DOCUMENTATION

emtrion
embedded systems

emCON-MX8MM Yocto Manual

Yocto Based BSP Manual

Rev. 5 / 12.05.2021
Author: Saptarshi Mitra
Last Change: May 12, 2021

Revision: Rev. 5 / 12.05.2021

Rev.	Date/Initial	Changes
1	23.10.2019/Mi	First Version of emCON-MX8MM Yocto Manual
2	05.11.2019/Mi	Minor changes for easier comprehension
3	11.11.2019/Mi	Audio example instructions added
4	09.01.2020/Mi	Minor modifications in introduction
5	12.05.2021/Rr	Changing to Yocto 3.0 Zeus

Contents

1 Introduction

Emtrion produces and offers various base boards and modules which are available in `https://support.emtrion.de/en/home.html`. One of the exciting products in Emtrion's product range is a module named emCON-MX8MM. This manual provides instructions and pointers for efficient use of Yocto project development tool with the hardware.

The **emCON-MX8MM** is based on the **i.MX processor i.MX8MM** from NXP Semiconductors. Among many others, it supports open source software development with a fully functional Linux kernel. The system can be tailored according to requirements using Yocto Project as well as Debian. The board is an MPU, providing the user benefits of a **Quad core Cortex A53** and **Cortex-M4** processor.

The **Yocto version** for the product is **Yocto Zeus (3.0**), launched in October 2019. The Yocto layers provided by Freescale and emtrion are based on this Yocto version. The layers used from freescale repositories are **meta-imx**, **meta-freescale**, **meta-freescale-3rdparty** and **meta-freescale-distro**. Emtrion provides two Yocto layers to add the necessary functions: **"meta-emtrion"** and **"meta-emtrion-bsp"** layers. While the meta-emtrion layer provides a general layer for all the functions common to the entire product range, the emtrion-bsp layer, as the name suggests is board specific. In case of the board under consideration, the relevant layer is **meta-emtrion-emcon-mx8mm**. Some new recipes have been added to the emtrion layers and existing recipes have been modified to tailor for the device.

The BSP is a **Linux Kernel**, sourced from the i.MX Linux repo and the version is **imx_5.4.70_2.3.0** based on Linux mainline version **5.4**. This release has a scheduled LTS support till Dec 2025, ideal for a stable product line. The developer kit is managed by the specific kernel configuration file and the device tree file, along with patches required for the correct operation of the associated peripherals.

The U-boot system implemented in the board is also based on the NXP modified repository branch **imx_v2020.04 _5.4.70_2.3.0**, using u-boot version **v2020.04**. Additional patches and have been provided in the meta-emtrion-bsp layer to give a working version for the emCON-MX8MM.

The important version numbers for the various tools and packages after the update are listed below.

Name	Version
Linux iMX	v5.4.70
Uboot iMX	v2020.04
Yocto	Zeus v3.0.4
Bitbake	v1.44.0
Gcc version	9.2.0
Openssh	v8.0
Busybox	v1.31.0
systemd	v1.243.2
Qt	v5.15.0
Gstreamer	v1.16.2
Weston	v9.0.0
Graphics support	3D GPU with OpenGL_ES 2.0

Table 1.1: *Packages and versions*

This manual describes the scope of the developer kit and the general information as well as instructions for the user.

It is assumed that users of Emtrion developer kits are already familiar with U-boot, Linux, Yocto and creating and debugging applications. General Linux and programming knowledge are out of the scope of this document. Emtrion is happy to assist you in acquiring this knowledge. If you are interested in training courses or getting support, please contact the Emtrion sales department.

Please understand we can not go into more details about Yocto Project inside the limited scope of this documentation, because Yocto/OpenEmbedded is a powerful but also complex build system. Emtrion offers paid support for problems regarding the developer kit. The official documentations can be referred to, however if that is not sufficient, we also offer training sessions about Yocto/OpenEmbedded.

The important resources that can be accessed for further in-depth knowledge are as follows:

1. Yocto Manual: `https://www.yoctoproject.org/docs/3.0/ref-manual/` (any question related to Yocto has some reference here)
2. Openembedded: `http://www.openembedded.org/` (information regarding openembedded layers for Linux development)
3. NXP: `https://www.nxp.com` (information about the NXP iMX8M Mini board, along with reference documents)
4. Linux documents for iMX: `https://www.nxp.com/webapp/Download?colCode=L5.4.70_2.3.0_LINUX_DOCS&location=null` (Linux, Yocto, graphics, VPU user guides as well as reference manual)
5. Yocto repositories: `https://git.yoctoproject.org/` (with the main Yocto repo: poky and other supplementary ones)
6. Openembedded repositories : `https://git.openembedded.org/` (with the required meta-openembedded layer and other supplementary)
7. Freescale repositories : `https://github.com/Freescale` (Freescale repositories for Yocto layers)
8. IMX repositories : `https://source.codeaurora.org/external/imx` (NXP repositories for Linux and Uboot)

2 Terms and definitions

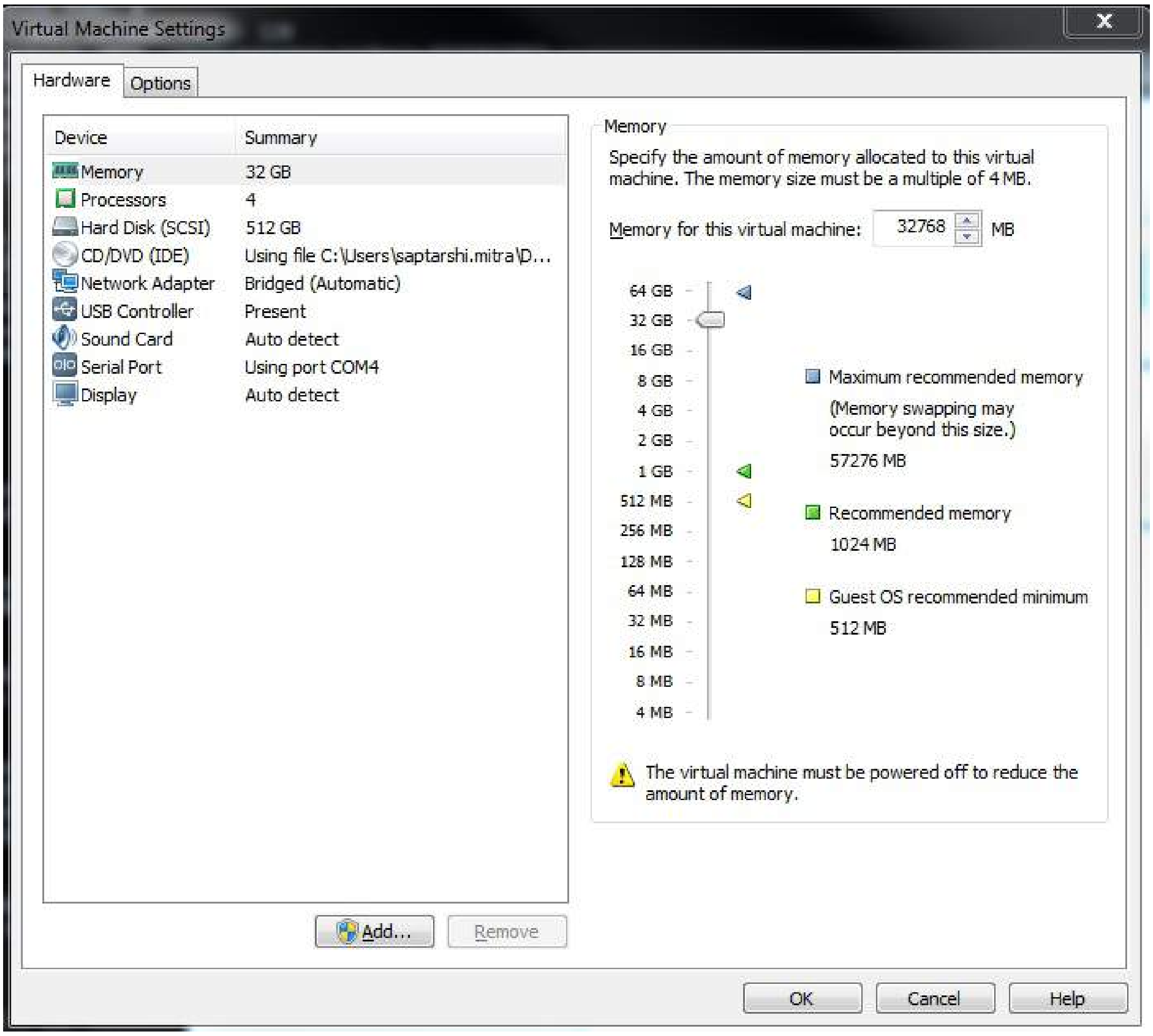
Term	Definition
Target	Module: emcon-mx8mm
Host	Workstation, Developer PC
Toolchain	Compiler, Linker, etc.
RootFS	Root file system, contains the basic operating system
Console	Text terminal interface for Linux
NFS	Network File System, which can share directories over network
NFS_SHARE	Location that is exported by the NFS for the purpose of updating and booting by using NFS
U-Boot	Bootloader, hardware initialization, updating images, starting OS
YP	Yocto Project
INST_DIR	Location where Yocto and the meta-layers are installed
MACHINE	Specifies the target device for which the image is built. The machine is named emcon-mx8mm for this kit
BUILD_DIR	Machine dependent build directory
BSP	Board Support Package
SDK	Software Development Kit

Table 2.1: *Terms and definitions*

3 Linux Virtual Machine

3.1 Test

The set up environment demonstrated or referred to in the entire document is a virtual machine running Linux having the following settings. However, it is expected that the discussion holds true for other comparable systems.



