# Index

## Manual for STM32MP-based Hardware and BSPs

### BSP



### Supported hardware

# BSP overview

## Supported Hardware

Kontron Electronics offers Eval-Kits to get you started quickly with your product and application design. The abilities of your customized hardware can be fitted to your specific needs.

For STM32MP157 hardware, there are Eval-Kits availible with following specs:

**SoCs**

- STMicroelectronics STM32MP157 SOC, ARM® Cortex® A7@648 MHz and ARM® Cortex®M4@196 MHz

**Memory**

- 512 MB DDR3 RAM
- 2MB QSPI NOR Flash
- 512 MB QSPI NAND
- 4GB eMMC

**Interfaces**

- 2 x Ethernet 10/100 MBit/s
- Display: RGB, DSI
- USB Host
- USB OTG
- SD card
- RS232
- RS485
- CAN
- Audio

- 2 x DIO
- 2 x AI

**Power supply**

- 24V supply voltage

List of supported hardware:

| Name | Kit Number | Board | SoM | Description |
|---|---|---|---|---|
| EVK STM32MP157 BL | 50099044 | 40099 131 | 40099 167 | Baseboard with STM32MP157A, without Display |
| EVK STM32MP157 DL | 50099045 | 40099 131 | 40099 167 | Demoboard with STM32MP157A 5" Display and capacitive Touch |

# Software

The Kontron linux BSP software is based on the widely used Yocto linux distribution and supports the Kontron Eval-Kits. The adaptions for the STM32MP157 SOC are integrated by the specialized Yocto layers of OpenST.

The characteristics of the Kontron BSP are:

- Linux kernel 4.19 or newer with board and SOC adaptations
- U-Boot bootloader 2019-11 or newer with board and SOC adaptions
- Linux userland based on the Yocto reference distribution with debian package format
- Boot process configruation in extlinux format
- sysvinit init manager
- GStreamer multimedia framework with hardware acceleration
- Qt 5.11 based on eglfs with OpenGL and QML/QtQuick support

For a quick start Kontron provides a VMware Image already preinstalled with all required development tools.

## Related links

- Yocto linux distribution
- STMicroelectronics STM32MP1 Wiki

# Software licensing

## Licences of Software Packages

The software for this board (BSP) contains open-source software with licence agreements that, among other things, restrict linking against closed-source applications (e.g. GPL, LGPL). Before using any of the libraries or applications, it is therefore necessary to check the licence agreements of the used source code. The licences are contained within the source code of the software packages. Furthermore it is necessary to check any valid patents and licence conditions of the used software, especially for multimedia formats (e.g. mp3 format). Kontron Electronics GmbH does not assume any liability for infringements of patents or licence agreements of parts of the provided BSP. Inside the Yocto build system you find a directory in `<builddir>/tmp/deploy/licenses` , that holds copies of all the licences of the packages, that were built. To get a list of all the packages and their licences included in your image, you can look at the file `license.manifest` in `<builddir>/tmp/deploy/licenses/<full-image-name>` .

If you have no access to those files, feel free to ask Kontron Electronics to provide them for you.

## Typical open-source licences

Here are some notes and further information on different licences.

> **Warning**
>
> The information in this chapter does not make any claim to be complete or to be legally correct.

## GPLv2 used by the Linux kernel and many other packages

If you release or redistribute a product, that includes software under GPLv2, you are bound to provide the source code of those parts with your product (copyleft).
You can either include the source code with your product and deliver it together, or you can include a written offer to provide the source code when requested.
*Attention: GPLv2 does not allow you to provide the source code via a network service (e.g as download). You must deliver it on physical media. Only GPLv3 allows delivery via downloads.*
Many GPLv2 licensed packages include the possibility to license it under a later version of the same licence (e.g. GPLv3), but the Linux kernel for example is GPLv2 only.
If you use code inside your application that is licensed under the GPL you must not keep your application code closed, you are obligated to use the GPL and have to provide the programs source code to your customers.
*Frequently Asked Questions (FAQ)*
A Practical Guide to GPL Compliance

## GPLv3

The GPLv3 is the modernized and updated version of the GPLv2.
See http://www.gnu.org/licenses/quick-guide-gplv3.html.

## LGPLv2.1 used by many libraries

The LGPL is similar to the GPL, but allows that your own proprietary applications link against libraries that are licensed under LGPL, without the need to make your application code public.
If you make changes or additions to the original software, you have to provide these changes to your customer.

## LGPLv3

The LGPLv3 is the modernized and updated version of the LGPLv2.

### MIT

The MIT licence even allows you to modify and distribute software packages, without the need to publish the source code (no copyleft). It is still necessary to include the licence notice in your product.

### BSD

The BSD licence is similar to the MIT licence and has no copyleft.

## Proprietary licences (e.g Freescale/NXP or other HW manufacturers)

The BSP might also contain packages, firmware or drivers, that are licensed under proprietary licences by the manufacturer or other third parties. Depending on your product, those agreements need to be checked for compliance.

# Licence compliance

> **Warning**
>
> The information in this chapter does not make any claim to be complete or to be legally correct.

It is essential to make sure, that your final product complies with all the licences of the included software packages. As mentioned in the last paragraph it depends on the used licences what you have to do for a full licence compliance.

## Notification in the manual

Almost all licences require that you inform your customers in your manual that you use open-source software. You have to mention that parts of your

software are open-source software and deliver a list of the components you are using and their particular licences.

## Source code delivery

If you are using software licensed as GPL or LGPL you have to deliver the source code and licence texts of this software to your customers or at least, make it possible that your customer can get it. There a several options to achieve this.
- Direct delivery with your product: You can accompany your product with a volume containing all used source code under the mentioned licences.
- Written offer: You can state in your manual that every customer of your product can get a copy of the source code as long as the delivery of your product is no longer ago than three years or as long as you deliver spare parts. It is allowed to ask for a small fee to cover your expenses.
- Download (For (L)GPL V3 code only): You can send your customers a link where they can download the source code. You have to guarantee that the link will be accessible for the same time as it would be for the written offer.
Kontron Electronics will automatically generate an archive with the used source code files of your product. Due to the size of this file of a few GB we will not send it to you every time it will be generated. Please feel free to contact us whenever you need this archive.
Please be aware that we only can integrate the source code of the programs we have access to. If you add further open-source software, you have to append these sources to our archive.

## Adding own software to your product

If you are using (L)GPL in version 3 licensed code you have, for version 2 it is recommended, to give your customer the opportunity to install his own programs on your product. For code under (L)GPL version 3 you are obligated to grant your customer the right to install his own programs on your product. Version 2 of the licence only recommends, but does not enforce this. To ensure the security of your device, this opportunity does not have to be included in your product from the beginning, it is appropriate to require your customer to send the product back to you and you will disable the necessary security features to allow the installation of custom programs. You can demand that all

given warranties of your product will expire at the time of applying custom software.

## Exemplary text for your manual:

**English**:
This product contains software components which are licensed as free respectively open-source software under the GNU General Public License, versions 2 or 3, or the GNU Lesser General Public License, versions 2.1 or 3. Everyone can get the source code of this software components from us on a data storage medium (CD-ROM, DVD, USB drive) if requested at our customer support at the following address within three years after the delivery of the product or as long as we offer spare parts or support for the product. [Name of the company]
[Contact person]
[Address]
Including the statement of the following product data:
[Product name]
[Serial number]
[Date of delivery]
We also require a fee of EUR 10,- for the costs of preparation of the medium and shipping to be transferred to the following bank account [Bank account]
Preventive it should be mentioned here that using the right of installing own versions of the open-source software components, which is guaranteed in the licence contract, will expire all certifications and warranties of the product. The operation of the manipulated product will happen on one's own authority.

**German**:
Dieses Produkt enthält Softwarebestandteile, die von den Rechteinhabern als Freie Software bzw. Open Source Software unter der GNU General Public License, Versionen 2 bzw. 3, bzw. der GNU Lesser General Public License, Versionen 2.1 bzw. 3.0, lizenziert werden. Jedermann kann den Quellcode dieser Softwarebestandteile von uns auf einem Datenträger (CD-ROM, DVD oder USB-Stick) erhalten, wenn innerhalb von drei Jahren nach der Auslieferung des Produkts an den Kunden oder solange, wie wir Ersatzteile oder Support für das Produkt anbieten, eine Anfrage an unsere Kundenbetreuung an folgende

Adresse
[Name der Firma]
[Ansprechpartner]
[Adresse]
mit Angabe folgender Produktdaten
[Name]
[Seriennummer]
[Auslieferungsdatum]
gestellt wird und EUR 10,- für die Kosten zur Erstellung des Datenträgers und dessen Versendung vorab auf folgendes Konto [Kontoverbindung] überwiesen werden.
Vorsorglich wird darauf hingewiesen, dass die Nutzung des im Lizenzvertrag zugesicherten Rechts, die Open Source Komponenten gegen eigene Versionen auszuwechseln, zum Erlöschen der Zertifizierung bzw. Garantie führt. Der Betrieb des entsprechend geänderten Gerätes erfolgt auf eigene Verantwortung.

## Building Yocto image from source code

Install Yocto as described here: https://www.yoctoproject.org/docs/current/mega-manual/mega-manual.html
Unpack the archive in the yocto main folder. Setup your environment and download the missing code to build the image. Attention: Some of the code could not be downloaded as it has a proprietary license. Remove these components of the image.
If you have questions, please contact support@kontron-electronics.de.

## Using Qt in a Product

There are several licensing options for Qt. You should decide on one of them before starting to design your application, as switching from the open-source licence to the commercial licence is not allowed.

Here are some sources for further information:

*Qt Licensing*

Qt: Making the right licensing decision

# Quickstart

Below you find a description of the fastest route to set up your build environment for developing your first application for your target hardware.

## Prerequisites on the development computer

- a powerful PC with 12 GB of RAM or more
- at least 80GB free space on hard disk
- VMware Player Version >= 10.x installed
- Kontron Electronics VMware Image
- a terminal application like TeraTerm on the Windows host, or you can use the `screen` program in the linux VMware virtual machine.

## Used Hardware

- mini-USB-B to USB-A cable
- USB-Serial adapter
- Power supply (in scope of delivery)
- Network infrastructure and network cable to establish a connection between host and target

## Connecting the board

> **Hint**
>
> Depending on your board the hardware setup and configuration may vary slightly for your setup. Consult the hardware description for your board, if necessary.

- Connect the debug console by USB interface to the host
  Therefore connect the Debug-USB connector on the board with the USB-Serial adapter and the adapter with a USB port of your host.
  Set the baud rate to 115200 baud/s, 8 data bits, no parity
  Using TeraTerm: Launch application, select COM-port of the FTDI-chip and set the baud rate
- Connecting the Ethernet port
  Depending on your infrastructure you might connect the board to a LAN, or directly to the host computer (e.g. via USB Ethernet adapter)
- Connecting the power supply
  Depending on the boot switches (see your board description) the device will boot from the internal flash or SD card. Watch the boot log in the terminal until finally you will see a login prompt.

## Configuring the board

- Login as user "root", no password is required
- Set network settings temporarily with `ifconfig` or permanently by editing `/etc/network/interfaces`

## Set up application

- Launch the Kontron Electronics VMware Image
- If necessary login with password `user`
- Launch QtCreator from the icon on the left

Open a project:

- c-app-demo (~/yocto-exceet/projects/qtcreator/c-app-demo/c-app-demo.pro) or
- kontron-demo (~/yocto-exceet/projects/qtcreator/exceet-demo/kontron-demo.pro) Specify the network settings of the target (for details see Using Qt-Cross-Toolchain and QtCreator)

## Build, deploy and run the application

Use the according commands in the "Build" menu or the buttons in the toolbar of QtCreator.

# Setting up and using the build environment

This chapter explains the steps necessary to download (clone) a copy of the Yocto-based BSP for Kontron Electronics STM32MP Boards. After setting up the system you can start a build and get a bootloader, kernel image and root filesystem to run on your target hardware.

Please note that Kontron Electronics might also provide you with a pre-configured virtual machine image. If you use this, the build system is already installed and you can immediately start to build an image for your target device. You can skip the next chapter and go on with Initialize the build environment.

## Installing prerequisites on linux PC

Ubuntu 16.04 LTS 64-bit is used as reference OS for the development PC. The newer Ubuntu 18.04 LTS 64-bit version should also work well and should be the preferred version for new installations.

If you start from the beginning, it might be necessary to install some prerequisites on your development PC. Therefore do

```
sudo apt update
```

to update your package index. Afterwards start the package manager *apt* to install the required packages:

```
> sudo apt-get install sed wget curl cvs subversion git-core
coreutils unzip texi2html texinfo docbook-utils gawk python-
pysqlite2 diffstat help2man make gcc
> sudo apt-get install build-essential g++ desktop-file-utils
chrpath libxml2-utils xmlto docbook bsdmainutils iputils-ping cpio
python-wand python-pycryptopp python-crypto
> sudo apt-get install libsdl1.2-dev xterm corkscrew nfs-common
nfs-kernel-server device-tree-compiler mercurial u-boot-tools
```

```
libarchive-zip-perl bison flex
> sudo apt-get install ncurses-dev bc linux-headers-generic gcc-
multilib g++-multilib libncurses5-dev libncursesw5-dev lrzsz
dos2unix lib32ncurses5 repo libssl-dev
```

Please also see the STMicroelectronics website and the official Yocto docs for additional packages, that might be needed.

On Ubuntu 18.04 CubeProgrammer requires the original OracleJava 8 JRE version to be installed. See Ubuntu Wiki page for installation instructions.

Installing Java 8 JRE the manual way without using the PPA works best.

## Gaining access to the repositories

To gain access to the Kontron Electronics repositories please request a useraccount on the Kontron Electronics GitLab server. We recommend to use SSH and authentication with keys.

> **Hint**
>
> In the Kontron VMware image a standard user account is already integrated which can fetch all required sources from the GitLab server. So for the first steps you don't neet to request a user account when you use the VMware image.

Please note that if port 9418 is blocked in your environment, you are not able to use the git-protocol. As some of the openembedded recipes for Yocto rely on the git-protocol, you won't be able to use the Yocto system properly in this case (fetch-tasks may fail).

### Generating a SSH-key on your machine

First check if you already have an existing SSH-key in `~/.ssh/` (id_rsa and id_rsa.pub). If yes you can use it in the next step. If not use the following commands to generate a key:

```
mkdir ~/.ssh
chmod 700 ~/.ssh
ssh-keygen -t rsa
```

You can add a passphrase for additional security when prompted. For more information on SSH authentication please visit the Ubuntu Help.

**Adding the SSH-key to your GitLab account**

Go to the Kontron gitlab server at https://git.kontron-electronics.de and log in. In the top right corner click on your profile picture. Click "Settings" and navigate to "SSH Keys" in the left navigation. Copy and paste your key and give it a name (e.g. work-pc). Copy the content of your `ida_rsa.pub` file from the previous step and paste it in the "Key" input field. Click "Add Key".

# Cloning repositories

## Cloning the core repository (yocto-ktn)

To clone the necessary repositories for your build, go to a directory on your system where you want all the data needed (including source files, build, cache, config, etc.) to be saved (usually $HOME). Please note, that - depending on your build - this usually requires a lot of disk space (> 50 GB). If you have to choose between a SSD and a HDD for running the build, use the SSD as this gives you a little extra speed.

```
cd ~
```

As the Kontron Electronics GitLab server is not in the list of known SSH hosts on your machine initially, add it once by running:

```
ssh git@git.kontron-electronics.de
```

Clone the main repository (yocto-ktn). Please note, that the subdirectory yocto-ktn is created automatically.

```
git clone https://git.kontron-electronics.de/yocto-ktn/yocto-
ktn.git
```

### Cloning additional build repositories

Customer-specific data like kernel configurations, devicetrees for custom boards, custom recipes, etc. is kept in a separate meta-layer within the yocto-ktn system. Customer-specific build configurations are also kept in a separate build directory. The default build configurations for Kontron Electronics Eval-Kits are also kept in repositories like `build-stm32mp` for the layer configuration and `meta-ktn-stm32mp` for the board adaptions. Different Yocto community branches can be fetched by choosing the appropriate branch (e.g. thud).

The most convenient way to initialize a build and clone all necessary repositories is by using the init-env script. Run this script with the desired build configuration (name of the build repo) as argument. See Initializing the build environment.

## Initialize the build environment

Before you are able to build anything for your board, you first have to fetch the yocto metadata which describes which components shall be used and how to build the software for your image.

The Kontron BSP distinguishes between the *core-repository*, *build-repository* and the *meta-layers*:

The **core-repository** is the directory `yocto-ktn` which you have already cloned. It contains scripts to fetch the yocto environments for your board.

The **meta-layers** are yocto layers placed in `yocto-ktn/layers` subdirectory. Yocto finds there all the recipes it needs to build the desired software and images.

The **build repository** is a directory named `build-stm32mp` (or `build-<customer>` for full customized boards). It contains the information what contains and how to create the yocto environment for your board. The most important files in this directories are:

- repo.conf
  It contains information which meta-layers in which version should be

fetched and placed in the `yocto-ktn/layers` subdirectory. This file is special to the Kontron BSPs and `init-env` tool.

- bblayers.conf
  is the yocto configuration file to configure which directories contain recipes to use
- local.conf
  is the yocto configuration file for local build settings

After a yocto build, the build repository also contains the build results and indermediate files.

To initialize your build environment you have to source the `init-env` script. By default this script runs the `meta-update` script (see Updating the repositories) to fetch all yocto layers with the version configured in the `conf/repo.conf` file. So this sets up your yocto environment with its layers.

Sourcing the init-env script automates the following tasks:

1. Running the `meta-update` script from `yocto-ktn/scripts/`
   a. Updating the core repository (yocto-ktn) to the latest revision
   b. Cloning/Updating the build-repository (only if `-u` option is used)
   c. Parsing the file `conf/repo.conf` in the build directory
   d. Cloning/Updating all meta layers to the revisions from `repo.conf`
2. Running the `oe-init-build-env` script from `layers/poky` or `layers/openembedded-core`
   a. Initialize the yocto build environment
   b. If no `conf/local.conf` file exists in the build-directory, create one from the template
3. Setting a machine if the `-m` option is used

For other options of init-env and meta-update, please run `. init-env -h` or `meta-update -h`.

