARTs

UART name	Device name in Linux	Note
uart1	/dev/ttymxc0	Default debug console
uart2	/dev/ttymxc1	RS485
uart3	/dev/ttymxc2	Enable when CAN1 is disabled
uart4	/dev/ttymxc3	can be with RTS/CTS
uart5	/dev/ttymxc4	can be with RTS/CTS

17.1 Console

In order to use serial debug console, connect i.MX6 board to the host via COM port and a null model cable.

Identify corresponding number of the serial port in host.

```
$ dmesg |grep tty

[ 0.000000] console [tty0] enabled
[ 0.680710] 00:03: ttyS0 at I/O 0x3f8 (irq = 4, base_baud = 115200) is a 16550A
```

In host, use a terminal program, such as *microcom* to connect to the target.

```
$ microcom /dev/ttyS0

connected to /dev/ttyS0
Escape character: Ctrl-\
Type the escape character followed by c to get to the menu or q to quit

root@imx6s-dmo-ppc:~#
```

The board serial console has the following configuration: baud rate 115200 8N1, no flow control.

For getting information about how programming a serial port, follow Linux man page.

17.2 RS485 half duplex

On PPC i.MX6 UART2 is used for RS485 half duplex mode, where the direction control will be handled by the UARTs CTS signal as described in the i.MX6 reference manual.

```
&uart2 {
  fsl,uart-has-rtscts;
  linux,rs485-enabled-at-boot-time;
  rs485-rts-active-high;
};
```

In the following, a sample C source shows how to configure and use RS485. The program is implemented based on Linux RS485 documentation.

```
static int ser_setup_serial(int fd, char* dev)
{
  int status;
  struct termios ti;
  struct serial_rs485 rs485conf;

  /* Enable RS485 mode: */
  rs485conf.flags |= SER_RS485_ENABLED;

  /* Set logical level for RTS pin equal to 1 when sending: */
  rs485conf.flags |= SER_RS485_RTS_ON_SEND;
  /* or, set logical level for RTS pin equal to 0 when sending: */
    /*   rs485conf.flags &= ~(SER_RS485_RTS_ON_SEND); */

  /* Set logical level for RTS pin equal to 1 after sending: */
    /*   rs485conf.flags |= SER_RS485_RTS_AFTER_SEND; */
  /* or, set logical level for RTS pin equal to 0 after sending: */
  rs485conf.flags &= ~(SER_RS485_RTS_AFTER_SEND);

  /* Set rts delay before send, if needed: */
  rs485conf.delay_rts_before_send = 100;

  /* Set rts delay after send, if needed: */
  rs485conf.delay_rts_after_send = 100;
```

```

  /* Set this flag if you want to receive data even whilst sending data */
    /*  rs485conf.flags |= SER_RS485_RX_DURING_TX; */

  status = ioctl (fd, TIOCSRS485, &rs485conf);
  if (status < 0) {
    printf("%s: Unable to configure port in 485 mode, status (%i)\n", dev, status);
    return -1;
  }

  /* Set the port speed */
  tcgetattr(fd, &ti);
    ti.c_cflag = CS8 | CLOCAL | CREAD;
    ti.c_iflag = IGNPAR | IGNCR;
    ti.c_oflag = ONOCR | OCRNL;
    ti.c_lflag = ICANON;
  cfsetospeed(&ti, BAUDRATE);
  cfsetispeed(&ti, BAUDRATE);
  tcsetattr(fd, TCSANOW, &ti);

  return 0;
}

...
  #define BAUDRATE B9600
  char rs485_dev[] = "/dev/ttymxc1";
  int fd;

  fd = open (rs485_dev, O_RDWR);
  if (fd < 0) {
    printf ("ERROR: cannot open %s\n", rs485_dev);
    exit (1);
  }

  if (ser_setup_serial(fd, rs485_dev))
    exit (1);
```

Now the interface can be used for reading and writing data.

18 CAN

On PPC i.MX6, two Flexible Controller Area Network(FLEXCAN) busses are available. FLEXCAN is a communication controller implementing the CAN protocol according to the CAN 2.0B protocol specification.

The FLEXCAN modules are supported by Linux drivers and are named *can1* and *can2* in the BSP device tree.

```
pinctrl_can1: can1grp {
  fsl,pins = <
    MX6QDL_PAD_GPIO_8__FLEXCAN1_RX    0x1b0b0
    MX6QDL_PAD_KEY_COL2__FLEXCAN1_TX  0x1b0b0
};
pinctrl_can2: can2grp {
  fsl,pins = <
```

```
    MX6QDL_PAD_KEY_ROW4__FLEXCAN2_RX  0x1b0b0
    MX6QDL_PAD_KEY_COL4__FLEXCAN2_TX  0x1b0b0
};

&can1 {
  pinctrl-names = "default";
  pinctrl-0 = <&pinctrl_can1>;
  status = "okay";
};
&can2 {
  pinctrl-names = "default";
  pinctrl-0 = <&pinctrl_can2>;
  status = "okay";
};
```

The Linux BSP supports two flexCAN buses, that are implemented with the help of test hardware schematics add-on boards. Linux CAN document provides related information in order to access CAN buses using the socket CAN library

The CAN utility, *canutils*, is installed on the PPC i.MX6 BSP. Following commands figure out if CAN buses are working probably.

1. List CAN interfaces and see whether the interface is up or down

```
$ ip link

2: can0: <NOARP,ECHO> mtu 16 qdisc noop state DOWN mode DEFAULT group default
    qlen 10
      link/can
3: can1: <NOARP,ECHO> mtu 16 qdisc noop state DOWN mode DEFAULT group default
    qlen 10
      link/can
```

2. Bring CAN interface up in each device

```
$ ip link set can0 type can bitrate 125000 triple-sampling on
$ ip link set dev can0 up
```

3. Receive on CAN interface(e.g. device1)

```
$ candump can0
```

4. Send on CAN interface(e.g. device2)

```
$ cansend can0 0xde 0xad 0xbe 0xef
```

5. Check CAN interface statistics

```
$ ip -d -s link show dev can0

2: can0: <NOARP,UP,LOWER_UP,ECHO> mtu 16 qdisc pfifo_fast state UNKNOWN mode
          DEFAULT group default qlen 10
      link/can promiscuity 0
      can <TRIPLE-SAMPLING> state ERROR-ACTIVE (berr-counter tx 0 rx 0)
          restart-ms 0
          bitrate 125000 sample-point 0.875
          tq 500 prop-seg 6 phase-seg1 7 phase-seg2 2 sjw 1
```

```
          flexcan: tseg1 4..16 tseg2 2..8 sjw 1..4 brp 1..256 brp-inc 1
          clock 30000000
          re-started bus-errors arbit-lost error-warn error-pass bus-off
          0 0 0 0 0 0
    RX: bytes packets errors dropped overrun mcast
    0 0 0 0 0 0
    TX: bytes packets errors dropped carrier collsns
    31972 7994 0 0 0 0
```

Note: It is possible that, there is an isolated CAN-Bus interface on a variant of PPC board, which is connected to one of the existing CAN-bus.

19 Temperature sensor

The PPC i.MX6 does not provide external sensor, but the i.MX6 core temperature is being monitored via IMX_THERMAL driver within Linux kernel. Corresponding action, regarding to the temperature changes, could be defined by user or a default kernel algorithm will be applied(CPU_THERMAL);

Default kernel algorithm, which is called CPU cooling, reacts on CPU temperature changes and modify CPU frequency to the proper value.

Thermal management in kernel is done via sysfs API. *Thermal zone attributes* as well as *cooling device attribute* are exposed to the user-space via sysfs. (see kernel sysfs directory /sys/class/thermal/thermal_zone0).

The current temperature can be read in milicelsius through the following directory:

```
$ cat /sys/class/thermal/thermal_zone0/temp
```

There are two trip points registered by IMX_THERMAL driver. These points are used by the kernel to manage the cooling behavior. If the temperature(temp) is above trip_point_0_temp, the CPU frequency is set to the lowest CPU frequency. If the temperature(temp) reaches trip_point_1_temp, the thermal management will shut down the system.

The *trip_point_0_temp* threshold can be changed through:

```
$ echo 60000 > /sys/class/thermal/thermal_zone0/trip_point_0_temp
```

The below tables explain attributes and their values.

