

AN14093

Fast Boot on i.MX 8 and i.MX 9 Using Falcon Mode and Kernel Optimizations

Rev. 3.0 — 21 January 2025

Application note

Document information

Information	Content
Keywords	AN14093, Falcon Mode, kernel Optimizations, U-Boot Optimizations, Fast Boot, Falcon Boot, i.MX 93, i.MX 95, i.MX 8M, i.MX 8ULP, i.MX 8QXP, Linux
Abstract	This document guides how to reduce the boot time for the i.MX 8 and i.MX 9 family.



1 Introduction

This document guides how to reduce the Linux boot time for the:

- i.MX 8 family (i.MX 8QuadXPlus LPDDR4 MEK, i.MX 8ULP LPDDR4 EVK)
- i.MX 8M family (i.MX 8M Quad LPDDR4 EVK, i.MX 8M Mini LPDDR4 EVK, i.MX 8M Nano LPDDR4 EVK, and i.MX 8M Plus LPDDR4 EVK)
- i.MX 9 family (i.MX 93 LPDDR4 EVK and i.MX 95 LPDDR5 EVK)

The objectives of this document are as follows:

- Bootloader optimizations
- Linux kernel and user space optimizations
- Comparison between default and improved boot time on all platforms

1.1 Software environment

An Ubuntu 22.04 PC is assumed.

This application note applies to the Yocto Project BSP Scarthgap release and Linux BSP release [6.6.36_2.1.0](#) and [6.6.52_2.2.0](#).

Hardware setup and equipment:

- Development kit [NXP i.MX 8QXP MEK LPDDR4](#)
- Development kit [NXP i.MX 8ULP EVK LPDDR4](#)
- Development kit [NXP i.MX 8MQ EVK LPDDR4](#)
- Development kit [NXP i.MX 8MM EVK LPDDR4](#)
- Development kit [NXP i.MX 8MN EVK LPDDR4](#)
- Development kit [NXP i.MX 8MP EVK LPDDR4](#)
- Development kit [NXP i.MX 93 EVK for 11x11 mm LPDDR4](#)
- Development kit [NXP i.MX 95 EVK for 19x19 mm LPDDR5](#)
- Micro SD card: SanDisk Ultra 32 GB Micro SDHC I Class 10 was used for the current experiment
- Micro-USB (i.MX 8) or Type-C (i.MX 9) cable for the debug port.

2 General description

This section is an overview of the typical modifications required in order to achieve shorter boot times.

2.1 Reducing bootloader time

To reduce the bootloader time, choose either of the following two ways:

- **Removing the boot delay:** It saves about two seconds compared to the default configuration while requiring minimal changes. It leads to U-Boot skipping the wait for key press stage during boot.
- **Implement Falcon Mode:** It saves about four seconds compared to the default configuration. It enables the Second Program Loader (SPL) – part of U-Boot to load the kernel directly, skipping the full U-Boot.

2.2 Reducing Linux kernel boot time

To reduce the Linux kernel boot time, choose either of the following two ways:

- **Reduce console messages:** It saves about three seconds. Add `quiet` to the kernel command line.

- **Slim down the kernel by removing drivers and filesystems:** By default, the kernel image contains a lot of drivers and filesystems (ex: UBIFS) to enable the majority of the functionalities supported for the board. The list of included drivers and filesystems can be trimmed according to your use case.

2.3 Reducing user-space boot time

To reduce the user-space boot time, try the following way:

- **Start your application before systemd:** It saves about 600 ms. Launch the desired process as soon as possible, considering its dependencies.


2.4 Measurements

The scope of the measurements is between the board Power-On Reset (POR) and the start of the INIT process.

The setup used in these measurements is described in the *i.MX 9 Family – Boot-Time Measurement Methodology* (document [AN14205](#)).