By sourcing `init-env` you also change to the build directory and therefore you are ready to run a `bitbake` command.

There are numerous predefined build configurations available for Kontron Electronics demo and evaluation hardware. Depending on which Yocto community branch they are based on, they can be found in different branches (e.g. thud).

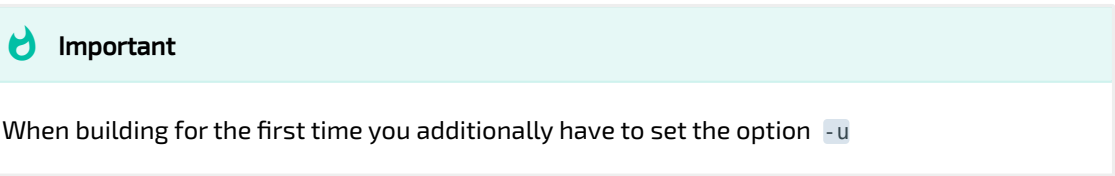
Often there is only one MACHINE for numerous STM32MP demo boards. In this case, different boards are supported by different device tree configurations in bootloader and linux kernel. When an image for this machine is built, all device trees are integrated in this image. So the same image can boot on all supported hardwares.

Which device configuration is chosen is determined by the setting of the 'HOUSING' variable in u-boot. The correct setting can be found in the hardware description for your Kontron board.

For example to initialize the build environment for a Kontron Eval-Kit type:

```
> cd ~/yocto-ktn
> . init-env -r thud build-stm32mp
```

With the `-r` option you can choose the appropriate revision of this git checkout. The revision can be a git branch (as used above) or a special git commit with its commit id or it can also be a git tag like `rel_BSP_stm32mp_1.3.0`.




If you have a full customized boards use your customized build repository `build-<customer>`:

```
. init-env build-<customer>
```

To use a `build-customer` with a specific Yocto BSP branch (only if multiple branches such as rokco, thud, etc. are available):

```
. init-env -r <BSP branch> build-<customer>
```

To initialize the environment and update to the latest revision of the build repository, use the `-u` update option:

```
. init-env -u build-<customer>
```

To initialize the environment and skip checkout errors (e.g. when you have local uncommited changes in some layer), use the `-s` option:

```
. init-env -s build-<customer>
```

Now you are ready to build a recipe, a complete image or the sdk for your machine with the yocto bitbake tool. See Using Bitbake for building what you want.

## Repository and directory Structure

This is how the directory tree will look after you cloned yocto-ktn, initialized your build environment and did an image build:

```
yocto-ktn                   # the main repository
|
├── build-stm32mp           # a build repository (after
initializing the build environment)
|   |
|   ├── conf
|   |   ├── repo.conf       # specifies the revisions of all layers
|   |   ├── local.conf      # specifies local settings for the
build
|   |   └── bblayers.conf   # specifies all layers that will be
parsed by bitbake
|   |
|   └── tmp                 # contains all of the build data
(after building an image)
|       ├── deploy
|       |   ├ images       # contains image files and binaries
for the target
|       |   ├ deb          # contains packages
|       |   ├ licenses     # contains licenses of the packages in
use
|       |   └ sdk          # contains SDK and toolchain binaries
|       |
|       └── work
```

```
│           └── ...          # contains all source and build files
for the packages
│
├── layers                  # contains all meta layers with
recipes (after initializing the build environment)
│   │                       # (each one is a git repository)
│   │
│   ├ meta-openembedded     # contains basic meta layers
│   ├ meta-st               # contains all specific stuff to
support the STM32MP SOC and the OpenST Linux distribution
│   ├ meta-ktn              # contains all Kontron adaptions and
modifications for all Yocto based boards
│   ├ meta-ktn-stm32mp      # contains all Kontron adaptions
specific to the STM32MP SOC
│   └ ...
│
├── scripts                 # contains scripts to automate certain
tasks
├── downloads               # contains all the files downloaded by
the fetcher
│                           # (shared by all builds)
├── sstate-cache            # contains the sstate cache (shared by
all builds)
└── init-env                # this is a script to initialize the
build environment
```

## Updating the Repositories

As time goes by new versions of the used layers may be available. For this you can use the `meta-update` utility. Updating the repositories is conveniently done by running the `meta-update` script in `yocto-ktn/scripts`. However this is often not necessary, because it is automatically run while initializing the build environment.

The `meta-update` script tries to fetch the most recent versions of the core repository (yocto-ktn) and the (customer) build repository (only if option `-u` is set) from the server and then parses the `repo.conf` file in the build repository.

The meta-layers with the specified revisions are then checked out to `yocto-ktn/layers`. The `meta-update` script needs to know the current build, but you usually don't need to set the `-b` option as the script gets the current build from an environment variable `BUILDDDIR`, that is set while running `init-env`.

To update the current build without using init-env you can run `meta-update` directly:

```
meta-update -u
```

If you only want to check out the meta-layers specified in `repo.conf`, maybe because you ran some manual `git checkout` commands in the layers and want to return to the state defined in `repo.conf`:

```
meta-update
```

> **Important**
>
> Please note that whenever you run `init-env` or `meta-update` and have local changes in one of the repositories, you can run into problems while the script tries to checkout a certain revision of the build repository or a meta layer. To resolve these problems, go to the repository and do one of the following steps, depending on your situation:
>
> 1. Discard your uncomittet changes if you do not need them anymore by running `git checkout -- .` or a similar command
>    *or*
> 2. Stash your changes for later reuse, see: git stash
>    *or*
> 3. Commit your changes and if necessary, push them to the remote. You might also want to update `repo.conf` afterwards.

## Other Helpful Scripts

The `yocto-ktn/scripts` directory contains some more scripts, you might find helpful:

1. `meta-bump` updates your `repo.conf`. You can set a certain layer to a specific revision, or you can update all layers to the latest revision by running the script without any arguments.
2. `meta-status` prints information about the current state of the meta layers
3. `init-remote2` initializes TFTP, NFS and a webserver on your local machine to use network boot on your target device and to be able to install

packages on your target from a local pacakge server. It also can get its configuration from a file. For examples see the 'init-remote_*' files in 'conf' subdirectory your build directory.

# Using bitbake

To build a single package, an image for the target or a toolchain, Yocto uses the `bitbake` command. Before you can use the `bitbake` command you must set up your build environment with `init-env` . See Setting up and using the build environment.

## Building a single recipe

To build a single recipe use:

```
bitbake <recipe-name>
```

To list all available recipes run:

```
bitbake -s
```

A single recipe can create multiple packages, such as -dev, -dbg or such. If you want to provide your package directory as package feed for your package manager on your target, recreate the package index by running:

```
bitbake package-index
```

See Package Management for more info.

> **Info**
>
> Please note, that building from scratch can take a long time (several hours!) and needs a lot of disk space and RAM! Especially when you build images with large libraries like Qt. To build as much as possible even when a recipe fails you can use the `-k` option for bitbake.

# Building an image

The Kontron BSP provides three basic images:

- image-ktn-minimal
  is a minimal image which simply boots the hardware
- image-ktn
  is a basic konsole image with utilities for debugging, package-manager and SSH access.
- image-ktn-qt
  is a image with Qt5/EGLFS support with demo applications

For information on which image is dedicated for your board see its hardware description.

To build, for example, the image-ktn type

```
bitbake image-ktn -k
```

and wait until yocto finished its work. You can find the image files in the `tmp/deploy/images/<yourmachine>` directory of your build repository.

> **Important**
>
> Before building an image, you have to read and accept the EULA document. Else the build will fail! See a newly generated `local.conf` where to find the license documents and how to accept them for your board (e.g. set `ACCEPT_EULA_stm32mp-t1000-s-multi = "1"` in `local.conf` )

After you have built your image successfully, you now can go on to boot this image on your machine. For this continue with Booting an image on your hardware.

## Building the SDK for your image

After you have built your image, it is possible to build a SDK which fits your machine and image contents. For the Kontron Eval-Kits there are already precompiled SDKs. See Prebuild BSP releases for more info.

To create an installer for the toolchain of your board and image combination type:

```
bitbake <image-recipe> -c populate_sdk
```

For example:

```
bitbake image-ktn -c populate_sdk
```

For Qt5 development there is a special recipe which contains the tools needed for Qt5 development:

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

**Info**

`meta-toolchain-qt5` and `image-ktn-qt` are two toolchains with a little difference:

- the `meta-toolchain-qt5` is the one choice when you want to develop an GUI application based on the Qt toolkit. This toolchain contains the biggest set of Qt extensions, but no additional libraries of your image. It contains the full set of Qt development tools.
- the `image-*` toolchain fits exactly to your image configuration. This toolchain contains, besides the Qt extensions, all libraries included into your image. This toolchain is the best choice when you create an application which needs special libraries which are only part of your image.

If you created the toolchain-installer with Yocto, you can find it in the directory `<build-dir>/tmp/deploy/sdk`

The installer is a shell-script, that can be executed like this:

```
sh ktn-glibc-x86_64-cortexa7t2hf-neon-vfpv4-image-ktn-qt-toolchain-thud_1.3.1.sh
```

The default installation path is `/opt/kontron/` if not specified otherwise.

# Booting an image on your hardware

## Boot chain overview

The Kontron yocto distribution for STM32MP boards implements the trusted bootchain which consists of

- **trusted-firmware-a** as first stage bootloader (fsbl)
- **u-boot** as second stage bootloader (ssbl)
- **linux** kernel

After powering up the device, the bootrom program located in SOC ROM searches for the boot device to use. The boot device is determined by boot strap pins BOOT0, BOOT1, BOOT2 and bits in OTP. This way it is possible to fetch the first stage boot loader from different storage media.

How to switch the boot source and what boot sources are available can be found in the Eval-Kit description. See there for more information what your board provides and how you can switch the boot source e.g. by dedicated boot switches on the board.

For the Kontron Eval-Kits the boot sources are SD-card and QSPI NOR. After fsbl and u-boot are loaded and started, u-boot loads the linux kernel and root filesystem from the media mentioned in `boot_targets` u-boot environment variable. See u-boot bootloader for more info.

For more information about the boot sequence see STM32MP trusted boot chain

### trusted-firmware loader (fsbl)

Upon supplying power to the hardware, the so called 'first stage bootloader' (fsbl) is loaded into internal SRAM and started from the STM32MP SOC.

The purpose of the fsbl is to do basic system initialisation tasks like DRAM and clock setup. Furthermore it assigns the peripherals to the secure and none

secure area of the device. Secure parts can only used by fsbl. For all other software like u-boot and linux, secure periphals can't be used without notifying the secure software part, and requesting some action from the secure monitor.

```
NOTICE:  CPU: STM32MP157AAD Rev.B
NOTICE:  Model: stm32mp-t1000-s (tf-a)
INFO:    Reset reason (0x4):
INFO:      Pad Reset from NRST
INFO:    Using NOR
INFO:      Instance 1
INFO:    Boot used partition fsbl1
INFO:    NOR: Memory mapped mode
INFO:    Product_below_2v5=1: HSLVEN update is
INFO:      destructive, no update as VDD>2.7V
NOTICE:  BL2: v2.0(debug):0.1.0.d-7-gd5e797c-dirty
NOTICE:  BL2: Built : 15:39:08, Apr 12 2019
INFO:    BL2: Doing platform setup
INFO:    RAM: DDR3-DDR3L 16bits 528Mhz K4B4G1646E
(v1,1066-7-7-7,cal)
INFO:    Memory size = 0x20000000 (512 MB)
INFO:    BL2 runs SP_MIN setup
INFO:    BL2: Loading image id 4
INFO:    Loading image id=4 at address 0x2fff0000
INFO:    Image id=4 loaded: 0x2fff0000 - 0x30000000
INFO:    BL2: Loading image id 5
INFO:    Loading image id=5 at address 0xc0100000
INFO:    STM32 Image size : 758445
WARNING: Skip signature check (header option)
INFO:    Image id=5 loaded: 0xc0100000 - 0xc01b92ad
INFO:    read version 0 current version 0
NOTICE:  BL2: Booting BL32
INFO:    Entry point address = 0x2fff0000
INFO:    SPSR = 0x1d3
NOTICE:  SP_MIN: v2.0(debug):0.1.0.d-7-gd5e797c-dirty
NOTICE:  SP_MIN: Built : 15:39:08, Apr 12 2019
INFO:    ARM GICv2 driver initialized
INFO:    stm32mp HSE (20): Secure only
INFO:    stm32mp PLL2 (27): Secure only
INFO:    stm32mp PLL2_R (30): Secure only
INFO:    SP_MIN: Initializing runtime services
INFO:    SP_MIN: Preparing exit to normal world
```

After fsbl has finished system setup, u-boot is started.

## u-boot bootloader (ssbl)

Upon supplying power to the hardware, the U-Boot bootloader stored in the NOR flash or SD card will be started (depending on the boot switch configuration).
The boot messages of the bootloader on the debug terminal will look something like this:

```
U-Boot 2018.11-stm32mp-r2 (Apr 11 2019 - 18:56:44 +0000)

CPU: STM32MP157AAD Rev.B
Model: stm32mp-t1000-s (u-boot)
Board: stm32mp1 in trusted mode (ex,stm32mp-t1000-s)
       Watchdog enabled
DRAM:  512 MiB
Clocks:
- MPU : 648 MHz
- MCU : 196 MHz
- AXI : 264 MHz
- PER : 24 MHz
- DDR : 528 MHz
NAND:  0 MiB
MMC:   STM32 SDMMC2: 0, STM32 SDMMC2: 1
Loading Environment from SPI Flash... SF: Detected mx25r1635f with
page size 256 Bytes, erase size 64 KiB, total 2 MiB
OK
In:    serial
Out:   serial
Err:   serial
Net:   Found Micrel KSZ8081 PHY, enable 50MHz RMII mode

eth0: ethernet@5800a000
Boot over nor0!
Hit any key to stop autoboot:  2
```

After waiting some seconds, the bootloader tries to boot the linux kernel and mount the root filesystem. Which kernel, device-tree and linux kernel command line is used and where the root filesystem is located, is configured in the `extlinux.conf` file u-boot searches for in the boot partition.

This boot partition can be located on different storage media. For example 'mmc0' for SD-cards, 'mmc1' for eMMC and 'ubifs0' for QSPI NAND. But this depends on your board and what it provides. Besides this, 'pxe' is a special case

for booting over network (see Boot from network for more info). Which boot medias are scanned for boot partitions is determined by the u-boot variable `boot_targets`, which contains a list of boot medias to be scanned. For example the Eval-Kits `boot_targets` variable is set to `mmc0 mmc1 ubifs0 pxe`. Please refer to the hardware documentation for information on what your board supports.

U-boot searches the `extlinux.conf` file in a directory `${boot_device}${boot_instance}_${board_name}`. This results for example in the directory name `mmc0_stm32mp-t1000-s` for 'boot on SD-card' for the board 'stm32mp-t1000-s'. U-boot searches this directory with prefix '/' or '/boot/' on the boot partition.

To enter the bootloader command line hit a key before the bootloader starts booting. With this command line it it possible to change u-boot variables which influences the boot mode.

```
STM32MP>
```

List of basic U-Boot commands:

| Command | Description |
|---|---|
| printenv | print all variables and their values |
| printenv <variable> | print content of <variable> |
| setenv <variable> <value> | Set <variable> to <value> |
| setenv <variable> | Delete <variable> |
| editenv <variable> | Edit value of <variable> |
| saveenv | save all variables to flash |
| env default -a | restore default environment |

Besides changing the boot variables u-boot provides commands for testing hardware, reading and writing storage media and much more. Use the u-boot command `help` to get a first impression what is possible.

The most important u-boot variables for booting are:

- **boot_device**
  Device string from where bootrom fetched tf-a and u-boot. For example `mmc` for SD-card and `nor` for QSPI NOR.
- **boot_instance**
  Instance of boot device, commonly '0'.
- **boot_targets**
  This is a list of boot devices which are scanned when u-boot searches the extlinux.conf boot configuration file. Available devices are `mmc0` for SD-card, `mmc1` for eMMC, `ubifs0` for QSPI NAND and `pxe` for network boot. Please have a look in the description for your Eval-Kit which boot devices are available for your board.
- **board_name**
  This variable is set from the device tree file for u-boot. It's the first compatible entry in the device tree file. This variable is used to build the search path of `extlinux.conf` on the boot partition.
- **housing**
  This variable is appended to the board_name. It can be changed by the user to enable some board variants. For example there is a board named 'stm32mp-t1000-s' which doesn't have a display. With the help of the `housing` variable some variants can be selected without the need to exchange u-boot and its device-tree configuration. For example setting the `housing` variable to '-50' selects the Eval-Kit variant with 5 inch display type ('stm32mp-t1000-s-50').
- **serverip**
  Ip address of tftp server from where pxe tries to fetch the boot files.
- **bootdelay**
  Delay in seconds for which u-boot waits for user input to get into u-boot console. If set to zero you can't get into u-boot console while booting. If

you want to have this possibility again, you have to modify the bootdelay variable in u-boot environment from a running linux.

> **Important**
>
> Setting `bootdelay` does not prevent starting u-boot console for all cases! If u-boot environment is not able to boot or detects a failure in boot processing it is possible that u-boot ends up in u-boot console.

Please note, that you can also modify the bootloader variables from within a running linux if the package u-boot-fw-utils is installed. Then you can use `fw_printenv` and `fw_setenv` the same way as you would use `printenv` and `setenv` in U-Boot.

## Partition layout structure

The linux userland uses two partitions to store the system and data:

- **borootfs**
  mounted on /, storage for linux userland software and also for the boot files: linux kernel, device tree board descriptions and extlinux boot in /boot directory
- **userfs**
  mounted on /usr/local, for user data storage
- **rescuefs** (optional)
  optional partition for a rescue boot system

Depending on the capabilities of the hardware and the setting of the boot switches the SOM can boot directly either from QSPI NOR, SD card or USB boot.

## Image types

Depending on the configuration, Yocto creates different types of images. You can find the image files in your build directory in `tmp/deploy/images/`

`<MACHINE>` . The variable `IMAGE_FSTYPES` can be used to set the types of images to be created.