Table 8: Thermal zone attributes

Attributes	r/w	Values	Description
cdev0_trip_point	r	-1/0	-1:cooling device is not associated with any trip point
mode	rw	enabled/disabled	Algorithm manages the thermal zone. enabled: default kernel algorithm, disabled: user space application
policy	rw	Step Wise/Fair Share/Bang Bang/User space	Thermal governor used for this zone
temp	r	-	Current temperature as reported by sensor
trip_point_0_temp	rw	-	The temperature above which trip point will be fired
trip_point_0_type	r	critical/hot/passive/active	Trip point0 type
trip_point_1_temp	r	-	The temperature above which trip point will be fired
trip_point_1_type	r	critical/hot/passive/active	Trip point1 type

Table 9: Cooling device attributes

Attributes	r/w	Values	Description
cur_state	rw	0...max_state	Current cooling state of the cooling device. 0 means no cooling
max_state	r	-	Maximum cooling state of the cooling device

20 Ethernet

The DATA Modul PPC i.MX6 provides a Gigabit Ethernet network connection. In addition, it is possible to have LAN port via PCIe or USB-LAN adapter. In this case, depending on which Ethernet-card or USB-LAN adapter is used, proper Linux driver should be built in kernel image.

The i.MX6 ethernet controller communicates through RGMII interface. The following nodes are required in the device tree to enable ethernet network connection.

```
pinctrl_fec: fecgrp { /* Ethernet */
  fsl,pins = <
    MX6QDL_PAD_ENET_MDC__ENET_MDC     0x0001B0B0
    MX6QDL_PAD_ENET_MDIO__ENET_MDIO     0x0001B0B0
    MX6QDL_PAD_ENET_REF_CLK__ENET_TX_CLK  0x0001B0B0
    MX6QDL_PAD_RGMII_RD0__RGMII_RD0     0x0001B030
    MX6QDL_PAD_RGMII_RD1__RGMII_RD1     0x0001B030
    MX6QDL_PAD_RGMII_RD2__RGMII_RD2     0x0001B030
    MX6QDL_PAD_RGMII_RD3__RGMII_RD3     0x0001B030
    MX6QDL_PAD_RGMII_RXC__RGMII_RXC     0x00013030
    MX6QDL_PAD_RGMII_RX_CTL__RGMII_RX_CTL 0x00013030
    MX6QDL_PAD_RGMII_TD0__RGMII_TD0     0x0001B030
    MX6QDL_PAD_RGMII_TD1__RGMII_TD1     0x0001B030
```

```
    MX6QDL_PAD_RGMII_TD2__RGMII_TD2     0x0001B030
    MX6QDL_PAD_RGMII_TD3__RGMII_TD3     0x0001B030
    MX6QDL_PAD_RGMII_TXC__RGMII_TXC     0x00013030
    MX6QDL_PAD_RGMII_TX_CTL__RGMII_TX_CTL 0x00013030
    MX6QDL_PAD_ENET_CRS_DV__GPIO1_IO25    0x0001B0B0 /* enet reset */
    MX6QDL_PAD_ENET_TX_EN__GPIO1_IO28   0x0001B0B0 /* INT/PWDN */
  >;
};

&fec {
  pinctrl-names = "default";
  pinctrl-0 = <&pinctrl_fec>;
  phy-mode = "rgmii";
  phy-handle = <&ethphy>;
  phy-reset-gpios = <&gpio1 25 0>; /* GPIO1_IO25  */
  phy-supply = <&sw4_reg>;
  status = "okay";

  mdio {
    #address-cells = <1>;
    #size-cells = <0>;
    ethphy: ethernet-phy@0 {
      reg = <0>;
      ti,rx-internal-delay = <DP83867_RGMIIDCTL_2_00_NS>;
      ti,tx-internal-delay = <DP83867_RGMIIDCTL_1_50_NS>;
      ti,fifo-depth = <DP83867_PHYCR_FIFO_DEPTH_4_B_NIB>;
      ti,clk_out = <DP83867_CLK_OUT_CHAN_A_RX_CLK>;
    };
  };
};
```

Data Modul Yocto image uses *systemd-networkd* utility for network configuration. Systemd-networkd is a system service that detects and manages network devices. It performs based on the configuration file, which is located under */etc/systemd/network*.

The configuration file should be named with *.network* at the end. In the Yocto image, the configuration file is placed in */etc/systemd/network/default.network*

```
$ vi /etc/systemd/network/default.network
```

The network can be set in dynamic or static IP connection.

- Dynamic IP connection

```
$ cat /etc/systemd/network/default.network

[Match]
Name=e*

[Network]
#DHCP setup
DHCP=v4
```

- Static IP connection

```
$ cat /etc/systemd/network/default.network
```

```
[Match]
Name=e*

[Network]
#Static setup
Address=10.0.0.2/24
Gateway=10.0.0.1
DNS=8.8.8.8
```

One of these two connections can be applied. By default, dynamic IP connection is enabled and the other one is stored as comments.

In order to test ethernet, *ping* command is a simple and quick try.

```
$ ping 8.8.8.8

PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=56 time=8.96 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=56 time=8.63 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=56 time=8.66 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=56 time=9.00 ms
```

21 Booting Kernel from Network

For developing purpose, one can load kernel and rootfs over NFS server on PPC board. It will significantly reduce time and no need to re-program the board every time. For this matter, some configurations should be done on *host* and *client*. Host means a development Linux PC and client means PPC i.MX6 board.

21.1 Host Network configuration

For the network connection between the host and client, a dedicated network should be used. Therefore a second network interface or USB ethernet adapter is needed. Alternatively the default network of the host system can be used, but be aware, that for the correct setup in this network the network administrator should be consulted.

It is required to have *DHCP server* because the client needs IP address. It depends on the network environment. One way to have DHCP server is to install *dnsmasq* and configure it to act as DHCP server.

```
$ sudo aptitude install dnsmasq
```

Besides *DHCP server*, it is needed to have *NFS server* to be able to load kernel and rootfs. For this part, it is necessary to install the following packages:

```
$ sudo aptitude install nfs-kernel-server nfs-common
```

Then add following line to the */etc/exports*:

```
PATH_TO_DIR_STORES_NFS_FILES *(rw,sync,no_root_squash)
```

PATH_TO_DIR_STORES_NFS_FILES is the path that stores kernel and rootfs, for example */development/export*.

Finally run the below command to make the folder accessible via network:

```
$ sudo exportfs -a
```

21.2 Client Network configuration

21.2.1 User space

The next printout gives the default settings for the Ethernet interface.

```
oot@imx6dl-dmo-ppc:~# ip addr show dev eth0
4: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
        group default qlen 1000
    link/ether 70:f1:76:00:01:69 brd ff:ff:ff:ff:ff:ff
    inet 10.1.1.92/24 brd 10.1.1.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::72f1:76ff:fe00:169/64 scope link
       valid_lft forever preferred_lft forever
```

These settings can be changed by using the ip-tool. For more information related to the persistent network configuration, check IP Linux man page.

For persistent configuration the /etc/network/interfaces should be edited. For dns-resolving, a name server entry should be provided in /etc/resolv.conf. Be aware that this file is a link to a RAM-Filesystem. To make the change permanent, remove this link and add a new file.

```
$ cat /etc/resolv.conf

nameserver 10.1.1.1
```

21.2.2 Bootloader

Barebox provides environment files for network configuration. There are two files that usually have to be changed */env/network/eth0* and */env/boot/nfs*. The first one contains the ethernet adapter settings i.e. dhcp or ip address, gateway, etc. The second one can be used to specify kernel, device tree and rootfs locations for network boot. The below example configuration can be used to load kernel, device tree and rootfs over NFS.

```
#!/bin/sh

path="/mnt/nfs"

global.bootm.image="${path}/zImage"

oftree="${path}/oftree"
if [ -f "${oftree}" ]; then
        global.bootm.oftree="$oftree"
fi

nfsroot="/development/export/target"
bootargs-ip

initramfs="${path}/${global.user}-initramfs-${global.hostname}"
if [ -f "${initramfs}" ]; then
        global.bootm.initrd="$initramfs"
else
        global.linux.bootargs.dyn.root="root=/dev/nfs␣nfsroot=$nfsroot,v3,tcp"
fi
```

global.bootm.image is the name of the Linux kernel image on the NFS server in this case “${path}/zImage”.

oftree is the name of the device tree on the NFS server in this case “${path}/oftree”

nfsroot is the directory name of the exported NFS rootfs.

Note: The ${path} variable is mandatory. In this way Barebox knows that it should use NFS to load the images. The directory "/mnt/NFS" instructs Barebox to start NFS transaction. Thus the above configuration will mount zImage, ofree and rootfs from the NFS server.