[Table 1](#) describes the measured intervals:

Table 1. Measured intervals

Time point	Interval between pulses	Location of the pulse	Boot stages	
BootROM	nRST -> before ddr_init()	board/freescale/<board>/spl.c/board_init_f()	SPL	 Timeline
DDR initialization	before ddr_init() -> after ddr_init()	board/freescale/<board>/spl.c/board_init_f()		
SPL initialization + Load U-Boot image	after ddr_init() -> before image_entry()	common/spl/spl.c/jump_to_image_no_args()		
U-Boot initializations (init_sequence_f)	before image_entry() -> start init_sequence_r	common/board_r.c/board_init_r()	U-BOOT	
U-Boot initializations (init_sequence_r)	start init_sequence_r -> u-boot main_loop	common/main.c		
Boot sequence	u-boot main_loop -> before load_image	include/configs/<board>.h		
Kernel Image Load	before loadimage -> after loadimage	include/configs/<board>.h		
Kernel Boot Until INIT process	after loadimage -> /sbin/init	get the timestamp during kernel boot	Kernel	

3 Bootloader optimizations

This chapter includes the following information.

- [Section 3.1 "Default boot mode"](#)
- [Section 3.2 "Falcon mode"](#)

3.1 Default boot mode

[Figure 1](#) describes the default boot sequence. After power on or reset, the boards execute the **BootROM** (the primary program loader), stored in its Read Only Memory (ROM).

BootROM configures the System-on-Chip (SoC) by performing basic peripheral initializations such as Phase Locked Loops (PLLs), clock configurations, memory initialization (SRAM). Then, it finds a boot device from where it loads a bootloader image, which can include the following component: U-Boot SPL, Arm Trusted Firmware (ATF), U-Boot, and so on.

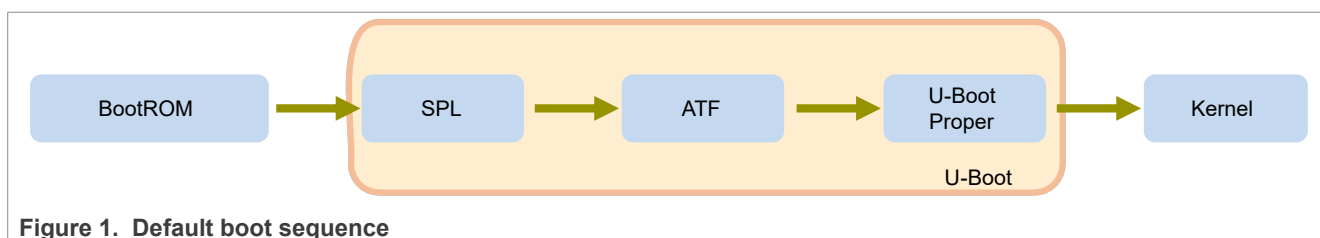


Figure 1. Default boot sequence

A typical U-Boot image does not fit inside internal SRAM, and therefore it is split into two parts: **SPL** and **U-Boot proper**.

SPL is the first stage of the bootloader, a smaller pre-loader that shares the same sources as U-Boot, but with a minimal set of code that fits into SRAM. SPL is loaded into SRAM. It configures and initializes some peripherals and, most importantly, DRAM. Subsequently, it loads the ATF and U-Boot proper into the DRAM. The final step is to jump to ATF, which, in turn, jumps to U-Boot proper.

ATF, included recently in i.MX8/9 families, provides a reference trusted code base for the Armv8 architecture. It implements various ARM interface standards, including Power State Coordination Interface (PSCI). The binary is typically included in the bootloader binary. It is started in the early stages of U-Boot. Without ATF, the kernel cannot setup the services which need to be executed in the Secure World environment.

U-Boot proper is the second stage bootloader. It offers a flexible way to load and start the Linux kernel and provides a minimal set of tools to interact with the board's hardware via a command line interface. It runs from DRAM, initializing additional hardware devices (network, USB, DSI/CSI, and so on). Then, it loads and prepares the Flattened Device Tree (FDT). The main task handled by the U-Boot is the loading and starting of the kernel image itself.

