

# emSBC-MX93 Yocto Manual

---

Yocto Based BSP Manual

© Copyright 2026 **emtrion GmbH**

All rights reserved. This documentation may not be photocopied or recorded on any electronic media without written approval. The information contained in this documentation is subject to change without prior notice. We assume no liability for erroneous information or its consequences. Trademarks used from other companies refer exclusively to the products of those companies.

Revision: Rev. 1 / 02.12.2025

Rev.	Date/Initial	Changes
1	22.11.2024/Mi	First Version of emSBC-MX93 Yocto Manual
1	26.08.2025/Mi	Second Version of emSBC-MX93 Yocto Manual
1	02.12.2025/Fre	Yocto Manual Updates for Customer-Accessible Build Setup

# Contents

<b>1</b>	<b>Introduction</b>	<b>4</b>
<b>2</b>	<b>Terms and definitions</b>	<b>6</b>
<b>3</b>	<b>Scope of the delivery</b>	<b>7</b>
<b>4</b>	<b>Setting up Yocto Buildsystem</b>	<b>9</b>
4.1	Installation Steps . . . . .	9
4.1.1	Installation Results . . . . .	9
4.2	Yocto Setup Script . . . . .	10
4.3	Emtrion's Yocto Layers . . . . .	10
<b>5</b>	<b>Image Creation</b>	<b>13</b>
5.1	Output files . . . . .	13
5.2	Flashing Bootloader and Root Filesystem Using UUU . . . . .	14
5.2.1	UUU Flashing Script Example . . . . .	14
<b>6</b>	<b>Device Startup</b>	<b>16</b>
<b>7</b>	<b>Bootting, Updating, and Restoring</b>	<b>20</b>
7.1	Default Boot . . . . .	20
7.2	Updating the System . . . . .	20
7.3	System Restore and NFS Boot . . . . .	21
7.4	U-boot . . . . .	21
7.4.1	Basic Uboot Operation . . . . .	21
7.4.2	Using U-Boot to modify the boot device or perform system updates . . . . .	21
7.4.3	Bootting from SD card . . . . .	22
7.5	NFS Setup . . . . .	23
7.6	Updating using swupdate . . . . .	24
7.6.1	Updating the Root Filesystem Using SWUpdate . . . . .	24
<b>8</b>	<b>Using ARM Cortex M33 processor</b>	<b>26</b>
8.1	Build a demo application . . . . .	26
8.1.1	Download the toolchain . . . . .	26
8.1.2	Install the SDK and toolchain . . . . .	26
<b>9</b>	<b>SDK</b>	<b>27</b>
9.1	Installing the SDK . . . . .	27
9.2	Setting up the SDK environment . . . . .	28
<b>10</b>	<b>Further information</b>	<b>29</b>

# 1 Introduction

Emtrion produces and offers a variety of baseboards and modules, all available on their its official website: <https://www.emtrion.de/en/>. One of the latest additions to the product lineup is the single-board computer **emSBC-MX93**. This manual provides a comprehensive guidance and tips for effectively using the Yocto Project development tools with this hardware.

The **emSBC-MX93** is built around the NXP **i.MX93 processor**. It supports open-source software development with a fully functional Linux kernel and can be tailored to specific requirements using the Yocto Project. The board features a powerful **Dual-Cortex A55** processor and a **Cortex-M33** coprocessor, offering robust performance for a wide range of applications.

The emSBC-MX93 integrates the emSTAMP-MX93 module directly onto its printed circuit board (PCB). It utilizes the i.MX9352CVV processor, which includes:

- A dual-core Cortex-A55 processor with a clock speed of 1.7 GHz.
- A Cortex-M33 coprocessor with a 250 MHz clock frequency.
- A Neural Processing Unit (NPU) for efficient AI task execution.

The emSBC-MX93 is compatible with the **Yocto Scarthgap LTS release (version 5.0)**, launched in April 2024. The Yocto layers provided by Freescale and emtrion are based on this Yocto version.

- **NXP Layers:** meta-freescale, meta-freescale-3rdparty, and meta-freescale-distro.
- **Emtrion Layer:** meta-emtrion, which provides additional functionality tailored to the hardware.

The BSP includes a Linux Kernel mirrored from the NXP's i.MX Linux repository, version **lf-6.6.3-1.0.0**, hosted on emtrion's Gitlab. This kernel is part of NXP's Long-Term Support (LTS) releases and is based on the upstream Linux **6.6** branch, customized with additional patches and optimizations for i.MX processors.

The developer kit is configured through a dedicated kernel defconfig and device tree, along with the required changes for the correct operation of the associated peripherals.

The U-Boot bootloader on the emSBC-MX93 is sourced from the i.MX U-Boot repository, using the branch **lf\_v2023.04**. This branch is based on upstream U-Boot v2023.04 with NXP patches It is also hosted on emtrion's Gitlab.

The following table outlines the key version numbers for tools and packages used in the development kit:

Name	Version
Linux iMX	6.6.3-emtrion
Uboot iMX	v2023.04
Yocto	Scarthgap v5.0.5
Bitbake	v2.8.0
Gcc version	13.3.0
Openssh	v9.6p1
Busybox	v1.36.1
systemd	v255.4
Gstreamer	1.22.5
Mesa	v24.0.7
Graphics support	OpenGL_ES 2.0
Graphics driver	Gcnano v6.4.9

**Table 1.1:** Packages and versions

This manual provides an overview of the developer kit, general information, and user instructions.

It is assumed that users of emtrion developer kits are already familiar with U-Boot, Linux, Yocto, application development and debugging. 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 services or getting support, please contact the emtrion sales department.

Given the complexity of the Yocto Project and the limited focus of this manual, an in-depth explanation of Yocto/OpenEmbedded is beyond scope. While Yocto/OpenEmbedded is a highly capable build system, it can also be challenging to master. For this reason, emtrion offers professional support for developer kit-related issues as well as training sessions on Yocto/OpenEmbedded, should the official documentation not be sufficient.

For further in-depth information, the following resources are recommended:

1. **Yocto Manual:** <https://www.yoctoproject.org/docs/3.0/ref-manual/> (A primary reference for questions related to Yocto).
2. **Openembedded:** <http://www.openembedded.org/> (Information about OpenEmbedded layers for Linux development).
3. **NXP:** <https://www.nxp.com> (Details about the NXP i.MX93 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](https://www.nxp.com/webapp/Download?colCode=L5.4.70_2.3.0_LINUX_DOCS&location=null) (Includes user guides for Linux, Yocto, graphics, VPU, as well as the reference manual).
5. **Yocto repositories:** <https://git.yoctoproject.org/> (Includes the main Yocto repository, **poky**, and other supplementary repositories).
6. **Openembedded repositories :** <https://git.openembedded.org/> (Includes the meta-openembedded layer and additional resources).
7. **Freescale repositories :** <https://github.com/Freescale> (Contains Freescale Yocto layer repositories).
8. **iMX repositories :** <https://source.codeaurora.org/external/imx> (NXP repositories for Linux and Uboot).