Device	Summary
Memory	32 GB
Processors	4
Hard Disk (SCSI)	512 GB
CD/DVD (IDE)	Using file C:\Users\saptarshi.mitra\D...
Network Adapter	Bridged (Automatic)
USB Controller	Present
Sound Card	Auto detect
Serial Port	Using port COM4
Display	Auto detect

Memory

Specify the amount of memory allocated to this virtual machine. The memory size must be a multiple of 4 MB.

Memory for this virtual machine: 32768 MB

🟦 Maximum recommended memory (Memory swapping may occur beyond this size.) 57276 MB

🟩 Recommended memory 1024 MB

🟨 Guest OS recommended minimum 512 MB

The virtual machine must be powered off to reduce the amount of memory.

Figure 3.1: *Virtual Machine settings*

However, the settings are strongly dependent on the PC and have to be adjusted later on your PC. A powerful PC will cut down time spent on debugging as well as image creation.

- Memory

- Number of processors
- Network adapter
- Serial port

3.2 Virtual Machine settings

System requirements

As Linux Distribution in the system, **Debian GNU/Linux 9 (stretch)** is used. Please note that if using Debian, Yocto Zeus used for this device, is supported by **Jessie and Stretch**. The other possible development environments are listed in the Yocto reference manual. The packages mentioned below are only indicative of the packages that may be required for other systems. The distribution was set up with general Yocto Project system requirements described in the chapter "1. System Requirements" of the Reference Manual Yocto Project 3.0.4 Release.

http://www.yoctoproject.org/docs/3.0/ref-manual/ref-manual.html

Further components are included

- Serial ports added to the virtual machine appear at /dev/ttySn, USB serial converters at **/dev/ttyUSBn**.
- A NFS server exporting the nfs share **/home/hico/nfs**.
- The serial terminal program **picocom** for connecting to the target.
- The Yocto-Layers **meta-emtrion** and **meta-emtrion-bsp**.
- Packages needed to build an image are listed in the following section. Some of the packages may already be installed in the system, but ensuring the presence of all will avoid any unforeseen problems during the build.

```
sudo apt-get install gawk wget git-core diffstat unzip texinfo \
build-essential chrpath socat libsdl1.2-dev libsdl1.2-dev xterm \
sed cvs subversion coreutils texi2html docbook-utils python-pysqlite2 \
help2man make gcc g++ desktop-file-utils libgl1-mesa-dev \
libglu1-mesa-dev mercurial autoconf automake groff curl lzop asciidoc

```

Listing 3.1: *Recommended Packages from NXP*

3.3 Starting the VM

The VM is a compressed ZIP archive. Changing settings and starting the VM is used by the VMware Player or VMware Workstation. Here is the link for downloading the VMware Player.

https://www.vmware.com/go/downloadplayer

The corresponding manual is available in the following link.

https://docs.vmware.com/en/VMware-Workstation-Pro/15.0/workstation-pro-15-user-guide.pdf

After decompressing of the VM and importing it by the VMware Player, first check if the settings above are fit to your PC. If not, adjust the settings by reading the corresponding chapters of the specified manual.

Please note, the size of the VM will increase up to 128 GB while you are working with it.

3.4 Preconfigured variables

Within the VM there are used some predefined locations. In the scope of this document the locations are assigned to specified placeholders. They are listed in the table below.

Placeholder	Assignment	Remark
MACHINE	emcon-mx8mm	machine name
DISTRO	fsl-xwayland-emtrion	distro name created for emcon-mx8mm with xwayland graphics backend
HOME_DIR	/home/hico	home directory of user hico (Replace it with the corresponding home directory in your PC)
NFS_SHARE	<HOME_DIR>/nfs	Location exported by the NFS
NFS_ROOTFS	<NFS_SHARE>/<MACHINE>/rootfs	nfs share for booting the root file system by using NFS
INST_DIR	<HOME_DIR/Yocto_emcon-mx8mm	Location where Yocto and all the meta-layers will be stored to
BUILD_DIR	<INST_DIR>/YoctoBuildDirectory/emtrion/machines/<MACHINE>	Location of the build system
BUILD_DWNL	<INST_DIR>/YoctoBuilddirectory/downloads	Location of the fetched downloads while the build process
BUILD_SSTATE	<INST_DIR>/YoctoBuildDirectory/sstate-cache	Location of the sstate-cache while the build process
HOME_DWNL	<HOME_DIR>/Downloads	Location of the pre-built images, SDK
SDK_DIR	/opt/fsl-xwayland-emtrion/4.14-thud/	SDK with tools, sources and libraries

Table 3.1: *Preconfigured Variables*

4 Installation

4.1 Emtrion development kit package

4.1.1 Package Contents

Emtrion software package for emCON-MX8MM development kit comes as an archive file, which on extraction reveals the following directories and files:

1. **layers**
   a) **meta-emtrion**: Contains the general recipes and files required for the majority of the products in the Emtrion product range.
   b) **meta-emtrion-bsp**: Contains the directory corresponding to the machine under consideration, with machine specific configuration files, recipes and patches.
2. **images**
   a) **ramdisk-emcon-mx8mm.cpio.gz**: This image is required for the EMPURS script, for updating and restoring the system.
   b) **rfs_image-<MACHINE>.rootfs.ext4**: Linux image file for the rootfs system created for the development kit.
   c) **rfs_image-<MACHINE>.rootfs.tar.gz**: This archive file can be extracted and the device can be booted and accessed using nfs boot.
   d) **Image**: Kernel image pertinent to the developer kit.
   e) **imx8mm-emcon-avari.dtb**: Device tree for the emcon-mx8mm board to be user with emtrion Avari baseboard, developed for the linux kernel.
   f) **imx8mm-emcon-bvari.dtb**: Device tree for the emcon-mx8mm board to be user with emtrion Bvari baseboard, developed for the linux kernel.
   g) **imx-boot-emcon-mx8mm-sd.bin-flash_evk**: Bootloader for emcon-mx8mm with bl31, uboot-spl, uboot packaged together.
3. **emcon-mx8mm-setup-release.sh**:This script can be utilized to setup the entire environment for Yocto development.
4. **Readme-emcon-mx8mm**: The readme file contains more information about the setup script, providing instructions for the user. Emtrion recommends reading this before starting with the installation, to have a clear idea about the modifiable variables and layers and to avoid problems in the later stages.

More information about the layers are available in the Emtrion's Yocto layers section. The setup script is described in details in the Yocto setup script section.

4.1.2 Installation steps

The installation of the Emtrion layers has to be performed in order of the following steps.

1. Create a directory with a name for your reference. It's called **<INST_DIR>** how.
2. Change into this directory and install NXP Yocto release according `https://source.codeaurora.org/external/imx/imx-manifest/tree/README?h=imx-linux-zeus`. Select **5.4.70-2.3.2** release.
3. Download the Yocto package for the emCON-MX8MM board from the Emtrion website to any location outside **<INST_DIR>**.
4. Extract the contents.
5. Move or copy **meta-emtrion** and **meta-emtrion-bsp** directories to **<INST_DIR>/sources** and the link file **emcon-mx8mm-setup-release.sh** to **<INST_DIR>**. (For those with direct access to the Emtrion git repositories, these steps are unnecessary, as a special manifest file will be availabe at emtrion.)
6. Set up of the machine dependent build directory by sourcing the setup script from the **<INST_DIR>**. e.g. ***. ./emcon-mx8mm-setup-release.sh***

4.1.3 Installation results

The installation steps shown in the chapter before result in a directory structure with all subdirectories necessary for compiling a rootfs with bitbake. When running ***. ./emcon-mx8mm-setup-release.sh*** for the first time, an NXP EULA for using the NXP source code is checked. Just scroll down and answer the question with **y**. Afterwards the build directory is created and the configuration files **local.conf** and **bblayer.conf** are prepared and copied to the subdirectory **conf**. It is possible to create an individual build directory by calling ***. ./emcon-mx8mm-setup-release.sh -b <my-build-dir>***. The machine name, the distro name and the baseboard are preset in the script file, but it is possible to configure them by setting environment variables.