Linux kernel runs from DRAM and takes over the system completely. The U-Boot has no longer control over the system from this point onward.

3.2 Falcon mode

Falcon mode is a feature in U-Boot that enables fast booting by allowing SPL to directly start the Linux kernel. It completely skips the U-Boot loading and initialization, with the effect of reducing the time spent in the bootloader.

[Figure 2](#) illustrates the Falcon mode booting sequence.



Figure 2. Falcon mode boot sequence

To implement this mode, the main steps are:

- Activate specific configurations for Falcon.
- Prepare the FDT in advance.
- Configure the ATF to jump to kernel.
- Generate the FIT/container image containing the ATF and the kernel image.

3.2.1 Falcon mode enablement

To enable the Falcon mode, follow the steps described in the [README](#) file of the source code. Make sure that the correct branch is selected according to the intended BSP release.

3.2.2 Falcon mode implementation details

The steps described in [Section 3.2.1 "Falcon mode enablement"](#) enable the Falcon mode through modifications of U-Boot, ATF, and `mkimage` tool. This section describes what happens in the background.

• The U-Boot patch

In the *meta-imx-fastboot/recipes-bsp/u-boot/files* directory, you can find a patch and a configuration file for each platform. For more details about the configured parameters, see the U-Boot documentation.

Each *0001-<board>-add-falcon-mode-support.patch* file:

- Implements the `spl_start_uboot()` function, located in `uboot-imx/board/freescale/<board>/spl.c`, where `<board>` is: `imx8qxp_mek`, `imx8ulp_evk`, `imx8mq_evk`, `imx8mm_evk`, `imx8mn_evk`, `imx8mp_evk`, `imx93_evk`, or `imx95_evk`. This function checks if SPL should start the kernel or U-Boot. If any key is pressed during boot, the function returns 1, meaning that U-Boot must be started. Otherwise, SPL should start the kernel.
- Creates two new U-Boot variables `prepare_fdt` and `mmcargs_fastboot`, in the `include/configs/<board>.h` file. The `prepare_fdt` variable generates and saves the fixed FDT. The `mmcargs_fastboot` is used by the `prepare_fdt` to set the kernel arguments.
- The patch for the i.MX 95 in addition implements the `spl_fit_read()` function in the `arch/arm/mach-imx/imx9/scmi/soc.c` file. Since the USDHC controller is a non-secure master, it cannot access the DDR secure region. This function is required only for i.MX 95 and it handles the container image loading from the storage device (SD or eMMC) to DDR.

The *0001-imx8m-reset-ethernet-phy-in-spl.patch* file resets the Ethernet PHY for the i.MX 8M Family. To bring it up in the operational state in which Ethernet MAC can interact with the PHY, this must be reset before starting the kernel. The PHY is reset in the `board_init_r()` function, located in the `uboot-imx/common/spl/spl.c` file.

• The ATF patch

In the *meta-imx-fastboot/recipes-bsp/imx-atf/files* directory, you can find a patch for each platform.

The patch adds support for jumping directly to the kernel. Since ATF does not support to jump directly to the kernel on NXP platforms, the FDT address must be passed as an argument, in the `bl31_early_platform_setup2()` function, located in `imx-atf/plat/imx/imx8m/<board>/<board>_bl31_setup.c` for i.MX 8M Family and `imx-atf/plat/imx/<board>/<board>_bl31_setup.c` for i.MX 8 Family and i.MX 9 Family.

• The mkimage patch

The patches for the `mkimage` are located in the *meta-imx-fastboot/recipes-bsp/imx-mkimage/files* directory.

Each `0001-<board>-add-falcon-mode-support.patch` file:

- Creates two new targets in the `soc.mak` file, to generate:
 - The image containing the ATF and the kernel.
 - `uImage` — used by the `spl export` command during the FDT preparation.
- In addition, the patch for the i.MX 8M creates the `mkimage_fit_atf_kernel.sh` script used for generating the FIT image containing the ATF and the kernel, and add the `os` properly to `uboot-1` node of the U-Boot FIT image source (`u-boot.its`). This property is required when loading U-Boot (the case when `spl_start_uboot()` returns **1**) while Falcon Mode is enabled. Otherwise, the U-Boot fails to boot.