## 2 Terms and definitions

Term	Definition
Target	emSBC-MX93
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	The directory exported via NFS, used for updating the system and booting over the network
U-Boot	Bootloader, hardware initialization, updating images, starting OS
YP	Yocto Project
INST_DIR	Directory where Yocto and its meta-layers are installed
MACHINE	Specifies the target device for which the image is built. The machine name is emtrion-emsbc-imx93
BUILD_DIR	Machine dependent build directory
BSP	Board Support Package
SDK	Software Development Kit

**Table 2.1:** *Terms and definitions*

## 3 Scope of the delivery

Emtrion provides the required software, prebuilt binaries, and development tools through a cloud link. Purchasers receive access to this link, which points to a directory containing the **emSBC-MX93\_Scarthgap.tar.gz** archive. This archive includes a predefined directory structure with prebuilt images, documentation, scripts, and a ready-to-use cross-development SDK.

```

emSBC-MX93_Data
├── images
│   ├── emtrion-imx93-image-core-emtrion-emsbc-imx93.rootfs.ext4
│   ├── emtrion-imx93-image-core-emtrion-emsbc-imx93.rootfs.tar.gz
│   ├── emtrion-imx93-image-core-emtrion-emsbc-imx93.rootfs.wic.gz
│   ├── flash.bin
│   ├── Image
│   ├── imx93-emtrion-linux.dtb
│   └── u-boot-sd-2023.04-r0.bin
├── Manual
│   ├── HW
│   └── SW
├── scripts
│   ├── setup-environment
│   └── uuu_script
├── README
└── sdk
    ├── imx93-distro-glibc-x86_64-emtrion-imx93-image-core-cortexa55-emtrion-emsbc-imx93-toolchain-6.6-scarthgap.host.manifest
    ├── imx93-distro-glibc-x86_64-emtrion-imx93-image-core-cortexa55-emtrion-emsbc-imx93-toolchain-6.6-scarthgap-host.spdx.tar.zst
    ├── imx93-distro-glibc-x86_64-emtrion-imx93-image-core-cortexa55-emtrion-emsbc-imx93-toolchain-6.6-scarthgap.sh
    ├── imx93-distro-glibc-x86_64-emtrion-imx93-image-core-cortexa55-emtrion-emsbc-imx93-toolchain-6.6-scarthgap.target.manifest
    ├── imx93-distro-glibc-x86_64-emtrion-imx93-image-core-cortexa55-emtrion-emsbc-imx93-toolchain-6.6-scarthgap-target.spdx.tar.zst
    └── imx93-distro-glibc-x86_64-emtrion-imx93-image-core-cortexa55-emtrion-emsbc-imx93-toolchain-6.6-scarthgap.testdata.json
  
```

Figure 3.1: Delivery package layout

The delivery package contains the following components:

1. **emSBC-MX93\_Data/** – Main directory containing all board-related deliverables.
2. **images/** – Prebuilt binary images ready for deployment:
  - `emtrion-imx93-image-core-emtrion-emsbc-imx93.rootfs.ext4`: Root filesystem image in EXT4 format
  - `emtrion-imx93-image-core-emtrion-emsbc-imx93.rootfs.tar.gz`: Root filesystem archive
  - `emtrion-imx93-image-core-emtrion-emsbc-imx93.rootfs.wic.gz`: Complete disk image containing bootloader, kernel, and root filesystem in WIC format
  - `flash.bin`: Combined firmware image for flashing via the Universal Update Utility (UUU)
  - `Image`: Linux kernel image
  - `imx93-emtrion-linux.dtb`: Device Tree Blob for the emSBC-MX93
  - `u-boot-sd-2023.04-r0.bin`: U-Boot binary for SD card boot
3. **Manual/** – Documentation directory:
  - **HW/**: Hardware documentation
  - **SW/**: Software documentation, including emSBC-MX93 Yocto Manual
4. **scripts/** – Utility scripts for setup and flashing:
  - `setup-environment`: Script to prepare the Yocto build environment
  - `uuu_script`: Script for flashing the device using the Universal Update Utility (UUU)
5. **sdk/** – Cross-development Software Development Kit (SDK):

- \*.sh: Installer script for the Yocto SDK
- \*.manifest: Host and target package manifests
- \*.spdx.tar.zst: SPDX license information
- \*.json: Test data and metadata files

6. **README** – Provides essential information about the delivery package, including setup instructions and usage notes

**Key Features:**

- Ready-to-flash images for immediate deployment
- Prebuilt Linux kernel, device tree, and bootloader binaries
- Cross-compilation SDK for application development on a host system
- Hardware and software documentation
- Utility scripts for environment setup and device flashing
- Based on the Yocto Project **Scarthgap** release for the NXP i.MX93 platform

# 4 Setting up Yocto Buildsystem

## 4.1 Installation Steps

Detailed instructions for setting up the Yocto build environment and compiling images for the emSBC-MX93 are provided in the README file included in the delivery package (emSbc-MX93\_Scarthgap/README).

The README contains:

- Prerequisites for the host system and required development packages
- Step-by-step instructions for extracting the BSP tarball and initializing the Yocto workspace
- Guidelines for copying the setup scripts and meta-layers
- Commands to set up the build environment for the emtrion-emSbc-imx93 machine
- Instructions to build images, SDKs, and SWUpdate packages
- Common build commands and troubleshooting tips

Users are encouraged to follow the README carefully to ensure the build environment is configured correctly.

### 4.1.1 Installation Results

After executing the setup procedure described in the README file, the Emtrion Yocto build environment is prepared with all necessary layers and directories for building images for the emSBC-MX93.

The resulting workspace is organized as follows:

Location	Remarks
<b>yocto-mirror/</b>	Root directory of the Yocto workspace
– <b>sources/</b>	Contains all Yocto layers, including meta-emtrion, meta-freescale, meta-openembedded, meta-qt6, meta-imx, poky, and other required layers
– <b>build/</b>	Build directory generated by the setup-environment script for a specific machine
– <b>conf/</b>	Configuration files such as local.conf and bblayers.conf
– <b>tmp/</b>	Temporary compilation outputs created by BitBake
– <b>sstate-cache/</b>	Shared state cache for speeding up subsequent builds
– <b>cache/</b>	BitBake metadata cache
– <b>downloads/</b>	Downloaded source archives and external dependencies
– <b>setup-environment</b>	Script to initialize the build environment
– <b>yocto-build.git, meta-*-git, poky.git, etc.</b>	Git repositories for local manifests and layer sources

Table 4.1: Yocto workspace directory structure after installation

After sourcing the setup script, the prompt switches to the build directory, from where images can be compiled using BitBake commands. All required layers are already present under `sources/`, ensuring a fully configured environment for building custom root filesystems, images, SDKs, and SWUpdate packages.

### Info

Note that some directories (e.g., `tmp/`, `sstate-cache/`) are created during the first BitBake run.

## 4.2 Yocto Setup Script

### Usage for creating a new build directory:

```
DISTRO=<distro> MACHINE=<machine> source setup-environment <build-dir>
```

This usage creates the build directory `<build-dir>`, configures it for the specified `<machine>` and `<distro>`, and prepares the calling shell environment to run BitBake from the build directory.

For the emSBC-MX93 Scarthgap setup, execute the following in the root of the Yocto workspace (`yocto-mirror/`):

```
DISTRO=imx93-distro MACHINE=emtrion-emsbc-imx93 source setup-environment build
```

### Info

The setup script must be **sourced**, not executed directly. Running it directly (e.g., `./setup-environment`) will result in errors when initializing the build environment.

### Usage for an existing build directory:

```
source setup-environment <build-dir>
```

This usage prepares the shell environment to run BitBake from an existing build directory `<build-dir>` without modifying its configuration.

When running the setup script for the first time, the files `local.conf` and `bbayers.conf` are generated in the specified build directory.

## 4.3 Emtrion's Yocto Layers

The Yocto layers included in the delivery package are organized as follows. Each layer contains recipes, configurations, and scripts necessary for building images for the emSBC-MX93 platform.

Location	Remarks
<code>meta-emtrion</code>	Emtrion's core meta layer providing machine-specific settings, recipes, and images for the emSBC-MX93.
<code> — conf</code>	Configuration files and distribution settings for this layer.