The NFS server IP address can be specified in */env/network/eth0*. If DHCP is used, specifying server IP is not needed. Below is an example which shows static IP configuration file.

```
#!/bin/sh

# ip setting (static/dhcp)
ip=static
global.dhcp.vendor_id=barebox-${global.hostname}

# static setup used if ip=static
ipaddr=10.2.1.6
netmask=255.255.255.0
gateway=10.2.1.1
serverip=10.2.1.1

# MAC address if needed
#ethaddr=xx:xx:xx:xx:xx:xx

# put code to discover eth0 (i.e. 'usb') to /env/network/eth0-discover

exit 0
```

One can change the above files by entering the Barebox shell and using the *edit* command.

```
$ edit /env/network/eth0
```

To exit, use Ctrl+d (save) and Ctrl+c (abort). After doing the desired changes and saving them (Ctrl+d), Barebox environment should be saved by using the *saveenv* command.

To start booting via network, one should use *boot nfs* command.
Note: Barebox booloader does not support mini-pcie ethernet cards.

22 Display output, resolution and timings

Two display outputs are supported in PPC i.MX6:

- LVDS single/dual channel
- HDMI

Monitors which is connected through HDMI port, gives supported modes via EDID file (Extended display identification data) to the operating system. However, embedded display that is connected via LVDS connector, does not provide any information for the operating system and all required data such as resolution will be given to the Linux inside device tree as a fixed mode.

Supported resolutions are listed in the following path.

```
$ cat /sys/class/drm/card1-HDMI-A-1/modes
$ cat /sys/class/drm/card1-LVDS-1/modes
```

In LVDS type, there is only one mode, therefore there is no possibility to change it. Whereas in HDMI type, EDID file provides list of supported modes. In this case, different display modes would be available. A way to configure the mode is to use *fbdev interface*. It is performed through kernel argument.

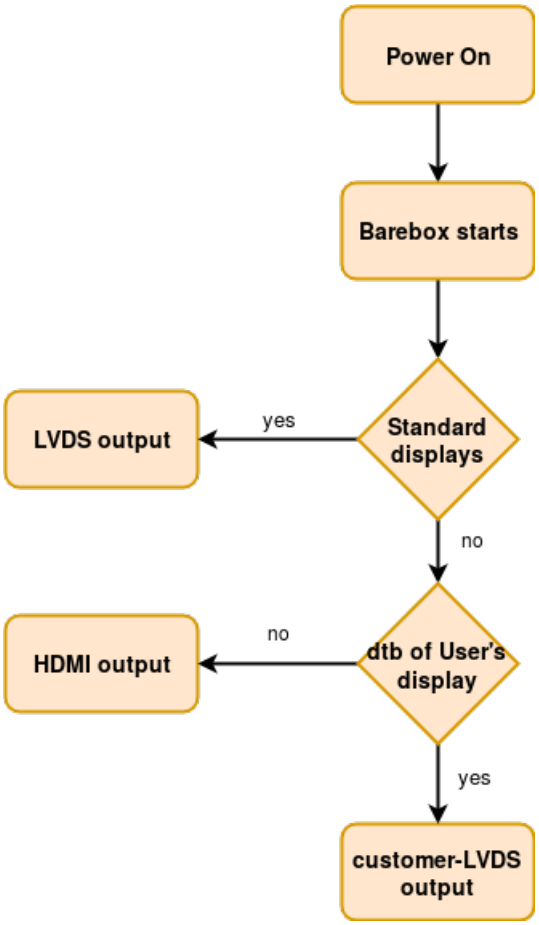
More information regarding display output control and how use internal modedb is available at Linux modedb documentation.

In case of using X11 PPC i.MX6 image, *classic X modes* is an available interface, which provides mode configuration via *xorg.conf* and X driver.

PPC i.MX6 supports one output at the time with the framebuffer /dev/fb0.

As a feature, a **Display-Detection** mechanism in Barebox is implemented. PPC i.MX6 supports three standard DATA-Modul Touch-LVDS-Displays sizes: 7", 10" and 12". If user connects one of these displays, automatic display-detection system recognizes the proper display timing and user does not need to configure it manually. When automatic display-detection cannot find neither **standard display** nor **customer-devicetree**, the output will be HDMI.

In a case, user needs to apply another LVDS display, he/she should set the proper display timing via Barebox-SWU 31.1.4. The below diagram shows the process.



Note: If user updates device-tree or display-timing, then HDMI output will not be available anymore. To use HDMI output again, user needs to update device-tree with standard HDMI-device-tree, which is located in the build directory.

22.1 LVDS

The LVDS embedded display has two encoder channels, which are presented inside the device tree with two LVDS-channel nodes. Both channels are disabled at this point in the general device tree file ***imx6qdl-dmo-ppc.dtsi***

```
&ldb {
  status = "okay";

  lvds-channel@0 {
    status = "disabled";
    fsl,data-mapping = "spwg";
    fsl,data-width = <24>;
  };

  lvds-channel@1 {
    status = "disabled";
    fsl,data-mapping = "spwg";
    fsl,data-width = <24>;
  };
};
```

For each cpu- and display-type there is a separate device tree file like:
imx6{cpu-type}-dmo-ppc-{vendor}-{type}.dts, where cpu-type is ***dl*** for iMX6S and iMX6DL or ***q*** for iMX6D and iMX6Q, **vendor** is the name and **type** is the type of the display, e.g. ***imx6dl-dmo-ppc-chimei-g070y2.dts***

```
/*
 * Copyright 2017 Data Modul AG, Reyhaneh Yazdani <reyhane.y84@gmail.com>
 *
 * The code contained herein is licensed under the GNU General Public
 * License. You may obtain a copy of the GNU General Public License
 * Version 2 or later at the following locations:
 *
 * http://www.opensource.org/licenses/gpl-license.html
 * http://www.gnu.org/copyleft/gpl.html
 */

#include "imx6dl-dmo-ppc.dts"

&ldb {
  lvds-channel@1 {
    status = "okay";
    displaytimings: display-timings {};
  };
};

#include "displaytiming-chimei-g070y2.dts"
```

For each display, a specific timing node is defined, which is done in a separate file like: ***displaytiming-{vendor}-{type}.dts***, e.g. ***displaytiming-chimei-g070y2.dts***. This file will be included in the respective dts file as shown above.

```
&displaytimings {
  native-mode = <&timing0>;
  timing0: Chimei-g070y2-L01-800x480 {
    clock-frequency = <28285712>;
```

```
    hactive = <800>;
    vactive = <480>;
    hfront-porch = <10>;
    hback-porch = <10>;
    hsync-len = <130>;
    hsync-active = <1>;
    vfront-porch = <5>;
    vback-porch = <5>;
    vsync-len = <10>;
    vsync-active = <1>;
  };
};
```

If an embedded display with single-channel is connected, LVDS-channel ***1*** node should be configured as shown in the dts example above.

Dual-channel displays are used for high quality display. In this case, *fsl-dual-channel* property should exist in LVDS node and only LVDS channel ***0*** be configured, e.g. ***imx6dl-dmo-ppc-auo-g173hw01.dts***

```
/*
 * Copyright 2017 Data Modul AG, Reyhaneh Yazdani <reyhane.y84@gmail.com>
 *
 * The code contained herein is licensed under the GNU General Public
 * License. You may obtain a copy of the GNU General Public License
 * Version 2 or later at the following locations:
 *
 * http://www.opensource.org/licenses/gpl-license.html
 * http://www.gnu.org/copyleft/gpl.html
 */

#include "imx6q-dmo-ppc.dts"

&ldb {
  fsl,dual-channel;

  lvds-channel@0 {
    status = "okay";
    displaytimings: display-timings {};
  };
};

#include "displaytiming-auo-g173hw01.dts"

&backlight1 {
  pwms = <&pwm1 0 200000>;
  brightness-levels = <0 1 10 20 30 40 50 60 70 75 80 90 100>;
  default-brightness-level = <7>;
  status = "okay";
};
```

When display settings are altered according to the different embedded displays, the device tree has to be modified. It can be done via SWU update. The procedure how to update device tree and apply SWU are described in subsection 31.1 in this document.

For adding a new LVDS-display in source code, please refer to section 34.

The default LVDS display is in landscape mode. Display rotation requires a lot of memory bandwidth. Thus it would recommend to not rotate the framebuffer. To have portrait mode, the rotation should be done by an application. The application will render on top. Qt5 and Qml2 will do this.

22.2 HDMI

PPC i.MX6 reads EDID information of the regarding connected HDMI-monitor. If no display mode is specified in kernel argument, then Linux chooses a default.

If no standard Data-Modul LVDS-displays is connected and no customer-device-tree is programmed on the device, then HDMI display will be the output. If user updates device-tree or display-timing, then HDMI output will not be available anymore. To use HDMI output again, user needs to update device-tree with standard HDMI-device-tree, which is located in the build directory.