3.2.3 Memory map

This section describes the memory map during Falcon mode.

SPL is loaded by BootROM and runs from On-Chip RAM (OCRAM - the memory of the internal processor). SPL initializes the dynamic RAM (DDR), loads the ATF into OCRAM, then loads the kernel device tree and the kernel image into DDR. SPL has a reserved memory space in DDR, for malloc. This area must not be overwritten while in SPL.

Figure 3 presents the memory map for i.MX93, as an example.

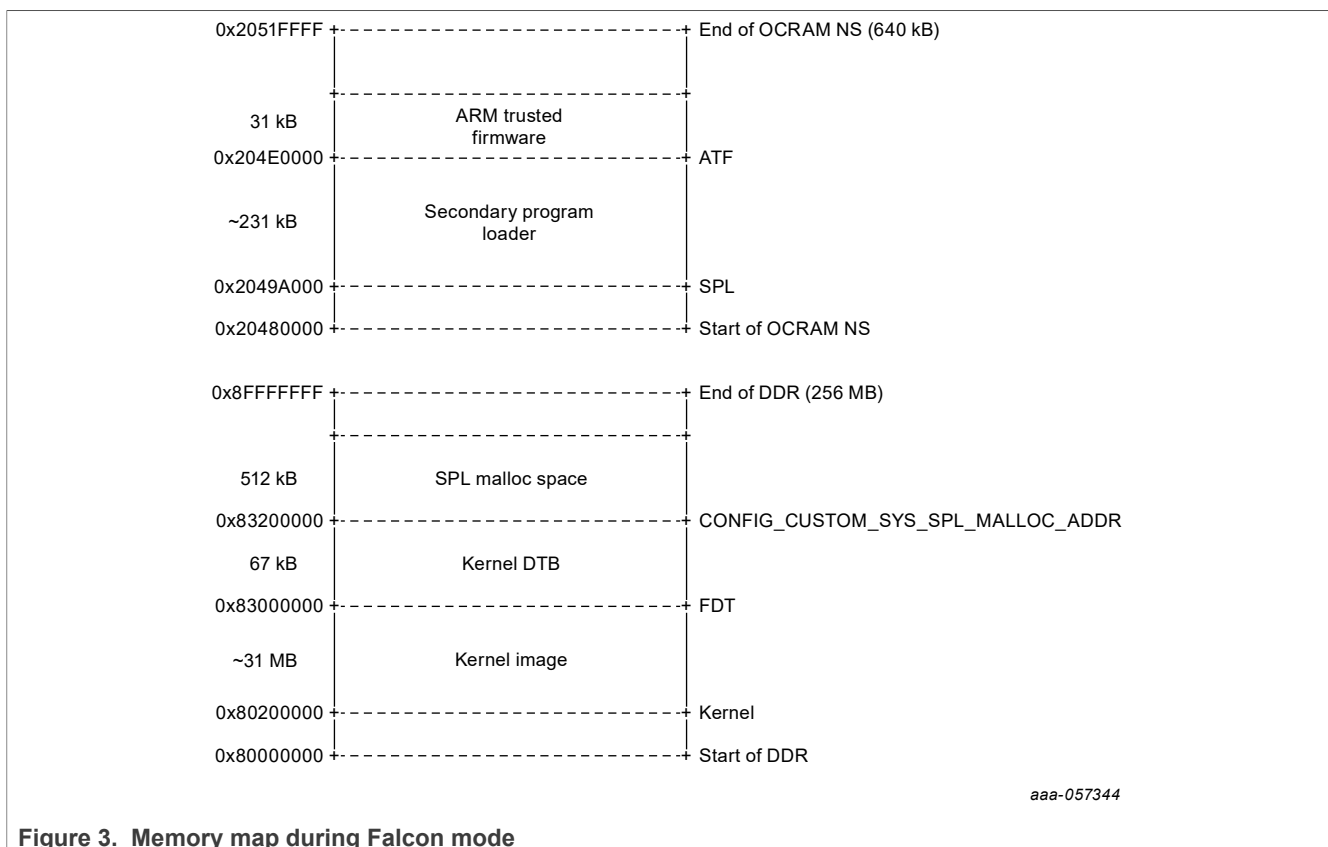


Figure 3. Memory map during Falcon mode

For the other platforms, only the addresses are different, as presented in Table 2.

Table 2. Memory address

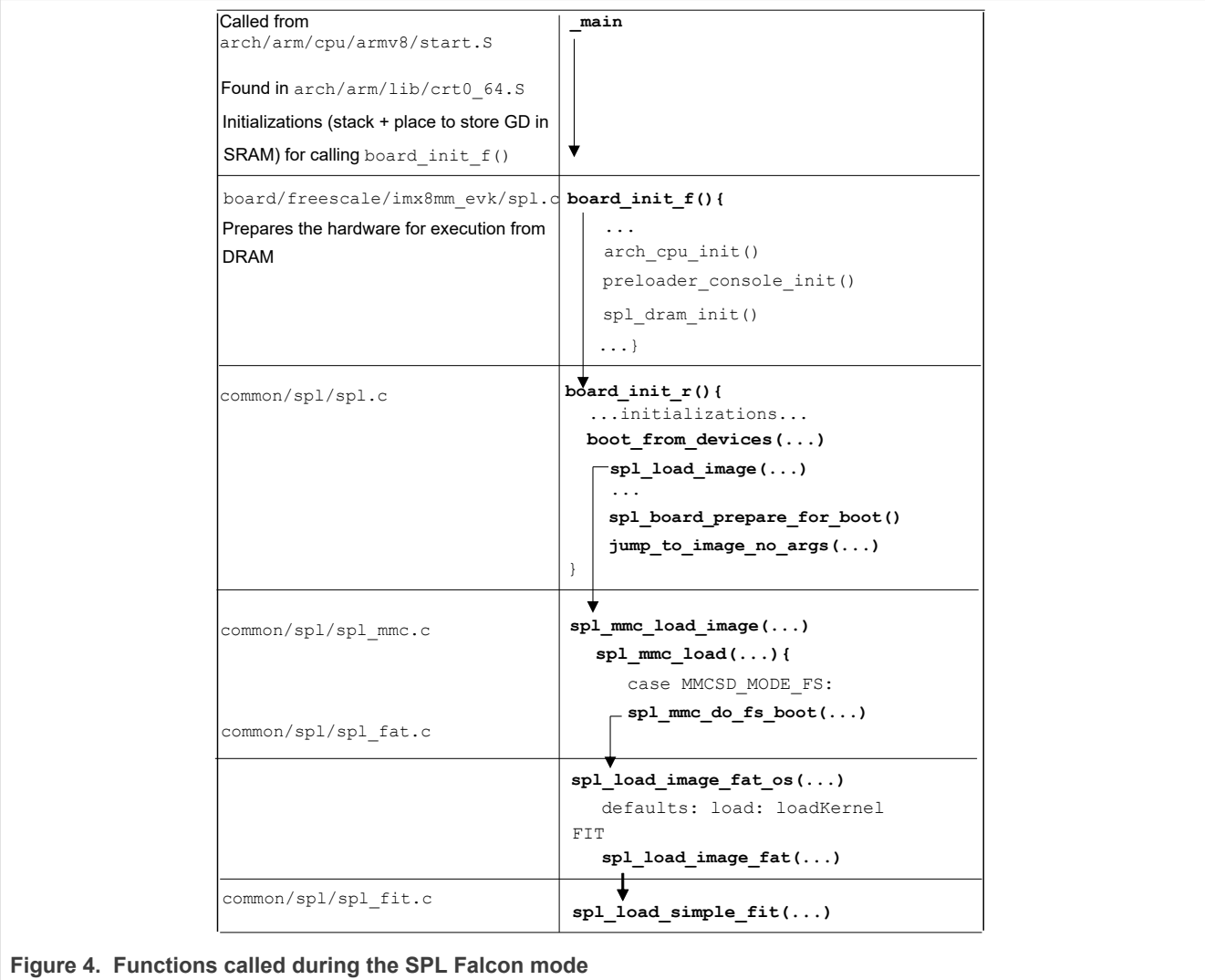
Platform	SPL	ATF	Kernel Image	Kernel DTB
i.MX 95	0x4aa00000	0x8A200000	0x90200000	0x93000000
i.MX 8M Mini	0x007e1000	0x00920000	0x40200000	0x43000000