<ul style="list-style-type: none"> <li> – distro <ul style="list-style-type: none"> <li> – imx93-distro.conf</li> <li> – include <ul style="list-style-type: none"> <li> – imx93-base.inc</li> <li> – imx93-preferred-env.inc</li> </ul> </li> <li> – layer.conf</li> </ul> </li> </ul>	Contains base settings, machine configuration defaults, and environment preferences.
– conf/machine	Machine-specific configuration files.
<ul style="list-style-type: none"> <li> – emtrion-emsbc-imx93.conf</li> <li> – include <ul style="list-style-type: none"> <li> – emsbc-imx93.inc</li> </ul> </li> </ul>	Contains definitions for the emSBC-MX93 machine.
– recipes-core	Core recipes for the layer.
<ul style="list-style-type: none"> <li> – psplash <ul style="list-style-type: none"> <li> – psplash_git.bbappend</li> <li> – files <ul style="list-style-type: none"> <li> – emtrion_logo.png</li> </ul> </li> </ul> </li> </ul>	Optional recipe to customize the boot splash screen.
– recipes-bsp	Board Support Package (BSP) recipes.
<ul style="list-style-type: none"> <li> – u-boot <ul style="list-style-type: none"> <li> – u-boot-emtrion_2023.04.bb</li> <li> – emtrion-u-boot-scr <ul style="list-style-type: none"> <li> – emtrion-u-boot-scr.bb</li> <li> – files <ul style="list-style-type: none"> <li> – boot.cmd</li> </ul> </li> <li> – imx93-11x11-emtrion_defconfig</li> </ul> </li> </ul> </li> </ul>	Contains U-Boot recipes, board-specific configuration, and boot scripts for the emSBC-MX93.
– recipes-images	Image recipes for the delivery package.
<ul style="list-style-type: none"> <li> – images <ul style="list-style-type: none"> <li> – emtrion-imx93-image-core.bb</li> <li> – files <ul style="list-style-type: none"> <li> – sw-description</li> </ul> </li> </ul> </li> </ul>	Defines a core image recipe with all required packages and dependencies for the emSBC-MX93.
– recipes-kernel	Linux kernel recipes.
<ul style="list-style-type: none"> <li> – linux <ul style="list-style-type: none"> <li> – emtrion-linux-emtrion-common.inc</li> <li> – linux-emtrion-imx93 <ul style="list-style-type: none"> <li> – defconfig</li> <li> – imx93-emtrion-linux.dts</li> </ul> </li> <li> – linux-emtrion-imx93_6.6.bb</li> </ul> </li> </ul>	Contains kernel recipe, machine-specific DTS, and defconfig file for building the Linux kernel.
– recipes-swupdate	SWUpdate framework integration.

<ul style="list-style-type: none"> <li> – swupdate <ul style="list-style-type: none"> <li> – emtrion-image-swu <ul style="list-style-type: none"> <li> – emtrion-emsbc-imx93 <ul style="list-style-type: none"> <li> – priv.pem</li> <li> – sw-description</li> <li> – update.sh</li> </ul> </li> <li> – emtrion-image-swu.bb</li> </ul> </li> <li> – swupdate <ul style="list-style-type: none"> <li> – defconfig</li> <li> – emtrion-emsbc-imx93 <ul style="list-style-type: none"> <li> – swupdate.cfg</li> <li> – swupdate.default</li> </ul> </li> <li> – public.pem</li> <li> – swupdate.service</li> </ul> </li> <li> – swupdate_2024.05.bbappend</li> </ul> </li> </ul>	<p>Provides recipes and configuration for integrating SWUpdate framework and signing updates.</p>
<ul style="list-style-type: none"> <li> – swupdate-autostart <ul style="list-style-type: none"> <li> – files <ul style="list-style-type: none"> <li> – board_part_info.conf</li> <li> – emsbc-mx93-swupdate.service</li> <li> – emsbc-mx93-swupdate.sh</li> </ul> </li> <li> – swupdate-autostart_1.0.bb</li> </ul> </li> </ul>	<p>Provides scripts and recipes for automatic SWUpdate service startup.</p>
<ul style="list-style-type: none"> <li> – scripts</li> </ul>	<p>Utility scripts for the layer.</p>
<ul style="list-style-type: none"> <li> – emtrion-imx93-setup-release.sh</li> </ul>	<p>Helper script for environment setup.</p>
<ul style="list-style-type: none"> <li> – wic <ul style="list-style-type: none"> <li> – image-imx93-emtrion.wks</li> </ul> </li> </ul>	<p>WIC scripts for image partitioning.</p> <p>Defines the partition layout and filesystem configuration for bootable images.</p>

## 5 Image Creation

After completing the Yocto build system setup described in the previous chapter, the next step is to build images for the emSBC-MX93.

The meta-emtrion layer includes a core image recipe specifically tailored for the emSBC-MX93 board. This recipe defines the essential packages, configurations, and dependencies required to create a minimal functional image.

To build an image, open a terminal within the configured build environment and run the BitBake command with the desired image recipe name:

```
bitbake <name_of_image_recipe>
```

For example, to build the core image for the emSBC-MX93:

```
bitbake emtrion-imx93-image-core
```

### 5.1 Output files

During the build process, various artifacts and images are generated. The most important outputs are stored in the following directories:

- `<BUILD_DIR>/tmp/deploy/images/<MACHINE>`: Contains the images built for the target machine.
- `<BUILD_DIR>/tmp/deploy/sdk`: Contains the generated SDK installer files.

The main output files relevant for flashing and development are listed below:

Image / File	Description
<b>Image</b>	Compiled kernel image for the target machine.
<b>imx93-emtrion-linux-emtrion-emsbc-imx93.dtb</b>	Device Tree Blob for the emSBC-MX93.
<b>emtrion-imx93-image-core-emtrion-emsbc-imx93-rootfs- {yyyymmddhhmmss}.tar.gz</b>	Root filesystem archive in tar format.
<b>emtrion-imx93-image-core-emtrion-emsbc-imx93-rootfs- {yyyymmddhhmmss}.ext4</b>	Root filesystem in ext4 format.
<b>emtrion-imx93-image-core-emtrion-emsbc-imx93-rootfs- {yyyymmddhhmmss}.wic.gz</b>	Complete disk image in WIC format (compressed), containing partitions and root filesystem; can be flashed via UUU or similar tools.
<b>u-boot-emtrion-emsbc-imx93.bin</b>	U-Boot bootloader for the emSBC-MX93, includes SPL and BL31.
<b>bl31-imx93.bin</b>	ARM Trusted Firmware (BL31) binary.
<b>imx-boot-emtrion-emsbc-imx93-sd.bin-flash_singleboot</b>	Packaged bootloader image suitable for direct SD/eMMC flashing.

<code>emtrion-image-swu-emtrion-emsbc-imx93.rootfs.swu</code>	SWUpdate image for updating the root filesystem on the target device.
<code>emtrion-imx93-image-core-emtrion-emsbc-imx93-_64-toolchain-scarthgap_v1.3.sh</code>	Generated SDK installer for cross-development.
<code>fw_env.config-emtrion-emsbc-imx93</code>	Flash environment configuration for U-Boot.

Table 5.1: Build output description

### Info

Note that some files are generated with timestamped names, symbolic links, or compressed formats. Always use the most recent version when flashing or packaging for deployment.

## 5.2 Flashing Bootloader and Root Filesystem Using UUU

To program the **emSBC-MX93**, the **UUU (Universal Update Utility)** tool is used to flash both the bootloader and the root filesystem. The Yocto build generates a bootloader binary (`flash.bin`) containing BL31, U-Boot SPL, and U-Boot, as well as a WIC disk image (`emtrion-imx93-image-core-emtrion-emsbc-imx93.rootfs.wic.gz`) containing the root filesystem and partition layout.