In addition, in kernel config the CONFIG_I2C_IMX_HDMI=y should be set.

23 Touch

Data Moduls easyMaxTouch "driverless" controller allows the user to change the configuration of the easyMax-Touch USB controller using a text file.

These are the steps to modify touch controller settings:

1. Find out the corresponding device of the easyMaxTouch USB controller in OS.

   ```
   $ ls -l /dev/disk/by-id/

   usb-DM_easyTOUCH-0:0 -> ../../sda
   ```

   Consider that touch USB controller can be detected in different letter after sd, it depends on the other block devices connected to the system.

2. Mount the device

   ```
   $ mount /dev/sda /mnt
   ```

3. Modify touch settings

   Configuration settings can be changed by overwriting data into the text file.

   ```
   $ vi /mnt/maxtouch.txt

   Firmware version 3.20 -Multi Touch ^M
   Controller mXT640T V1.5 ^M
   FW_Rev 12017029_001 ^M
   Self test disabled ^M
                                                    ^M
   Sensitivity threshold = 45 ^M
   Switch X and Y = 0 ^M
   Invert X coordinates = 1 ^M
   Invert Y coordinates = 0 ^M
   Initial movement hysteresis = 4 ^M
   Next movement hysteresis = 2 ^M
   Touch detection integration = 3 ^M
   Number of reported touches = 10 ^M
   Touch automatic calibration = 100 ^M
   ```

```
Clipping X low = 0 ^M
Clipping X high = 0 ^M
Clipping Y low = 0 ^M
Clipping Y high = 0 ^M
Enable right click = ---^M
```

Note that while working with the configuration file on the mass storage drive, there is no connection between the data that is written into the file and the data that can be read from the file, because the operating system may try to buffer the data. After buffering the data, the operating system overwrites the data written to the file as current content. therefore, changes could be applied by short delay.

If the maxtouch.txt file does not show the settings that where just entered, it may be necessary to reset the controller using the reset command or restart the OS.

4. Unmount the device

```
$ umount /mnt
```

24 Audio

The PPC i.MX6 BSP supports the sgtl500 audio codec. It contains Advanced Linux Sound Architecture(ALSA), which provides APIs and command-line utilities to use audio. PPC i.MX6 supports audio stereo output and MIC_IN.

The installed ALSA-utilities are *alsamixer*, *aplay* and *arecord*. Moreover, *Gstreamer* can be used to play *ogg* files.

In the following some useful commands are presented, which make using audio simple.

- Check if sound card is present

```
$ cat /proc/asound/cards

0 [imx6qdldmoppcsg]: imx6qdl-dmo-ppc -imx6qdl-dmo-ppc-sgtl5000
                     imx6qdl-dmo-ppc-sgtl5000
```

- Enable audio amplifier

```
$ echo 7 > /sys/class/gpio/export
$ echo high > /sys/class/gpio/gpio7/direction
```

- List device name

```
$ aplay -L
```

- Play *wav* file. In case of more than one device, with *-D* option, the desired device can be selected.

```
$ aplay test.wav

$ aplay -D sysdefault:CARD=imx6qdldmoppcsg file.wav
```

- You can also play *ogg* and *wav* files with Gstreamer

```
$ gst-play-1.0 file.ogg
$ gst-play-1.0 file.wav
```

- Record audio from microphone with *arecord* utility. It will record sound from the default sound device.

```
$ arecord -d 10 file.wav
```

- Alsamixer is a graphical program for ALSA, which is used for configuration.

Note: HDMI Audio is not enable by default. This feature would be enabled through Linux configuration.

25 Bluetooth

The PPC i.MX6 provides Bluetooth functionality in different ways.

- Onboard Wi-Fi-M.2-module soldered on the PCB
- The Mini-PCIe socket mounted with Wi-Fi/Bluetooth Mini-PCIe-Module(check section 14)
- ***USB Bluetooth sticks, which are not tested so far.***

By default, Bluetooth module is not enabled in the PPC i.MX6 Linux BSP. However, various Mini-PCIe Wi-Fi/Bluetooth modules based on Intel chip sets (e.g.4965AGN, 3160) are successfully tested.

To get Bluetooth working on the PPC i.MX6, the module needs to be configured, respective kernel driver and bluetooth profiles are required to be installed.

- The dhcp-client is required to be installed (The default BSP covers it).
- Enable Wireless networking support, including wireless extensions compatibility in Linux kernel.

```
1    CONFIG_CFG80211=y
2    CONFIG_CFG80211_WEXT=y
3    CONFIG_MAC80211=y
```

- Enable the respective driver for the used wireless module, e.g. IntelÂő Dual Band Wireless-AC 3160, which is handled by the Intel Wireless WiFi Next Gen AGN - Wireless-N/Advanced-N/Ultimate-N (iwlwifi) driver.

```
1    CONFIG_IWLWIFI=y
2    CONFIG_IWLWIFI_LEDS=y
3    CONFIG_IWLDVM=y
4    CONFIG_IWLMVM=y
```

- And for Bluetooth, additionally the HCI drivers and respepective protocols (here Intel protocol support) need to be enabled in the Linux kernel:

```
1    CONFIG_BT_HCIBTUSB=y
2    CONFIG_BT_HCIUART=y
3    CONFIG_BT_HCIUART_INTEL=y
```

After booting the kernel with above configuration, the module should show up as described in section 14

```
$ lspci -v

    ...

01:00.0 Network controller: Intel Corporation Wireless 3160 (rev cb)
        Subsystem: Intel Corporation Dual Band Wireless-AC 3160
        Flags: bus master, fast devsel, latency 0, IRQ 304
```

```
        Memory at 01100000 (64-bit, non-prefetchable) [size=8K]
        Capabilities: [c8] Power Management version 3
        Capabilities: [d0] MSI: Enable- Count=1/1 Maskable- 64bit+
        Capabilities: [40] Express Endpoint, MSI 00
        Capabilities: [100] Advanced Error Reporting
        Capabilities: [140] Device Serial Number e4-02-9b-ff-ff-c6-42-fb
        Capabilities: [14c] Latency Tolerance Reporting
        Capabilities: [154] Vendor Specific Information: ID=cafe Rev=1 Len=014 <?>
        Kernel driver in use: iwlwifi
```

Thereafter, Bluetooth should be configured by using *bluetoothctl*. For further information please refer to the Bluetooth article.

26 Wi-Fi

The PPC i.MX6 provides Wi-Fi functionality in different ways.

- Onboard Wi-Fi-M.2-module soldered on the PCB
- The Mini-PCIe socket mounted with Wi-Fi Mini-PCIe-Module(check section 14)
- ***USB Wi-Fi sticks, which are not tested so far.***

By default, Mini-PCIe Wi-Fi module is not enabled in the PPC i.MX6 Linux BSP. However, various Mini-PCIe Wi-Fi modules based on Intel chip sets (e.g. 4965AGN, 3160) are successfully tested.

To get Wi-Fi working on the PPC i.MX6, some pre-conditions are needed to be fulfilled:

- The dhcp-client is required to be installed (The default BSP covers it).
- Enable Wireless networking support, including wireless extensions compatibility in Linux kernel.

```
1  CONFIG_CFG80211=y
2  CONFIG_CFG80211_WEXT=y
3  CONFIG_MAC80211=y
```

- Enable the respective driver for the used wireless module, e.g. IntelÂő Dual Band Wireless-AC 3160, which is handled by the Intel Wireless WiFi Next Gen AGN - Wireless-N/Advanced-N/Ultimate-N (iwlwifi) driver.

```
1  CONFIG_IWLWIFI=y
2  CONFIG_IWLWIFI_LEDS=y
3  CONFIG_IWLDVM=y
4  CONFIG_IWLMVM=y
```

After booting the kernel with above configuration, the module should show up as described in section 14

```
$ lspci -v

    ...

01:00.0 Network controller: Intel Corporation Wireless 3160 (rev cb)
        Subsystem: Intel Corporation Dual Band Wireless-AC 3160
        Flags: bus master, fast devsel, latency 0, IRQ 304
        Memory at 01100000 (64-bit, non-prefetchable) [size=8K]
        Capabilities: [c8] Power Management version 3
```

```
        Capabilities: [d0] MSI: Enable- Count=1/1 Maskable- 64bit+
        Capabilities: [40] Express Endpoint, MSI 00
        Capabilities: [100] Advanced Error Reporting
        Capabilities: [140] Device Serial Number e4-02-9b-ff-ff-c6-42-fb
        Capabilities: [14c] Latency Tolerance Reporting
        Capabilities: [154] Vendor Specific Information: ID=cafe Rev=1 Len=014 <?>
        Kernel driver in use: iwlwifi

$ ifconfig -a

    ...

wlp1s0 Link encap:Ethernet HWaddr e4:02:9b:c6:42:fb
          BROADCAST MULTICAST MTU:1500 Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)
```

Then using Linux command line to set up the Wi-Fi.