The image files of the latest build can easily be accessed by using the links, that always point to the files most recently created. Older image files will be kept in this directory until they are deleted.

After compilation two filesystem images are available: *borootfs* and *userfs*. The names of these filesystems are composed of the image name and the distribution and machine. For a configuration with

image: `image-ktn` , distribution: `ktn` , machine: `stm32mp-t1000-s-multi`

the image names will be for:

- borootfs: `image-ktn-qt-stm32mp-t1000-s-multi`
- userfs: `image-ktn-qt-userfs-stm32mp-t1000-s-multi-thud`

The Kontron Yocto BSP packages only generate compressed filesystems with tar.gz ending. These images are intended to be flashed by booting with SD card and flashing with the mptool Tool. If you wish to generate other images you have to enable them in `IMAGE_FSTYPES` .

- **image-*.tar.gz** This file contains the compressed filesystem and can be used to flash the contents with mptool tool or to load the filesystem via network after it was extracted
- ***.sdcard** This file has first to be created out of the image files in the image directory. See section create a bootable SD-card for this. It contains the borootfs and userfs partition besides the tf-a and u-boot bootloader. In default configuration it also contains the flash contents itself to flash them directly with mptool to the internal flashes.
- **uImage or zImage** This is the kernel image containing the linux kernel.
- **u-boot-*-trusted.stm32** This is the u-boot bootloader for the STM32MP trusted boot chain. This can be directly flashed to NOR Flash, eMMC or SD card.

- **tf-a-*-trusted.stm32** This is the first stage bootloader (fsbl) for the STM32MP trusted boot chain. This can be directly flashed to NOR Flash, eMMC or SD card.

# Booting your board

There are sevaral options to boot your software on the board:

1. boot your software from a SD-card to run your software for testing or to burn it into the internal flashes with mptool
2. boot your software from internal flash e.g. for productive envrionments
3. boot your software via network while developing
4. boot a u-boot via USB boot for flashing internal flashes with STM32CubeProgrammer on a totally blank device

# Booting from SD card

## Creation of a bootable SD card

Before you can *boot* from SD card, you have to *create* a bootable SD card. There are two ways to create one:

1. Download a already prepared SD card image for your board and write it to the SD card. This is the simplest way to update the firmware on your board, but is limited to the SD card images provided from Kontron.
2. Create your own SD card image with yocto which can be customized to your needs.

For both ways you have to provide an SD card reader which works in your environment!

### Using prebuilt Kontron image

The simplest way to get a bootable SD card is to download prebuilt images from the Kontron file server. Search the *.sdcard image files for your board and the desired BSP version.

If you want to create the SD card with your PC and Windows os you have to use a suitable imager program for Windows to write the image contents to your SD card. One such easy to use program is 'Etcher'. You can download it from https://github.com/balena-io/etcher/releases

> **Danger**
>
> The usage of imager programs can be dangerous. If you select the wrong disk, it might cause severe loss of data on your hard disk or other drives!

If you want to write this prebuild *.sdcard file on a linux host, you can go on with Create image with yocto and simply skip the image creation procedure with 'create-sd-card.sh'.

### Create image with yocto

To create a bootable SD card from your image tar.gz files you can use the script `create-sd-card.sh` from your Yocto tmp directory `tmp/deploy/images/<MACHINE>/script-mp` or you can download *.sdcard image files or the tar.gz files of a prebuilt image from the Kontron file server.

The layout and contents of the image is configured by a '*.layout' configuration file. Preconfigured files exist for different configurations of machine, distro and image.

To create the tar.gz files of your image, you have to build your images with bitbake. Then change into the images directory and run the 'create-sd-card.sh' script **as root**. Root is required for mounting and unmounting the image contents and unpack the contents with the right access rights.

```
> bitbake image-ktn
[...]
> cd tmp/deploy/images/stm32mp-t1000-s-multi/script-mp
> sudo ./create-sd-card.sh -c image-ktn-qt-ktn-stm32mp-t1000-s-
multi.layout
```

> **Important**
>
> If you want to use this sd card to write your firmware to the internal flashes mit mptool, don't forget to use the `-p y` command line option or `PROD_IMG=y` configuration file option!

After this you find a file named `image-ktn-qt-ktn-stm32mp-t1000-s-multi.sdcard` in `tmp/deploy/images/stm32mp-t1000-s-multi` directory.

Now fill the SD card with the image contents:

> **Danger**
>
> The usage of the `dd` command in combination with `sudo` can be dangerous. If you use wrong parameters, this might cause severe loss of data on your hard disk or other drives!

```
sudo dd if=../image-ktn-qt-ktn-stm32mp-t1000-s-multi.sdcard of=/dev/sdb bs=8M conv=fdatasync status=progress
```

## Boot into SD card

To boot from the SD-card,

- configure your boot switches for 'SD card boot' (for Kontron Eval-Kits this is not necessary, because the first boot source is the SD card),
- prepend the u-boot environment variable `boot_targets` with `mmc0` or set it to `setenv boot_targets mmc0 mmc1 ubifs0 pxe` (this is the default setting for Kontron Eval-Kits)
- optionally set the boot variant in `housing` for boards with display (e.g. `-50` for the `stm32mp-t1000-s-50` board and device tree). This requires saveenv and a reboot to take effect.

> **Hint**
>
> See your board documentation for information for your board about default boot devices

If you want your settings to be persistent, don't forget to run `saveenv`.
Then you are ready to boot by running `run bootcmd`.

# Booting from internal flash

## Update firmware in flash

Before the the device can boot from internal flash, you have to write your software to it.

Depending on your hardware it is possible that you have two locations where your filesystems can be stored: QSPI NAND or eMMC flash.

The simplest way to populate your flash with the new filesystem contents is to boot from sd card and use the mptool to flash the device with the new contents.

First create your SD card image with create-sd-card.sh tool and the `-p y` option to include your firmware files in the SD card image in `/usr/local` directory. See Creation of a bootable SD card for this.

After your device has booted into the linux system, login and start updating your device:

```
mptool flash-bl-fs -m EMMC
```

mptool has different options to control what shall be updated. The most common are:

| Option | Description |
| --- | --- |
| flash-tfa | Flash tf-a (first stage) bootloader |
| flash-u-boot | Flash u-boot (second stage) bootloader |
| flash-full-bl | Flash all bootloaders (tf-a and u-boot) |
| flash-full-fs | Flash all filesystem partitions |
| flash-bl-fs | Flash all bootloaders and all filesystem partitions |
| -m | Flash filesystem partitions to medium (e.g. NAND or EMMC) |

See the mptool description or `mptool -h` for more options.

**Export eMMC as USB flash drive**

In case of eMMC memory it is also posible to publish the eMMC contents as USB flash drive when the OTG port of the device is connected thru USB cable to your development machine.

This way the internal eMMC flash can be handled like a normal USB attached flash device on your development machine. It is very handy if you only want to explore the file system on the board or if you want to exchange only a few files.

To activate USB flash mode in u-boot use the `ums` command, here internal eMMC (mmc 1) on a Kontron Eval-Kit:

```
STM32MP> ums 0 mmc 1
```

Now you can mount the internal eMMC memory like an USB flash drive in your linux development environment. You can then use all the tools linux provides for partitionning, formatting and writing contents to USB flash drives.

## Boot into flash

After the flash is fully populated with your software (see Update firmware in flash), you can boot from flash:

- configure your boot switches for 'NOR boot', or remove the bootable SD card,
- prepend the u-boot environment variable `boot_targets` with `mmc1` for eMMC flash or `ubifs0` for QSPI NAND flash. Or set it to `setenv boot_targets mmc1` or `setenv boot_targets ubifs0`
- optionally set the boot variant in `housing` for boards with display (e.g. `-50` for the `stm32mp-t1000-s-50` board and device tree). This requires saveenv in u-boot and a reboot to take effect.

If you want your settings to be persistent, don't forget to run `saveenv` in u-boot.
Then you are ready to boot by running `run bootcmd`.

> **Hint**
>
> You can find in the appropriate board description, which boot_targets and and boot variants are supported for your board.

Here is an example for selecting NAND for boot, saving it to environment and reboot to take this setting into effect:

```
STM32MP> printenv boot_targets
boot_targets=mmc0 mmc1 ubifs0 pxe
STM32MP> setenv boot_targets ubifs0
STM32MP> setenv housing -50
STM32MP> saveenv
Saving Environment to SPI Flash... SF: Detected mx25r1635f with
page size 256 Bytes, erase size 64 KiB, total 2 MiB
Erasing SPI flash...Writing to SPI flash...done
OK
STM32MP> reset
```

# Booting via network adapter (TFTP/NFS)

In a development environment where you need to do a lot of testing while changing the kernel and rootfs, it is often helpful to have the system on your development machine and share the files via network with your target device. Please consult the hardware description of your board for the information which ethernet interfaces are supported in u-boot.

In combination with the Yocto build system you can export the `<builddir>/tmp/deploy/images` directory via TFTP to allow the target to fetch the kernel image, devicetrees, etc. To allow the target to use a rootfs over the network, you need to export the rootfs via NFS on your development machine.

The Kontron BSP uses PXE boot for booting over network. See STMicroelectronics Wiki for more information.

## Setup TFTP on the host

To install the TFTP-Server:

```
> sudo apt-get install tftp-hpa
```

Change the TFTP-settings by changing the content of `/etc/default/tftp-hpa` :

```
TFTP_USERNAME="tftp"
TFTP_DIRECTORY="/tftproot"
TFTP_ADDRESS="[::]:69"
TFTP_OPTIONS="--secure"
```

After installation enable and restart the tftp server:

```
> sudo systemctl enable tftpd-hpa
> sudo systemctl restart tftpd-hpa
```

## Setup NFS on the host

To install the NFS server and create nfs directories:

```
> sudo apt-get install nfs-kernel-server
> sudo mkdir /nfsroot /nfs
```

Change the NFS settings by changing the content of `/etc/exports` :

```
/nfsroot *(rw,no_root_squash,sync,no_subtree_check)
/nfs *(rw,no_root_squash,sync,no_subtree_check)
```

After installation enable and restart the nfs server:

```
> sudo systemctl enable nfs-kernel-server
> sudo systemctl restart nfs-kernel-server
```

With `exportfs` you can check what your nfs server exports:

```
> sudo exportfs
/home/user      <world>
/nfsroot        <world>
/nfs            <world>
```

## Populate NFS and TFTP directories

To extract the rootfs and tftp contents, we suggest to use the `init-remote2` script in `yocto-ktn/scripts` :

```
init-remote2 -f <configfile> [-t] [-n] [-o] [-c] [-h]
```

The most important configuration options are:

- f: use a configuration file
- t: generate TFTP contents
- n: generate NFS contents
- o: overwrite NFS contents if they already exist
- c: clean NFS directory if it already exists
- h: print out help text

You can get more inforation calling the script with the help (-h) option. Although all information can be given on the command line, we suggest to use config files to ease the use of this tool. Config options on the command line have a higher priority as options in the config file. This way the config file can contain default settings which can be overwritten by command line options.

Example configuration files can be found in `build-stm32mp/conf` directory. This is the example file `init-remote_t1000-s-multi_console.conf` for the console image:

```
# this is a simple config file for init-remote2

# name of builddir
OPT_BUILDDIR=build-stm32mp

# name of machine
OPT_MACHINE=stm32mp-t1000-s-multi

# name of used distro
OPT_DISTRO=ktn-eglfs

# name of rootfs image file
OPT_IMAGE=image-stm32mp-console

# compatibility option
OPT_SINGLEBOARD=0

# generate tftp directory in /tftproot/$OPT_TFTPNAME
OPT_GEN_TFTP=0
OPT_TFTPNAME=t1000-s-multi_console

# generate nfs directory in /nfs/$OPT_NFSNAME
OPT_GEN_NFS=0
OPT_NFSNAME=t1000-s-multi_console
# overwrite existing files
OPT_OVERWRITE_NFS=0
# clean nfs directory before deploying a new one
OPT_CLEAN_NFS=0

# generate debian packacke repos in $OPT_BUILDDIR/pkg-repos
OPT_GEN_PKGREPOS=0
```

For example to generate the NFS and TFTP contents for the console image:

```
init-remote2 -f conf/init-remote_t1000-s-multi_console.conf -t -n
```

The files in `/nfs/t1000-s-multi_console` and `/tftproot/t1000-s-multi_console` are populated. `init-remote2` uses `sudo` to populate some files for NFS. So it might be that you are being asked for the root password of your development machine.

To update the directoies with modified content use

```
init-remote2 -f conf/init-remote_t1000-s-multi_console.conf -t -n -o
```

With the `-o` option all files out of the image are replaced with the new content. Files in the NFS directory which are not contained in the image stay untouched in their location.

If you want to wipe out an old NFS directory and get one brand-new use the `-c` option:

```
init-remote2 -f conf/init-remote_t1000-s-multi_console.conf -n -c
```

> **Hint**
>
> `init-remote2` only deploys borootfs image. userfs and rescuefs images are not deployed!

## Create PXE configuration for your board

You have to create a PXE configuration for every board you want to boot over network. The PXE configuration is stored in the `pxelinux.cfg` subdirectory of your TFTP server directory. For every board you have to create a configuration file named similar to its MAC address.

First get the MAC address of your board from u-boot prompt:

```
STM32MP> pri ethaddr
ethaddr=70:82:0e:99:96:52
```

Create and edit the config file for this board. Prepend config file name with `01-` and replace all `:` by `-`. Here is an example:

```
> gedit /tftproot/pxelinux.cfg/01-70-82-0e-99-96-52
```

These are the PXE config file contents for this example:

```
#bootfile for t1000-s
DEFAULT t1000-s-50_console
TIMEOUT 20
LABEL t1000-s-50_console
    KERNEL t1000-s-multi_console/uImage
    FDT t1000-s-multi_console/stm32mp-t1000-s-50.dtb
    APPEND root=/dev/nfs nfsroot=192.168.1.240:/nfs/t1000-s-
multi_console,nfsvers=3 rootwait rw earlyprintk
console=ttySTM0,115200 ip=192.168.1.11
```

For more information about booting over network see also STMicroelectronics Wiki

**Important**

- You have to adapt the NFS server ip and the board's ip address to your setup

## Boot from network

In the bootloader prepend or replace the list of `boot_targets` with the `pxe` entry (e.g. `boot_targets=pxe mmc0 mmc1 ubifs0`). Furthermore you should adapt the `serverip` environment variable to your host's ip address. If you want your settings to be persistent, don't forget to run `saveenv`.

Then you are ready to boot by running `run bootcmd`.

```
STM32MP> setenv boot_targets pxe mmc0 mmc1 ubifs0
STM32MP> setenv serverip 192.168.1.240
STM32MP> saveenv
STM32MP> run bootcmd
```

## Initial booting with USB boot

The STM32MP1 SOCs have the ability to load images and data from a host PC via USB to the internal SRAM. With the help of u-boot it is possible to write images to the connected storage devices. This is mostly used in development or production environment for board bring up. For Kontron SOCs flashing QSPI NOR, eMMC and SD card is currently supported.

> **Hint**
>
> For the Kontron BSPs the filesystem partions aren't generated with default settings. STM32CubeProgrammer is only used to flash the bootloaders 'tf-a' and 'u-boot' if booting from sd card is not recommended. Further flashing is done by booting from SD card and using 'mptool'.

To use this feature the SOC has to be USB boot mode, which means you set the boot switches to USB boot or the internal NOR flash must be empty.

> **Hint**
>
> Depending on the OTP-Fuses and if they were already burned, it might not be possible to put the SOC into USB boot mode anymore.

When the SOC is in USB boot mode and you connect a Micro-USB cable to the OTG-Port of the board, your host PC should detect a new USB device.

On Linux you can check for the device with `lsusb` :

```
> lsusb | grep STM
Bus 003 Device 039: ID 0483:df11 STMicroelectronics STM Device in
DFU Mode
```

To download and flash content from your host PC on the device you have to use the STM32CubeProgrammer

## Flashing with STM32CubeProgrammer

To load a full system (tf-a, u-boot, borootfs, userfs) or only part of it into the QSPI NOR eMMC or SD card you can use the `*.tsv` programmer files from the yocto machine directory `meta-ktn-stm32mp/conf/machine`.

First check the connection to the board:

```
> STM32_Programmer_CLI -c port=usb1

-------------------------------------------------------------------
                        STM32CubeProgrammer
v2.0.0

-------------------------------------------------------------------



USB speed   : High Speed (480MBit/s)
Manuf. ID   : STMicroelectronics
Product ID  : DFU in HS Mode @Device ID /0x500, @Revision ID /
0x0000
SN          : 002500413338510B34383330
FW version  : 0x0110
Device ID   : 0x0500
Device name : STM32MPxxx
Device type : MPU
Device CPU  : Cortex-A7
```

If the programmer can't get the connection, you might adapt your UDEV rules to grant your user account to communicate with this device. In this case you have to create an appropriate UDEV rule for the programmer. This can be done for example by creating the file `/etc/udev/rules.d/50-usb-stmicro.rules` with the following content:

```
# Grant permissions for all STMicro devices
SUBSYSTEMS=="usb", ATTRS{idVendor}=="0483", GROUP="users",
MODE="0666"
```

When connection is ok, you can now program the flash contents. In this example we'll program the QSPI NOR and eMMC flash of the board. For this you

should have the tf-a, u-boot, bootfs, rootfs and userfs image files in your images directory (for example `tmp/deploy/images/stm32mp-t1000-s-multi` ):

> **Info**
>
> Depending on the size of the filesystem images this programming requires a few minutes to finish.

```
> cd build-stm32mp
> cp ../layers/meta-ktn-stm32mp/conf/machine/FlashLayout_t1000-s-
multi_qt_nor-emmc.tsv tmp/deploy/images/stm32mp-t1000-s-multi/
> STM32_Programmer_CLI -c port=usb1 -tm 100000 -w tmp/deploy/
images/stm32mp-t1000-s-multi/FlashLayout_t1000-s-multi_qt_nor-
emmc.tsv

-------------------------------------------------------------------
                        STM32CubeProgrammer
v2.0.0

-------------------------------------------------------------------



Warning: Timeout is forced to 100000 ms


USB speed   : High Speed (480MBit/s)
Manuf. ID   : STMicroelectronics
Product ID  : DFU in HS Mode @Device ID /0x500, @Revision ID /
0x0000
SN          : 002500413338510B34383330
FW version  : 0x0110
Device ID   : 0x0500
Device name : STM32MPxxx
Device type : MPU
Device CPU  : Cortex-A7


Start Embedded Flashing service



Memory Programming ...
Opening and parsing file: tf-a-stm32mp-t1000-s-trusted.stm32
  File          : tf-a-stm32mp-t1000-s-trusted.stm32
  Size          : 245360 Bytes
```

```
  Partition ID  : 0x01

Download in Progress:
[==================================================] 100%

File download complete
[...]

Memory Programming ...
Opening and parsing file: st-image-userfs-ktn-eglfs-stm32mp-t1000-
s-multi.ext4
  File          : st-image-userfs-ktn-eglfs-stm32mp-t1000-s-
multi.ext4
  Size          : 64 MBytes
  Partition ID  : 0x12

Download in Progress:
[==================================================] 100%

File download complete
Time elapsed during download operation: 00:00:53.085

RUNNING Program ...
  PartID:      :0x12
Start operation done successfully at partition 0x12
Flashing service completed successfully
```

After successful flashing the device you can change the boot switches to eMMC boot and start your linux system.