Connect the board to your host PC in **USB Serial Downloader (SDP) mode**. Then, execute the prepared UUU script. This script first flashes the bootloader to the device, and then programs the root filesystem from the 'wic' image. Using the UUU script automates these steps, eliminating manual partitioning and copying, and ensures that the system is flashed consistently and reliably.

### 5.2.1 UUU Flashing Script Example

The following simplified UUU script demonstrates flashing the bootloader and the WIC image to the emSBC-MX93:

```
uuu_version 1.2.39

# This command will be run when i.MX6/7 i.MX8MM, i.MX8MQ
SDP: boot -f flash.bin

# This command will be run when ROM support stream mode
# i.MX8QXP, i.MX8QM
SDPS: boot -f flash.bin

# Commands for SPL (deprecated SDPU example)
SDPU: delay 1000
SDPU: write -f flash.bin -offset 0x57c00
SDPU: jump

SDPV: delay 1000
SDPV: write -f flash.bin -skipspl
SDPV: jump
```

```
FB: ucmd setenv fastboot_dev mmc
FB: ucmd setenv mmcdev ${emmc_dev}
FB: ucmd mmc dev ${emmc_dev}
FB: flash -raw2sparse all emtrion-imx93-image-core-emtrion-emsbc-imx93.rootfs.wic
FB: flash bootloader flash.bin
FB: ucmd if env exists emmc_ack; then ; else setenv emmc_ack 0; fi;
FB: ucmd mmc partconf ${emmc_dev} 0 1 0

FB: ucmd mmc dev ${emmc_dev}

FB: ucmd mmc read ${loadaddr} 0x880 0x1000
FB: ucmd md.b ${loadaddr} 0x1000

FB: Done
```

This script ensures the bootloader and root filesystem are correctly installed, allowing the board to boot immediately after flashing.

## 6 Device Startup

Connect the developer kit to your host machine via the serial port and ensure it is also connected to the network. Open a serial terminal using the following command:

```
minicom -D /dev/ttyUSB0 -b 115200
```

Here, `-b 115200` sets the baud rate to 115200, which is the typical default for the emSBC-MX93 serial console.

Power on the developer kit. The serial terminal will display boot messages from the following stages:

- **U-Boot SPL:** Secondary Program Loader initializing DDR and early hardware.
- **U-Boot:** Main bootloader responsible for loading the Linux kernel.
- **Linux kernel:** Kernel startup messages and device initialization.

```
U-Boot SPL 2023.04-lf_v2023.04-00028-g54d4988dafc (Nov 26 2024 - 12:14:27 +0100)
SOC: 0xa1009300
LC: 0x2040010
PMIC: PCA9451A
PMIC: Low Drive Voltage Mode
DDR: 1866MTS
M33 prepare ok
Normal Boot
Trying to boot from BOOTROM
Boot Stage: Primary boot
image offset 0x0, pagesize 0x200, ivt offset 0x0
Load image from 0x4a400 by ROM_API
NOTICE: BL31: v2.8(release):lf-6.6.3-1.0.0-0-g8dbe28631
NOTICE: BL31: Built : 11:24:01, Oct 11 2024
```

Figure 6.1: Device SPL Startup example

```

U-Boot 2023.04-lf_v2023.04-00032-g7d0f88da508-dirty (Jul 09 2025 - 08:22:12 +0200)

CPU:   i.MX93(52) rev1.1 1700 MHz (running at 1692 MHz)
CPU:   Industrial temperature grade (-40C to 105C) at 36C
Reset cause: POR (0x1)
Model: emtrion emSBC-MX93 board
DRAM:  2 GiB
Core:  213 devices, 31 uclasses, devicetree: separate
MMC:   FSL_SDHC: 0, FSL_SDHC: 1
Loading Environment from MMC... *** Warning - bad CRC, using default environment

[*]-Video Link 0fail to find output device
probe video device failed, ret -19

        [0] lcd-controller@4ae30000, video
In:     serial
Out:    serial
Err:    serial

BuildInfo:
- ELE firmware version 1.2.0-38f309fe

switch to partitions #0, OK
mmc0(part 0) is current device
Device specific data not found in current environment. Searching for factory default data. Please wait...
switch to partitions #0, OK
mmc0(part 0) is current device

MMC read: dev # 0, block # 384, count 896 ... 896 blocks read: OK
switch to partitions #1, OK
mmc0(part 1) is current device
No valid Factory defaults data found. Cannot update environment.
UID: 0x415f0ee3 0xf3474f80 0x64fb7a8d 0xdb79e3cf
Flash target is MMC:0
Net:
Warning: ethernet@428a0000 (eth1) using random MAC address - 36:2a:2c:04:ff:33

Warning: ethernet@42890000 (eth0) using random MAC address - 22:67:3e:b5:c5:97
eth0: ethernet@42890000, eth1: ethernet@428a0000 [PRIME]
Fastboot: Normal
Normal Boot
Hit any key to stop autoboot:  0
Working FDT set to 83000000
libfdt fdt_path_offset() returned FDT_ERR_NOTFOUND
libfdt fdt_path_offset() returned FDT_ERR_NOTFOUND
libfdt fdt_path_offset() returned FDT_ERR_NOTFOUND
libfdt fdt_path_offset() returned FDT_ERR_NOTFOUND
switch to partitions #0, OK
mmc0(part 0) is current device
Scanning mmc 0:1...
Found U-Boot script /boot.scr
715 bytes read in 1 ms (698.2 KiB/s)
## Executing script at 83500000
35017216 bytes read in 135 ms (247.4 MiB/s)
44536 bytes read in 2 ms (21.2 MiB/s)
## Flattened Device Tree blob at 83000000
   Booting using the fdt blob at 0x83000000
Working FDT set to 83000000
   Using Device Tree in place at 0000000083000000, end 000000008300ddf7
Working FDT set to 83000000

```

Figure 6.2: Device u-boot Startup example

Once the kernel has booted, you should reach the login prompt of the target system. Enter the following credentials:

Login: root

```

Starting kernel ...
0.000000] Booting Linux on physical CPU 0x0000000000 [0x412fd050]
0.000000] Linux version 6.6.3-emtrion-g15903441be51 (oe-user@oe-host) (aarch64-poky-linux-gcc (GCC) 13.3.0, GNU ld (GNU Binutils) 2.42.0.20240723) #1 SMP PREEMPT Wed Apr 23 10:18:36 UTC 2025
0.000000] KASLR disabled due to lack of seed
0.000000] Machine model: emtrion emSBC-MX93 board
0.000000] efi: UEFI not found.
0.000000] Reserved memory: created CMA memory pool at 0x0000000b000000, size 256 MiB
0.000000] OF: reserved mem: initialized node linux,cma, compatible id shared-dma-pool
0.000000] OF: reserved mem: 0x0000000b00000000..0x0000000bfffffff (262144 KiB) map reusable linux,cma
0.000000] OF: reserved mem: 0x000000002021e000..0x000000002021efff (4 KiB) nonap non-reusable rsc-table@2021e000
0.000000] OF: reserved mem: 0x00000000a4000000..0x00000000a400ffff (32 KiB) nonap non-reusable vdev@vring@0a4000000
0.000000] OF: reserved mem: 0x00000000a4008000..0x00000000a400ffff (32 KiB) nonap non-reusable vdev@vring1@0a40008000
0.000000] OF: reserved mem: 0x00000000a4010000..0x00000000a401ffff (32 KiB) nonap non-reusable vdev@vring@0a4010000
0.000000] OF: reserved mem: 0x00000000a4018000..0x00000000a401ffff (32 KiB) nonap non-reusable vdev@vring1@0a4018000
0.000000] Reserved memory: created DMA memory pool at 0x00000000a4020000, size 1 MiB
0.000000] OF: reserved mem: initialized node vdev@buffer@0a4020000, compatible id shared-dma-pool
0.000000] OF: reserved mem: 0x00000000a4020000..0x00000000a411ffff (1024 KiB) nonap non-reusable vdev@buffer@0a4020000
0.000000] OF: reserved mem: initialized node ele-reserved@0a4120000, compatible id shared-dma-pool
0.000000] OF: reserved mem: 0x00000000a4120000..0x00000000a421ffff (1024 KiB) nonap non-reusable ele-reserved@0a4120000
0.000000] Reserved memory: created CMA memory pool at 0x00000000c0000000, size 256 MiB
0.000000] OF: reserved mem: initialized node ethosu_region@C0000000, compatible id shared-dma-pool
0.000000] OF: reserved mem: 0x00000000c0000000..0x00000000cfffffff (262144 KiB) map reusable ethosu_region@C0000000
0.000000] earlycon: lpuart32 at MMIO32 0x0000000044380000 (options '')
0.000000] printk: bootconsole [lpuart32] enabled
0.000000] NUMA: No NUMA configuration found
0.000000] NUMA: Faking a node at [mem 0x0000000000000000-0x00000000fffffff]
0.000000] NUMA: NODE_DATA [mem 0xffbc56c0-0xffbc7fff]
0.000000] Zone ranges:
0.000000]   DMA      [mem 0x0000000000000000-0x00000000fffffff]
0.000000]   DMA32   empty
0.000000]   Normal  empty
0.000000] Movable zone start for each node
0.000000] Early memory node ranges
0.000000]   node 0: [mem 0x0000000000000000-0x00000000a3fffffff]
0.000000]   node 0: [mem 0x00000000a4000000-0x00000000a421ffff]
0.000000]   node 0: [mem 0x00000000a4220000-0x00000000fffffff]
0.000000] Initmem setup node 0 [mem 0x0000000000000000-0x00000000fffffff]
0.000000] pscl: probing for conduit method from DT.
0.000000] pscl: PSCLv1-i detected in firmware.
0.000000] pscl: Using standard PSCL v0.2 function IDs
0.000000] pscl: MIGRATE_INFO_TYPE not supported.
0.000000] pscl: SMC Calling Convention v1.2
0.000000] percpu: Embedded 22 pages/cpu s50792 r8192 d31128 u90112
0.000000] detected VIPT I-cache on CPU0
0.000000] CPU features: detected: CIC system register CPU interface
0.000000] CPU features: detected: Virtualization Host Extensions
0.000000] CPU features: detected: Qualcomm erratum 1009, or ARM erratum 1286807, 2441009
0.000000] CPU features: detected: ARM errata 1105522, 1319367, or 1530923
0.000000] alternatives: applying boot alternatives
0.000000] kernel command line: console=ttyLPU0,115200 earlycon root=/dev/mmcblkp2 rootwait rw
0.000000] Dentry cache hash table entries: 262144 (order: 9, 2097152 bytes, linear)
0.000000] Inode-cache hash table entries: 131072 (order: 8, 1048576 bytes, linear)
0.000000] Fallback order for Node 0: 0
0.000000] Built 1 zonelists, nobllity grouping on. Total pages: 516096
0.000000] Policy zone: DMA
0.000000] mem auto-init: stack:all(zero), heap alloc:off, heap free:off
0.000000] software IO TLB: area num 2.
0.000000] software IO TLB: mapped [mem 0x00000000f9000000-0x00000000fd000000] (64MB)
0.000000] Memory: 1429116K/2097152K available (20800K kernel code, 1620K rwdata, 7650K rodata, 3960K init, 632K bss, 143748K reserved, 524288K cma-reserved)

```

Figure 6.3: Device login example

## Device Network Setup

By default, the developer kit is configured to obtain an IP address via **DHCP**. This behavior can be modified using the U-Boot environment variable `ip-method`, which can be set to either `dhcp` or `static`.

To verify that the device has obtained a valid IP address from the Linux system after boot, use the following command:

```
ip addr show
```

Alternatively, the older command:

```
ifconfig
```

```
root@emtrion-emsbc-imx93:~# ifconfig
eth0      Link encap:Ethernet  HWaddr 1A:99:45:A5:F7:22
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

eth1      Link encap:Ethernet  HWaddr 3A:B9:A9:8A:59:F3
          inet addr:172.26.1.12  Bcast:172.26.255.255  Mask:255.255.0.0
          inet6 addr: 2003:f9:5824:1:38b9:a9ff:fe8a:59f3/64 Scope:Global
          inet6 addr: fe80::38b9:a9ff:fe8a:59f3/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:907 errors:0 dropped:0 overruns:0 frame:0
          TX packets:179 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:147229 (143.7 KiB)  TX bytes:20534 (20.0 KiB)
          Interrupt:104

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:103 errors:0 dropped:0 overruns:0 frame:0
          TX packets:103 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:8949 (8.7 KiB)  TX bytes:8949 (8.7 KiB)
```

Figure 6.4: Device Network example

### Info

If the network is not configured correctly, the user must set it manually by adjusting the `ip-method` variable in the bootloader. For detailed instructions, please consult the bootloader configuration documentation.

# 7 Booting, Updating, and Restoring

The Emtrion emSBC-MX93 development kit can be booted, updated, or restored using various methods. One of the most commonly used methods is via U-Boot, which allows the device to be controlled using user-defined environment variables during the boot process. Note that this approach requires a functional U-Boot image to already be present on the device.

The boot process begins with a power-on reset, after which the ROM searches for a valid image in different locations, depending on the states of specific registers, eFuses, and GPIOs. Supported boot mechanisms include NOR flash, eMMC, SD cards, and serial downloader.

The boot device is selected using the double DIP switch acting as boot pins. The correct pin positions can be found in the hardware manual for the emSBC-MX93.

## 7.1 Default Boot

To boot from the eMMC, set the module's boot pins to the "00" position. After connecting the board's console port to a host machine via a USB-to-serial cable, pressing the reset button starts the device as described in the "Device Startup" chapter.

The **root filesystem** is stored in a user-defined ext4 partition on the **eMMC**. After the bootloaders are loaded, U-Boot executes the boot script `/boot.scr` from the eMMC. This script loads the Linux kernel image **Image** and its corresponding device tree blob (`imx93-emtrion-linux.dtb`) into memory, after which the kernel is started.

## 7.2 Updating the System

The Emtrion-provided **uuu** (Universal Update Utility) script can be used to update the **Kernel**, **U-Boot**, and **Root Filesystem (Rootfs)**. This script automates the flashing process, ensuring a consistent and reliable update of the device's eMMC.

- **Updating U-Boot:**

The command `FB: flash bootloader emtrion-emsbc-imx93-sd.bin-flash_singleboot` writes a new U-Boot binary to the appropriate location, updating the bootloader to the desired version.

- **Updating the Root Filesystem (Rootfs):**

The command `FB: flash -raw2sparse all emtrion-imx93-image-core-emtrion-emsbc-imx93.rootfs.wic` flashes a new root filesystem image, overwriting the existing rootfs to ensure the system runs with the updated files and configuration.

## 7.3 System Restore and NFS Boot

The device can also be restored using U-Boot commands that fetch images from an NFS server and write them to the connected flash memories. This allows the system to be restored in case of rootfs corruption or other failures.

Additionally, U-Boot can be configured to boot the root filesystem directly from an NFS server. This is particularly useful during development and testing, as it allows rapid modifications to the filesystem without repeatedly reflashing the eMMC.

## 7.4 U-boot

U-Boot's primary function is to load the operating system from bulk storage into RAM and launch the kernel. Additionally, it facilitates updates for the kernel, root filesystem, and U-Boot itself. It can be configured to determine the boot medium, such as eMMC or NFS, and allows the specification of the root filesystem's location.