27 Video playback

The BSP supports hardware video decoding and scaling of H.264 videos. The scaling is limited by factor 4 in both directions and can cause performance issue in playing video with high resolution.

Additionally, the BSP provides a gstreamer script to simplify playing videos.

This script uses Gstreamer tools such as **gst-discoverer-1.0**, **gst-launch-1.0**, . . . to playback video. Moreover, it utilizes Video4Linux API in the pipeline to expose hardware codecs in a standard way by applying **v4l2** plugin in **gst-plugins-good**.

- Full screen playback with audio.

  This script plays full screen video without scaling, therefore the video and display resolutions should match

  ```
  $ dmo-video-play.sh -t fs -o 1 -a file.mov
  ```

- Hardware overlay window

  ```
  $ dmo-video-play.sh -t ovl -o 1 -w 50,50,540,380 -a file.mov
  ```

  Calling the scripts without any arguments shows all possible options.

It is possible that the decoder is not detected by the gst-launch, as the moment gst-launch is loading, the related hardware driver is not available and the related registry file does not contain the decoder. For this matter, at first run the following command to be sure that the decoder exists:

```
$ gst-inspect-1.0 |grep v4l2
```

The output:

```
video4linux2: v4l2src: Video (video4linux2) Source
video4linux2: v4l2sink: Video (video4linux2) Sink
video4linux2: v4l2radio: Radio (video4linux2) Tuner
video4linux2: v4l2deviceprovider (GstDeviceProviderFactory)
```

```
video4linux2: v4l2video7dec: V4L2 Video Decoder
video4linux2: v4l2video0convert: V4L2 Video Converter
video4linux2: v4l2video1convert: V4L2 Video Converteri /*in case of dual-quad cpu*/
```

Then, if the output is not as above, removing the gstreamer registry in *.cache/gstreamer-1.0/registry.arm.bin* and run the *gst-inspect* command again to create the new registry file.

```
$ gst-inspect-1.0 |grep v4l2
```

Note: The firmware for play back depends on the type of hardware. If the dual-lite hardware is used, the related rootfs for this hardware should be used. In other words, if the unrelated rootfs is applied, the coda firmware which is needed for the playback would not be installed. It can be detected by:

```
$dmesg |grep -i coda
```

Output:

```
[ 1.550031] coda 2040000.vpu: Direct firmware load for v4l-coda960-imx6dl.bin
failed with error -2
[ 1.559080] coda 2040000.vpu: Falling back to user helper
[ 12.022112] coda 2040000.vpu: Firmware code revision: 46072
[ 12.027740] coda 2040000.vpu: Initialized CODA960.
[ 12.032640] coda 2040000.vpu: Unsupported firmware version: 3.1.1
[ 12.122102] coda 2040000.vpu: codec registered as /dev/video[3-4]
```

28 Framebuffer

The frame buffer device provides an abstraction for the graphics hardware. It represents the frame buffer of some video hardware and allows application software to access the graphics hardware through a well-defined interface, so software doesn't need to know anything about the low-level (hardware register) stuff(Taken from Linux framebuffer documentation).

Framebuffers are mapped into device nodes: /dev/fb0, /dev/fb1, . . .

The DRM framework provides a legacy /dev/fb0 framebuffer, which supports i.MX6 graphic device. This driver has access to connected display through /dev/fb0.

- Check the framebuffer features:

```
$ fb-test

fb-test 1.1.0 (rosetta)
fb res 800x480 virtual 800x480, line_len 3840, bpp 16
```

- Get color depth of the framebuffer:

```
$ cat /sys/class/graphics/fb0/bits_per_pixel

16
```

- Get the list of video interface:

```
$ ls -l /sys/class/drm

card0 -> ../../devices/platform/Vivante GCCore/drm/card0
card1 -> ../../devices/soc0/display-subsystem/drm/card1
card1-HDMI-A-1 -> ../../devices/soc0/display-subsystem/drm/card1/card1-HDMI-A-1
card1-LVDS-1 -> ../../devices/soc0/display-subsystem/drm/card1/card1-LVDS-1
controlD64 -> ../../devices/soc0/display-subsystem/drm/controlD64
```

28.1 Virtual Framebuffer and VSYNC

Framebuffer panning provides a possibility to have a resolution that is higher than what a display supports but only show it in a resolution it does support. For example if there is a 1920x1080 display with pan enable, it would be possible to have a 1920x1080 area of the overall display and could pan over the entire 2560x1600 picture as moving the mouse.

The PPC i.MX6 Linux BSP supports framebuffer panning by creating multi buffer for **cma fbdev**, such as double buffer and triple buffer.

The Linux kernel configuration item **CONFIG_DRM_FBDEV_OVERALLOC** sets the panning parameter and defines the fbdev buffer overallocation in percent. Default is 100 and it corresponds to single buffer. Typical values for double buffering will be 200, triple buffering 300. In order to enable multi buffering, the kernel should be compiled with appropriate fbdev buffer allocation.

From the user applications, after opening */dev/fb0* and get handle, user can use *ioctl* for frame buffer in the kernel via */dev/fb0*. Checking the size of the frame buffer by ioctl and mapping it to the user space. After writing data to the buffer in the user space, the data will be displayed at the next vsync timing. User can get this timing by blocking ioctl **FBIO_WAITFORVSYNC** to synchronize drawing or buffer flip for double buffering. After returning the ioctl, user can reuse the buffer.

Please test this feature to be sure it is matched with your desired requirements.

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <linux/fb.h>
#include <sys/mman.h>

#define COLOR1 12
#define COLOR2 30
#define COLOR3 45
#define COLOR4 70

int main(int argc, char* argv[])
{
  int fbfd = 0;
  struct fb_var_screeninfo vinfo_orig;
  struct fb_var_screeninfo vinfo;

  // Open the file for reading and writing
  fbfd = open("/dev/fb0", O_RDWR);
  if (!fbfd) {
    printf("Error: cannot open framebuffer device.\n");
    return(1);
  }
  printf("The framebuffer device was opened successfully.\n");

  // Get variable screen information
  if (ioctl(fbfd, FBIOGET_VSCREENINFO, &vinfo)) {
    printf("Error reading variable information.\n");
```

```
}

// Store for reset
memcpy(&vinfo_orig, &vinfo, sizeof(struct fb_var_screeninfo));

// Set variable info
vinfo.xres = 320;
vinfo.yres = 240;
vinfo.xres_virtual = 320;
vinfo.yres_virtual = 480; // double the physical height
vinfo.bits_per_pixel = 8;
if (ioctl(fbfd, FBIOPUT_VSCREENINFO, &vinfo)) {
  printf("Error setting variable information.\n");
}

long int screensize = vinfo.xres * vinfo.yres;
long int bufsize = vinfo.xres_virtual * vinfo.yres_virtual;
char *fbp = 0;
int vs = 0;
int y = 0;

// Map fb to user mem
fbp = (char*)mmap(0, bufsize, PROT_READ | PROT_WRITE, MAP_SHARED, fbfd, 0);

if ((int)fbp == -1) {
  printf("Failed to mmap.\n");
}
else {
  // draw a COLOR1 rect at the upper half of screen
  memset(fbp, COLOR1, screensize / 2);
  // draw a COLOR2 rect at the lower half of screen
  memset(fbp + screensize / 2, COLOR2, screensize / 2);
  // draw a COLOR3 rect at the upper half of 2nd screen
  memset(fbp + screensize, COLOR3, screensize / 2);
  // draw a COLOR4 rect at the lower half of 2nd screen
  memset(fbp + screensize + screensize / 2, COLOR4, screensize / 2);

  sleep(2);

  // change to show lower half
  if (ioctl(fbfd, FBIO_WAITFORVSYNC, 0)) {
    printf("VSync failed.\n");
  }
  vinfo.yoffset = vinfo.yres;
  if (ioctl(fbfd, FBIOPAN_DISPLAY, &vinfo)) {
    printf("Pan failed.\n");
  }

  // wait
  sleep(2);
}

// Cleanup
munmap(fbp, screensize);
```

```
86   if (ioctl(fbfd, FBIOPUT_VSCREENINFO, &vinfo_orig)) {
87     printf("Error re-setting variable information.\n");
88   }
89   close(fbfd);
90
91   return 0;
92 }
```

29 GUI framework: Qt

One of the possible distro for PPC i.MX6 is *QT5 variant*.

Qt5.x.x has removed its own windowing system. That means it is required to use EGLFS for framebuffer mode or X11 as a defined windowing system.

EGLFS is a platform plugin for running Qt5 applications on top of EGL and OpenGL ES 2.0 without an actual windowing system like X11.

It may be required before starting a QT5 application, the *EGLFS* is selected as the platform plugin. This can be done by setting an application argument (-platform eglfs) or environment parameter:

```
$ export QT_QPA_PLATFORM=eglfs
```

For screen rotation, there is not possible to run a Qt application in a *rotated framebuffer*. To have portrait mode, the rotation should be done by an application. The application will render on top. Qt5 and Qml2 will do this.

Moreover, currently X11 image does not support screen rotation.

Besides OpenGL and Qt Quick2 applications, which expliot HW-acceleration(GPU), EGLFS supports also SW-rendering(CPU), as an example for QWidget applications.

30 X-Server

The Xorg Server is the core of the X Window system. It is a server that is responsible for creating a graphical environment and managing graphical display.

X-server can be started with *startx* command.