The created directory structure in INST_DIR is as follows:

Location	Remarks
InstallationDirectory/	
|– .repo	stores repositories and manifest files created by repo init
|– downloads	stores fetched data required for the build process
|– sources	all meta-layers and poky directory
|– build	build directory of one machine created by script
| |– conf	contains local.conf, bblayers.conf created by script
| |– sstate-cache	contains the build states (created during the build process)
| |– cache	contains the build cache (created during the build process)
| |– buildhistory	for caching (created during the build process)
| |– tmp	compiling results (created during the build process)
| |– deploy	images, sdk and so on (created during the build process)

Table 4.1: *Installation Directory structure*

Info

Note, some of the directories are created after the first run of bitbake.

After sourcing the script, the prompt automatically changes to the build directory, where images can be created using bitbake commands.

4.2 Yocto Setup Script

To get the initial build setup please execute the following command in the INST_DIR:

MACHINE=<my-machine> BASEBOARD=<my-base> DISTRO=<my-distro> . ./emcon-mx8mm-setup-release.sh -b

Info

Please note that the **script has to be sourced** from the directory mentioned and not directly executed. Doing otherwise will lead to an error while setting up the environment for building images. The script requires at least either f or i argument to run successfully.

4.2.1 Parameters

- **-b**: Build directory. This allows choosing an individual build directory name for handling several build side by side. Default value is **build**.
- **MACHINE=<my-machine>**: Default value is **emcon-mx8mm**. No need to change this but for using NXP machines set the suitable value here.
- **BASEBOARD=<my-baseboard>**: As a default value **aravi** is preset, other baseboards are also available from emtrion or it will be possible to configure customer specific basebards here.
- **DISTRO=<my-distro>**: The default distro is **fsl-xwayland-emtrion**. Other distros have not been tested yet.

When running the script for the first time, the files **local.conf** and **bblayer.conf** are generated. In addition copies of those files are created. In the following calls the copies are used for updating **local.conf** and **bblayer.conf**.

4.3 Emtrion's Yocto Layers

The Yocto layers provided with the software package have been described in the following section, with the associated directory structure.

Location	Remarks
meta-emtrion	Emtrion's basic meta layer for general Yocto development
|– conf | |– layer.conf	Configuration file for this meta layer
|– recipes	Recipes created by emtrion
| |– empurs-scripts | | |– empurs-scripts_0.1.bb | | |– files | | | |– devtmpfs | | | |– emPURS | | | |– empurs.service | | | |– emPURSinit.sh	Recipes and scripts for system update and restore

| |– emtrion-config | | |– emtrion-config_0.2.bb | | |– files | | | |– profile.qt | | | |– telnetd.sh	Basic configuration files
| |– gpio-utils | | |– files | | | |– gpio-utils.tar.bz2 | | |– gpio-utils_0.1.bb	Example of using GPIOs
| |– images	Recipes for various images offered by Emtrion
| | |– core-image-opengles.bb	Image with opengles support (not needed in this board)
| | |– core-image-purs.bb	Recipe for ramdisk image creation
| | |– emtrion-devkit-image.bb	Recipe for image of the board with rootfs
|– recipes-connectivity	
| |– openssh | |– files | | |– sshd_config | |–openssh_%.bbappend	Recipes for openssh to use ssh login in the boards
|– recipes-core	
| |– busybox | | |– busybox_1.31.0.bbappend | | |– files | | | |– busybox-udhcpc | | | |– defconfig	Recipe for busybox, providing a host of tools for linux
| |– initscripts | | |– files | | | |– umountfs | | |– initscripts_1.0.bbappend	Recipes for initscripts, used during system boot
| |– netbase | |– netbase-5.6 | | |– interfaces | |– netbase_5.6.bbappend	Recipes for netbase, important for TCP/IP communications
|– recipes-kernel	
| |– gator | | |– gator-daemon | | | |– makefile_21.patch | | |– gator-daemon_5.21.bb | | |– gator-driver | | | |– emtrion_activate_ETM_fwding_21.patch | | |– gator-driver_5.21.bb	Recipes not tested for this board
| |– linux | | |– files | | | |– linux.inc	Include file containing various linux parameters
|– recipes-qt | |– images | | |– qt5-image-slim.bb | |– packagegroups | | |– packagegroup-qt5-slim.bb	Packagegroups and images required for Qt5 support
|– recipes-support | |– devmem2 | | |– devmem2 | | | |– devmem2-fixups-2.patch | | |– devmem2.bb	Recipes for devmem, used for reading and writing to different memory locations

Location	Remarks

meta-emtrion-bsp	Board specific Emtrion meta layer
|– meta-emtrion-emcon-mx8mm	Particular board specific components, relevant to this board
|– conf	Configuration files
| |– distro	
| | |-fsl-xwayland-emtrion.conf	
| | |-include	Configuration and distro for this meta layer
| | | |-fsl-base.inc	
| | | |-fsl-emtrion.inc	
| |– layer.conf	
| |– machine	Machine specific configuration file, containing important information regarding the build
| | |– emcon-mx8mm.conf	
|– default-config	Default files required for proper environment set up (may or may not be modified by the user while using the setup script for installation)
| |– bblayers.conf	
| |– local.conf	
|– recipes	Recipes created or modified by emtrion for the specific board
| |– empurs-scripts	
| | |– empurs-scripts_0.1.bbappend	Recipe for update or restore of system
| | |– files	
| | | |– emPURS	
| |– emtrion-config	
| | |– emtrion-config_0.2.bbappend	
| | |– files	Recipe involving uboot parameters, pointer calculation, graphics window initialization as well as platform specific parameters for recovery and update
| | | |– emPURS_plat	
| | | |– pointercal	
| | | |– qt-wayland.sh	
| | | |– uboot_script	
| | | |–weston.ini	
| |– images	Images relevant to the dev kit
| | |– core-image-purs.bbappend	Ramdisk image creation for system update and core features
| | |– emtrion-devkit-image.bbappend	Image for the actual kit, with all the necessary attributes
|– recipes-bsp	Recipes related to the particular bsp sourced from NXP
| |– imx-atf	Recipe for adding atf binaries to the bootloader
| | |– imx-atf_2.2.bbappend	
| |– imx-mkimage	Recipe packaging bootloader to create a single binary
| | |– imx-boot_1.0.bbappend	

```
|    |– u-boot
|    |    |– files
|    |    |    |– 0001-Added-support-for-emCON-MX8MM-boards-fi
|    |    |    |– 0002-emCON-MX8-added-support-for-1GB-RAM-ass
|    |    |    |– 0003-set-QSPI-frequency-to-20MHz-bug-fix-set-FSP
|    |    |    |– 0004-emCON-MX8-Bug-Fix-for-SDC1-change-drive-
|    |    |    |– 0005-Added-support-for-4GB-LPDDR4-memory.-No
|    |    |    |– 0006-BugFix-now-the-4GB-is-working-correctly-eve
|    |    |    |– 0007-Remove-original-gitlab-ci-yaml.patch
|    |    |    |– 0008-Port-from-v2019.04_4.19.35_1.1.0-to-v2020.04_
|    |    |    |– 0009-Added-initial-version-of-emtrion-gitlab-ci-yar
|    |    |    |– 0010-BugFix-now-MMC-is-working-with-U-Boot-v2c
|    |    |    |– 0011-Added-Android-support-for-board-emCON-MX
|    |    |    |– 0012-BugFix-correct-default-environment-for-Andrc
|    |    |    |– 0013-Enhance-default-boot-command.patch
|    |    |    |– 0014-Enable-support-for-Android-AB.patch
|    |    |    |– 0015-tidyup-defconfig.patch
|    |    |    |– 0016-correcting-ldo1-and-ldo2-settings.patch
|    |    |    |– 0017-script-for-compiling-bootloader-added.patch
|    |    |    |– 0018-device-tree-for-SPL-bootloader-added.patch
|    |    |    |– 0019-rearranging-settings-from-config-header-file-t
|    |    |    |– 0020-Setting-sys-extra-options-to-correct-cfg-file-a
|    |    |    |– 0021-this-spl-needs-no-device-tree.patch
|    |    |    |– 0022-useless-comments-removed.patch
|    |    |    |– 0023-Booting-linux-from-eMMC-per-default.patch
|    |    |    |– board
|    |    |    |    |– emtrion
|    |    |    |    |    |– common
|    |    |    |    |    |    |– emtrionlogo.bmp.gz
|    |– u-boot-common.inc
|    |– u-boot-emcon-mx8mm_2020.04.bb
|    |– u-boot-emcon-mx8mm-fw-utils_2020.04.bb
|    |– u-boot-emcon-mx8mm-mfgtool_2020.04.bb
|    |– u-boot-tools_%.bbappend
```

Patches, device trees, configuration file and recipes responsible from creation of the u-boot from the freescale layer

```
|– recipes-graphics
|    |– wayland
|    |    |– weston
|    |    |    |– emcon-mx8mm
|    |    |    |    |– weston.ini
|    |    |– weston-init
|    |    |    |– 71-weston-drm.rules
|    |    |    |– init
|    |    |    |– profile
|    |    |    |– weston-start
|    |    |    |– weston.ini
|    |    |    |– weston.service
|    |    |– weston-init.bbappend
```