Programming can also be done for other flash devices on the board. For more examples see the other tsv files in the yocto machine directory mentioned above.

It is also possible to flash only some parts of the device. For this the ID field of the tsv file can be modified:

- **-**: do nothing
- **P**: update the partition data, do not modify GPT partition table
- **PE**: (re)create GPU partition table, but do not update partition data
- **PD**: (re)create GPT partition table and update partition data
- **PDE**: (re)create GPT partition table and empty partition data

**Hint**

- The tsv file format is sensitive for tabs and whitespaces! So be sure to separate the fields in one row by one single tab. Else reading the file by STM32CubeProgrammer will fail!
- If only some flash parts of device shall be programmed, the complete partition layout in the tsv file has to fit to the real device even the partition is not touched. Else the programmer complains and refuses programming.

For further information how to install and use the tool and how to configure the contents to be flashed, read the documentation on STMicroelectronics website

**Important**

For the correct function of CubeProgrammer it is important that the u-boot launches the command `stm32prog usb` if it is started by USB boot. It is required for communication with CubeProgrammer. For the Kontron u-boot this is done in the preboot command. If you modify u-boot for your own requirements you should have this in mind!

**Important**

The Kontron u-boot loaded with CubeProgrammer reads the u-boot environment on start. This can lead to situations where these settings prevent the usage of CubeProgrammer because, for example it doesn't start `stm32prog usb` . In this case the u-boot environment has to be erased. This can be done on u-boot command line with the command `mtd erase env`

## Login

After a successful boot you will be greeted with a login prompt. The default (and only) user in an unmodified Konton image is the `root` user and it has no password.

# Package management

In case you need additional software packages, that are not preinstalled on the Kontron Electronics hardware you have two choices on how to get them running. You can either ...

1. ... setup the full Yocto-based build environment and rebuild the image with the needed package included,
2. ... or you can use package-files that only need to be installed on the already running hardware.

To obtain a package-file for the software you need, that is compatible with the Kontron Electronics BSP, you can create a package from within the Yocto build environment by using `bitbake` .

To build a recipe use:

```
bitbake <recipe-name>
```

To list all available recipes run:

```
bitbake -s
```

A single recipe can create multiple packages, such as -dev, -dbg or such. After the build recreate the package index by running:

```
bitbake package-index
```

In case that a certain package is not available through an existing recipe (check at oe-Index), please contact Kontron Electronics for help.

# Install the package manager

In general the package manager `dpkg` and `apt-get` is already installed on the Kontron Electronics hardware. It is included in the Kontron images in Yocto by default.

# Set up a local package server

If you have the Yocto environment installed on your development computer and you want to be able to directly fetch the created packages located in `tmp/deploy/deb` in your build directory by your target, you can set up a webserver like Apache, that exports this directory via http.

For a very simple setup you can use the in webserver module in python3:

```
cd build-stm32mp/tmp/deploy/deb
python3 -m http.server 8000
```

This starts a simple webserver in the current working directory located on port 8000

# Using the package manager 'dpkg' and 'apt-get'

## Configuration file sources.list

To change the apt-get settings and to add/remove package repositories, please edit the file /etc/apt/sources.list

```
nano /etc/apt/sources.list
```

**Info**

Yocto generates the sources.list file automatically with the appropriate repositories for your image. Which server URL is used is determined by the Yocto variable `PACKAGE_FEED_URIS` . Set this variable in local.conf or any other appropriate place.

For example: `PACKAGE_FEED_URIS = "http://192.168.1.240:8000/"`

## Examples

Install the package 'tree' from the remote server:

```
apt-get update
apt-get install tree
```

# Using Qt cross toolchain and QtCreator

By using a cross toolchain on your development computer, you can easily create Qt5 or bare C/C++ applications to run on your Kontron hardware. This is the fastest way to get your application run on your target device. With this setup you are not bound to the yocto workflow. After you have developed your application, you can integrate it into your own yocto layer to deploy your application with your image.

## Setting up the toolchain

To be able to compile for your target system you must have the appropriate SDK for your system installed. See Building the SDK for your image for more information on how to do that.

> **Hint**
>
> The version of the SDK should match the firmware version which runs on your device.

## Setting up Qt/QtCreator

### Installing Qt/QtCreator

Check the download page for the latest release of QtCreator and download the `qt-creator-opensource-linux-x86-X.X.X.run` .

Install QtCreator by double-clicking the *.run-file in the file-manager or by running

```
chmod +x qt-creator-opensource-linux-x86_64-3.2.1.run
./qt-creator-opensource-linux-x86_64-3.2.1.run
```

Alternatively you might also want to consider installing the full Qt environment for your desktop, including sources, tools and QtCreator. The advantage is, that

apart from the Yocto-based target toolchain, you also have a toolchain available for your desktop environment. This enables you to switch between deploying to the target and deploying to your desktop machine for testing.

The online installer for the latest OpenSource edition of Qt can be found here for x86_64 bit linux systems:

http://download.qt.io/official_releases/online_installers/qt-unified-linux-x64-online.run

## Starting QtCreator

To run QtCreator with the Kontron cross environment for your board you have to source the yocto environment before starting the QtCreator.

This can be accomplished thru different ways:

1. Manual sourcing of the yocto environment setup script and starting QtCreator on command line
2. Using Kontrons `qtcreator-embedded.sh` script

The simplest way is using `qtcreator-embedded.sh` script which can be found in Kontrons VMware image.

### Manual sourcing

Open a shell and source the yocto environment script, then start QtCreator:

```
source /opt/kontron/stm32mp-t1000-s-multi/thud_1.3.1/environment-setup-cortexa7t2hf-neon-vfpv4-ktn-linux-gnueabi
qtcreator
```

### Using `qtcreator-embedded.sh` script

You can find the `qtcreator-embedded.sh` script in Kontrons VMware image at `/home/user/Qt/Tools/QtCreator/bin/qtcreator-embedded.sh`

After launching you can choose the toolchain you want to use. Choosing 'Cancel' doesn't source any yocto environment script.

# Setting up your kit

For compiling and debugging you need an appropriate so called 'kit' for your device. This kit provides all information needed for debugging with your hardware like

- cross toolchain
- debugger binary
- debug symbols and headers for the libraries the target uses (sysroot)

Kontron suggests to use one of the provided kits for the demo boards. If your have to modify one of the kits or you want to create your own kit, these kits are a good starting point.

Kits can be modified in the *Kits* view ( `Tools -> Options -> Kits` ). Here you can configure your kit with:

- its name (choose what you want)
- the device type (only *Generic Linux Device* for Kontron devices)
- the device to be used (see also Adding your own device)
- the sysroot - headers and libraries for the Yocto firmware (*toolchain base directory*/sysroots/cortex*)
- c cross-compiler (*toolchain base directory*/sysroots/x86*/usr/bin/arm-*-linux/arm-*-gcc)
- c++ cross-compiler (*toolchain base directory*/sysroots/x86*/usr/bin/arm-*-linux/arm-*-g++)
- cross-debugger (/usr/bin/gdb-multiarch)
- Qt version (*toolchain base directory*/sysroots/x86*/usr/bin/qt5/qmake)

The configuration entries for compiler, debugger and Qt version have several tabs to enter the matching configuration settings. In the kits view you can only select already pre-defined settings.

### Adding your own device

To be able to use the deploy and debugging features within QtCreator with your own device you can modify an existing configuration or create a new one.

If you want to create a new configuration in QtCreator or modify an existing one go to `Tools -> Options -> Devices` and add your device by clicking `Add ...`. Choose `Generic Linux Device` from the drop down menu. Set the appropriate values for the device name, ip address and the username and password. The connection can be tested with the `Test` button in the devices tab.

## Deploying to the target

If you are using a development environment provided by Kontron, and you are running a build of a demo image on your hardware (includes ssh-server, sftp-server, gdbserver), you should be able to deploy your app to the target via network. To change the settings of the remote device (IP-Adress, ...) in QtCreator, go to `Tools -> Options -> Devices -> Kontron stm32mp DemoKit`. QtCreator uses sftp and ssh to copy the files and run the application remotely.

The deployment of your files is configured in your qmake project file by setting the `INSTALLS` variable. The Kontron demo programs contain some simple deployment rules. For more information see the Qt/qmake documentation.

> **Important**
>
> Before starting your own Qt application on the device, you have to stop any already running Qt applications! If e.g. the kontron-demo application is running on the device you can simple do a `killall kontron-demo` command to stop this application.

### Copy files manually to the target

Using the protocols sftp and scp files can be copied to the target. If the target has the IP address `192.168.0.10` the content of the target can be shown by using the URL `sftp://root@192.168.0.10/` in a browser window.

To copy files via command line the program scp can be used.
You can also start a remote shell on the target via ssh:

```
user@ubuntu:~$  ssh root@192.168.0.10
root@192.168.0.10:~
```

# Qt environment

Qt offers several options concerning framebuffer, eglfs, input devices, etc. that can be set via environment variables.
Please use this page as a reference for the available options: Qt for Embedded Linux

## QML software rendering

To be able to use QML/QtQuick applications on SoCs without GPU, or if you don't want to use the proprietary hardware acceleration libraries it is possible to select the included software renderer. Some features such as shaders, etc. won't be available and you might experience performance issues, depending on how your application is designed.

To use the software backend, set `QT_QPA_PLATFORM=linuxfb` and `QT_QUICK_BACKEND=software` in your environment. The full environment for Kontron 5 inch Eval-Kit for example:

```
QT_QPA_PLATFORM=linuxfb:mmsize=208x130
QT_QUICK_BACKEND=software
QTWEBENGINE_DISABLE_SANDBOX=1
```

## QML accelerated rendering

For accelerated rendering of QML/QtQuick applications on GPU these settings are used for Kontron 5 inch Eval-Kit:

```
QT_QPA_PLATFORM=eglfs
QT_QPA_EGLFS_ALWAYS_SET_MODE=1
QT_QPA_EGLFS_INTEGRATION=eglfs_kms
```

```
QT_QPA_EGLFS_PHYSICAL_HEIGHT=130
QT_QPA_EGLFS_PHYSICAL_WIDTH=208
QTWEBENGINE_DISABLE_SANDBOX=1
```

**Important**

The current revision of the OpenGL library doesn't play well with Qt. In most cases using software rendering for QML/QtQuick applications leads to a more responsive UI experience.

# Debbuging

If gdbserver is running on the target (default in Kontrons demo images) you should be able to use the debugging features in QtCreator.

If you want to debug QML based applicaitons ensure, that (QML-)debugging is enabled in the project settings. If you are trying to debug a QML application and you get an error message "Invalid Signal", then try to skip the message by clicking 'OK' and the use the button with the green arrow to continue debugging.

## Further hints for debugging

- Disable optimization
  Debugging an optimized binary might be difficult because the compiler may reorder or optimize away some code. To disable optimization for debugging purposes you can set some QMAKE variables in your project file to disable optimization. But be aware that this might lead to differnt timings for your applicaiton!

```
QMAKE_CXXFLAGS_DEBUG += -O0
QMAKE_CFLAGS_DEBUG += -O0
```

- Rejected loading of shared libraries
  Loading of shared libraries is sometimes rejected by gdb due to 'insecure path settings'. Put these gdb commands into your QtCreator configuration for gdb to disable secure path setting. `set auto-load safe-path /` You

can configure additional start commands for gdb thru `Tools -> Options -> Debugger -> GDB`.

- Using gdb-multiarch
  Qt Creator uses so called *pretty printers* to provide a easy interface to basic Qt classes like e.g. Qt strings. These pretty printers are implemented with the help of python functions in gdb. The cross toolchain for the devices may lack some python libraries Qt creator needs for its pretty printers. To circumvent this use the `gdb-mulitarch` debugger of your development machine for remote debugging. You can install `gdb-multiarch` on your development host by running `sudo apt install gdb-multiarch`.

## Creating a Yocto recipe for a Qt application

To build your Qt application with bitbake and include it in a Yocto image, you have to create a recipe similar to the following example. Create it as `your-app_version.bb` in your custom layer `meta-customer/recipes-customer/your-app/`. The example fetches the sources from a GIT repository. Instead of that, the sources can also be fetched from local files in the recipe directory, from a tarball or a SVN server. See the Yocto-Dev-Manual for more information on how to write your own recipe.

```
SUMMARY = "The Qt5 Demo QML Application for the Kontron DevKit"
HOMEPAGE = ""
LICENSE = "GPLv2"
LIC_FILES_CHKSUM = "file://${COREBASE}/meta/
COPYING.GPLv2;md5=751419260aa954499f7abaabaa882bbe"

# Use qmake
inherit qmake5

OE_QMAKE_PATH_HEADERS = "${OE_QMAKE_PATH_QT_HEADERS}"

# Build Dependencies
DEPENDS += "qtbase qtdeclarative qtmultimedia"

# Runtime Dependencies for the kontron-demo package
RDEPENDS_${PN} += "qtmultimedia-qmlplugins qtmultimedia-plugins \
           gst-plugins-base-meta gst-fsl-plugin \
           gst-plugins-good-meta gst-plugins-bad-meta \
```

```
           qtdeclarative-qmlplugins qtbase-plugins qtbase-fonts"

# Specify the path to the source files
S = "${WORKDIR}/git"

FILESEXTRAPATHS_prepend := "${THISDIR}/files:"

# Fetch the latest version from the master branch of the GIT repo
SRCBRANCH = "master"
SRCREV = "${AUTOREV}"

# Where to fetch the sources from
SRC_URI = "git://${EXCEET_GIT_APPS}/exceet-
demo.git;protocol=ssh;branch=${SRCBRANCH}"
SRC_URI_http = "git://${EXCEET_GIT_APPS}/exceet-
demo.git;protocol=http;user=exceet-user:user;branch=${SRCBRANCH}"

# Which files to include in the kontron-demo package
FILES_${PN} = "/opt/exceet_demo_qml/* \
           /usr/src/debug/exceet-demo/* \
           /usr/bin/exceet_demo_qml \
        "

# Which files to include in the exceet-demo-dbg package
FILES_${PN}-dbg = "/opt/exceet_demo_qml/.debug* \
          "
```

# Modify the BSP

## Local or temporary modifications

On the first initialization a `local.conf` file is created in the `conf` directory of your build. This file is usually not tracked by git and is meant to be used only for local or temporary changes. Please check `local.conf` and read the comments in the file to find out about some default options.

The default `local.conf` file includes the `sourcecode-version.conf` and `user.conf` file if available. `sourcecode-version.conf` is meant to contain the sourcecode version number for all sourcecode compiled (can be used as software release number). It should be kept in sync with `repo.conf` .

Furthermore the `user.conf` file is meant to hold user specific settings that may be different between different developers. One such example is the URL variable for the package server (PACKAGE_FEED_URIS).

> **Warning**
>
> If you have changes in your `local.conf` that should not stay local, but need to be set as default for everyone who uses the build, then find a way to move these changes to the correct file. Some popular places in the meta layers are:
>
> 1. The image recipe in `recipes-core` for image-specific settings
> 2. The distro config in `conf/distro` for distro-specific settings
> 3. The machine config in `conf/machine` for machine-specific settings
> 4. Recipe of some package for package-specific settings

## Create your own layers

If you want to make modifications to the BSP, we suggest to create your own layers to keep your modifications reproducible and to separate them from other layers. For customer boards Kontron uses a `build-<CUSTOMER>` and a `meta-<CUSTOMER>` layer to keep the modifications for special customer boards.

The `build-<CUSTOMER>` layer describes which layers in which version are required to build the product. This is exactly the same as the *build-stm32mp* layer does for Kontron boards. So the *build-stm32mp* layer is a good blueprint for your own build layer.

The `meta-<CUSTOMER>` layer holds all adaptions for this board. This can be additional recipes or adaptions to some recipes, separate images, configurations and so on.

See also the Yocto documentation for creating layers.

## Device tree concept

For Kontron boards the device tree files are generated by different inclues. One file for:

- the SoM layer,
- the board layer and
- the housing layer

These includes are combined in the device tree file of the machine. For special cases some include files are obsolete (e.g. hosing include files) or are combined into the final device tree file for the machine. This structure is roughly used for all software components which use the device tree (tf-a, u-boot, linux kernel), except where they do not make sense (e.g. housing configuration for tf-a).