### 7.4.1 Basic Uboot Operation

To interact with U-Boot, use a terminal program such as minicom to connect to the board via its serial interface. When the U-Boot prompt appears, the system is ready to accept commands. Detailed U-Boot documentation is available at <http://www.denx.de/wiki/U-Boot/Documentation>.

U-Boot uses a set of environment variables to store essential information required for booting the operating system. These variables can include details like IP addresses or even complete scripts that define a sequence of actions to execute. By default, the environment variables are stored in the eMMC (MMC0). The following commands illustrate the basic operations for managing these variables:

U-boot command	Explanation
<b>printenv</b> [variable]	This command displays the value of the specified variable. If no variable is provided, it will display the entire environment.
<b>setenv</b> [variable] [value]	This command sets a variable to a specified value. If no value is provided, the variable will be deleted.
<b>editenv</b> [variable]	This command allows you to edit or modify the value of an existing variable.
<b>saveenv</b>	This command saves your changes permanently, ensuring they persist after a power cycle or reboot. The environment is written to the NOR flash memory.

Table 7.1: Uboot commands

### 7.4.2 Using U-Boot to modify the boot device or perform system updates

This section explains how to configure U-Boot for updating and booting.

The `serverip` variable must be set to the IP address of the Host. To find the IP address, use the command `sudo ip a` or `sudo ifconfig`

in the terminal of your Host.

Take the IP-address of the corresponding network adapter and assign it to the variable `serverip` in the U-Boot console.

```
emSBC-MX93 U-Boot > setenv serverip <IP-address>
```

The **"bootcmd"** variable can be configured in the environment to define the default command executed when U-Boot starts. This command specifies the location of the root file system for booting the device. Valid examples of **bootcmd** include "run flash\_boot" or "run net\_boot".

- run **flash\_boot**: This would trigger a command to load the kernel and the root filesystem from the eMMC storage.
- run **net\_boot**: This would trigger a command to fetch the kernel and root filesystem from a remote NFS server.

To facilitate easy updates and boot procedures using **NFS**, both the NFS server and the client must be properly configured.

### 7.4.3 Booting from SD card

Users have the option to create a fully portable image to boot the device from an SD card. The images generated from Yocto can then be added to the SD card. Insert the SD card into a card reader on your Linux-based development system, where it will appear as /dev/sdb, /dev/sdc, or a similar name. For this example, we'll refer to it as /dev/sdX. If the SD card is automatically mounted (you can check using the mount command, which lists mounted devices and their mount points), be sure to unmount it first. Root privileges are required to follow the steps below for setting up the SD card:

1. sudo fdisk -l : lists all the available disks and their partitions.
2. sudo dd if=/dev/zero of=/dev/sdX bs=1M status=progress : erases all data on the device.
3. sudo fdisk /dev/sdX
4. Write gpt partition table: g
5. Create a new partition: n
6. Partition number: 1
7. Start sector:default
8. End sector:default
9. Check partition:p
10. Write to disk:w
11. sudo mkdir -p /mnt/sd
12. sudo mount /dev/sdX1 /mnt/sd
13. cd /mnt/sd
14. tar xvzf <rootfs\_file\_name>.tar.gz
15. sync
16. sudo umount /dev/sdX\*

Insert the SD card into the board's SD card slot. Ensure that the bootloader is already present on the eMMC. Set the boot pins to "10" to boot the system from the SD card. Power on the device and interrupt the auto-boot process by pressing any key, then run the following commands.

```

1 1. fatload mmc 1:${mmcpart} ${kernel_addr_r} ${image}
2 2. fatload mmc 1:1 ${fdt_addr_r} imx93-emtrion-linux.dtb
3 3. booti ${kernel_addr_r} - ${fdt_addr};

```

**Listing 7.1:** Boot from root file system in SD card

## 7.5 NFS Setup

The update process for U-Boot, the kernel, or the RootFS is performed using NFS. To streamline development and testing, it is also recommended to use NFS for booting as well, allowing you to avoid the update process after each modification.

To enable this, an NFS server must be set up on the host system, and an <NFS\_SHARE> must be properly exported. The process of setting up an NFS server and exporting an NFS share can vary depending on the Linux distribution being used. Ensure the setup is configured correctly by referring to the specific instructions for your distribution. For Ubuntu 22.04, the following steps are recommended.

1. Install NFS Server on the Host System

```
sudo apt-get install nfs-kernel-server
```

2. Edit the NFS Exports File, by adding the following entry to /etc/exports to export the desired directory:

```
/home/<USER_NAME>/nfs/emtrion-emsbc-imx93/rootfs *(rw, sync, no_subtree_check, crossmnt, no_root_squash)
```

3. Prepare the Root Filesystem: Unpack the rfs\_image-emsbc-mx93.rootfs.tar.gz file into the directory specified above (/home/<USER\_NAME>/nfs/emtrion-emsbc-imx93/rootfs).

Ensure that the boot folder within this directory contains all the required boot images.

4. Restart the NFS Server to apply the changes:

```
sudo service nfs-kernel-server restart
```

On the Target System (via Minicom):

1. Stop Autoboot : Interrupt the autoboot process by pressing any key during startup.

2. Run the Following U-Boot Commands:

Set the host system's IP address (NFS server IP)

```
emSBC-MX93 U-Boot > setenv serverip xxx.xxx.xxx.xxx (Replace xxx.xxx.xxx.xxx with the appropriate IP addresses.)
```

Set the NFS root directory path:

```
emSBC-MX93 U-Boot > setenv nfsroot /home/<USER_NAME>/nfs/emSBC-mx93/rootfs
```

Run the following command in U-Boot to set the fdtfile variable. This ensures that the appropriate device tree file is used during the boot process:

```
emSBC-MX93 U-Boot > setenv fdtfile imx93-emtrion-linux.dtb
```

Run the following command in U-Boot to set the correct bootargs:

```
emSBC-MX93 U-Boot > netargs=setenv bootargs console=ttyLP0,115200 earlycon ip=dhcp root=/dev/nfs rw nfsroot=<serverip>:<nfsroot>,nfsvers=4,tcp
```

Configure the IP method (static or DHCP):

```
emSBCMX93 U-Boot > setenv ip-method static/dhcp
```

For static IP only, Set the target system's IP address:

```
emSBC-MX93 U-Boot > setenv ipaddr xxx.xxx.xxx.xxx
```

For static IP only, Set the subnet mask **emSBC-MX893 U-Boot > setenv netmask xxx.xxx.xxx.xxx**

Save the environment variables:

```
emSBC-MX93 U-Boot > saveenv
```

Verify the changes

```
emSBC-MX93 U-Boot > printenv
```

In general, the <NFS\_SHARE> is configured to point to the unpacked RootFS. During the boot or update process, the boot directory within the RootFS plays a central role, as it contains all the components required by different processes.

Because of the importance of the boot directory, it is not necessary to have the entire RootFS unpacked when performing updates. In such cases, <NFS\_SHARE> only needs to include a sub-directory named boot, containing the required files as outlined in earlier sections.

If you want to update the RootFS using its archive, the archive file must also be located within <NFS\_SHARE>.

Basic Structure of <NFS\_SHARE>:

<NFS\_SHARE>/boot

This ensures that the NFS setup includes all the essential components for booting or updating.

After setting up the NFS system as mentioned, to boot from the NFS sever, the command is:

*U-Boot # run net\_boot*

## 7.6 Updating using swupdate

### 7.6.1 Updating the Root Filesystem Using SWUpdate

The emSBC-MX93 development kit supports updating the root filesystem using the **SWUpdate** framework. All relevant recipes and configuration files are provided in the Yocto meta-emtrion/recipes-support/swupdate layer, including the update script, certificates, and configuration files.