Table 2. Memory address...continued

Platform	SPL	ATF	Kernel Image	Kernel DTB
i.MX 8M Nano	0x00912000	0x00960000	0x40200000	0x43000000
i.MX 8M Plus	0x00920000	0x00970000	0x40200000	0x43000000
i.MX 8M Quad	0x007e1000	0x00910000	0x40200000	0x43000000
i.MX 8ULP	0x22020000	0x20040000	0x80200000	0x83000000
i.MX 8QXP	0x00100000	0x80000000	0x80020000	0x83000000

3.2.4 Function calls during Falcon mode

For reference, here are the important functions called during SPL Falcon mode.



4 Kernel space optimizations

This chapter includes the following information.

- [Section 4.1 "Adding quiet"](#)
- [Section 4.2 "Removing unnecessary drivers and file systems"](#)

4.1 Adding quiet

To reduce the kernel time by about a half, the `quiet` argument can be used in the kernel `bootargs`. It suppresses the debug messages during the Linux start-up sequence. This argument was already added in the Yocto image built in [Section 3.2.1 "Falcon mode enablement"](#), using the `mmcargs_fastboot` variable.

4.2 Removing unnecessary drivers and file systems

Depending on your use case, you can slim down the kernel by removing unnecessary drivers and file systems. You may want to first analyze the kernel timing during boot using **bootgraph**.

To create a `bootgraph`, perform the following steps:

1. Add `initcall_debug` to the kernel `bootargs`.
 - a. Keep any key pressed while booting, to enter in U-Boot.
 - b. Edit the `mmcargs_fastboot` parameter by adding `initcall_debug`.

```
u-boot=> edit mmcargs_fastboot
edit: setenv bootargs "${jh_clk} console=${console} root=${mmccroot} quiet
initcall_debug
u-boot=> saveenv
Saving Environment to MMC... Writing to MMC(1)... OK
```

2. Regenerate the `fastboot` device tree, to use the new `bootargs`:

```
u-boot=> run prepare_fdt
u-boot=> reset
```

3. Boot the board and get the kernel log.

```
root@imx8mn-lpddr4-evk-fastboot:~# dmesg > boot.log
```

The `boot.log` file contains data like this. In this way can be analyzed how much time each function spend during the kernel boot.

```
[2.583922] initcall deferred_probe_initcall+0x0/0xb8 returned 0 after 895357
[2.583955] calling genpd_power_off_unused+0x0/0x98 @ 1
[2.583977] initcall genpd_power_off_unused+0x0/0x98 returned 0 after 12 usec
[2.583984] calling genpd_debug_init+0x0/0x90 @ 1
[2.584312] initcall genpd_debug_init+0x0/0x90 returned 0 after 321 usecs
[2.584333] calling ubi_init+0x0/0x23c @ 1
[2.584627] initcall ubi_init+0x0/0x23c returned 0 after 286 usecs
```

4. Copy the resulted `boot.log` file on the host PC. Go back on the host PC and create the graph using the following commands:

```
$ cd <build_dir>/tmp/work/imx8mnevk_fastboot-poky-linux/linux-imx/<git_rev>/
git/scripts
$ ./bootgraph.pl boot.log > boot.svg
```

You will obtain something like [Figure 5](#) and can analyze how the kernel boot time is used.