For example the device tree file `stm32mp-t1000-s-50.dts` for Kontron Eval-Kits consists of these device tree components:

| dts file | Description | Includes |
| --- | --- | --- |
| stm32mp-t1000-s-50.dts | Board with display (housing) | stm32mp-housing-50.dtsi *stm32mp-t1000-s.dts* |
| *stm32mp-t1000-s.dts* | Board without display | stm32mp-board-s.dtsi stm32mp-som-t1000.dtsi stm32mp157c-t1000-s-mx.dts |

- `stm32mp157c-t1000-s-mx.dts`
  Device tree file **generated from CubeMX** (maybe generated by user for his own baseboard). This is intended to do the pin multiplexing for the board. Contains all but also incomplete configuration for the board of:
  - clock settings
  - pinmux functions
  - DDR settings

- `stm32mp-som-t1000.dtsi`
  Include file for **SoM** component (fixed configurations for SoM)
  This is intended to fix parts which shouldn't be changed when using a SoM
  - overwirtes DDR settings
  - overwrites pinmux and functions implemented on SoM (e.g. DDR, ethernet, nand, nor, ...)
  - overwrites base clock domain settings (done in tf-a)

- `stm32mp-board-s.dtsi`
  Include file for **baseboard** (by user for his baseboard)
  - complete the device tree configuration for the board by adding nodes and properties

- `stm32mp-housing-50.dtsi`
  Include file for **housing** (optionally, by user for his board)
  This is intended to do settings which aren't parts of the baseboard so that these parts can be changed without changing the baseboard (e.g. display timings or touch controller settings)

The include file for the SoM component in tf-a (and u-boot) also defines default clock settings and default system partitioning settings (ETZPC, all set to DECPROT_NS_RW, DECPROT_UNLOCK). They overwrite the settings from the CubeMX device tree file. To prevent this, set `#define RCC_FROM_MX` or `#define ETZPC_FROM_MX` before including the SoM file.

If you want to create your own board, the best way is to start from a already existing CubeMX board configuration and modify it for your board.

## Create your own board

When you want to create your own board, we propose to execute these steps:

- Create your own build- and meta layer, if not already done
- Create your own machine description. Use one of the existing ones as blueprint
- Create an extlinux configuration for your board
- Create the devicetree files for your board

Depending on how far your design differs from Kontron Eval-Kits the creation of device tree files may vary a lot!

If your pin multiplexing and clock setting doesn't differ from the Eval-Kits you can use the device tree files for tf-a from this Eval-Kit. Else you have to start your own with CubeMX and integrate that in your device tree stack.

For u-boot you have to create your configurataion with your own device tree file which contains the name of your board in the compatible string. This string is used from extlinux to search the right board description. So your compatible string has to match the extlinux configuration.

Kontron provides CubeMX configurations and device tree files for all Eval-Kits and also for all bare SoM devices. The difference between these configuration is, that the Eval-Kits own a rich pin multiplexing. It is a good starting point if you only want to do small configuration changes.

In contrast, the configurations of SoM devices only contain configurations for the bare minimum peripherals. Normally this is limited to all recommended signals: serial console, SD card interface, USB host and OTG interface and LED on PA13. Also the not changeable interfaces are included (like NOR, NAND, Ethernet or GPIOs on some SoMs). All other pins are left in default configuration.

This way the configuration of the SoM device should boot on all baseboards with this SoM and the recommended signals implemented.

> **Important**
>
> Alhtough it is possible to change the pin multiplexing for SD card (SDMMC1), LED1 and serial console (UART4), **it is highly recommended to leave these settings untouched** for your board! When you change these recommended settings, you will have to modify various software parts (in case of console) or will loose the ability to boot from SD card with default otp settings. LED1 is also used by BOOTROM for signalling USB downloader state.
>
> For this reason, these recommended pins are already configured in the bare minimum CubeMX configurations for your SoM.

# Modifying the kernel configuration

The kernel configuration can be modified using the following steps:

## Make modifications and save them as config fragment

```
bitbake virtual/kernel -c menuconfig
bitbake virtual/kernel -c diffconfig
```

Adapt the kernel recipe file with the config fragment as needed.

### Rebuild the kernel for testing

```
bitbake virtual/kernel -C compile -f
```

Test your changes

### Create reduced defconfig

```
bitbake virtual/kernel -c savedefconfig
```

### Rebuild kernel

```
bitbake virtual/kernel
```

More informations can be found in STMicroelectronics Wiki

## Recompile kernel device tree

Sometimes it is required to change the device tree configuration of your setup and test it on your board. A possibility to do this without the complete Yocto bitbake process is to use the bitbake `devshell` command.

A simlpe workflow could be like this:

A prerequisite is that the linux kernel is already compiled. Then you can go to the source directory (maybe the workspace of a devtool workflow) and change your device tree files. Then start the bitbake devshell:

```
bitbake virtual/kernel -c devshell
```

Now source `recipe.env` which contains the environment variables where the source code and the build output for the linux kernel is located. Then build your device tree

```
> source recipe.env
Output directory O=/home/user/[...]/build-stm32mp/tmp/work/
```

```
stm32mp_t1000_s_multi-ktn-linux-gnueabi/linux-stm32mp/4.19-r0/
linux-stm32mp-t1000-s-multi-standard-build
Kernel source directory KDIR=/home/user/[...]/build-stm32mp/tmp/
work/stm32mp_t1000_s_multi-ktn-linux-gnueabi/linux-stm32mp/4.19-r0/
git
Makeflags MF=-j6 ARCH=arm O=/home/user/[...]/build-stm32mp/tmp/
work/stm32mp_t1000_s_multi-ktn-linux-gnueabi/linux-stm32mp/4.19-r0/
linux-stm32mp-t1000-s-multi-standard-build CROSS_COMPILE=arm-ktn-
linux-gnueabi- LOADADDR=0xC2000040
CROSS_COMPILE=arm-ktn-linux-gnueabi-

> makey stm32mp-t1000-s.dtb
[...]

> ls -l $O/arch/arm/boot/dts
-rw-r--r-- 1 1000 1000 71560 Nov 14 16:26 stm32mp-t1000-s.dtb
```

**Info**

The 'makey' command is an alias and calls make with the appropriate directory parameters

## Modifying the u-boot configuration

The u-boot configuration is currently split into two concepts: In the past u-boot configuration was only done by setting configuration variables in header files for your board. Nowadays u-boot also contains a menuconfig interface to configure many aspects of u-boot functionality. Unfortunately not all configuration can be done with menuconfig.

Yocto doesn't provide the menuconfig way as easy as it is provided for the linux kernel. So a different workflow has to be used:

Fist start the devshell for the bootloader

```
> bitbake virtual/bootloader -c devshell
```

Then try to source the `recipe.env` file:

```
> source recipe.env
ERROR: Set DEFCONFIG in your environment and source again!
Currently available configs:
```

```
    export DEFCONFIG=stm32mp15_basic_defconfig
    export DEFCONFIG=stm32mp1-t1000_defconfig
    export DEFCONFIG=stm32mp1-t1000-evk50_defconfig
    export DEFCONFIG=stm32mp1-t1000-evk50sdcard_defconfig
    export DEFCONFIG=stm32mp1-t1000-evkmanualtest_defconfig
    export DEFCONFIG=stm32mp1-t1000-manualtestevk_defconfig
    export DEFCONFIG=stm32mp1-t1000-sdcard_defconfig
bash: unalias: makey: not found
```

When you have multiple u-boot configurations the soucing fails and you have to select which u-boot configuration you want to change. Set the `DEFCONIG` variable to the appropriate value, source again and start the configuration GUI:

```
> export DEFCONFIG=stm32mp1-t1000_defconfig
> source recipe.env
Output directory O=/home/user/[...]/build-stm32mp/tmp/work/
stm32mp_t1000_s_multi-ktn-linux-gnueabi/u-boot-stm32mp/2018.11-r0/
build/stm32mp1-t1000_defconfig
Makeflags MF=-j6 ARCH=arm CROSS_COMPILE=arm-ktn-linux-gnueabi- O=/
home/user/[...]build-stm32mp/tmp/work/stm32mp_t1000_s_multi-ktn-
linux-gnueabi/u-boot-stm32mp/2018.11-r0/build/stm32mp1-
t1000_defconfig
CROSS_COMPILE=arm-ktn-linux-gnueabi-
> makey menuconfig
```

After the configuration is finished, create a `defconfig` file:

```
> makey savedefconfig
make[1]: Entering directory '/home/user/[...]/build-stm32mp/tmp/
work/stm32mp_t1000_s_multi-ktn-linux-gnueabi/u-boot-stm32mp/
2018.11-r0/build/stm32mp1-t1000_defconfig'
  HOSTCC  scripts/kconfig/conf.o
  HOSTLD  scripts/kconfig/conf
scripts/kconfig/conf  --savedefconfig=defconfig Kconfig
make[1]: Leaving directory '/home/user/[...]/build-stm32mp/tmp/
work/stm32mp_t1000_s_multi-ktn-linux-gnueabi/u-boot-stm32mp/
2018.11-r0/build/stm32mp1-t1000_defconfig'

> ls -l $O/defconfig
-rw-rw-r-- 1 root root 2631 Nov 14 16:44 /home/user/[...]/build-
stm32mp/tmp/work/stm32mp_t1000_s_multi-ktn-linux-gnueabi/u-boot-
stm32mp/2018.11-r0/build/stm32mp1-t1000_defconfig/defconfig
```

This file now has to be integrated in the u-boot-stm32mp recipe of Yocto.

## Using devtool to work on source code

To work on the source code of any package it is most convenient to use the `devtool` utility. As an example we will show how to modify the kernel code.

> **Tip**
>
> For more information about the mighty `devtool`, please visit the Yocto Manual.

To start working on the `linux-stm32mp` code:

```
devtool modify linux-stm32mp
```

A separate workspace layer will be created and the kernel source tree will be extracted there. Do your code changes and run a build with:

```
bitbake linux-stm32mp
```

Test your changes and create patches if necessary. To reset to the previous state and build without the changes in the workspace run:

```
devtool reset linux-stm32mp
```

# Sample Projects

This chapter provides a short description on how to create and debug applications on the M4 coprocessor of the STM32MP1 with the help of two example applications.
For a more general and detailed description of the M4 functionality please refer to the STM32 MPU wiki by ST .

## Prerequisites

For M4 debugging you need an appropriate JTAG debugger. What we recommend is

- ST-Link/V2 debugger and
- Olimex LTD ARM-JTAG-20-10 connector

See the board documentation for the location of the debug plug. For Kontron Eval-Kits the outline of the debug plug can be found here.

Before continuing make sure that:

- A running linux image is present on your device
- The SystemWorkbench IDE for STM32 with STM32-CoPro-MPU plugin is installed

## Debug modes of the M4 coprocessor

The M4 firmware can be run and debugged in two different modes:

- Production mode: M4 firmware is loaded from network and started with the remoteproc service running on the A7.
- Engineering mode: M4 firmware is loaded and started from the debugger similar to normal STM32 procedure. In this mode the A7 is not running.

> **Info**
>
> Due to a missing boot switch the engineering mode is currently not available on our Kontron Eval-Kits. Thus, following description is for the production mode.

More detailed information on the boot modes can be found in the ST wiki.

## Debugging the M4 coprocessor

Step by step guide on how to debug the example application on the M4 in production mode and the Kontron Eval-Kit.

- Connect the debug console to the host (via USB interface)
- Connect the Eval-Kit to the network and figure out the ip address with `ifconfig`
- Connect the ST-Link with Olimex connector to the board and the pc
- Open the System WorkBench
- Open the example project "m4_ktn-simple-monitor" which can be downloaded from the following repository `git@git.kontron-electronics.de:stm32mp/m4_ktn-simple-monitor.git`
- Rebuild the project
- Right click on the project and go to `Debug as..`
  double click `ST's STM32 MPU Debugging`
  go to the `Startup` tab and input the ip address of the board
- Set a breakpoint at the start of `main()`
- Press `Debug` and input `root` as username and no password
- After confirming the log in data a few times, you should end up in the Debug view similar to normal STM32 debugging.

The next section describes the capabilities of this application and how to use it.

# Description of the "m4_ktn-simple-monitor" example application

This example is based on the `OpenAMP_TTY_echo` example application from ST which can be found here.
Similar to the example application provided by ST it will create two virtual uart channels between the A7 and the M4 cores based on the Linux frameworks RPMsg , VirtIO] and IPCC and the OpenAmp framework on the M4 site.

As the name implies the application offers a simple monitor, which allows to control the LED1 on the evaluation board from the linux console. This simple monitor operates on virtual uart channel 0 while virtual uart channel 1 echos all incoming messages back.

If the application is up and running (default state with the evaluation board image) you can simply write a command on the virtual uart channel 0 (/dev/ttyRPMSG0) by using `rpmsg0` . For example `rpmsg0 "help"` which will show all available commands to control the LED on the evaluation board. Use `rpmsg1` for messages on the second channel, which will only echoed back.

# Creating a new project

New projects might be created with the System Workbench or by taking the ST template from https://github.com/STMicroelectronics/STM32CubeMP1.

This repository also includes all HAL_Drivers and lots of examples which can be used for building your application.

However, these example applications are designed for the evaluation boards of ST and have complex include paths structures which are not very friendly for extracting project parts like the OpenAMP framework.

Thus, it is less stressful to use the sample project "m4_ktn-eval-usart6" as starting point as it includes the rsc_table, basic Drivers and system libraries as well as all necessary header and source files. Or the "m4_ktn-simple-monitor" project if you want to use the OpenAMP framework.

## Assigning peripheral devices to the M4 coprocessor

To use a peripheral device with the M4 coprocessor it has to be assigned to it in the device tree prior to the linux boot. This can be accomplished by creating your own board device tree. For the Kontron Eval-Kits this is done in the file `stm32mp-board-s.dtsi` in the directory `layers/meta-ktn-stm32mp/recipes-kernel/linux/linux-stm32mp`.

In the following the usart6 peripheral is taken as an example. On the Kontron Eval-Kit the usart6 is converted into a RS232 signal on connector x16 (pin 1 Tx and pin 3 Rx). For other boards see the documentation.

First of all, the respective node has to be disabled for A7 cores with linux. This can be accomplished by adding:

```
&usart6{
    status = "disabled";
};
```

Next we enable the peripheral for the M4:

```
&m4_usart6{
    status = "okay";
};
```

To rebuild the device tree, either build the respective image or only the linux kernel image with `bitbake linux-stm32mp`.

Afterwards replace the old device tree in the `borootfs` and boot the board. To check if the resource is really disabled for the A7 read the node status with `cat /proc/device-tree/soc/serial\@44003000/status`. Additionally, check if the M4 resource was enabled successfully with `cat /proc/device-tree/m4\@0/m4_system_resources/serial\@44003000/status`.

A simple example for M4, which initializes the usart6 and continuously sends out a string, can be downloaded from `git@git.kontron-electronics.de:stm32mp/m4_ktn-simple-monitor.git`.

This example can be debugged similar to the `m4_ktn-simple-monitor` example described above.

Additionaly, for an easy and graphical configuration of a new peripheral the CubeMX tool can be used to generate the respective System WorkBench project. As a starting point the CubeMX project with the initial evaluation board configuration can be used, which can be found in the `meta-ktn-stm32mp` layer at `conf/machine/cubemx` . Here, it is again simplier to take the initialization of the respective peripheral from the project created by CubeMX and use it in an already working project.

More detailed informations on assigning peripheral devices can be found in the ST wiki.

In this application also the resource manager is implemented to check for conflicts in peripheral assignment.

## Starting Software with system service

A system service called `m4_fw-autoload.sh` is already included in the linux image for the Kontron Eval-Kits. It autoloads the above mentioned example `m4_ktn-simple-monitor` on system boot. The service can be controlled with the commands `start` and `stop` . For example this stops the M4 processor:

```
/etc/init.d/m4_fw-autoload.sh stop
```

By changing the `APPLICATION_NAME` in `/etc/default/m4_fw-autoload.conf` to `m4_ktn-eval-usart6` this example will be used by the service.

If you want to include your own binary file into the service you need to copy it to the directory `recipes-extended/m4-fw-autoload/files/binary` in your own layer and create a `m4-fw-autoload.bbappend` file and a adapted `m4_fw-autload.conf` file. This should lead to the binary being copied into the image on build and started from the system service.

## Starting software manually

In case the system service is not used, the M4 firmware can be started manually with the following principle

- Load the binary file into `/lib/firmware`
- `echo -n "firmware_name" > /sys/class/remoteproc/remoteproc0/firmware` where `firmware_name` is the name of your firmware binary file
- `echo -n start > /sys/class/remoteproc/remoteproc0/state`
- read state of M4 via `cat /sys/class/remoteproc/remoteproc0/state`
- stop the M4 via `echo -n stop > /sys/class/remoteproc/remoteproc0/state`

Alternatively you can also use the `fw_cortex_m4.sh` script present in the Remoteproc folder of the application.

## Getting logs from M4

For all of the demo projects a trace buffer is added to the resource table. This enables reading out the log buffer of M4 in the linux context:

```
root@stm32mp-t1000-s-multi:~# cat /sys/kernel/debug/remoteproc/
remoteproc0/trace0
[00000.000][INFO ]Cortex-M4 boot successful with STM32Cube FW
version: v1.0.0
[00000.008][INFO ]Virtual UART0 OpenAMP-rpmsg channel creation
[00000.009][INFO ]Virtual UART1 OpenAMP-rpmsg channel creation
```

# Tools and demos

## STMicroelectronics Tools

STMicroelectronics provides some tools to ease development for their STM32MP products. The Kontron BSP is built to work with these tools.

- STM32MPCubeProgrammer
  is the tool to flash images on virgin devices.
- STM32CubeMX
  for doing the pin multiplexing and for generating the linux and bootloader device tree files and M4 IDE project with HAL libraries.
- SystemWorkbench for STM32 with STM32-CoPro-MPU plugin
  as IDE for M4 development and debugging.

These tools are already installed in the Kontron VmWare image for a quick start of development.

## Kontron Tools and Demos

The Kontron BSPs include or provide several demo applications and tools, which can be helpful to get started and to configure your hardware.

### Prebuild BSP releases

You can find prebuild BSP releases for your board in https://files.kontron-electronics.de/stm32mp. There you can find images, sdk, open source sourcecode and license texts for your board. Check this location for new pre-built software for your Eval-Kit.

### Kontron VMware Image

Kontron provides a VMware Image with preinstalled tools and access to the sources located on the Kontron GitLab server. You can access the newest

version of this VMware image under https://files.kontron-electronics.de/stm32mp. The image is based on a current Ubuntu LTS release.