The SWUpdate service runs as a background daemon and can be managed using standard service commands.

Root file system updates can be applied this way:

#### Using the web interface:

Open a browser and navigate to `http://<DEVICE_IP>:8080` (replace <DEVICE\_IP> with the IP of your board). Drag and drop the .swu update file into the interface. The update will be applied automatically.

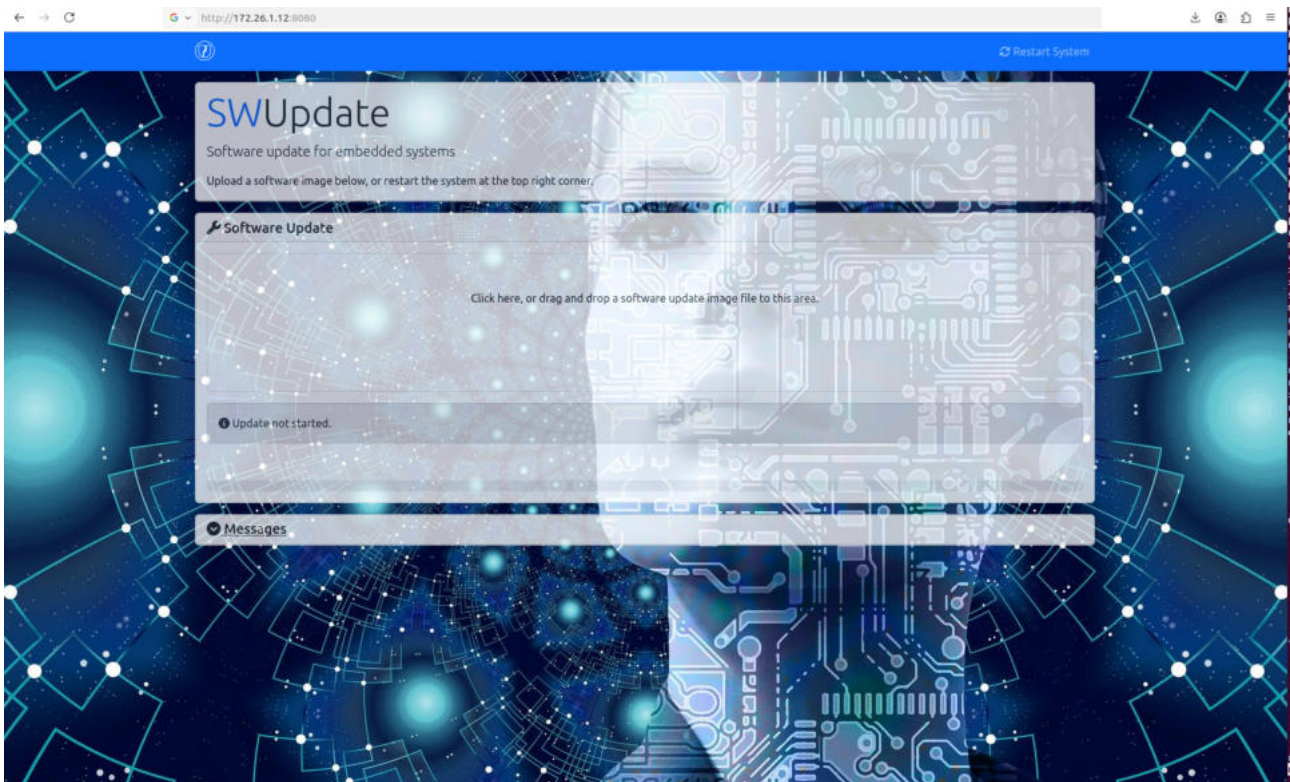


Figure 7.1: swupdate web interface

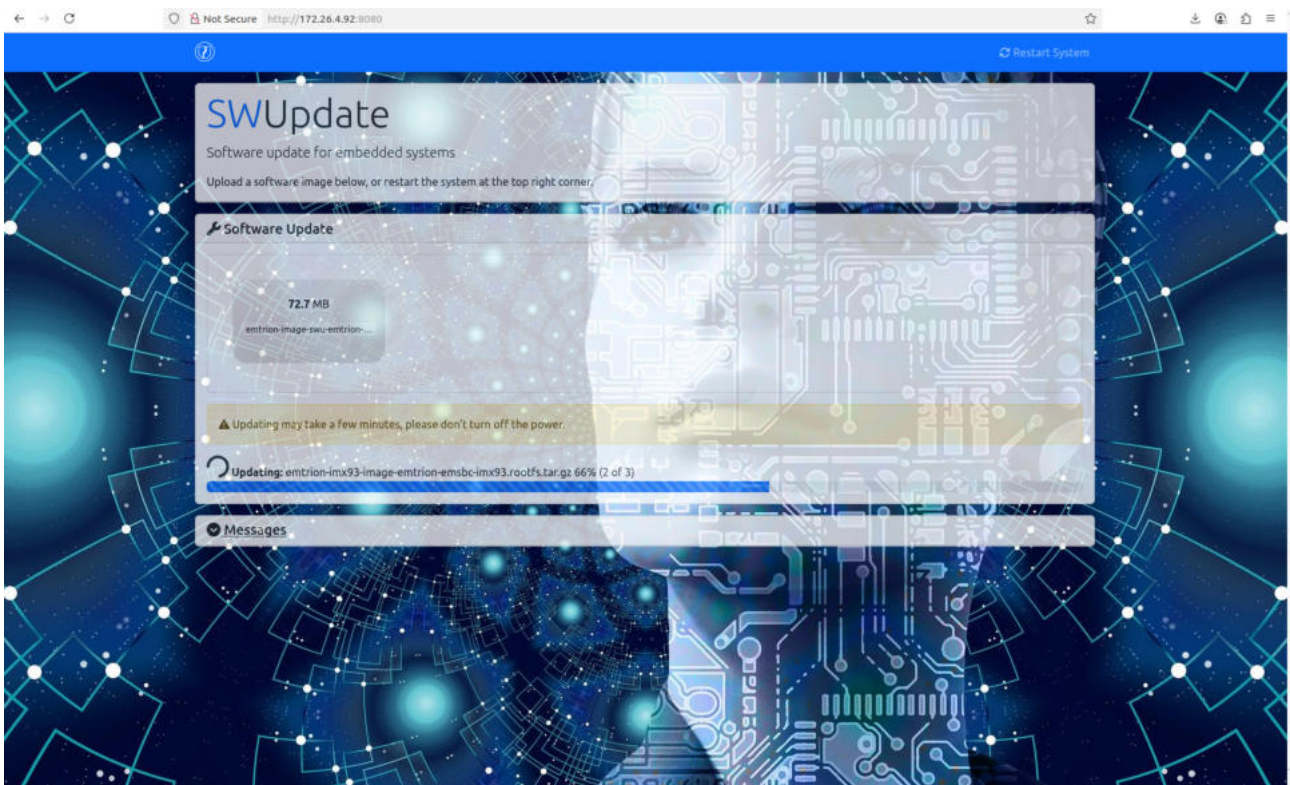


Figure 7.2: SWUpdate: Update in progress

Once the update is completed, the device will reboot with the new root filesystem.

