AN14113

How to Implement Falcon Mode of U-Boot on i.MX 8M Nano and i.MX 9 Series

Rev. 3.0 — 13 July 2026 Application note

Document information

Information	Content
Keywords	AN14113, i.MX 8M, i.MX 93, i.MX 943, i.MX 95 U-Boot, Falcon, fast boot
Abstract	This application note describes a method to implement Falcon mode.

1 Introduction

Falcon mode is a feature of U-Boot, which is introduced to speed up the booting process, allowing it to start the Linux kernel directly from the Secondary Program Loader (SPL).

The default i.MX releases support booting the kernel from U-Boot with boot commands (`booti`, `bootz`, or `bootm`, and so on). One drawback in the process is that the kernel is loaded relatively late.

Using Falcon mode, we can shorten this boot time. Falcon mode is helpful when a user needs a fast boot feature for the kernel.

This application note describes a method to implement Falcon mode.

The following are the target audiences of the document, who:

1. Need the kernel to boot at an early stage.
2. Want to know more on the use of ROM APIs.
3. Want to know more usage of imx-mkimage.
4. Want to know more on the SPL boot mechanism.

The mechanism described in this application note can also be applied to other i.MX 8M and i.MX 9 series chips.

2 Introduction to U-Boot SPL

SPL is a stripped-down version of U-Boot. It helps to do some initial hardware configurations (such as DRAM initialization using CPU cache as RAM) and load the larger, fully featured version of U-Boot.

When the board starts, besides ROM code, the *first-stage bootloader* is U-Boot SPL; and the *second-stage bootloader* is a regular U-Boot (or U-Boot proper). The SPL stands for Secondary Program Loader, which means that ROM code is the first thing that loads (and executes) another program, and SPL is the second thing that loads (and executes) another program.

Figure 1 shows the common boot sequence.

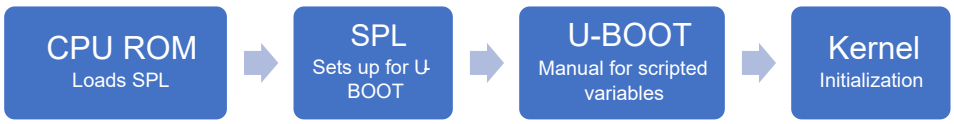

Figure 1. Boot sequence

Note: *The sequence is a bit different on i.MX 8M and i.MX 9 platforms.*

On Arm V8, Arm has a specified preferred way to boot Secure Component with the Arm Trusted Firmware (ATF). The ATF first loads the OP-TEE OS. The OP-TEE operating system initializes the secure world. Then, the ATF loads U-Boot that modifies the DTB on the fly to add a specific node to load Linux TEE drivers. Then, the Linux operating system is booted.

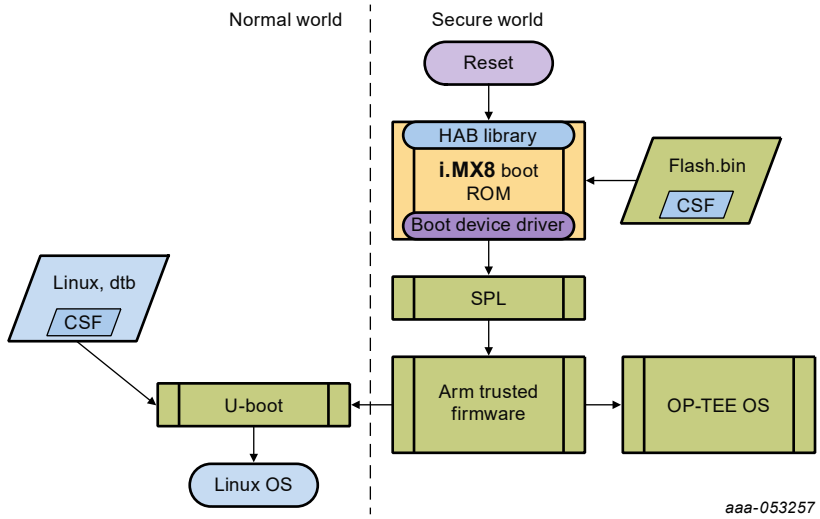

Figure 2. Boot Linux operating system

3 Falcon mode in U-Boot

In U-Boot, Falcon mode is introduced to speed up the booting process, allowing it to boot a Linux kernel (or whatever image) without launching a full U-Boot.

Falcon mode relies on the SPL framework. In most implementations, SPL is used to start U-Boot. Falcon mode extends this way allowing to start the Linux kernel directly from SPL.

4 Load and boot Linux kernel from SPL

To load the Linux kernel image from SPL, perform the following steps:

1. To enlarge the bootpartition size to 50 MB, modify the Yocto source code.
2. To allow launching an image with parameters in ATF, modify the ATF code.
3. Create a falcon `dts` in the kernel. The content of the falcon `dts` involves bootargs and memory information.
4. Build the Linux kernel image and kernel Falcon `dtb` into the `flash.bin` file.
5. (Optional) When using `uuu` to download the `flash.bin` to the card, update the U-Boot code to get the actual bootpartition size.