For using the VMware image you should have

- installed VMware Player Version 10 or higher,
- at least 12GB of RAM on your PC
- at least 80GB free space on hard disk

This VMware provides all tools required for developing software with STM32MP1:

- **Yocto build environment for linux image**
  Yocto is the environment to build your customized linux image for your board. The VMware contains the Kontron Yocto environment for STM32MP1 devices.
- **STM32 CubeMX**
  for clock tree setup and pin multiplexing. This is required if you need your own pin multiplexing or want to tweak clock settings of your board. It creates the basic device tree files for the bootloader and linux. It also can generate a code skeleton for M4 development.
- **STM32 CubeProgrammer**
  to equip a totally blank device with software or to burn OTP fuses in the device.
- **QtCreator**
  Development environment for Qt5 programming.
- **SW4STM32 IDE**
  Development environment for programming M4.

**Issues with VMware STM32MP 1.3.1 r1**

- On this VMware image the tftp and nfs directories do not exist. See Setup tftp and nfs on the host for how to set it up.

## Tools

### create-sd-card.sh

Kontron provides the 'create-sd-card.sh' script to generate bootable sd card images for your board from your built images. For instructions how to create a bootable sd card see documentation here.

To create the sd card you have to provide:

- the images to burn (normally located in your images directory)
- a layout configuration file which descirbes your partitions, contents and sizes

The sdcard image file is located alongside your image contents in your images directory and called like your layout file with the ending of '.sdcard'.

This script requires **root rights** to generate the sdcard image for mounting and populating the images with the correct access rights. So use *sudo* to run this script.

Layout configuration files for the Kontron EvalKits can be found in the /script-mp directory after an image was built. Additionally a file called 'sd-card.layout' is placed there with all available settings and comments how to use. Other preconfigured layout files exist for different configurations of machine, distro and image in the `meta-ktn-stm32mp/recipes-ktn/production-tool/mptool/common` directory. These configurations are a good starting point when you want to create a layout for your board.

Example for the layout file of EVK STM32MP157:

```
# image directory set by recipe
FW_DIR="./.."

TFA_IMG="tf-a-stm32mp-t1000-s-trusted.stm32"
UBOOT_IMG="u-boot-stm32mp-t1000-s-sdcard.stm32"
UBOOT_IMG_PROD="u-boot-stm32mp-t1000-s-trusted.stm32"

# borootfs (boot + rootfs)
BOROOTFS_IMG="image-ktn-qt-stm32mp-t1000-s-multi.tar.gz"
BOROOTFS_SIZE="500"
```

```
# userfs
USERFS_IMG="image-ktn-qt-userfs-stm32mp-t1000-s-multi-thud.tar.gz"
USERFS_SIZE="50"
```

The script supports the partitions *borootfs*, *userfs* and optionally *rescuefs*. In normal configuration it also places the tar.gz archives of these filesystems and the bootloader on the sd card. This way you can use this sd card to flash your internal storage with your firmware and *mptool*.

The most important command line options are:

| Option | Config file setting | Description |
|---|---|---|
| -p y\|n | PROD_IMG | Expand userfs and copy tar files to it (default y) |
| -c | - | Use alternate config file (default ist /etc/mptool.config) |
| -u | FW_DIR | Set base directory for update files (default is FW_DIR) |

Command line options always have precedence of config file settings

> **Hint**
>
> If you want to create a bootable SD card to update the internal flashes with `mptool` and new software, don't forget the `-p y` option on the command line or the `PROD_IMG=y` option in the config file. With this all firmware image files are stored in the SD card image under `/usr/local` so that mptool can use them directly.

### mptool

mptool is a set of scripts used to execute tasks for production purposes, such as flashing firmware to memory, setting the mac address, programming OTP locations or running tests for usage on the device.

To view a list of available tasks, run:

```
mptool -h
```

To run a specific task:

```
mptool <task_name>
```

Settings for partition sizes, image names and flash memory (NAND / EMMC) can be configured in the mptool.config. This config file can be found either in the `meta-ktn-stm32mp` layer in the directory `recipes-ktn/production-tool/mptool/<MACHINE>` or on the running linux image in the `/etc` directory.

The images used to flash or update the memory needs to be in the `/usr/local` directory. For the bootfs, rootfs and userfs the image needs to be in `tar.gz` format while for the tf-a and u-boot image a `.stm32` image is needed.

To partition the memory device defined in the config file and afterwards flash the respective images:

```
mptool flash-full-fs
```

All partitions can be seperately updated with e.g `update-rootfs`. Furthermore, with the option `-f=imagename` it is possible to use a different image than the one specified in the config file.

The MAC address can be written into the respective OTP register with 'set-lan-mac'.

### System service autostart-eglfs

The system service `autostart-eglfs` is used to start an Qt application on bootup with all desired environment variables set.

This service has a configuration file `/etc/default/autostart-eglfs`. The configuration is done by setting appropriate environment variables:

- **APPLICATION**
  absolute path to Qt application
- **APPLICATION_OPTIONS**
  command line parameters for application

- **QT_QPA_PLATFORM** linuxfb|eglfs
  configuration settings wheter hardware acceleration(eglfs) or software rendering (linuxfb) shall be used.
- **DISPLAY_PHYS_WIDTH**, **DISPLAY_PHYS_HEIGHT**
  display dimensions in milimeters

Further informations can be achieved by examining the companion file `/usr/bin/autostart-eglfs.sh` .

## Demo applications

### C-app-demo

A simple Hello-World application in C. Only available in the VMware image.

### kontron-demo (QML)

A QML-based demo application, featuring a simple touch UI. It contains demos for Qt widgets, multitouch controls, slideshow and QtWebEngine webbrowser. This application is started when the Eval-Kit is powered on.

### imagegestures and animatedtiles (Qt Widgets)

Two modified Qt examples to show performance of Qt-Widgets-based applications (also usable without GPU).

### Web viewer with virtual keyboard

The package `webengine-vk` contains a simple web browser with integrated onscreen touch keyboard. You can load a specific website or rotate the screen by running the application by launching the application manually.

When using `autostart-eglfs` set your parameters in `/etc/default/autostart-eglfs`

```
## Simple webbrowser based on Qt
APPLICATION=/opt/webengine-vk/webengine-vk
APPLICATION_OPTIONS=rot 180 http://www.kontron-electronics.de
```

and start the application by typing:

```
/etc/init.d/autostart-eglfs-initscript.sh start
```

**Hint**

- Please note that the web viewer is only capable to display a single page at the same time. No tab browsing is possible.
- This application is not included in the standard image for Kontron Eval-Kits. It has to be compiled with Yocto.

# Linux tools

## screen

Screen is a terminal multiplexer which is also very handy to open serial terminals. To use screen for a serial connecton on `/dev/ttyUSB0` with 115200 baud call it with following parameters:

```
screen /dev/ttyUSB0 115200
```

To exit this screen type `Ctrl-a k`. To scroll back Type `Ctrl-a ESC` and use your cursor keys to scroll back. Stop the scrollback mode with pressing `ESC` again.

There are various cheat sheet for screen, for example https://catonmat.net/ftp/screen.cheat.sheet.pdf

# Using the hardware

This page shows you how to access different peripherals and features of the hardware in general. To find out details about the board configuration and peripherals on a specific board, navigate to the hardware description page of that board.

## Reserved OTP ressources

The following locations in OTP are reserved for special usage by Kontron. These locations may be already programmed with appropriate content and can't be used by customers for their needs!

| OTP word | Intended usage |
|---|---|
| 59 | serial number of SoM |
| 60 | serial number of baseboard |
| 61 | reserved |
| 62 | reserved |
| 63 | reserved |
| 64 | reserved |
| 65 | reserved |
| 66 | reserved |

With the help of mptool it is possible to read and write these values.

# SOC and board features

## Sleep modes

The system can be suspended to the suspend-to-ram sleep mode. Please note that the actual state of the board and the devices in sleep mode depends on the hardware and software configuration.

For more information about this topic see STMicroelectronics Wiki Power overwiew

### Suspend to RAM

```
> echo mem > /sys/power/state
```

In this state the CPU and if possible any peripheral devices and buses will be halted. Only the RAM remains powered to keep the state of the system for wakeup.

### Wakeup

To wake up from standby, multiple wakeup sources can be defined. By default only the built in RTC is enabled as wakeup source.

To list all drivers that expose a wakeup property in sysfs, you can run:

```
> find /sys -name wakeup
[...]
/sys/kernel/irq/65/wakeup
/sys/kernel/irq/27/wakeup
/sys/kernel/debug/tracing/events/ftrace/wakeup
/sys/devices/platform/soc/5c004000.rtc/power/wakeup
/sys/devices/platform/soc/49000000.usb-otg/usb1/power/wakeup
/sys/devices/platform/soc/4000e000.serial/power/wakeup
/sys/devices/platform/soc/4000e000.serial/tty/ttySTM1/power/wakeup
/sys/devices/platform/soc/40013000.i2c/power/wakeup
/sys/devices/platform/soc/5c002000.i2c/power/wakeup
/sys/devices/platform/soc/5800d000.usbh-ehci/usb2/2-1/power/wakeup
/sys/devices/platform/soc/5800d000.usbh-ehci/usb2/2-1/2-1.1/power/
wakeup
/sys/devices/platform/soc/5800d000.usbh-ehci/usb2/power/wakeup
/sys/devices/platform/soc/5800a000.ethernet/power/wakeup
/sys/devices/platform/soc/40010000.serial/power/wakeup
```

```
/sys/devices/platform/soc/40010000.serial/tty/ttySTM0/power/wakeup
/sys/devices/platform/soc/5800c000.usbh-ohci/usb3/power/wakeup
/sys/devices/platform/soc/4c001000.mailbox/power/wakeup
```

Example:
To wake up the system after 20 seconds using the internal RTC:

```
> echo +20 > /sys/class/rtc/rtc0/wakealarm; echo mem > /sys/power/
state
```

### Thermal management

The STM32MP1 contains a temperature sensor to measure the CPU temperature.

Measure the CPU temperature in millis of °C:

```
> cat /sys/class/thermal/thermal_zone0/temp
59000
```

### CPU core management

Some devices out of the STM32MP1 series of SOC have more than one A7 CPU core where linux runs on. It is possible to disable and enable CPU cores while linux is running. Normally all CPU cores are activated.

Disable the second CPU core:

```
echo 0 > /sys/devices/system/cpu/cpu1/online
```

## Serial Interfaces

The following section provides general information on the serial interfaces found on Kontron hardware. For specific information on a certain board, please see the according hardware description.

## Debug

There's usually one UART, that is used as a debug and control interface to connect with terminal application on a PC via a Micro-USB header.

Depending on the hardware, the board contains an FTDI-USB-Chip or requires an external USB-Adapter to translate the UART-Signals to USB. To connect with the debug interface you can use terminal applications like TeraTerm (Windows), GtkTerm (Linux with GUI), screen (Linux shell) or the like. Set your terminal to 115200 baud 8N1.

UART4 is fixed for debug console usage!

## RS232

Some hardware also feature one or more RS232 interfaces.
Please note, that there might be **no handshake lines** available on the RS232 connector.

Normally linux treats the RS232 (and RS485) interfaces as tty devices. For historical reasons in linux tty devices are complex devices with e.g. line editing and character translation features (you can find a good overview on this on https://www.linusakesson.net/programming/tty).

There is the `stty` command line tool to control these features. To set the serial interface to a transparent mode which doesn't modify and interpret the data stream, you have to set particular options on the serial interface.

Set the serial device ttySTM2 into transparent mode at 115200 baud:

```
> stty -F /dev/ttySTM2 raw -echo -echoe -echok 115200
```

After this you can realize a simple input and output mirroring on this serial interface:

```
> cat < /dev/ttySTM2 > /dev/ttySTM2
```

### RS485

If your hardware features a RS485 port, it is usually initialized by the driver and can be used in your application right away, just the same as you would use any other serial interface. The DE-Signal for the RS485-Transceiver is handled by the driver.

> **Important**
>
> RS485 is a bus and supports mostly only half duplex data transfer. This means only one parcitipant on the bus can send data at the same time! So when the master sends data on the bus the slave device must wait until all data from the slave is sent. Only then the slave is allowed to send its response.

### USB

#### USB host

The USB host ports can be used to attach common USB devices like thumbdrives, keyboards, etc. Please note, that some device drivers might need to be enabled in the kernel config first, before those devices work correctly. Using USB storage devices is also possible in the bootloader.

#### USB OTG

A USB OTG port can be used as an additional host interface, or as USB device. If the hardware supports it, it can be switched to OTG mode to provide host or device functionality depending on the connected device.
The mode for the OTG port is defined in the devicetree.

## Network interfaces

### Ethernet

All STM32MP1 SOCs have an internal ethernet controller. On most SoMs this port is connected to an external PHY which is located on the SoM. SoMs without integrated PHY provide a RMII interface. See your hardware

description for details. The SOC internal interface is available on eth0 by default and it can also be used in the bootloader (e.g. for network booting). Thruput measurements can be done with the `iperf3` tool.

On the server side (e.g. PC)

```
> iperf3 -s
```

On the device side e.g.

```
> iperf3 -c 10.255.255.1
```

## CAN bus

If your hardware features a CAN port and it is correctly setup in the devicetree, you can use it in Linux by installing the packages and `can-utils`.
If the Packages are installed you can set up the interface by using the command:

```
ip link set can0 up type can bitrate 1000000 berr-reporting on
```

This configuration shows a can interface with 1MBit/s speed. The device can also use CAN-FD. For this the configuration of the interface changes a little bit (using 4MBit/s in fd mode):

```
ip link set can0 up type can bitrate 1000000 dbitrate 4000000 fd on
```

Afterwars you'll see the interface `can0` listed when calling `ifconfig` without parameter.

Sending and receiving from the interface can be done with the commands `cansend` and `candump`. To send a can telegram on the previously configured bus

```
cansend can0 -i 0x01 0x00 0x10 0x20 0x30 0x40 0x50 0x60 0x70
```

To receive from the can interface:

```
candump can0
```

To get statistics:

```
ip -det -statistics link show can0
```

> **Hint**
>
> There are two packages of can tools available: `canutils` and `can-utils`. The commands above are for the package `canutils`. The syntax for tools from `can-utils` package may vary.

> **Hint**
>
> Because `can-utils` seems to be the more modern package, it might be used in newer releases.

## Display interfaces

In general there are two possible interfaces to connect a display to a Kontron STM32MP1 SOMs: RGB and DSI. Only one interface can be used once a time.

Depending on the kernel and bootloader settings, the display interfaces are mapped to framebuffer devices ( `/dev/fb0` )

To test the display without running an application you can first enable it:

```
> echo 0 > /sys/class/graphics/fb0/blank
```

and then write some random data to the display to get colored pixels:

```
> dd if=/dev/urandom of=/dev/fb0
```

The display interfaces are configured in the devicetree for linux.

### Backlight brightness

To change the screen backlight the backlight file in sysfs has to be modified. Dependend on your hardware this could be `/sys/class/backlight/backlight/brightness` . Values from zero (maximal brightness) to seven (dark) are valid. In the following example the backlight will be first set to dark and afterwards changed to maximal brightness.

```
> echo 7 > /sys/class/backlight/backlight/brightness
# dark
> echo 0 > /sys/class/backlight/backlight/brightness
# maximal brightness
```

It should be mentioned, that the settings need to be saved as ASCII string into the file. If the file will not be closed after writing the values, the separate strings of the set brightness have to end with a newline ( `\n` ).

## Touch devices

Kontron offers different display solutions with different touch panels. Depending on the setup, there's usually a touch controller connected to the SOC via $I^2C$ or USB.

The `evtest` program can be used to test wheter the touch device works. When you touch on your device `evtest` will print out the event information.

## Storage

### NAND flash

The NAND flash chip usually contains the filesystems including devicetrees and kernel image. Dependent on your configuration the filesystems 'bootfs', 'rootfs' and 'userfs' are available.

### NOR flash

The NOR flash chip usually contains the tf-a loader (fsbl) U-Boot bootloader(ssbl) and the U-Boot environment.

### SD card

Most Kontron boards feature an onboard micro SD card slot. The SD card can be used as boot device, or as general file storage. Please note, that depending on what card you use, there are limitations on the write cycles and general durability of the card. For a longer lifespan, especially in applications with heavy SD card usage, we recommend to not use commercial grade, but industrial grade cards.

### eMMC

Some Kontron boards have an onboard eMMC chip instead of or additional to the NAND flash. eMMC can be used the same way as a normal SD card.

## GPIOs

For gpios there are two interfaces in the linux kernel:

- the older sysfs interface or
- the newer libgpiod interface

### Sysfs interface

Before using the GPIOs with sysfs they have to be exported. See your hardware description which GPIO number you have to use.

To export GPIO507, wich is the digital out dout1 on STM32MP157 Eval-Kit:

```
> echo 507 > /sys/class/gpio/export
```

The configuration as input or output or both takes place in the devicetree.
To read an input you can do:

```
echo in > /sys/class/gpio/gpio507/direction
cat /sys/class/gpio/gpio507/value
```

To set an output you can do:

```
echo out > /sys/class/gpio/gpio507/direction
echo 1 > /sys/class/gpio/gpio507/value
0
```

## libgpiod interface

With libgpiod and its tools it is also possible to set and read gpio lines. Only gpios which aren't already claimed by drivers can be used. This is also true for gpios exported via sysfs interface.

The following example shows the usage of GPIO507 (gpiochip8.3), wich is the digital out dout1 on STM32MP157 eval kit:

How to show the configuration of a gpio:

```
> gpioinfo gpiochip8
gpiochip8 - 8 lines:
        line   0:      unnamed       unused   input  active-high
        line   1:      unnamed       unused   input  active-high
        line   2:      unnamed       unused   input  active-high
        line   3:      unnamed       unused   input  active-high
        line   4:      unnamed       unused   input  active-high
        line   5:      unnamed       unused   input  active-high
        line   6:      unnamed       "LED3"  output   active-low
[used]
        line   7:      unnamed      "reset"   input  active-high
[used]
```

Read as input:

```
> gpioget gpiochip8 3
0
```

and set as output:

```
> gpioset gpiochip8 3=1
```

**Hint**

Reading back digital output pin gpiochip8.3 on the STM32MP1 Eval-Kit always reads 0 due to the hardware design, regardless whether it was previously set to a different value. The reason for this is that this gpio is configured as input and reads back the real value on the pin.

For more information see libgpiod git repository

# RTC

The STM32MP1 has an internal RTC for timekeeping. Due to the chip technology this RTC is more power consuming as dedicated RTC chips.