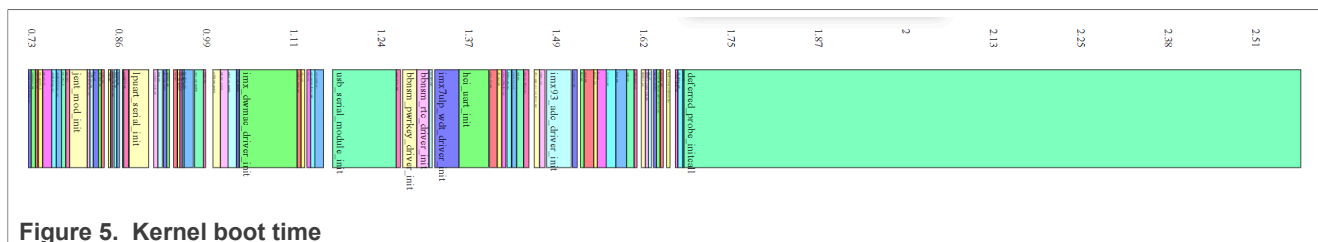


Figure 5. Kernel boot time

5. To disable a driver or a feature, change the kernel configuration. As an example, disable the kernel debug and the UBI file system.

- a. Run the following commands to enter the kernel menuconfig:

```
$ bitbake -c menuconfig linux-imx
```

From the menuconfig, disable `CONFIG_UBIFS_FS` and `CONFIG_DEBUG_KERNEL` and save the configuration. The resulting `.config` file contains the following lines:

```
# CONFIG_UBIFS_FS is not set
# CONFIG_DEBUG_KERNEL is not set
```

- b. Build the new kernel image:

```
$ bitbake -f -c compile linux-imx
```

- c. Generate the image containing the new kernel and the ATF and copy it on the SD card using the following commands:

```
$ bitbake -f -c compile imx-boot
$ sudo mount /dev/sd<x>1 /mnt
$ cp <build_dir>/tmp/work/<machine_name>-poky-linux/imx-boot/<git_rev>/
  git/<board>/kernel-atf<-container>.itb<.img> /mnt
$ umount /mnt
```

5 User space optimizations

This chapter includes the following information.

- [Section 5.1 "Start an application before systemd"](#)

5.1 Start an application before systemd

If required, a program can be started before systemd.

- Create a script `/home/root/newinit.sh` which starts your program before systemd. Below is a simple example on how to do this. Replace the `echo` line with your desired application.

```
#!/bin/sh

echo "Early start" > /dev/kmsg

exec /lib/systemd/systemd
```

- Make the script executable:

```
$ chmod +x newinit.sh
```

- Link `/sbin/init` to your `newinit.sh` script.

```
$ ln -sf /home/root/newinit.sh /sbin/init
```

Note:

To return to the initial configuration, use the following command:

```
$ ln -sf /lib/systemd/systemd /sbin/init
```

- Reboot the board and check the kernel log. In `dmesg`, it can be seen that the `newinit.sh` script is executed before the init process, by searching after **Early start** string.

6 Results

This section presents, for reference, the timing results on our test boards. The measurements for the i.MX 95 are based on the 6.6.23_2.0.0 BSP, while the rest are based on the 6.1.22_2.0.0 BSP.