```
$ startx
```

The **xinit** program gives the possibility to manually run an Xorg display sever. The *startx* is a front-end script for *xinit*. The *~/.xinitrc* is a shell script used by xinit as an argument, that starts the X-server to run some applications automatically in the X-server.

The file **xorg.conf** is a configuration file used to give the X-server information about the hardware components are used. Now in modern system, the X-server can avoid using it, because of auto configuring. It is located in */etc/X11/xorg.conf*. PPC i.MX6 X11 image contains a default configuration file(xorg.conf). It specifies a driver to load, which is Vivante.

In addition, **xrandr** is an extension that is implemented by X.org Server. It provides required support regarding resolution modification and rotation screen. The point should be considered is that rotation is implemented completely in software by the X-Server and is not accelerated by GPU.

Based on a technical issue, PPC i.MX6 X11 image does not support XRandR. Thus it is disabled and not possible to change resolution and rotate screen through it.

Above Xorg Xserver placed **Xfce** as a Linux Desktop environment. It allows starting other programs. Xfce contains a collection of programs, such as Window Manager, File Manager, Desktop Manager and Panel.

31 Bootloader: Barebox

Barebox is an open source, primary bootloader used in embedded devices. Currently, in PPC i.MX6 module, Barebox bootloader is used.

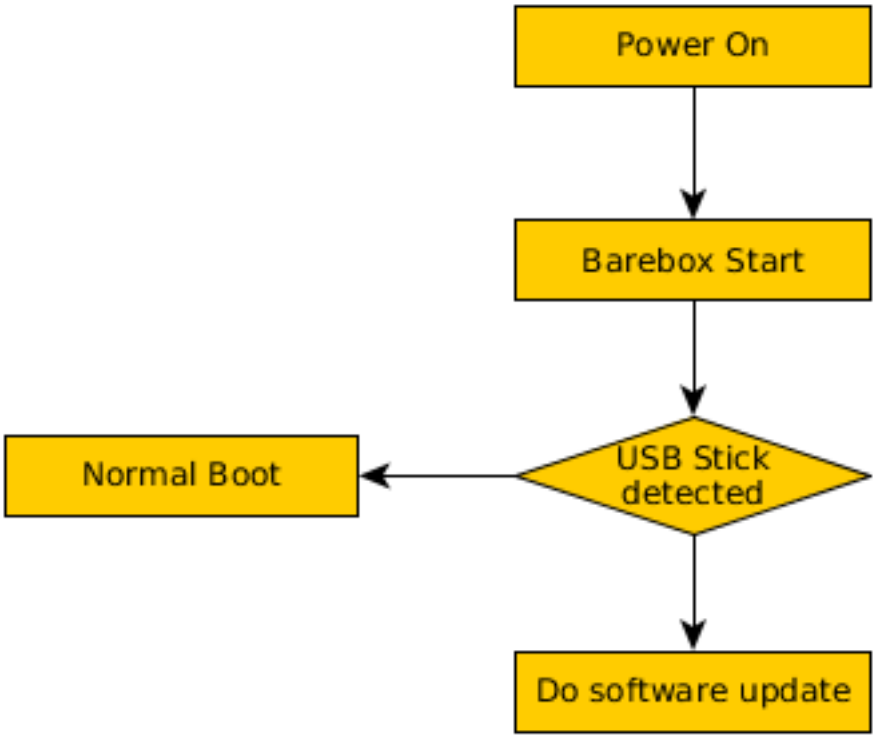
The first 512KB of the SPI flash storage is partitioned for Barebox and the next 128KB is assigned to Barebox-environment.

31.1 Barebox SWU

Barebox provides the facility to update boot partitions over USB-stick. It contains an *inifile*, which specifies the images and target devices to be updated.

31.1.1 Update process

The below diagram shows how Barebox Software Update(SWU) performs the update process.



31.1.2 Config file

It is a simple ini file that is needed to be created in Linux text format. It defines the images and the target devices to be update. If the file is created on different OS it should be converted to Linux text format.

Lines beginning with "#" are treated as comments.

The first line should specify the version information. The applicable version is #$< ver = 0.2 >$.

There is only one parameter syntax supported at the moment: *VAR*=$< VALUE >$. No space before and after equals sign is allowed. At the end of file, an empty line is required to use as **carriage return**. The config file should be named *inifile* and located in the root of the USB-stick along with other image files.

There is also special type of "inifile" for LVDS parameters update. It is described in the subsection 31.1.5.

The image parameters of the *inifile* are:

```
#<ver=0.2>

# enables writing the "update.log" to the USB media
LOGGING=1

# defines the device where the operating system i.e. kernel, dts, rootfs,
# homefs etc. is stored.
# When used without an image file provided on USB stick only the boot source will
# be changed inside selected BB_TARGET_DEV (BB Environment -> Current Boot).
 OS_TARGET_DEV= mmc2 | mmc3 | ata0
# mmc2 implies SD-Card
# mmc3 implies emmc
# ata0 implies Sata

# defines the device where the Barebox and Barebox-environment is stored
 BB_TARGET_DEV= m25p0 | mmc2
# m25p0 implies Spiflash
# mmc2 implies SD-Card

# specifies the file on the USB stick that is the Barebox binary image
BAREBOX_IMAGE=barebox-imx6q-dmo-edm-qmx6.bin

# specifies the file on the USB stick that is the Barebox-environment image or
# DEFAULT to get back to the default Barebox-environment
 BAREBOX_ENV=DEFAULT

# specifies the file on the USB stick that is the kernel binary image
 KERNEL_IMAGE=zImage-dmo

# specifies the file on the USB stick that is the unflattened binary device tree
 DTS_IMAGE=imx6q-dmo-edmqmx6-auo-g173hw01.dtb

# specifies the file on the USB stick that is rootfs image
 ROOTFS_IMAGE=dmo-image-release-imx6q-dmo-edm-qmx6-20140709074749.rootfs.ext3

# specifies the file on the USB stick that is full os image consisting of kernel
# partition, rootfs and homefs
FULL_IMAGE=dmo-image-release-imx6q-dmo-edm-qmx6-20140709074749.with-homefs.sdcard2
```

Note: It is not advisable to update all of the image parameters at once. For example it is no problem to update Barebox on flash and full image on emmc but it does not make sense to update rootfs and full os image. The full image contains already the rootfs. To prevent undefined results, in case of having both FULL_IMAGE

and other images (except Barebox image), only FULL_IMAGE update will be executed.

Note: The software update works only with one inserted USB-stick.

Note: The USB-stick has to be formatted in vfat/fat32 filesystem.

31.1.3 Signature Check

The image integrity check can be applied by creating checksum file. The checksum file should be named as the image and ending with suffix *.md5sum*. Example shows how to create checksum file.