For optimal power consumption an external RTC should be used.

## Using the rtc

The RTC can be accessed via the hwclock command (see Linux documentation for further information).

To perform synchronisations of system time and RTC manually, you can use `hwclock` :

**RTC to system time:**

```
> hwclock --hctosys
```

**system time to RTC:**

```
> hwclock --systohc
```

# Audio

To list all soundcards you can use the following command:

```
> cat /proc/asound/cards
 0 [WM8510         ]: WM8510 - WM8510
                      WM8510
```

To test your soundcard you can use the command `speaker-test` :

```
speaker-test -t sine -f 1000
```

## PWM beeper

The STM32MP157 Eval-Kit features a PWM controlled beeper. It is registered in evdev and can be controlled via the userspace tool `beep` , by sending a `BEL` character to the console ( `echo -e "\a" > /dev/tty0` ), or by using `ioctl("/dev/input/eventX",KIOCSOUND,<tone>)`

# Known issues and limitations

## Software

- Hardware acceleration with Qt disabled due to performance issues with hw acceleration and Qt
- Video playback with demo application disabled

## Hardware

See hardware description for the boards you have.

## Changes

### Changes from BSP version 1.1.0 to 1.3.1

- The init manager changed from systemd to sysinitv
- New distro *ktn* is introduced. The old distro *ktn-eglfs* is deprecated.
- The new images *image-ktn* and *image-ktn-qt* replaces the old stm32* images. They provide almost the same functionality
- A new partition layout with *borootfs* and *userfs* is introduced. The tools *mptool* and *create-sd-card.sh* are adapted to this layout.
- libusb added
- Minimal configuratons for SoM t1000 and t1001 added
- Minor updates of tf-a, u-boot and linux kernel
- linux kernel: Pinctrl strict checking deactivated for linux kernel
- linux kernel: include driver for ralink53xx wireless usb
- tf-a: writ-lock OTP location after writing and reread contents after update

## Changes from BSP version 1.0.0 to 1.1.0

An ongoing effort is to adjust yocto layers and concepts to share with kontron yocto imx products. For some parts this leads to profound changes.

This is a list of the most important changes:

- Some layers, recipes and files renamed from exceet to ktn
- The flash partitioning scheme changed from bootfs/rootfs/userfs (OpenST) to borootfs/userfs
- A new distro `ktn` with sysvinit init system is introduced and will become the default distro in the near future. Distro `ktn-eglfs` with systemd will then become deprecated.
- Only tar files are generated for all Kontron DemoKits. These files are the basis for `mptool` and `create-sd-card.sh` scripts for updating the flash contents by e.g. sd card boot.
- The can package changed from 'can-utils' to 'canutils'

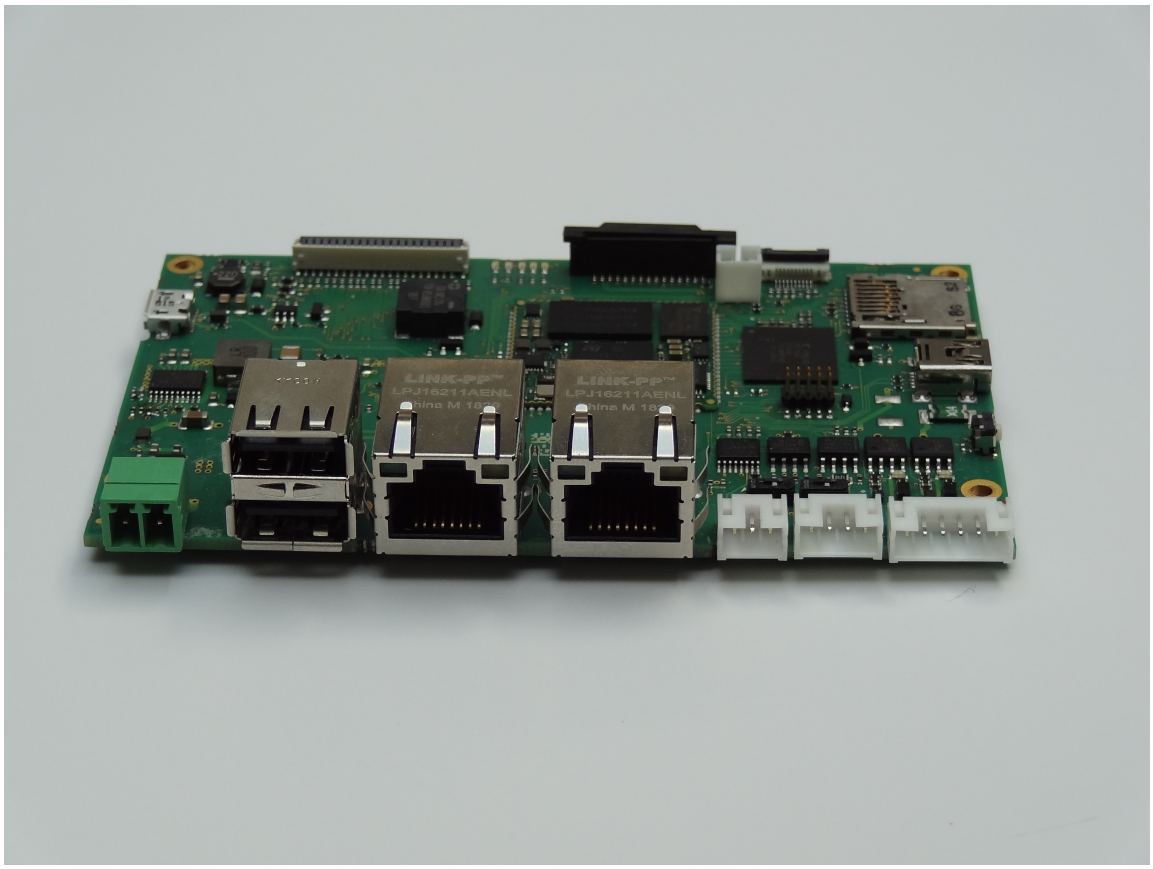
# Hardware index

## Eval-Kit EVK STM32MP157 BL/DL

# Quickstart for EVK STM32MP157 BL/DL Eval-Kits

This quickstart will guide you thru the first steps with your brand new Eval-Kit!

# Kit identification




Eval-Kit EVK STM32MP157 BL (50099 044) without display

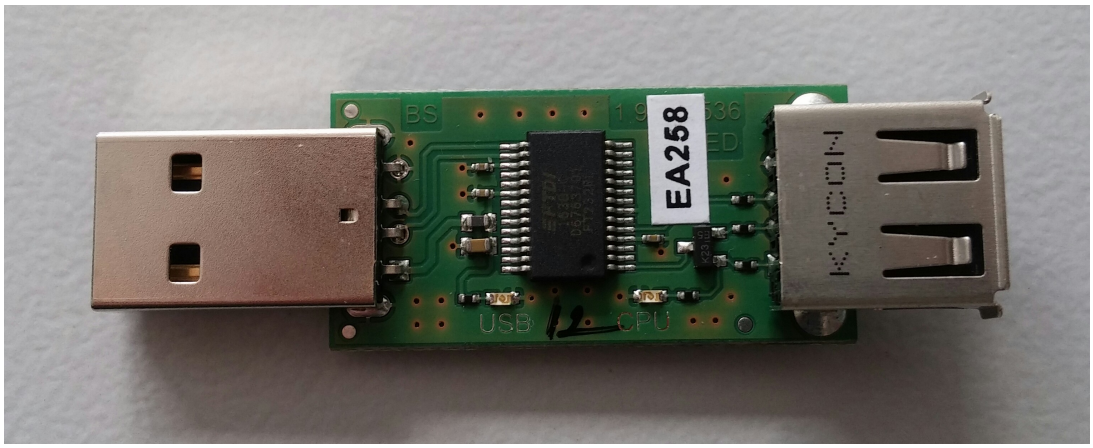
Eval-Kit EVK STM32MP157 DL (50099 045) with 5 inch display and capacitive multitouch. The baseboard is mounted on the backside of the display.

# What contains the Eval-Kit

The Eval-Kit contains all components to start quickly with this board.

- 24V power supply with appropriate power plug

- USB to serial converter for linux console access




- Connectors with cable for 3-pin RS232 rifle, 4-pin RS485/CAN and 6-pin DIO and AIN rifle.
- Mini-USB-B to USB-A cable for connection of USB to serial converter and the debug console connector of the board

These things are not contained in the Eval-Kit, but recommended for further steps

- PC with serial terminal program
- RJ45 ethernet cable
- Network infrastructure with dhcp server

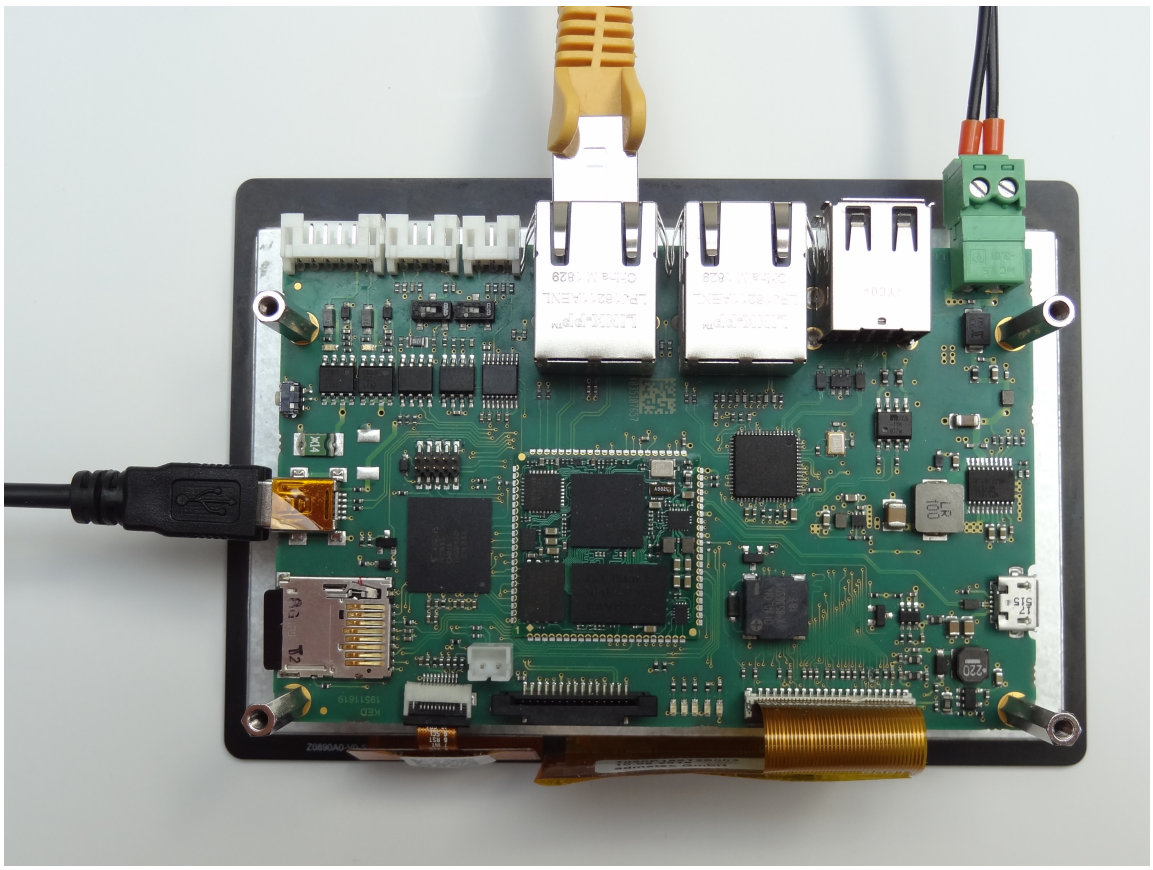
Later, for a full-featured developemt environment this is recommended

- VmWare player or workstation for linux based development installed on a powerful host
- Kontron VmWare image already prepared with all required tools
- ST-Link/V2 debugger with Olimex LTD ARM-JTAG-20-10 connector for M4 development

## Prepare, connect and boot

1. **Prepare your PC: Installation of Tera Term**
   If you don't already have a serial terminal program, it is recommended to install TeraTerm on your PC to get access to the linux serial console of your device. If you already have one other program you can use it instead of Tera Term.
   To install Tera Term download the newest release from its download page https://osdn.net/projects/ttssh2/releases and install it on your PC.
2. **Connect serial terminal**
   First plug in the USB serial converter and open Tera Term. Make a new connection on the serial device with baudrate 115200, 8 bits and no parity. Plug in the Mini-USB-B to USB-A cable into the serial converter and into the serial console interface (don't be confused, it's a Mini-USB socket)
3. **Connect network interface**
   If available connect the first ethernet interface to your network with the RJ45 ethernet cable. If your network provides a dhcp server the Eval-Kit should get an ip address.

4. **Connect power supply**
   Now connect the power supply and observe the boot messages on Tera Term. After switching on power the green 3.3V and 5V leds will light up. After the board has booted into linux LED3 will indicate a heartbeat blink and LED2 is controlled by M4 and blinks with approximately 2Hz. If the board has a display, the Kontron Qt5 Demo application is started.



Eval-Kit connected for the first start.

# First steps

## Login on serial console

After the board has booted you are prompted for a login user name

```
Kontron Electronics Reference Distro 1.3.0 stm32mp-t1000-s-multi /
dev/ttySTM0

stm32mp-t1000-s-multi login:
```

Login as 'root' without password. As you can see the current software version is printed out before the login prompt is requested.

If you want to know the BSP version out of a running system you can retrieve this from the file `/etc/os-release` :

```
root@stm32mp-t1000-s-multi:~# cat /etc/os-release
ID="ktn"
NAME="Kontron Electronics Reference Distro"
VERSION="1.3.0 (thud)"
BUILD_ID="20191030103231"
BUILD_VERSION="eaf450f"
BUILD_HOST="ubuntu"
BUILD_BB_VERSION="1.40.0"
BUILD_GCC_VERSION="8.%"
BUILD_GLIBC_VERSION="2.28%"
```

## Getting ssh access

The Eval-Kit provides a secure shell daemon (OpenSSH) for encrypted remote login via ip network. This can be used to get a shell login almost like the login via serial console, but it is also very handy to transfer files from development host to device or vice versa.

Precondition for this is that you have a Ethernet network where you can connect your Eval-Kit to. On boot the Eval-Kit will ask the dhcp server of your network for configuration settings. If your network provides a dhcp server this is the simplest way to get connected. The only thing you need is to know the ip address of your device. This can be done using the serial console:

```
root@stm32mp-t1000-s-multi:~# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 70:82:0E:99:96:52
          inet addr:192.168.1.65  Bcast:192.168.1.255  Mask:
255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:18 errors:0 dropped:0 overruns:0 frame:0
          TX packets:24 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:3051 (2.9 KiB)  TX bytes:5804 (5.6 KiB)
          Interrupt:59 Base address:0xa000
```

In this example, the device got the ip address `192.168.1.65` . Now you can use for example TeraTerm to connect to your Eval-Kit via network. Create a new connection, select 'TCP/IP' and put in your ip address in the server field of the dialog. Check that service 'SSH' is selected an press 'OK' to start the connection. Enter your username 'root' and leave password empty (or put your password there if you changed it already). Then the connection should be openend in a few seconds.

If your network doesn't provide a dhcp server you have to configure the network settings by hand. Therefore open the serial console and configure the network with the settings appropriate to your network (ask your administrator if unsure).

Here we do a configuration of the interface with ip adddress `192.168.1.70` on a class C network with network mask `255.255.255.0` :

```
root@stm32mp-t1000-s-multi:~# ifconfig eth0 192.168.1.70 netmask
255.255.255.0 up
```

Afterwards your Eval-Kit should be accessible under this ip address (provided, of course, that your computer has access to the same network and has the same network mask). Keep in mind that this configuration will be lost, wenn the device is rebooted!

With your ssh shell you can now explore the contents of the device a little bit and e.g. change some configurations. For example you can make your ip configuration reboot save by editing the file '/etc/network/interfaces' with the nano editor.

## Controlling M4 led

When the Eval-Kit starts a simple program is loaded into the M4 microcontroller by linux and started. This leads to LED2 blinking.

On the command line you can send commands to the M4 microcontroller which influences the led state:

```
root@stm32mp-t1000-s-multi:~# rpmsg0 "led status"
led state:BLINKING
root@stm32mp-t1000-s-multi:~# rpmsg0 "led off"
switch led off
root@stm32mp-t1000-s-multi:~# rpmsg0 "led on"
switch led on
root@stm32mp-t1000-s-multi:~# rpmsg0 "led blink"
switch led to blinking
```

## Next steps

- **Install VmWare and download Kontron VmWare image**
  In the next step you should download Kontron VmWare image to get a ready to run developent environment with all the tools you need preinstalled:
  Yocto, SDK, development tools, Kontron tools, STMicroelectronics tools
- **Read the Eval-Kit documentation**
  to get more information about your board
  - available boot media
  - available connectors and interfaces
- **Read the BSP documentation**
  - to update your board
  - to develop for M4
  - to use yocto for tweaking your own image

## License information

This Eval-Kit contains open-source software which grants you the rights to use, copy, modify and distribute the software. For more information which open-source componets and licenses are included, how to get the sources for this product and how to build them see the Eval-Kit and BSP documentation (https://docs.kontron-electronics.de/stm32mp/build-stm32mp) for this product.

# Description for Eval-Kit EVK STM32MP157 BL/DL

## Identification

| Name | Kit Number | Board | SoM | Description |
|---|---|---|---|---|
| EVK STM32MP157 BL | 50099044 | 40099 131 | 40099 167 | Baseboard with STM32MP157A, without Display |
| EVK STM32MP157 DL | 50099045 | 40099 131 | 40099 167 | Demoboard with STM32MP157A 5" Display and capacitive Touch |

**Boards**
are only boards without housing and peripherals. Different kits may be based on the same board.

**Kits**
are complete Eval-Kits for a quick start with the product. It often contains connectors, power plugs, housing and display based on a specific board and SoM.