Recipe for graphics using weston, along with the configuration file to be used

```
|– recipes-kernel
```

```
|     |– linux
|     |     |– files
|     |     |– 0001-First-basic-support-for-the-emCON-mx8mm-modu
|     |     |– 0002-arm64-dts-Added-missing-I2C-device-address-to-
|     |     |– 0003-arm64-defconfig-Initial-commit-of-a-emtrion-spe
|     |     |– 0004-arm64-dts-Fallback-to-SPI-Mode-for-QSPI-flash.pa
|     |     |– 0005-arm64-dts-minor-change-in-compatible-string.pa
|     |     |– 0006-gpu-drm-bridge-Added-support-for-the-SN65DSI
|     |     |– 0007-arm64-dts-Added-support-for-the-dsi-lvds-bridge
|     |     |– 0008-arm64-dts-Initial-commit-of-bvari-devicetree-files
|     |     |– 0009-arm64-adaptation-of-emCON-MX8MM-configurat
|     |     |– 0010-gpu-drm-changes-in-sn65dsi83-driver-due-to-por
|     |     |– 0011-arm64-dts-Disabled-support-for-CSI.patch
|     |     |– 0012-arm64-dts-Added-support-for-PCIe.patch
|     |     |– 0013-arm64-dts-Added-USB-host-support.patch
|     |     |– 0014-arm64-dtsi-Changed-pinmuxing-for-sd-card.patch
|     |     |– 0015-arm64-dtsi-Restricted-uSDHC2-interface-to-3-3V.p
|     |     |– 0016-arm64-dtsi-Reactivated-gpu-and-vpus-on-mx8.pa
|     |     |– 0017-arm64-dtsi-Added-the-RTS-DTS-pins-to-UART2.pa
|     |     |– 0018-dt-bindings-pinctrl-Added-missing-pin-muxings.p
|     |     |– 0019-arm64-dtsi-Added-second-flexspi-an-ecspi.patch
|     |     |– 0020-arm64-dtsi-Changed-default-startup-backlight-le
|     |     |– 0021-arm64-dtsi-Cleanup-of-unneeded-parts.patch
|     |     |– 0022-arm64-dtsi-Additional-cleanup-of-devicetree-files
|     |     |– 0023-arm64-dtsi-Added-support-for-avari-and-bvari.pa
|     |     |– 0024-arm64-config-Stripped-down-the-defconfig.patch
|     |     |– 0025-arm64-config-Stripped-down-the-defconfig.patch
|     |     |– 0026-arm64-defconfig-Removed-autoversion-and-adde
|     |     |– 0027-scripts-builddeb-Added-emboot-dependency.pat
|     |     |– 0028-arm64-defconfig-Activated-SDMA-driver-in-emtri
|     |     |– 0029-arm64-dts-Fixed-wrong-pinmux-settings-for-SAI2.
|     |     |– 0030-arm64-dts-Fixed-settings-for-simple-audio-device
|     |     |– 0031-drm-bridge-Bugfix-SN65DSI83-bridge-correct-calc
|     |     |– 0032-arm64-dts-Changed-Backlight-PWM-frequency-fr
|     |     |– 0033-emcon-mx8mm-bugfix-for-PCIe-interface-correcti
|     |     |– 0034-Added-support-for-PCIe-module-with-Qualcomm
|     |     |– 0035-Added-support-for-PCIe-module-with-Intel-AC316
|     |     |– 0036-Added-info-message-about-DSI-clock-parameters
|     |     |– 0037-emCON-MX8MM-new-defconfig-for-builds-of-Linu
|     |     |– 0038-emCON-MX8MM-BugFixing-adaptation-of-the-De
|     |     |– 0039-emcon-mx8mm-BugFix-added-define-in-defconfi
|     |     |– 0040-Added-initial-GitLab-CI-yaml.patch
|     |     |– 0041-GitLab-CI-yaml-modified-parameters-for-building
|     |     |– 0042-BugFix-correction-of-author-name.patch
|     |     |– 0043-GitLab-CI-yaml-correction-of-kernel-package-buil
|     |     |– 0044-GitLab-CI-yml-get-used-docker-image-from-emtri
|     |     |– 0045-GitLab-CI-yml-bugfix-correct-move-of-created-pa
|     |     |– 0046-drm-bridge-disable-dev_info-for-clocks.patch
|     |     |– defconfig
|     |– linux-emcon-mx8mm_5.4.bb
```

Patches and recipes responsible from creation of the kernel from the NXP developed linux with the relevant device tree and defconfig file support

```
|– recipes-qt
|     |– qt5
|     |     |– qtbase_%.bbappend
|     |     |– qtwayland_%.bbappend
```

Recipe extensions for Qt5 support with wayland

5 Image Creation

The next step after setting up the build system as instructed in the previous chapter Installation, is to start building recipes and images for the emcon-mx8mm.

As mentioned before, the layer meta-emtrion and meta-emtrion-bsp provides the required images for the device. You can start building an image by prompting bitbake following the name of the image recipe. Enter in the terminal of the build environment:

***bitbake** <name_of_ image_ recipe>*

bitbake core-image-purs

Builds the *initramfs* that is used for Emtrion devices' update mechanism. As the image is included by the other image emtrion-devkit-image, the image is automatically built while bitbaking this image, but only if the image was still not yet built. For this reason the image has to build explicitly, if any changes were made before building one of the other images.

bitbake emtrion-devkit-image

Builds the image for emcon-mx8mm. It creates a root file system with all the required components as well as QT5 support. Additionally it includes the initramfs, the kernel and device tree.

5.1 Output files

During the build process various objects and images are created. However, the most relevant images are installed in:

<BUILD_DIR>/tmp/deploy/images/<MACHINE> and **<BUILD_DIR> /tmp/deploy/sdk**.

The exact names of the images are listed below. Note: Some of them are symbolic links.

Images	Description
Image*	Kernel
imx8mm-emcon-avari.dtb*	Device tree supporting Avari baseboard
imx8mm-emcon-bvari.dtb*	Device tree supporting Bvari baseboard
rfs_image-<MACHINE>.rootfs.tar.gz	Root file system archive
rfs_image-<MACHINE>.rootfs.ext4	Root file system in ext4 format
ramdisk-<MACHINE>.cpio.gz	Ramdisk for update mechanism
imx-boot*	Packaged bootloader which can directly be flashed
imx-boot-tools	Directory with the separate bootloader files
fsl-xwayland-emtrion-glibc-x86_64-rfs_image-aarch64-toolchain-3.0.sh	SDK installer

Table 5.1: *Build output description*

(*) means a symbolic link

5.2 Root File System

As shown in the list above, the output of the root file system is a tar.gz archive. You can decompress it by the tar command. For testing or development we recommend to decompress the archive to the **<NFS_SHARE>** and then starting the device via *nfs*. Navigate to the directory **<BUILD_DIR>** and call

sudo tar xvzf tmp/deploy/images/<MACHINE> /name_of_rootfs_archive –C <NFS_ROOTFS>

Don't forget "sudo" otherwise the kernel won't be able to modify the files during starting of the system.

5.3 Boot directory

The directory structure of the root file system includes a directory called boot. In addition to the **kernel image, device tree and ramdisk** (restore root file system) a file **uboot_script** is located there.

This file implements some U-Boot command sequences. It can be used for the purpose of updating and booting the RootFS by using NFS.

However, the environment of the U-Boot has to be set up before. This is discussed in detail in the chapter concerning booting.

Alternatively, **imx-boot and rootfs** archive can later be stored in this directory to flash eMMC using NFS in linux environment.