See the blue block where we start the kernel image.

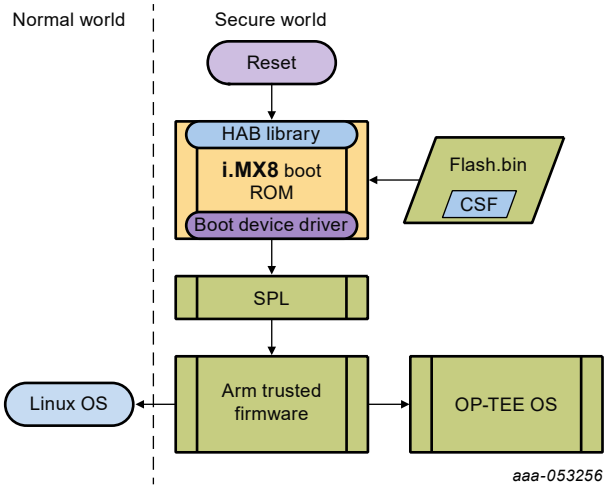

Figure 3. Load and boot Linux kernel from SPL

4.1 Modify Yocto source code to enlarge bootpartition size to 50 MB

In the default release, Yocto reserves 8192 kB for bootpartition that is used to store the MBR/GPT table and `flash.bin`.

When placing the Linux kernel image into the `flash.bin` file, the size of 8192 kB is no longer sufficient to accommodate a 30 MB kernel image. So, we enlarge the bootpartition to 50 MB in the Yocto source.

The patch is as below:

```
--- ./sources/meta-freescale/wic/imx-imx-boot-bootpart.wks.in.ori
 2023-08-22 15:57:03.817563907 +0800
+++ ./sources/meta-freescale/wic/imx-imx-boot-bootpart.wks.in   2023-05-17
 17:49:43.742919645 +0800
@@ -14,7 +14,8 @@
 #   ${IMX_BOOT_SEEK} 32 or 33kiB, see reference manual
 #
 part u-boot --source rawcopy --sourceparams="file=imx-boot" --ondisk mmcblk --
no-table --align ${IMX_BOOT_SEEK}
-part /boot --source bootimg-partition --ondisk mmcblk --fstype=vfat --label
 boot --active --align 8192 --size 64
+#part /boot --source bootimg-partition --ondisk mmcblk --fstype=vfat --label
 boot --active --align 8192 --size 64
+part /boot --source bootimg-partition --ondisk mmcblk --fstype=vfat --label
 boot --active --align 51200 --size 64
 part / --source rootfs --ondisk mmcblk --fstype=ext4 --label root --align 8192
```

See the bold text that changes the bootpartition size from 8192 kB to 51,200 kB.

For more details about the part command in Yocto, see https://docs.yoctoproject.org/ref-manual/kickstart.html

4.2 Modify ATF code to allow launching an image with parameters in ATF

In the default release, ATF launches the image without any parameters. However, to boot the kernel and pass the DTB address as a parameter to the kernel, modify ATF.

The patch for i.MX 8M Nano is as follows:

```
diff --git a/plat/imx/imx8m/imx8mn/imx8mn_bl31_setup.c b/plat/imx/imx8m/imx8mn/
imx8mn_bl31_setup.c
index c87748a18..0a5a61003 100644
--- a/plat/imx/imx8m/imx8mn/imx8mn_bl31_setup.c
+++ b/plat/imx/imx8m/imx8mn/imx8mn_bl31_setup.c
@@ -244,6 +244,14 @@ void bl31_early_platform_setup2(u_register_t arg0,
 u_register_t arg1,
 #endif
 #endif

+#if defined(USE_FALCON_MODE)
+       /* RAM FDT address */
+       bl33_image_ep_info.args.arg0 = BL32_FDT_ADDR;
+       bl33_image_ep_info.args.arg1 = 0U;
+       bl33_image_ep_info.args.arg2 = 0U;
+       bl33_image_ep_info.args.arg3 = 0U;
+#endif
+
 #if !defined(SPD_opteed) && !defined(SPD_trusty)
        enable_snvs_privileged_access();
```