- For a quick start with these kits see the Quickstart document.
- For connector interfaces and board layout see appendix

## Overview of components and features

This board consists of two main units. The SoM which is the more complex component including processor and DDR3-RAM. The second component is actually a baseboard including additional storage and containing all necessary connectors.

The SoM is soldered to the baseboard.

## SOM t1000 (40099 167)

- STM32MP157A SOC (2xCortex-A7@650MHz, 1xCortex-M4@200MHz)
- 512 MB DDR3 RAM
- 2 MB QSPI NOR flash
- 512 MB QSPI NAND flash
- 1x Ethernet PHY (100MBit/s)
- 1x I2C GPIO Expander

## Baseboard s (40099 131)

- 4GB eMMC
- Micro SD card slot
- 2 x USB host
- 1 x USB OTG
- 1 x USB Ethernet (100 MBit/s)
- 2 x Ethernet Connector (RJ45)
- 1 x RS232
- 1 x RS485 or CAN
- 3 x Debug LEDs
- 2 x Digital IO (GPIO expander)
- 2 x Analog IN
- 1 x PWM beeper
- 1 x Audio out
- Display Interfaces: RGB or DSI

### Display and Touch (only DL variant)

- 5 inch display with 800x480 resolution
- capacitive multitouch

## Software support

These Eval-Kits are supported by the kontron yocto BSP

| Name | shortname | devicetree file | housing | Description |
|---|---|---|---|---|
| EVK STM32MP157 BL | t1000-s | stm32mp-t1000-s.dts | | |
| EVK STM32MP157 DL | t1000-s-50 | stm32mp-t1000-s-50.dts | -50 | |

STM32CubeMx configuration for board is `t1000-s.ioc` .

**Important**

The `housing` variable in u-boot has to be set to select the correct kit configuration for the board. See u-boot bootloader for more info.

**Notice**

The build repository base URL is https://git.kontron-electronics.de/stm32mp. It is recommended to use the `init-env` script to checkout and populate all yocto layers and to setup the yocto environment for this device. See initializing the yocto build environment for more information

### Yocto configurations

The dedicated software configurations for these Eval-Kits

| | |
|---|---|
| build repository | build-stm32mp |
| branches | thud |
| machine | stm32mp-t1000-s-multi |
| init-env command | `. init-env -t thud build-stm32mp` |
| EULA accept variable | ACCEPT_EULA_stm32mp-t1000-s-multi = "1" |
| distros dedicated for these kits | ktn |
| images dedicated for these kits | image-ktn, image-ktn-qt |
| available boot devices | mmc0 (SD-card), mmc1 (eMMC), ubifs0 (QSPI NAND), pxe (Ethernet1) |
| latest prebuild binaries | https://files.kontron-electronics.de/stm32mp |

## Licence information

This product contains software components which are licensed as free respectively open-source software under the GNU General Public License, versions 2 or 3, or the GNU Lesser General Public License, versions 2.1 or 3 or any other open-source licence. Everyone can get the source code of this software components from us by download or by storage medium within three years after the delivery of the product or as long as we offer spare parts or support for the product.

To get the source code on a data storage medium (CD-ROM, DVD, USB drive), please send a request to our customer support at the following address

```
Kontron Electronics GmbH
Kantstrasse 10
72663 Grossbettlingen
Deutschland
```

```
Web: www.kontron-electronics.de
E-Mail: support@kontron-electronics.de
```

Including the statement of the following product data:

- Product name and product number
- Date of delivery

We also require a fee of EUR 10,- for the costs of preparation of the medium and shipping to be transferred.

Preventive it should be mentioned here that using the right of installing own versions of the open-source software components, which is guaranteed in the licence contract, will expire all certifications and warranties of the product. The operation of the manipulated product will happen on one's own authority.

If you want to download the source code covered by open-source licenses for this product use these URLs:

- For getting binaries, license texts and source code:
  https://files.kontron-electronics.de/stm32mp
  Look for the appropriate BSP version in this directory or in the *archive* subdirectory.
- For instructions how to build the software:
  https://docs.kontron-electronics.de/stm32mp/build-stm32mp

## Known issues and limitations

- It is not possible to set the device into the 'Engineering Mode' (M4 debugging without linux running)
- This board provides very limited peripherals for using with M4
- DSI interface is not tested

# Description of board components

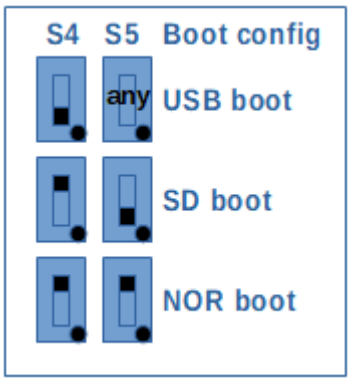
The following sections describe the hardware components and, if available their interface to to linux os.

## Power supply

The power supply is located on X4. The nominal supply voltage is 24V. The average current for the board with display is lower than 200mA.

## Boot switches

The boot source can be selected by boot switches on the bottom side of the baseboard. For devices with display you have to disassemble the baseboard to be able to change the boot switches. See Appendix to locate the boot switches on your board.




Some remarks to the boot modes:

- As long as the STM32MP1 SOC has no USB connection in USB boot mode, LED1 blinks fast.
- The boards are delivered with SD boot setting.

For these Eval-Kits a boot configuration is written in OTP memory:

```
BOOT_CONFIG3 = 0x22000000
```

This means the boot priority are as follos:

| Boot switch setting | BOOT0/1/2 pins on SoM | boot source search sequence |
|---|---|---|
| USB boot | tbd. | USB OTG port waits for CubeProgrammer to send data |
| SD boot | tbd. | SD-card, NOR boot, USB boot |
| NOR boot | tbd. | SD-card, NOR boot, USB boot |

As a conclusion, the setting of SD/NOR boot switch doesn't matter! Only USB boot switch is useful if the device should be started with CubeProgrammer.

See ST Wiki for documentation of boot fuses settings.

## Serial Interfaces

Also see BSP / Using the Hardware / Serial Interfaces.

| STM32MP1 | Used as | Linux access | Connector | Usage |
|---|---|---|---|---|
| uart4 | console | `/dev/ttySTM0` | X11 (MiniUSB) | |
| usart2 | RS485 | `/dev/ttySTM1` | X18 (4pol) | Multipexed with CAN interface |
| usart6 | RS232 | `/dev/ttySTM2` | X16 (3pol) | Reserved for M4 demo program |

How to configure RS485/CAN mode for X18 can be found in Board Layout

**Info**

You need an additional adapter to translate the 3.3V console UART signals (provided on the Mini-USB port) to USB.

## CAN Interfaces

| STM32MP1 | Used as | Linux access | Connector | Usage |
|---|---|---|---|---|
| can1 | CAN | `SocketCAN:can0` | X18 (4pol) | Multipexed with RS485 interface. |

How to configure RS485/CAN mode for X18 can be found in Board Layout

## Ethernet

| Name | Connector | Linux device | Remark |
|---|---|---|---|
| Ethernet 1 | X2 | eth0 | Native SOC interface |
| Ethernet 2 | X6 | eth1 | USB ethernet, not available in u-boot bootloader |

## Digital IOs

Two digital inputs/outputs (either or) are available. If used as output the state can be read back from the associating input.

The table below shows number and function of available GPIOs. You can access them via the standard GPIO sysfs interface `/sys/class/gpio` or by libgpiod (commands `gpio*` ).

| Name | direction | GPIO number (sysfs) | Accessible via (libgpiod) | Connector |
|---|---|---|---|---|
| dout1 | output | 507 | `gpiochip8.3` | X17_DIO1 |
| din1 | input | 506 | `gpiochip8.2` | X17_DIO1 |
| dout2 | output | 505 | `gpiochip8.1` | X17_DIO2 |
| din2 | input | 504 | `gpiochip8.0` | X17_DIO2 |

## Analog inputs

There are two analog inputs available on the board, connected to the internal ADC

| Name | Accessible via | Connector |
|---|---|---|
| AIN1 | `/sys/bus/iio/devices/iio\:device0/in_voltage5_raw` | X17_AIN1 |
| AIN2 | `/sys/bus/iio/devices/iio\:device0/in_voltage16_raw` | X17_AIN2 |

For voltage calculation in mV from the raw value see STMicroelectronics Wiki
The scaling formula for the board is:
`Uconnector = Uadc * 11`

For simplicity there is a `adcread` script to read adc values which observes all offsets, and scaling factors. To read out channel 5 on device adc 0 call:

```
> adcread 0 5
ADC0.5: 16 mV
```

## LEDs

There are 3 debug LEDs available and can be controlled by linux user space or are used by M4 demo program

| Name | Interface | Accessible via | Used as |
|---|---|---|---|
| LED1 | SOC PORTA 13 | M4 demo | M4 demo LED |
| LED2 | SOC PORTA 14 | `/sys/class/leds/LED2` | |
| LED3 | I2C GPIO exander | `/sys/class/leds/LED3` | heartbeat |

**Hint**

LED1 (PORT A13) is also used to indicate USB boot mode. When STM32MP1 waits for USB connection, LED1 blinks fast.

## I2C busses

On this board there are two i2c busses used:

| Name | Accessible via | Used by |
|---|---|---|
| I2C2 | i2c-1 | GPIO SOM, Touch |
| I2C4 | i2c-2 | AUDIO, DSI |

**Important**

i2c-1 interface is internally used for GPIO port expander TCA6408A on SoM at address 0x20

### Devices on i2c-1

| Adresse | Location | Komponente |
|---|---|---|
| 0x20 | SOM | GPIO port expander TCA6408A |
| 0x14 | Housing | Goodix touch |

**Devices on i2c-2**

| Adresse | Location | Komponente |
| --- | --- | --- |
| 0x1a | Board | Audio WM8510 |

## USB host

Two USB 2.0 host interfaces are available on connector X9.

## USB OTG

One USB OTG port is available on connector X7. This USB OTG port is required for USB boot mode.

## Audio

A speaker can be plugged in X10

## Display Interfaces

The display is connected via the 40 pin RGB 24 interface (X13). The touch lines for the touch controller are pinned separately via X5. The Display can be connected directly to the standard Kontron Display without need of an adapter.

The board is additionally equiped with an 50 pin RGB interface (X12). The touch controller pins are included. With this interface customer specific displays can be connected via a convenient display adapter (bonded on the display).

On X1 there is also a DSI interface.

> **Important**
>
> Only one interface (RGB or DSI) can be active at the same time.

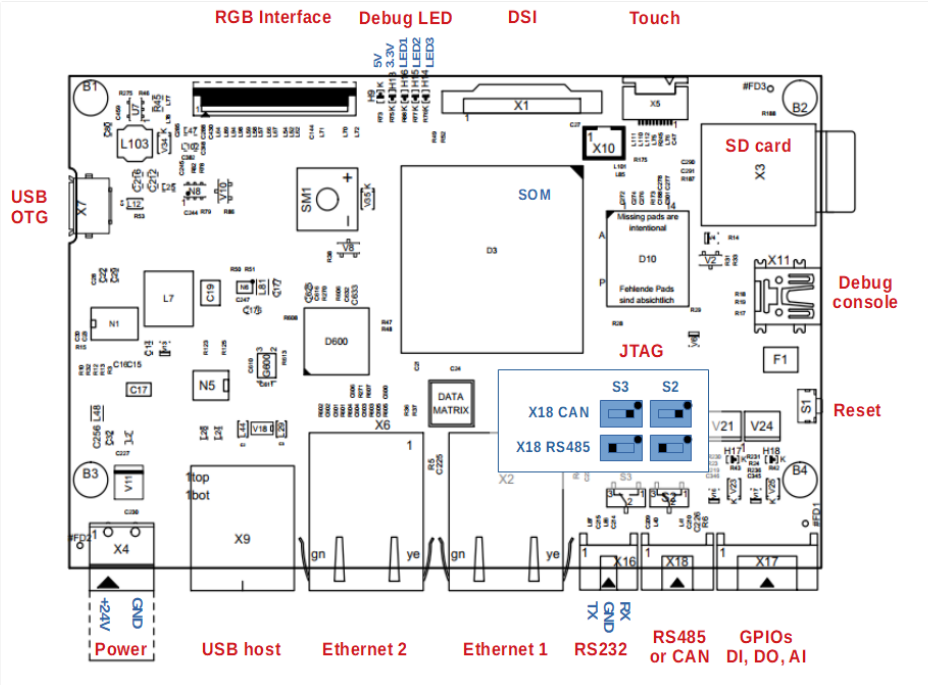
## RTC

On this board the internal RTC is used.

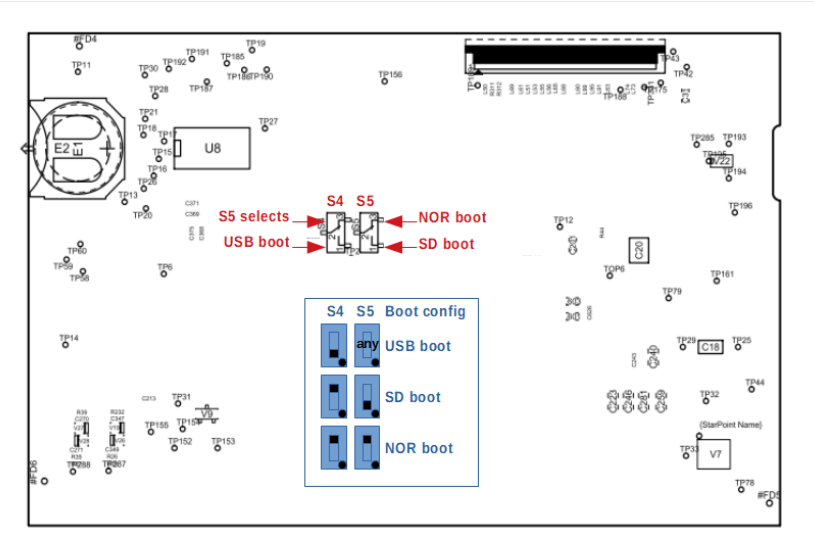
# Appendix

## Board Layout

### Top view

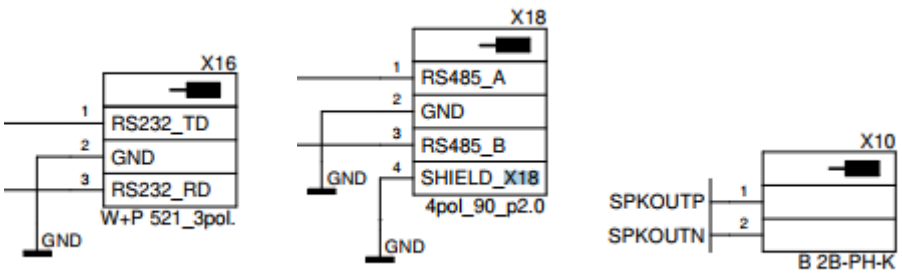


### Bottom view



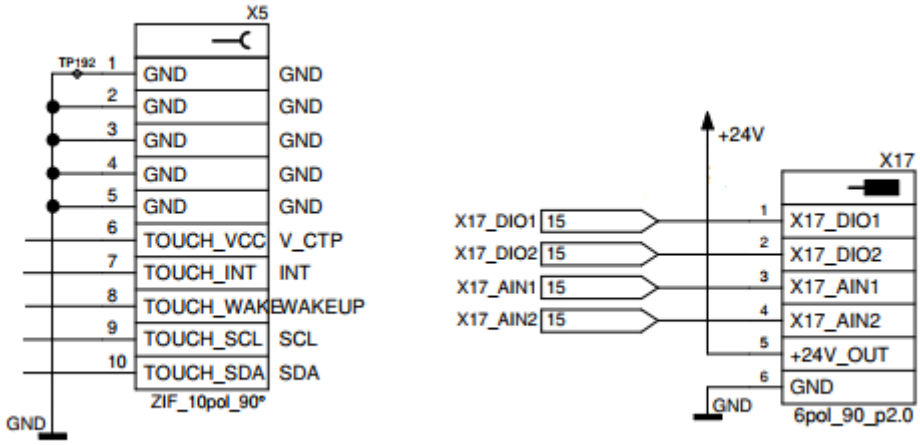

## Connector Pinouts

### RS232, RS485, Speaker (X16, X18, X10)

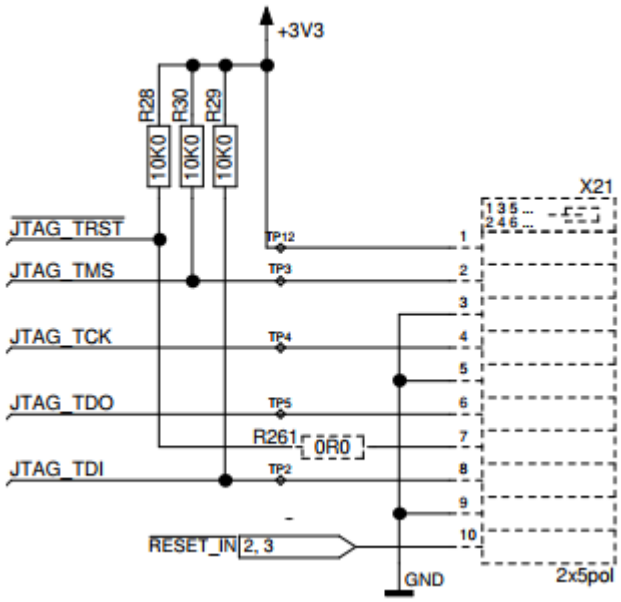



X16 uses the DTE configuration (TD is output, RD is input)
CAN configuration for X18 is RS485_A -> CAN_H, RS485_B -> CAN_L
CAN has an internal 120 ohms termination resistior

## Touch, GPIO (X5, X17)




## JTAG (X21)




- Location on board
- ST-Link with Olimex connector to the board and the PC

# FAQ

### My board doesn't boot any more

If your board doesn't boot any more check these things:

- Check if 5V and 3.3V led are lighten green. If none lights up, check your power supply. If only one lights up, your board is damaged!
- Check the boot switches. If LED1 flashes fast immediately after the power is switched on, it seems, that the device is in USB downloader mode. Either the boot switches are set to USB boot or the boot device doesn't contain a valid tf-a boot loader.
- Check your serial connection (115200 baud, 8bits, no parity)
- Try to boot from a fresh written SD card with bootable content