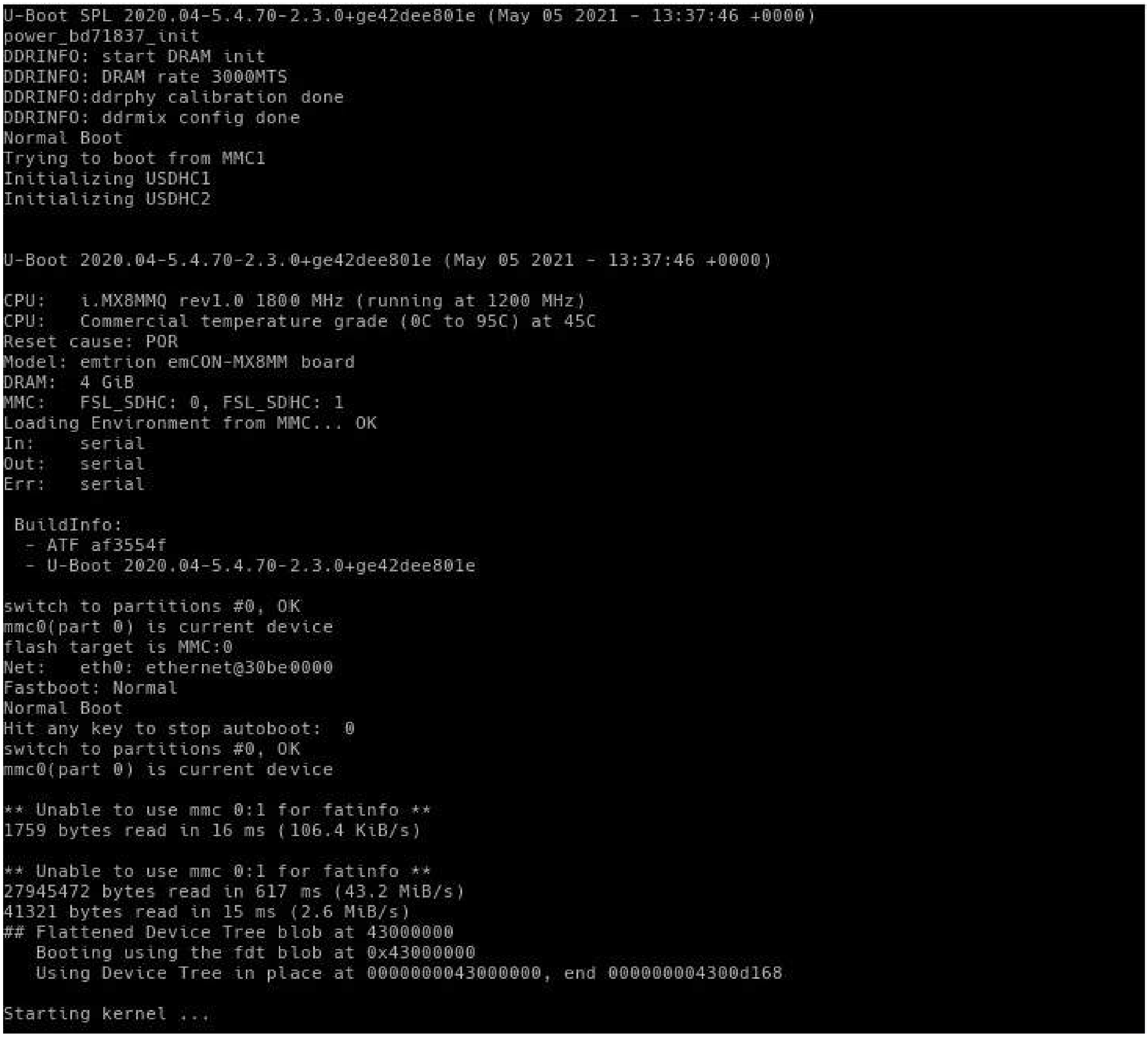
6 Device Startup

Connect the developer kit to the serial port attached to the virtual machine and your network. Open a console in the VM and open a serial terminal by entering

picocom -–b 115200 /dev/ttySx

In the case of using an USB serial adapter replace it by the corresponding **ttyUSBx**.

The developer kit is ready to be powered on. The terminal should then show messages for U-Boot-SPL, U-boot and Linux kernel.

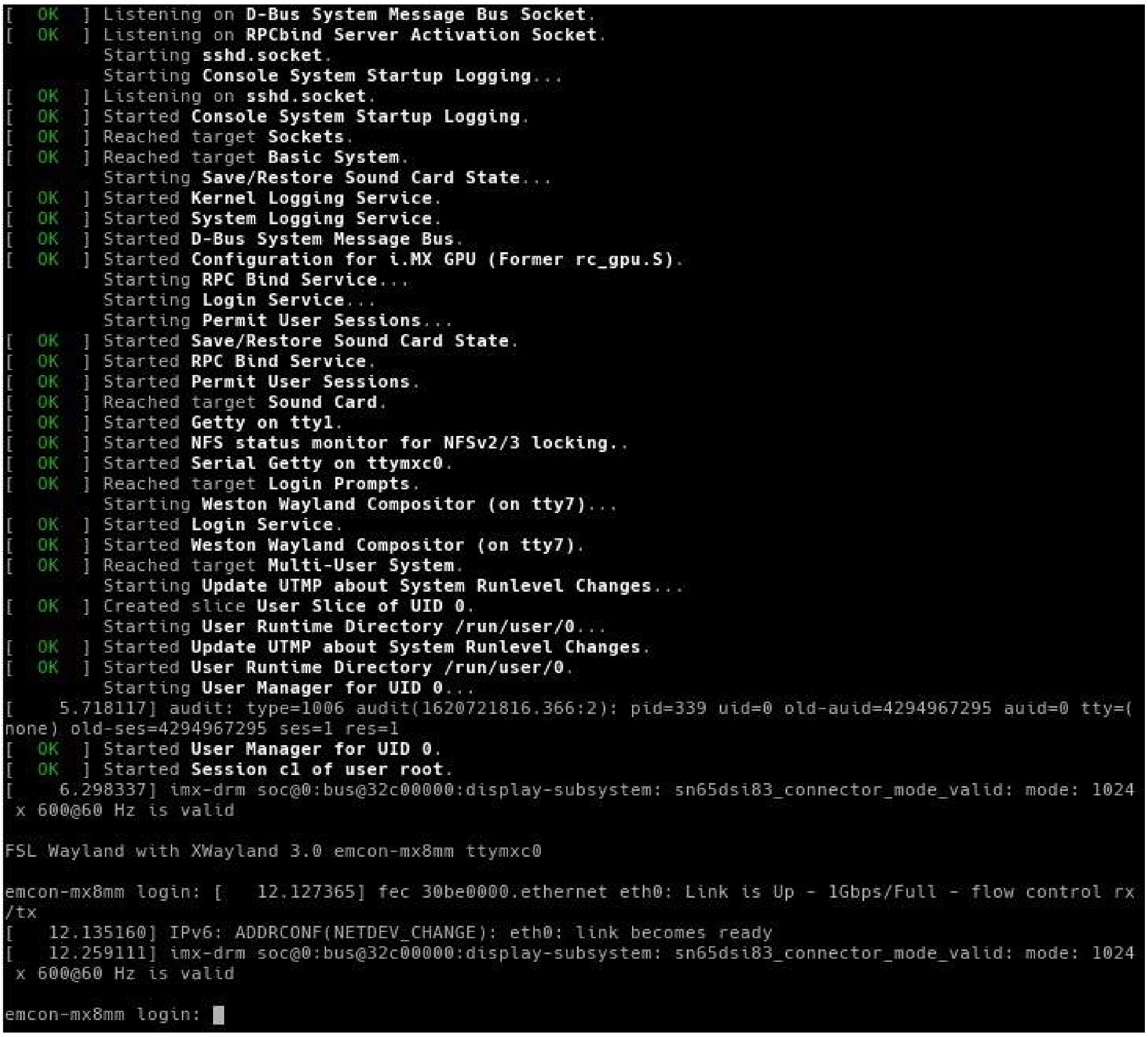


```
U-Boot SPL 2020.04-5.4.70-2.3.0+ge42dee801e (May 05 2021 - 13:37:46 +0000)
power_bd71837_init
DDRINFO: start DRAM init
DDRINFO: DRAM rate 3000MTS
DDRINFO:ddrphy calibration done
DDRINFO: ddrmix config done
Normal Boot
Trying to boot from MMC1
Initializing USDHC1
Initializing USDHC2


U-Boot 2020.04-5.4.70-2.3.0+ge42dee801e (May 05 2021 - 13:37:46 +0000)

CPU:   i.MX8MMQ rev1.0 1800 MHz (running at 1200 MHz)
CPU:   Commercial temperature grade (0C to 95C) at 45C
Reset cause: POR
Model: emtrion emCON-MX8MM board
DRAM:  4 GiB
MMC:   FSL_SDHC: 0, FSL_SDHC: 1
Loading Environment from MMC... OK
In:    serial
Out:   serial
Err:   serial

 BuildInfo:
  - ATF af3554f
  - U-Boot 2020.04-5.4.70-2.3.0+ge42dee801e

switch to partitions #0, OK
mmc0(part 0) is current device
flash target is MMC:0
Net:   eth0: ethernet@30be0000
Fastboot: Normal
Normal Boot
Hit any key to stop autoboot:  0
switch to partitions #0, OK
mmc0(part 0) is current device

** Unable to use mmc 0:1 for fatinfo **
1759 bytes read in 16 ms (106.4 KiB/s)

** Unable to use mmc 0:1 for fatinfo **
27945472 bytes read in 617 ms (43.2 MiB/s)
41321 bytes read in 15 ms (2.6 MiB/s)
## Flattened Device Tree blob at 43000000
   Booting using the fdt blob at 0x43000000
   Using Device Tree in place at 0000000043000000, end 000000004300d168

Starting kernel ...
```