Table 3. Initial boot time measurements

Board	SPL			U-Boot				KERNEL	Total time
	BOOT ROM	DDR initialization	SPL initializations + Load U-Boot image	U-Boot initializations (init_sequence_f)	U-Boot initializations (init_sequence_r)	Boot sequence	Kernel image load	ATF + Kernel boot until INIT process	
	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)
i.MX 8MN	161	241	162	363	790	2894	333	3506	8450
i.MX 8MP	162	301	175	373	1726	4181	345	3627	10890
i.MX 8MM	142	265	117	412	812	2970	396	5002	10116
i.MX 93	369	111	117	628	1172	3271	412	3090	9170
i.MX 95	350	5052	216	482	652	4142	373	6250	17527

Table 4. Optimized boot time measurements

Board	SPL				KERNEL	
	BOOT ROM	DDR initialization	SPL initializations	Kernel Image Load ^[1]	ATF + Kernel Boot until INIT process ^[2]	Total time
	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)
i.MX 8MN	203	240	86	376	1185	2090
i.MX 8MP	187	301	97	382	1237	2204
i.MX 8MM ^[3]	139	265	63	1336	2956 ^[4]	4759
i.MX 93	374	111	89	366	1391	2330
i.MX 95	350	20 ^[5]	184	410	3307	4271

[1] CONFIG_DEBUG_KERNEL disabled, resulting in a smaller kernel image size => decreases kernel image loading.

[2] kernel log messages are suppressed using quiet.

[3] The Kernel image loading time is longer for i.MX 8MM because MMC UHS is not supported in SPL.

[4] i.MX 8M Mini EVK does not come with an integrated Wi-Fi Module connected to the PCIe port (unlike i.MX 8M Plus). Therefore, the PCIe PHY initialization consumes time, waiting for an active link. If a Wi-Fi module is attached to the PCIe interface, the Kernel boot time decreases to 1215 ms, so the total boot time is 3018 ms.

[5] DDR Quickboot enabled.

7 Note about the source code in the document

The example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2025 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials must be provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

8 Revision history

[Table 5](#) summarizes the revisions to this document.

Table 5. Revision history

Document ID	Release date	Description
AN14093 v.3.0	21 January 2025	<ul style="list-style-type: none">• Added the i.MX 8ULP, i.MX 8QXP, and i.MX 8MQ boards• Updated the implementation for the 6.6.36_2.1.0 BSP• Re-structured the document
AN14093 v.2.0	26 September 2024	<ul style="list-style-type: none">• Re-structured the document• Added the i.MX95 EVK board• Updated the implementation to use Yocto, based on LF6.6.23 release
AN14093 v.1.2	26 February 2024	Copyright date in source code.
AN14093 v.1.1	24 January 2024	Added support for i.MX 93 A1
AN14093 v.1.0	09 October 2023	Initial public release

Legal information

Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at <https://www.nxp.com/profile/terms>, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Suitability for use in non-automotive qualified products — Unless this document expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

HTML publications — An HTML version, if available, of this document is provided as a courtesy. Definitive information is contained in the applicable document in PDF format. If there is a discrepancy between the HTML document and the PDF document, the PDF document has priority.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately. Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP B.V. — NXP B.V. is not an operating company and it does not distribute or sell products.

Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

Fast Boot on i.MX 8 and i.MX 9 Using Falcon Mode and Kernel Optimizations

AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, μ Vision, Versatile — are trademarks and/or registered trademarks of Arm Limited (or its subsidiaries or affiliates) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved.

EdgeLock — is a trademark of NXP B.V.

Microsoft, Azure, and ThreadX — are trademarks of the Microsoft group of companies.

Contents

1	Introduction	2
1.1	Software environment	2
2	General description	2
2.1	Reducing bootloader time	2
2.2	Reducing Linux kernel boot time	2
2.3	Reducing user-space boot time	3
2.4	Measurements	3
3	Bootloader optimizations	4
3.1	Default boot mode	4
3.2	Falcon mode	4
3.2.1	Falcon mode enablement	5
3.2.2	Falcon mode implementation details	5
3.2.3	Memory map	6
3.2.4	Function calls during Falcon mode	7
4	Kernel space optimizations	8
4.1	Adding quiet	8
4.2	Removing unnecessary drivers and file systems	8
5	User space optimizations	9
5.1	Start an application before systemd	9
6	Results	10
7	Note about the source code in the document	11
8	Revision history	11
	Legal information	12

Please be aware that important notices concerning this document and the product(s) described herein, have been included in section 'Legal information'.