```
$ md5sum image_kernel.img > image_kernel.img.md5sum
```

Note: The PPC i.MX6 Yocto build system generates automatically the md5sum file for each image.

31.1.4 LVDS Parameter Update

In addition to the above items, Barebox SWU provides the capability to update **LVDS settings**, without programming new device tree file.

For adding a new lvds-display in source code, please refer to section 34.

The table 10 describes the config file parameters necessary to update LVDS settings. It also explains the correspondence between the config file and device tree parameters. The possible values for each parameter is in the third column. The OS_TARGET_DEV is still needed because it is required to manipulate the device tree for the new LVDS settings.

Table 10: LVDS display timing

Config file	Device tree	Allowed Values
TFT_LVDS_PANEL_MODIFY_PARAMETER	-	Y/N
TFT_LVDS_PANEL_OUTPUT	-	okay/disabled
TFT_LVDS_PANEL_BITCONFIG_COLOURMAPPING	fsl,data-mapping	spwg/jeida
TFT_LVDS_PANEL_BITCONFIG_SINGLE_DUAL_LINK	fsl,dual-channel	0/1
TFT_LVDS_PANEL_BITCONFIG_18_24	fsl,data-width	18/24
TFT_LVDS_PANEL_TIMING_PIXELCLK_HZ	clock-frequency	integer
TFT_LVDS_PANEL_TIMING_H_ACTIVE_LINES	hactive	integer
TFT_LVDS_PANEL_TIMING_V_ACTIVE_LINES	vactive	integer
TFT_LVDS_PANEL_TIMING_H_FPORCH	hfront-porch	integer
TFT_LVDS_PANEL_TIMING_HSYNC_BPORCH	hback-porch	integer
TFT_LVDS_PANEL_TIMING_HSYNC_WIDTH	hsync-len	integer
TFT_LVDS_PANEL_SIGNAL_HSYNC_POL_ACTIVE	hsync-active	0/1
TFT_LVDS_PANEL_TIMING_V_BLANC	vfront-porch	integer
TFT_LVDS_PANEL_TIMING_VSYNC_OFFSET	vback-porch	integer
TFT_LVDS_PANEL_TIMING_VSYNC_WIDTH	vsync-len	integer
TFT_LVDS_PANEL_SIGNAL_VSYNC_POL_ACTIVE	vsync-active	0/1

Note: If value range is "0 | 1" then "0" is disabled/low and "1" is enabled/high.

31.1.5 Example Config Files

- The below example shows the contents of the inifile to update Barebox on spiflash and Kernel, DTS, rootfs on mmc3.

```
#<ver=0.2>

#LOGGING=1

OS_TARGET_DEV=mmc3
BB_TARGET_DEV=m25p0

BAREBOX_IMAGE=barebox-dmo-ppc.img
BAREBOX_ENV=DEFAULT

KERNEL_IMAGE=zImage
DTS_IMAGE=imx6s-dmo-ppc-chimei-g070y2.dtb
ROOTFS_IMAGE=dmo-image-imx6s-ppc.rootfs.ext4

# FULL_IMAGE=dmo-image-release-with-homefs.sdcard2
```

- Template for a Chimei-800x480 Display

```
#<ver=0.2>

OS_TARGET_DEV=mmc3
BB_TARGET_DEV=m25p0

##########################################
# TFT LVDS PARAMETERS (Chimei-800x480) #
##########################################
TFT_LVDS_PANEL_MODIFY_PARAMETER=Y
TFT_LVDS_PANEL_OUTPUT=okay
TFT_LVDS_PANEL_BITCONFIG_COLOURMAPPING=spwg
TFT_LVDS_PANEL_BITCONFIG_SINGLE_DUAL_LINK=0
TFT_LVDS_PANEL_BITCONFIG_18_24=24
TFT_LVDS_PANEL_TIMING_PIXELCLK_HZ=28285712
TFT_LVDS_PANEL_TIMING_H_ACTIVE_LINES=800
TFT_LVDS_PANEL_TIMING_V_ACTIVE_LINES=480
TFT_LVDS_PANEL_TIMING_H_FPORCH=10
TFT_LVDS_PANEL_TIMING_HSYNC_BPORCH=10
TFT_LVDS_PANEL_TIMING_HSYNC_WIDTH=130
TFT_LVDS_PANEL_SIGNAL_HSYNC_POL_ACTIVE=1
TFT_LVDS_PANEL_TIMING_V_BLANC=5
TFT_LVDS_PANEL_TIMING_VSYNC_OFFSET=5
TFT_LVDS_PANEL_TIMING_VSYNC_WIDTH=10
TFT_LVDS_PANEL_SIGNAL_VSYNC_POL_ACTIVE=1
```

- Template for a AUO-g173hw01-1920x1080

```
#<ver=0.2>
OS_TARGET_DEV=mmc3
BB_TARGET_DEV=mmc2

##################################################
# TFT LVDS PARAMETERS (AUO-g173hw01-1920x1080) #
```

```
################################################
TFT_LVDS_PANEL_MODIFY_PARAMETER=Y
TFT_LVDS_PANEL_OUTPUT=okay
TFT_LVDS_PANEL_BITCONFIG_COLOURMAPPING=spwg
TFT_LVDS_PANEL_BITCONFIG_SINGLE_DUAL_LINK=1
TFT_LVDS_PANEL_BITCONFIG_18_24=24
TFT_LVDS_PANEL_TIMING_PIXELCLK_HZ=127285714
TFT_LVDS_PANEL_TIMING_H_ACTIVE_LINES=1920
TFT_LVDS_PANEL_TIMING_V_ACTIVE_LINES=1080
TFT_LVDS_PANEL_TIMING_H_FPORCH=0
TFT_LVDS_PANEL_TIMING_HSYNC_BPORCH=0
TFT_LVDS_PANEL_TIMING_HSYNC_WIDTH=180
TFT_LVDS_PANEL_SIGNAL_HSYNC_POL_ACTIVE=1
TFT_LVDS_PANEL_TIMING_V_BLANC=2
TFT_LVDS_PANEL_TIMING_VSYNC_OFFSET=0
TFT_LVDS_PANEL_TIMING_VSYNC_WIDTH=50
TFT_LVDS_PANEL_SIGNAL_VSYNC_POL_ACTIVE=1
```

31.1.6 Software Update Status

The final status of the update procedure(fail/successful) and detail information about the update are shown on the serial console. The *Update status: 0* means that the software update process is successful.

Below is an example output during software update of Barebox and its environment on serial console. Please note that it may look different depending on content of inifile.

```
Board: Data Modul AG eDM-SBC-iMX6-PPC-DL/S-1G
detected i.MX6 Solo revision 1.3
m25p80 m25p80@00: s25fl164k (8192 Kbytes)
imx-usb 2184000.usb: USB EHCI 1.00
imx-usb 2184200.usb: USB EHCI 1.00
imx-esdhc 2198000.usdhc: registered as 2198000.usdhc
imx-esdhc 219c000.usdhc: registered as 219c000.usdhc
malloc space: 0x2fefbce0 -> 0x4fdf79bf (size 511 MiB)
barebox-environment environment-sd.13: probe failed: No such file or directory
environment load /dev/env0: No such file or directory
Maybe you have to create the partition.
running /env/bin/init...
set parameter: no such device eth0
usb: USB: scanning bus for devices...
usb: Bus 001 Device 001: ID 0000:0000 EHCI Host Controller
usb: Bus 002 Device 002: ID 0000:0000 EHCI Host Controller
usb: Bus 002 Device 003: ID 0424:2514
usb: Bus 002 Device 004: ID 8564:1000 Mass Storage Device
Using index 0 for the new disk
usb: 4 USB Device(s) found
mounting usb media...
SWU preparation...
Checking config file version.
conf ver = 0.2
reading ini file
Preparing Software update!
<<< SWU START >>>
Software update in progress!
```

```
mmc3: detected MMC card version 4.41
mmc3: registered mmc3
update: bb dev: m25p0 os dev: mmc3
integrity check skipped /mnt/disk/barebox-imx6dl-dmo-ppc.img.
update succeeded
integrity check skipped /mnt/disk/barebox-imx6dl-dmo-ppc.img.
update barebox status: 0
running signature check.
integrity check skipped /mnt/disk/barebox_zero_env.
update block device: S:/mnt/disk/barebox_zero_env -> D:/dev/m25p0.barebox-environment
integrity check skipped /mnt/disk/barebox_zero_env.
update succeeded
update barebox env status: 0
update status: 0
Software update successfully finished!
please remove usb media and reset the board.
```

*Note: If serial console is not available, the update output can be logged in the USB stick by activating logging(adding **LOGGING=1** in **inifile**). The **update.log** file will be stored on the USB stick after update process is finished.*

32 Splash screen

The splash screen is the logo that is shown during boot on the screen. By default, Data Modul's logo is displayed.

32.1 Providing the final image with the customer's logo

For applying customer's logo, a picture correspond to the size of the display is required. For example, for a display with 1280x800 size, a picture with 1280x800 pixel should be created. The format of the picture must be *png* or *bmp*.

Before building the final image according to the section 2, customer's logo must be added to the Yocto source code by the following steps:

- Copy the logo (*.png* or *.bmp*) corresponding to the size of the display to the directory poky/meta-dmo/conf/logos
- Edit the file poky/meta-dmo/conf/machine/splash-screen.inc according to the logo's properties:

```
# splash-screen.inc
# Customer logo will be configured here

# -Copy logo image to folder meta-dmo/conf/logos/
# and set logo name in SPLASH_SCREEN_IMAGE
# -Adapt resolution by setting SPLASH_SCREEN_DIMENSION
# -Set screen type by commenting/uncommenting the respective
# SPLASH_SCREEN_TYPE

# The image source can be set as url path or file.
# If the splash image is defined with file, it needs to be placed in the
# logos folder.
SPLASH_SCREEN_IMAGE ?= "<customer-logo>.png"

# It seems, that the logo size is capped somwhere around WXGA (1280x800) or
```

```
# to be a bit more precise, the byte size.
# The convert tool resizes the image to match either the height or the width
# depending on the source image (portrait/landscapea)
# To force the given image, add an ! to the dimension. e.g. 800x480!
SPLASH_SCREEN_DIMENSION ?= "800x480!"