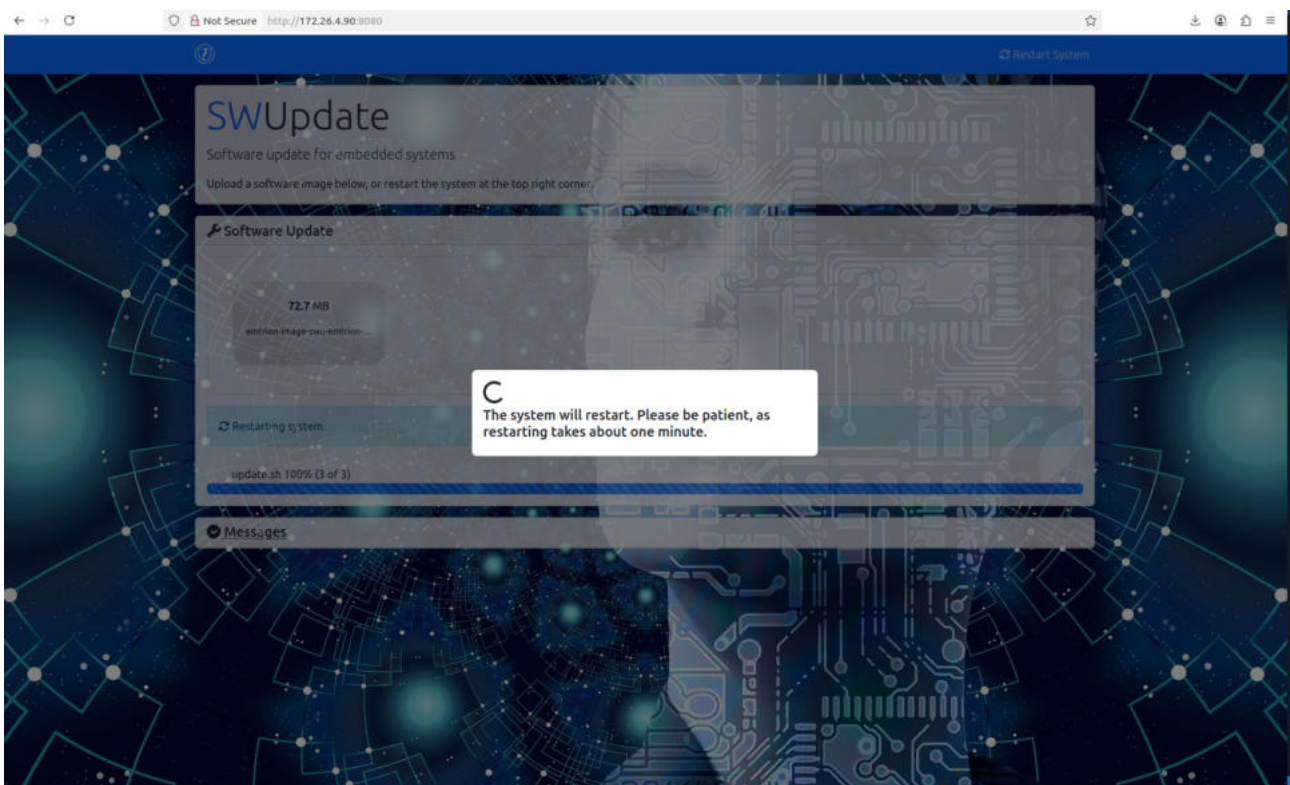


Figure 7.3: SWUpdate: Device rebooting after successful update

## 8 Using ARM Cortex M33 processor

The i.MX93, being an MPU, offers the flexibility to use the Cortex-M33 coprocessor alongside the Cortex-A55 main processor. The application needs to be specifically designed to work with the Cortex-M33 microcontroller, and the corresponding firmware must be developed accordingly to leverage both the main processor and the coprocessor efficiently.

NXP offers an SDK that facilitates application development for the Cortex-M33 core. The SDK provides examples demonstrating how to access the peripherals available on this subsystem.

1. Go to <https://mcuxpresso.nxp.com/en/builder>.
2. Click Boards > i.MX.
3. Select MCIMX93-EVK.
4. Click the Build MCUXpresso SDK button.
5. Select your Toolchain/IDE and Host OS.
6. Click Download SDK.

### 8.1 Build a demo application

#### 8.1.1 Download the toolchain

You must also download the toolchain to build your firmware application.

Go to <https://developer.arm.com/tools-and-software/open-source-software/developer-tools/gnu-toolchain/gnu-rm/downloads>.

Select the latest available version for "GNU Arm Embedded Toolchain".

#### 8.1.2 Install the SDK and toolchain

Decompress and install the SDK in your Linux PC.

```
tar -xvf <SDK_VER>_MEK-MCIMX93-EVK.tar.gz -C <SDK_VER>_MCIMX93-EVK>
```

Install the toolchain.

```
tar -xvf gcc-arm-none-eabi-7-2018-q2-update-linux.tar.bz2 -C <toolchain_install_path>
```

This link [https://www.digi.com/resources/documentation/digidocs/embedded/dey/4.0/cc93/yocto-develop-cortex-t\\_93.html](https://www.digi.com/resources/documentation/digidocs/embedded/dey/4.0/cc93/yocto-develop-cortex-t_93.html) offers a detailed, step-by-step guide for building a demo application using the provided toolchains to generate the binary.

## 9 SDK

To develop applications outside of the Yocto build system, the host development environment needs to be set up. The Yocto Project offers several methods for setting up the environment, one of which involves creating an SDK using the build directory. To do this, use the following command:

```
bitbake emtrion-imx93-image-core -c populate_sdk
```

This command generates an SDK installer that contains the toolchain and a sysroot matching the target root file system. The installer is located in:

```
<BUILD_DIR>/tmp/deploy/sdk/
```

### 9.1 Installing the SDK

An **SDK** folder is provided with the development kit and contains the SDK installer file:

```
imx93-distro-glibc-x86_64-emtrion-imx93-image-core-cortexa55-emtrion-emsbc-imx93-toolchain-6.6-scarthgap
```

When running the SDK installer, you will be prompted to select an installation directory. The default installation path is:

```
/opt/imx93-distro/6.6-scarthgap
```

This default can be accepted by pressing **Enter**.

To start the installation, navigate to the **<BUILD\_DIR>** and execute the installer using the following command:

```
./imx93-distro-glibc-x86_64-emtrion-imx93-image-core-cortexa55-emtrion-emsbc-imx93-toolchain-6.6-scarthgap.sh
```

After confirmation, the installer extracts the SDK and completes the setup. Once finished, the message:

```
"Setting it up... Done"
```

will be displayed, indicating that the SDK installation was successful.

```
NXP i.MX Release Distro SDK installer version 6.6-scarthgap
=====
Enter target directory for SDK (default: /opt/imx93-distro/6.6-scarthgap):
You are about to install the SDK to "/opt/imx93-distro/6.6-scarthgap". Proceed [Y/n]? y
[sudo] password for entissar.fredj:
Extracting SDK.....
.....done
Setting it up...done
SDK has been successfully set up and is ready to be used.
Each time you wish to use the SDK in a new shell session, you need to source the environment setup script e.g.
$ . /opt/imx93-distro/6.6-scarthgap/environment-setup-cortexa55-poky-linux
```

Figure 9.1: SDK installation

## 9.2 Setting up the SDK environment

Before starting application development, the build environment must be initialized. During the SDK installation, an environment setup script is installed in the SDK directory at `<SDK_DIR>`.

To configure the development environment, source the script using the following command:

```
source <SDK_DIR>/environment-setup-cortexa55-poky-linux
```

After sourcing the script, the cross-compilation tools, compiler flags, and environment variables required for development are automatically configured.

### Info

The environment setup is valid only for the current terminal session. A new terminal requires sourcing the script again.

## 10 Further information

### Online resources

Further information can be found on the Emtrion support pages.

<https://www.emtrion.de/en/>

<https://www.emtrion.de/en/contact-us/support/>

### We support you

Emtrion offers a variety of services, including Support, Training, and Engineering. For more information or technical support, feel free to contact us at [sales@emtrion.com](mailto:sales@emtrion.com).