Figure 6.1: *Device Startup example*

After the developer kit is booted the user is prompted for login:

emCON-MX8MM **login: root**



```
[  OK  ] Listening on D-Bus System Message Bus Socket.
[  OK  ] Listening on RPCbind Server Activation Socket.
         Starting sshd.socket.
         Starting Console System Startup Logging...
[  OK  ] Listening on sshd.socket.
[  OK  ] Started Console System Startup Logging.
[  OK  ] Reached target Sockets.
[  OK  ] Reached target Basic System.
         Starting Save/Restore Sound Card State...
[  OK  ] Started Kernel Logging Service.
[  OK  ] Started System Logging Service.
[  OK  ] Started D-Bus System Message Bus.
[  OK  ] Started Configuration for i.MX GPU (Former rc_gpu.S).
         Starting RPC Bind Service...
         Starting Login Service...
         Starting Permit User Sessions...
[  OK  ] Started Save/Restore Sound Card State.
[  OK  ] Started RPC Bind Service.
[  OK  ] Started Permit User Sessions.
[  OK  ] Reached target Sound Card.
[  OK  ] Started Getty on tty1.
[  OK  ] Started NFS status monitor for NFSv2/3 locking..
[  OK  ] Started Serial Getty on ttymxc0.
[  OK  ] Reached target Login Prompts.
         Starting Weston Wayland Compositor (on tty7)...
[  OK  ] Started Login Service.
[  OK  ] Started Weston Wayland Compositor (on tty7).
[  OK  ] Reached target Multi-User System.
         Starting Update UTMP about System Runlevel Changes...
[  OK  ] Created slice User Slice of UID 0.
         Starting User Runtime Directory /run/user/0...
[  OK  ] Started Update UTMP about System Runlevel Changes.
[  OK  ] Started User Runtime Directory /run/user/0.
         Starting User Manager for UID 0...
[    5.718117] audit: type=1006 audit(1620721816.366:2): pid=339 uid=0 old-auid=4294967295 auid=0 tty=(
none) old-ses=4294967295 ses=1 res=1
[  OK  ] Started User Manager for UID 0.
[  OK  ] Started Session c1 of user root.
[    6.298337] imx-drm soc@0:bus@32c00000:display-subsystem: sn65dsi83_connector_mode_valid: mode: 1024
 x 600@60 Hz is valid

FSL Wayland with XWayland 3.0 emcon-mx8mm ttymxc0

emcon-mx8mm login: [   12.127365] fec 30be0000.ethernet eth0: Link is Up - 1Gbps/Full - flow control rx
/tx
[   12.135160] IPv6: ADDRCONF(NETDEV_CHANGE): eth0: link becomes ready
[   12.259111] imx-drm soc@0:bus@32c00000:display-subsystem: sn65dsi83_connector_mode_valid: mode: 1024
 x 600@60 Hz is valid

emcon-mx8mm login: █
```

Figure 6.2: *Device login example*

Device Network Setup

By default, the developer kit is setup to use a dhcp server. This is configurable by a bootloader environment variable "ip-method". This variable can have the values *"dhcp"* or *"static"*.

You can check if there is a valid ip address with the command **ip**.

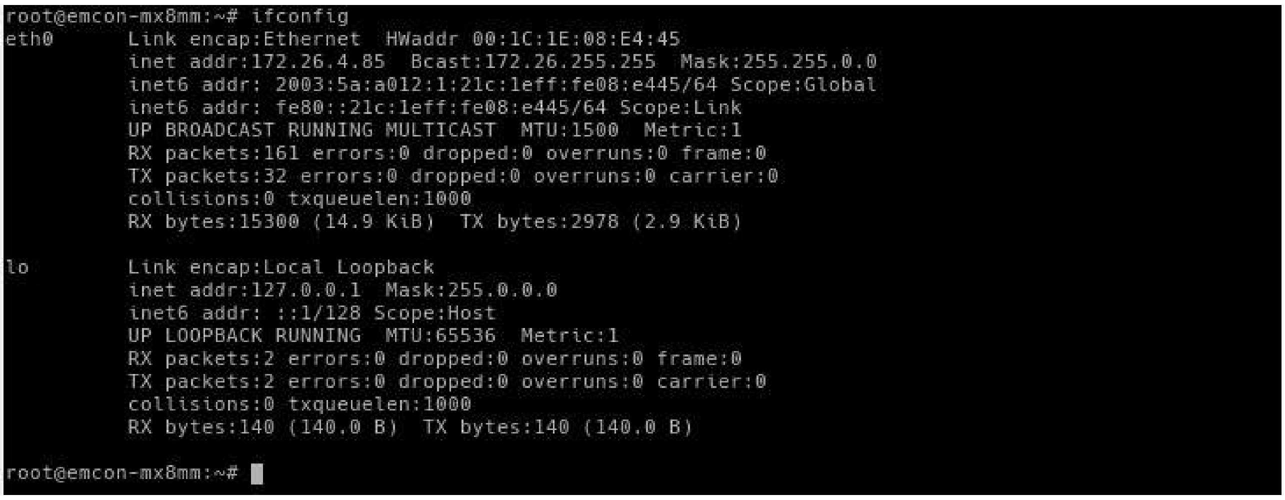

```
root@emcon-mx8mm:~# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:1C:1E:08:E4:45
          inet addr:172.26.4.85  Bcast:172.26.255.255  Mask:255.255.0.0
          inet6 addr: 2003:5a:a012:1:21c:1eff:fe08:e445/64 Scope:Global
          inet6 addr: fe80::21c:1eff:fe08:e445/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:161 errors:0 dropped:0 overruns:0 frame:0
          TX packets:32 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:15300 (14.9 KiB)  TX bytes:2978 (2.9 KiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:2 errors:0 dropped:0 overruns:0 frame:0
          TX packets:2 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:140 (140.0 B)  TX bytes:140 (140.0 B)

root@emcon-mx8mm:~#
```

Figure 6.3: *Device Network example*

If the setup is not correct the user should do it manually. Please check the description of the bootloader configuration on how to set up the variable *"ip-method"*.

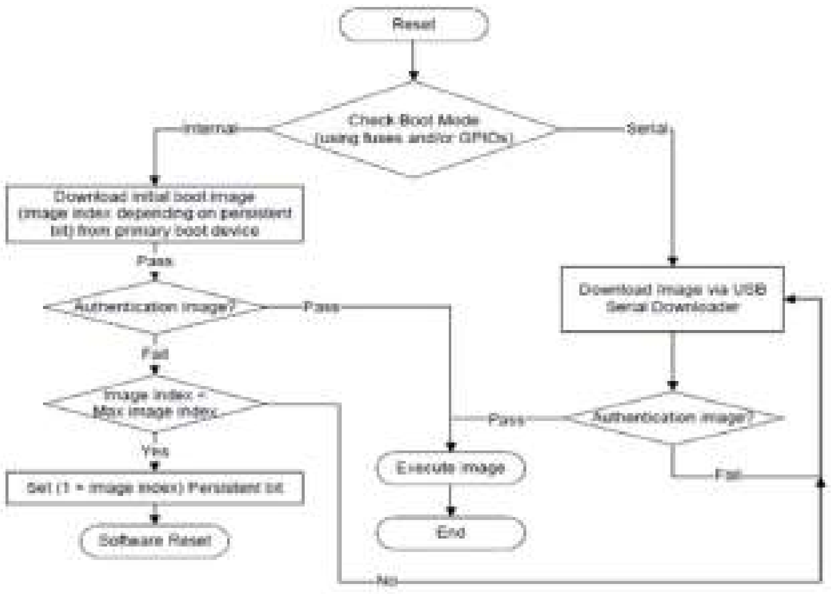
7 Booting, updating and restoring

The Emtrion emCON-MX8MM development kit can be booted, updated or restored using various methods. The U-boot is one of the most popular methods, where using the already integrated U-boot image, the device can be controlled with a few user defined variables, during boot time. However, this can only be used in presence of a working U-boot.

The boot process is initiated by a power on reset, which is followed by the ROM looking for a valid image in the possible locations, depending on the states of various registers, eFuses and GPIOs. The boot mechanisms supported can vary from NOR flash, eMMC, SD cards, serial downloader.

The module has 4 DIP switches acting as the boot pins; where the last 3 pins are to be controlled for different boot options (e.g. 0010 for eMMC boot). The relevant pin positions can be found in the hardware manual for the module. The baseboards also contain DIP switches which have to be in proper position to allow the module to boot (e.g. 10100000 for booting emCON-MX8MM). This can be obtained from the respective hardware manuals for the baseboards (Avari or Bvari).

The general flow of boot for the module as obtained from NXP reference document is as follows:



Reset
Check Boot Mode (using fuses and/or GPIOs)
Internal
Serial
Download initial boot image (image index depending on persistent bit) from primary boot device
Pass
Authentication image?
Pass
Fail
Image index < Max image index
Yes
Set (1 + image index) Persistent bit
Software Reset
Download Image via USB Serial Downloader
Authentication image?
Pass
Fail
Execute image
End
No

Figure 7.1: *Boot flow in iMX8M Mini*

7.1 Default boot

By default, the emCON-MX8MM contains the bootloaders in the integrated **eMMC**. They are pre flashed in the fixed boot0 and boot1 partitions of the eMMC. The boot pin positions on the module for this boot should be "0010". After connection to the workstation via an USB to serial cable, with console port of the baseboard and pressing the reset switch should start the device as mentioned in the "Device Startup" chapter.