SPLASH_SCREEN_PATH = "${COREBASE}/meta-dmo/conf/logos/"

# With dietsplash type,Linux penguin symboles are displayed besides the logo
SPLASH_SCREEN_TYPE ?= "dietsplash"
# With kernel type, only logo is displayed
# SPLASH_SCREEN_TYPE ?= "kernel"
```

- Build the new image and update the board; customer's logo will be displayed on the screen during boot.

33 SSH

The OpenSSh suit is a connectivity tool for remote login based on SSH protocol. It is installed on the PPC Yocto image. Therefore, user can use it for encrypted transfer between a host and the module.

```
$ ssh root@ip-add

root@ip-add's password:
Welcome to Ubuntu 16.04.3 LTS (GNU/Linux 4.4.0-98-generic x86_64)

 *Documentation: https://help.ubuntu.com
 *Management: https://landscape.canonical.com
 *Support: https://ubuntu.com/advantage

0 packages can be updated.
0 updates are security updates.

***System restart required ***
Last login: Mon May 15 09:22:11 2017

 *keychain 2.8.1 ~ http://www.funtoo.org
 *Found existing ssh-agent: 3609
 *Known ssh key:

root@DMO:~$
```

34 How to add LVDS-Dislplays to PPC i.MX6 Linux BSP

34.1 Introduction

This chapter describes how to add a new LVDS-Display to Data-Modul PPC i.MX6 Linux Board-Support-Package(BSP).

There are two ways to add a new LVDS-Display:

- Using the Barebox Software Update (SWU) mechanism as described in subsection 31.1 to update the display parameters.
- Adding the display parameters on source code level and rebuild the kernel and device tree.

34.2 Add new LVDS-Display in the kernel source

To add the new Display parameters to the device-tree, some preconditions need to be fulfilled:

- The kernel source tree has to be available (can be downloaded from DMO GitHub repository)
- ARM-toolchain needs to be installed to build new kernel and device-tree(described in subsection 2.9)

Moreover, it is required to follow these steps:

- Add a new display-timing file accordance with the LVDS-Display specification
- Add a new display file for correspond SoC variant(duallight/quad)
- Add the new device-tree file into the Makefile
- Compile kernel source code
- Install the new device-tree file (and kernel if needed) on the target
- Booting the target

34.2.1 Cloning the kernel Source tree

Creating a source and a separate built directory and clone the repositories

```
$ mkdir sources
$ mkdir build-arm
$ cd sources
$ git clone https://github.com/data-modul/linux-imx6
```

34.2.2 Setup build environment

```
$ cd build-arm
$ . /opt/poky/2.2.1/environment-setup-cortexa9hf-neon-poky-linux-gnueabi
```

34.2.3 Add a new display-timing to the device-tree

The device-tree files are located in: ***arch/arm/boot/dts.***

Existing display-timing files can be used as template to create a new one.

The file name should follow this format: ***displaytiming-{vendor}-{type}*** and with ***dts*** file extension at the end, e.g. ***displaytiming-auo-g804sn05.dts***

The content of the file looks like:

```
1  &displaytimings {
2    native-mode = <&timing0>;
3    timing0: AUO-G084SN05-800x600 {
4      clock-frequency = <39800000>;
5      hactive = <800>;
6      vactive = <600>;
7      hfront-porch = <28>;
8      hback-porch = <58>;
9      hsync-len = <170>;
10     hsync-active = <0>;
11     vfront-porch = <4>;
```

```
12      vback-porch = <6>;
13      vsync-len = <18>;
14      vsync-active = <0>;
15    };
16  };
```

Some variable values can be taken directly from the display datasheet and the remaining values are calculated via CVTd6r1.xls.

Inside the excel file, enter the values for HActive (Horizontal Pixel), VActive (Vertical Pixel) and VFreq (Pixel Clock) and verify the respective total values. If needed, set Blanking to reduced (= y in 6th entry row in the Entry Mask, see below). Then take over the result values(listed below) to the display-timing file.

- H FRONT PORCH(hfront-porch)
- H BACK PORCH(hback-porch)
- HOR SYNC(hsync-len)
- V FRONT PORCH(vfront-porch)
- V BACK PORCH(vback-porch)
- VER SYNC(vsync-len)

These values also can be used in the ***inifile*** when using the Barebox SWU to update the LVDS-settings.

1) Enter Desired Horizontal Pixels Here =>		1280	
2) Enter Desired Vertical Pixels Here =>		800	
3) Enter If You Want Margins Here (Y or N) =>		n	
4) Enter If You Want Interlace Here (Y or N) =>		n	
5) Enter Vertical Scan Frame Rate Here =>	NOTE: Actual frame rate will be within +/- 0.6Hz due to pixel clock rounding to 0,25MHz.	60	Hz
6) Enter If You Want Reduced Blanking Here (Y or N) =>		y	
STATUS:	OK		

34.2.4 Add a new display file to the device-tree

In this step, new display file has to be created, correspond to the SoC. Please use the already existing files, e.g. *imx6dl-dmo-ppc-auo-g804sn05.dts* for iMX6DL/S and *imx6q-dmo-ppc-auo-g804sn05.dts* for iMX6Q/D as template.

The name convension should follow this rule:

- iMX6DL/S: imx6dl-dmo-ppc-{vendor}-{type}.dts
- iMX6Q/D: imx6q-dmo-ppc-{vendor}-{type}.dts

The content of these files should look like the following. The file is defined for iMX6DL/S with auo-g804sn05 LVDS-Display:

```
#include "imx6dl-dmo-ppc.dts" /* for iMX6DL/S */
#include "imx6q-dmo-ppc.dts" /* for iMX6Q/D */

&ldb {
  lvds-channel@1 { /* in case of dual-channel LVDS-Display, use lvds-channel@0 */
    status = "okay";
    displaytimings: display-timings {};
  };
};

#include "displaytiming-auo-g804sn05.dts" /* the new display-timing file*/
```

Note: LVDS-Display has two channels. For displays that use only one channel, channel 1 should be enabled. For display with dual-channel, only channel 0 should be enabled.

34.2.5 Add new device-tree file into the Makefile

The new device-tree file needs to be added into the Makefile to be built in the next step. Open "arch/arm/-boot/dts/Makefile" in an editor and add the respective file in the section **dtb-$(CONFIG_SOC_IMX6Q) +=** \ by following the existing structure.

Note: As there is no separate section for iMX6DL/S, the device-tree files for these SOCs also are added here!

34.2.6 Compile kernel source code

Now it is required to build the new device-tree. This is done with the usual kernel build process.

```
$ cd build-arm
$ make -C ../sources/linux-imx6 O=$PWD dmo-imx6-ppc_defconfig
$ make -j4
```

The build results exist in the below paths:

- build-arm/arch/arm/boot/zImage
- build-arm/arch/arm/boot/dts/new-device-tree.dtb

34.2.7 Install the new device-tree file on the target

In order to install the new kernel and device-tree, use ***Barebox SWU***. If only the device tree should be updated, comment the KERNEL_IMAGE line in the inifile.

```
#<ver=0.2>

BB_TARGET_DEV=m25p0
OS_TARGET_DEV=mmc3

#KERNEL_IMAGE=zImage2
DTS_IMAGE=imx6dl-dmo-ppc-auo-g804sn05.dtb
```

34.2.8 Booting the target

Follow the instructions during SWU and finally reboot the target. The new LVDS-Display settings should be applied and content displayed as expected.

DATA MODUL

ALL TECHNOLOGIES. ALL COMPETENCIES. ONE SPECIALIST.

DATA MODUL AG

Landsberger Straße 322
DE-80687 Munich
Phone: +49-89-56017-0

DATA MODUL WEIKERSHEIM GMBH

Lindenstraße 8
DE-97990 Weikersheim
Phone: +49-7934-101-0