The **root file system** is stored in a user defined ext4 partition of the **eMMC**. After the bootloaders are loaded, the default boot command from U-boot results in a search for a valid root file system system in this eMMC to start up the linux kernel. If the autoboot doesn't launch the kernel, you can use **run flash_boot** command to start the kernel manually.

Emtrion has scripts integrated in the image which can be used to **update the Kernel, U-boot or the Rootfs** systems of the kit. Similarly there is a possibility of running a system restore command in U-boot to fetch data from the NFS server and push boot images to the flash memories connected to the board. It restores a system if something goes wrong with the rootfs. The U-boot also has the capability to make the device look for the rootfs in the NFS server and boot from it, making it less cumbersome, to test and modify a system during development.

7.2 U-boot

The basic task of U-Boot is to load the operating system from bulk memory into RAM and then start the kernel. It can also be used it to initiate an update of the kernel, the RootFS and of U-Boot itself. Furthermore, it can be configured to dictate from which medium the operating system is to be booted from, for example eMMC or NFS. The location of the root file system can also be specified.

7.2.1 Basic Uboot Operation

To work with U-Boot, first use a terminal program like picocom to connect to the serial line of the board. As soon as the U-Boot prompt appears in the terminal, U-Boot is ready to receive commands. The general U-Boot documentation can be found here: `http://www.denx.de/wiki/U-Boot/Documentation`

U-Boot has a set of environment variables which are used to store information needed for booting the operating system. Variables can contain details such as IP addresses, but they can also contain a whole script of actions to perform sequentially. By default the eMMC (MMC0) is the location of storing the u-boot variables. The following commands explain the basic handling of environment variables:

U-boot command	Explanation
printenv [variable]	This shows the value of the specified variable. If no variable is specified, the whole environment is shown.
setenv [variable] [value]	Set a variable to a specific value. If no value is specified, the variable gets deleted.
editenv [variable]	Edit/modify a variable with an existing value.
saveenv	Make your changes permanent, so they remain after power off or reboot.The environemnt is written in the NOR flash

Table 7.1: *Uboot commands*

7.2.2 Using U-Boot to change boot device or update the system

This section describes how U-Boot has to be setup for updating and booting.

The variable serverip has to be set to the IP-address of the VM. You can get the IP-address by prompting

ip a or **ifconfig**

in the terminal of your VM.

Take the IP-address of the corresponding network adapter and assign it to the variable serverip in the U-Boot console. The format of [IP-address] is dot decimal notation.

emCON-MX8MM U-Boot > setenv serverip <IP-address>

The **"bootcmd"** variable can be set in the environment variable to dictate the default command to be used, when the u-boot is started. This indicates the location of the root file system to start the kit. Tested valid examples for **bootcmd are "run flash_boot" or "run net_boot".**

To use the NFS for easy update and boot procedures, the **NFS** server and the client has to be correctly set up, which is described in the **Section 7.3**.

7.2.3 Updating, booting and restoring the system

7.2.3.1 Updating of the system (root file system, u-boot and kernel)

As the image archive contains the root file system as well as the kernel, updating of the system affects always both.

Before performing the update process, following steps have to be done.

On VM

Copy the rootfs tar.gz archive from the **<BUILD_DIR>** to the **<NFS_ROOTFS>** directory. Extract the archive. Ensure that the rootfs contains all the required images for the update. The required images are located in the boot directory inside rootfs. The minimum expected files are:

emcon-mx8mm device trees emPURS_plat script ramdisk-emcon-mx8mm.cpio.gz uboot_script Image or the Kernel

On U-Boot

On starting the Uboot, the NFS has to be set up properly. The process is mentioned in details in the Section 7.3. When this is successfully completed, the following commands can be executed.

To update rootfs:

Copy an image of the rootfs.tar.gz archive created with the Yocto from the **<BUILD_DIR>** to the **<NFS_ROOTFS>**/boot directory in the VM. The files mentioned as minimum requirements should also be present in the directory. Then run the following command.

emCON-MX8MM U-Boot > run restore_sys

This loads the rootfs to the eMMC of the development kit. This starts the update process. Please be patient as the process of fetching the root file system image via network and decompressing it to the flash storage can take a few minutes.

To update uboot:

Copy images of the imx-boot-emcon-mx8mm-sd.bin-flash_evk file created with the Yocto from the **<BUILD_DIR>** to the **<NFS_ROOTFS>**/boot directory in the VM. The files mentioned as minimum requirements should also be present in the directory. Then run the following commands.

```
1. run configure-ip
2. nfs ${loadaddr} ${nfsroot}/boot/imx-boot-emcon-mx8mm-sd.bin-flash_spl_uboot
3. dcache flush
4. mmc dev 0 1
5. mmc write ${loadaddr} 0x42 0xfff
6. mmc dev 0 0
```

Listing 7.1: *Uboot update on eMMC*

This loads the bootloaders to the eMMC. Reset the power to the device and on being powered on the new uboot should start.

7.2.3.2 Booting

The default boot method is already mentioned in the Section 7.1. However, there are alternate options for the system startup which is explained in this section.

Booting with root file system in SD card

SD card boot support has not been provided in the first release. However the user can place the SD card for convenient test runs. It is sufficient to create only 1 partition in the SD card. The root file system created from Yocto can now be added to the SD card. Insert an SD card into a card reader on the Linux based development system (VM). It will show up as /dev/sdb or /dev/sdc or similar. We will use /dev/sdX as an example. If it gets automounted (the command mount will give you a list of mounted devices and their mount point), first unmount it. You need to have root rights to do the following steps to create such an SD card as listed below:

1. sudo dd if=/dev/zero of=/dev/sdX bs=1M status=progress
2. sudo fdisk /dev/sdX
3. Write gpt partition table: g
4. Create a new partition: n
5. Partition number: 1
6. Start sector:default
7. End sector:default
8. Check partition:p
9. Write to disk:w
10. sudo mkdir -p /mnt/sd
11. sudo mount /dev/sdX1 /mnt/sd
12. cd /mnt/sd
13. tar xvzf <rootfs_file_name>.tar.gz
14. sync
15. sudo umount /dev/sdX*

Place the card in the SD card slot of the baseboard. The bootloader should be present in the eMMC. Power on the device and stop the auto boot in uboot with a random key press.

Then run the following commands.

```
1. ext2load mmc 0:1 ${loadaddr} /boot/uboot_script
2. env import -t ${loadaddr} ${filesize}
3. ext2load mmc 0:1 ${loadaddr} /boot/${image.linux}
4. ext2load mmc 0:1 $fdt_addr /boot/${image.devicetree}
5. setenv bootargs "console=ttymxc0,115200 root=/dev/mmcblk1p1 rootwait"
6. booti ${loadaddr} - ${fdt_addr}
```

Listing 7.2: *Boot from root file system in SD card*

Booting from NFS server

Set up the NFS system as mentioned in Section 7.3 with all the required directories. In order to boot from the NFS sever, the command is:

U-Boot # run net_boot

7.2.3.3 Restoring

There is a script attached to the uboot which can be used to restore a system from a NFS system, provided that the Uboot has loaded successfully. The <NFS_ROOTFS>/boot must contain the files mentioned in Section 7.2.3.1 for updating rootfs. The command is:

U-Boot # run restore_sys

This script is responsible for initiating the emPURS_plat script of Emtrion. It loads the new device tree, kernel and rootfs, after deleting the existing partitions in the flash memories. The boot partitions are formatted and renewed with the images inside the NFS server. The new rootfs is installed and realtime clock is reset from NTP servers. Finally the system is rebooted.

7.3 NFS Setup

Update of the u-boot, kernel or RootFS is done as mentioned earlier, using NFS. Avoiding the update process for test purpose after each modification while developing, it is recommended to use NFS for booting, too.

For that a NFS-Server must be available on the Host and a <NFS_SHARE> has to be exported. However, setting up a NFS-Server and exporting a NFS-share can be different from the linux distribution. Be sure to make this work correct, please inform yourself how this work has to be done at your distribution. For the VM used in this example, the following steps are advisable.

1. Get nfs server in the host system
   sudo apt-get install nfs-kernel-server
2. Edit in etc/exports
   /home/hico/nfs/emcon-mx8mm/rootfs *(rw,sync,no_subtree_check,crossmnt,no_root_squash)
3. Unpack rfs_image-emcon-mx8mm.rootfs.tar.gz in the folder mentioned above. Ensure that the boot folder contains the required images.
4. Restart server in host after nfs addition
   sudo service nfs-kernel-server restart

In the target system connected via picocom:

1. Stop autoboot by pressing a key during the startup
2. Run the following Uboot commands:
   emCON-MX8MM U-Boot >setenv serverip xxx.xxx.xxx.xxx (obtain host system IP for the VM)
   emCON-MX8MM U-Boot >setenv nfsroot /home/hico/nfs/emcon-mx8mm/rootfs
   emCON-MX8MM U-Boot >setenv ip-method static/dhcp (either static or dhcp)
   emCON-MX8MM U-Boot >setenv ipaddr xxx.xxx.xxx.xxx (set a target IP ,only for static)
   emCON-MX8MM U-Boot >setenv netmask xxx.xxx.xxx.xxx (only for static)
   emCON-MX8MM U-Boot >saveenv (to save the added variables)
   emCON-MX8MM U-Boot >printenv (to verify that the changes have been updated in the Uboot environment)

In general the <NFS_SHARE> has pointed to the unpacked RootFS. While booting or updating, the directory boot of the RootFS takes a central position. It includes all the needed components used by the different processes. Due to of the central position of the directory "boot", the need of an unpacked RootFS is not necessary while updating. In this case the <NFS_SHARE> must only contain a sub-directory boot which includes the required files mentioned in previous sections. If you want to update the RootFS using its archive, you also have to locate the archive there.

The basic structure of the <NFS_SHARE> looks like as following. **<NFS_SHARE>/boot**

8 Using ARM Cortex M4 processor

The iMX8M Mini being an MPU gives the user the flexibility to use the M4 coprocessor in conjuction with the A53 processor. The application has to be built to be suitable for the controller, along with the firmware.

An application can be created for M4 with resources from MCUXpresso SDK site.
`https://mcuxpresso.nxp.com/en/welcome`.
Configure the application properly for the emCON-MX8MM board. Build the application with the provided toolchains to obtain the binary. Then copy the M4 demo binary file to boot directory either in NFS or SD card root file system as desired. Power on the device and stop autoboot with a key press. In U-boot with NFS

1. run configure-ip
2. nfs 0x7e0000 $nfsroot/boot/imx8mm_m4_application.bin
3. bootaux 0x7e0000

This should lead to a message as follows:
Starting auxiliary core at 0x007E0000 ...
If Linux is then booted, on boot it shows:
[0.000000] M4 is started
Now the console on UART2/3/4 should show the output of the M4 with applications like picocom or Putty. If M4 is ran in parallel with the Linux kernel, the emtrion provided device tree would need modifications. e.g. UART4 can be used for the console output of the M4 processor, then it has to be disabled in device tree to prevent any conflict in resource sharing.

An idea about the modifications can be obtained from the M4 device tree provided for the iMX8MM evaluation board, **viz. fsl-imx8mm-evk-m4.dts**.

Info

Please note that all the available UART ports for the baseboards are not simple serial ports, but proprietary of emtrion and needs an adapter to be connected to the PC for monitoring. More details can be found in the respective baseboard hardware manuals.

9 Audio with Yocto

In Avari baseboard from Emtrion an audio codec SGTL5000 from Freescale is already incorporated. The emCON-MX8MM module can be combined with this and can be used directly for audio playback. Using the device tree and the Linux kernel patches, ALSA libraries and plugins, along with the firmware provided by NXP for the SDMA controller, the solution has been achieved.

To list the sound capable devices for playback or record, the following commands can be used:

root@emcon-mx8mm: aplay -l
root@emcon-mx8mm: arecord -l

The ouptut for playback devices should be similar to the following message:

```
**** List of PLAYBACK Hardware Devices ****
card 0: SGTL5000Card [SGTL5000-Card], device 0: 30020000.sai-sgtl5000 sgtl
    5000-0 []
Subdevices: 1/1
Subdevice #0: subdevice #0
```

Listing 9.1: *Playback device list*

Basic speaker test for checking the audio can be done by the following command and monitored using the headphone output.

root@emcon-mx8mm: speaker-test -t sine -f 440 -c 2

Raw audio files of the "wav" format can be played using something similar to the following command.

root@emcon-mx8mm: aplay -D sysdefault:CARD=SGTL5000Card /path to file/test.wav

Gstreamer can be used along with the ALSA sink plugins for mp3 audio playback. The command for Gstreamer should be similar to the following.

root@emcon-mx8mm: gst-launch-1.0 playbin3 uri=file:///media/test.mp3 audio-sink="alsasink"

Volume and miscellaneous control for audio can be performed using the **alsamixer** command in Linux Yocto.

10 SDK

In order to develop applications outside of the Yocto build system the host development system needs to be set up. For this purpose the YP offers several installation methods. One of the methods for creating an SDK is by using the build directory. Use the following command for performing.

bitbake emtrion-devkit-image –c populate_sdk

The result is a SDK installer containing the toolchain and the sysroot which includes and matches the target root file system. The installer is stored in

<BUILD_DIR>/tmp/deploy/sdk/

Installing the SDK

While running the SDK installer, the user is asked for the installation directory. The default location is **/opt/fsl-xwayland-emtrion/3.0**. The default can be left as it is and confirmed. From inside the location **<BUILD_DIR>** start the installer as follows.

./tmp/deploy/sdk/fsl-xwayland-emtrion-glibc-x86_64-rfs_image-aarch64-toolchain-3.0.sh

Default installation location: **/opt/fsl-xwayland-emtrion/3.0**

Requires a confirmation for installation from user. On confirmation, the script will start extracting the SDK and then show the message "Setting it up....Done", indicating that the script has finished successfully.

Setting up the SDK environment

Before one can start developing applications, the environment for the terminal or the application has to be set up. For that purpose a script is installed during the installation process of the SDK. The script is stored in the SDK's directory of **<SDK_DIR>**.

Performing the setup procedure, the script has to be sourced as follows.

source <SDK_DIR>/environment-setup-aarch64-fsl-linux

The environment is only valid in the context of the terminal where this script has been called.

10.1 Qt5 with Yocto development

To run Qt5 with wayland support, the following scripts have to be executed before starting with the development.

1. cd /etc/profile.d
2. ./qt-wayland.sh

However, the script should run automatically on login and if there is the emtrion recommended screen connected a Weston window will be started.

If the script fails to run, modifications can be done in the **weston.ini** configuration file located in the path: **/etc/xdg/weston**.

The following steps are to be followed for configuring the SDK in Qt Creator. This is a general method which is shown as an example and the user may have make adjustments depending on the system and the build environment.

The following is only an example on how to setup a Yocto Project standard SDK built using Debian distribution for a Linux x86_64 host in Qt Creator 4.2.0 based on **Qt 5.11.3** with GCC in a 64 bit machine:

1. Open the "Tools", "Options..." menu and select the **Build & Run** section
2. Use **/opt/fsl-xwayland-emtrion/4.14-thud/sysroots/x86_64-pokysdk-linux/usr/bin/qmake** as qmake location in **Qt versions** tab. The version for development is **5.11.3**, the latest supported by product. Add a name parameter to the Qt version for easy identification in the future e.g. qt5 sdk.
3. Use **/opt/fsl-xwayland-emtrion/4.14-thud/sysroots/x86_64-pokysdk-linux/usr/bin/aarch64-poky-linux/aarch64-poky-linux-g++** as C++ compiler path in the Compilers tab. Add a name to the manually added compiler for future reference.
4. Use **/opt/fsl-xwayland-emtrion/4.14-thud/sysroots/x86_64-pokysdk-linux/usr/bin/aarch64-poky-linux/aarch64-poky-linux-gcc** as C compiler path in the Compilers tab. Add a name to the manually added compiler for future reference.
5. Use **/opt/fsl-xwayland-emtrion/4.14-thud/sysroots/x86_64-pokysdk-linux/usr/bin/aarch64-poky-linux/aarch64-poky-linux-gdb** as debugger path in Debuggers tab
6. If CMake is required, use **/opt/fsl-xwayland-emtrion/4.14-thud/sysroots/x86_64-pokysdk-linux/usr/bin/cmake** as cmake path in CMake tab
7. Open the **Devices** section
8. In the **Devices** tab create a new device of type "Generic Linux Device", specify IP address and authentication details
9. Return to the **Build & Run** section
10. Create a new kit with name "**emcon-mx8mm**" selecting the configurations **/opt/fsl-xwayland-emtrion/4.14-thud/sysroots/x86_64-pokysdk-linux** as sysroot path.
11. Remove any **Qt mkspec** if it has been added by default.
12. Press Apply and exit Qt Creator
13. At this point the environment set up script has to be already executed in a terminal.
14. Start Qt Creator from the current command line, where the environment has been modified. Use the location where Qt has been installed in the host PC.
15. Another method is to copy the entire environment set up file and add it to the **Environment** variable in QT options. This gives the Qt creator the flexibility to be started from any shortcuts, not only the actual modified terminal.
16. Create a new project otherwise build and compile an existing project like "Qt Quick Demo- Clocks" from the Examples in QT creator.
17. By running the successfully built program on the target device emCON-MX8MM, the clocks should be visible in the screen attached the board.

Alternately, the **Qteverywheredemo** from Qt is also built in the image rootfs and located under the directory /usr/share. This demo can also be executed to see the results in a supported screen.

11 Further information

Online resources

Further information can be found on the Emtrion support pages.

`www.support.emtrion.de`

We support you

Emtrion offers different kinds of services, among them Support, Training and Engineering. Contact us at `sales@emtrion.com` if you need information or technical support.