The patch for i.MX 93 is as follows:

```
diff --git a/plat/imx/imx93/imx93_bl31_setup.c b/plat/imx/imx93/
imx93_bl31_setup.c
index 4c12347cd..1ce739daa 100644
--- a/plat/imx/imx93/imx93_bl31_setup.c
+++ b/plat/imx/imx93/imx93_bl31_setup.c
@@ -80,7 +80,7 @@ void bl31_early_platform_setup2(u_register_t arg0,
 u_register_t arg1,
         * tell BL3-1 where the non-secure software image is located
         * and the entry state information.
         */
-       bl33_image_ep_info.pc = PLAT_NS_IMAGE_OFFSET;
+       bl33_image_ep_info.pc = (u_register_t)PLAT_NS_IMAGE_OFFSET;
        bl33_image_ep_info.spsr = get_spsr_for_bl33_entry();
        SET_SECURITY_STATE(bl33_image_ep_info.h.attr, NON_SECURE);

@@ -100,6 +100,15 @@ void bl31_early_platform_setup2(u_register_t arg0,
 u_register_t arg1,
        bl33_image_ep_info.args.arg3 = BL32_FDT_OVERLAY_ADDR;
        bl32_image_ep_info.args.arg3 = BL32_FDT_OVERLAY_ADDR;
 #endif
+
+#if defined(USE_FALCON_MODE)
+       /* RAM FDT address */
+       bl33_image_ep_info.args.arg0 = (u_register_t)BL32_FDT_ADDR;
+       bl33_image_ep_info.args.arg1 = 0U;
+       bl33_image_ep_info.args.arg2 = 0U;
+       bl33_image_ep_info.args.arg3 = 0U;
+#endif
+
 }
```

The patch for i.MX 95 is as follows:

```
diff --git a/plat/imx/imx9/imx95/imx95_bl31_setup.c b/plat/imx/imx9/imx95/
imx95_bl31_setup.c
index 6f8d68b60..7d241b7f6 100644
--- a/plat/imx/imx9/imx95/imx95_bl31_setup.c
+++ b/plat/imx/imx9/imx95/imx95_bl31_setup.c
@@ -104,6 +104,14 @@ void bl31_early_platform_setup2(u_register_t arg0,
 u_register_t arg1,
    bl32_image_ep_info.args.arg3 = BL32_FDT_OVERLAY_ADDR;
 #endif
 #endif
+
+#if defined(USE_FALCON_MODE)
+   /* RAM FDT address */
+   bl33_image_ep_info.args.arg0 = (u_register_t)BL32_FDT_ADDR;
+   bl33_image_ep_info.args.arg1 = 0U;
+   bl33_image_ep_info.args.arg2 = 0U;
+   bl33_image_ep_info.args.arg3 = 0U;
+#endif
 }
```

The patch for i.MX 943 is as follows:

```
diff --git a/plat/imx/imx9/imx94/imx94_bl31_setup.c b/plat/imx/imx9/imx94/
imx94_bl31_setup.c
index 0a9891a44..9f167463b 100644
--- a/plat/imx/imx9/imx94/imx94_bl31_setup.c
+++ b/plat/imx/imx9/imx94/imx94_bl31_setup.c
@@ -107,6 +107,15 @@ void bl31_early_platform_setup2(u_register_t arg0,
 u_register_t arg1,
       bl32_image_ep_info.args.arg3 = BL32_FDT_OVERLAY_ADDR;
 #endif
 #endif
+
+#if defined(USE_FALCON_MODE)
+       /* RAM FDT address */
+       bl33_image_ep_info.args.arg0 = (u_register_t)BL32_FDT_ADDR;
+       bl33_image_ep_info.args.arg1 = 0U;
+       bl33_image_ep_info.args.arg2 = 0U;
+       bl33_image_ep_info.args.arg3 = 0U;
+#endif
+
 }
```

For the full patch, see AN14113SW.

4.3 Create a falcon dts in the kernel

To boot the kernel, pass not only the DTB address as a parameter to the kernel, but also kernel bootargs and memory layout information. These details are passed through the DTB.

To boot the kernel, create a falcon `dts` file and include the required bootargs, memory layout, and other information.

Here is a template for a Falcon mode `dts` file:

```
// SPDX-License-Identifier: (GPL-2.0+ OR MIT)
/*
```

```
 * Copyright 2019 NXP
 */
#include "imx8mn-evk.dts"

/ {
        model = "Falcon Mode Device Tree";
        compatible = "falcon-mode";

        memory {
                reg = <0x00 0x80000000 0x00 0x80000000>;
                device_type = "memory";
        };

        chosen {
                bootargs = "console=ttymxc1,115200 root=/dev/mmcblk1p2 rootwait
 rw";
        };
};
```

This `dts` template provides a basic structure for Falcon mode, including memory layout and kernel boot arguments. You can customize it further to suit your specific requirements.

Below are some details about how to get the required information from U-Boot for the `dts`. This required information is bootargs and memory layout.

Note: *Normally, the memory layout is defined in the U-Boot `dts` and passed to the kernel, or directly defined in the kernel `dts`. So that the kernel can know the size of memory available for use. For most platforms, we have the memory layout in both kernel and U-Boot `dts`. For most platforms, for example, i.MX 8M, i.MX 8Q, there is no need to define the memory layout in the Falcon `dts`. Add the memory layout in the Falcon `dts` if it is not present in the kernel `dts`, for example, i.MX 93 `dts`. For more details on memory node in dts, see* *Section 4.3.2**.*

4.3.1 How to get bootargs from U-Boot env

To get bootargs, there are two methods:

1. Get bootargs from `<U-Boot>/include/configs/<board name>.h`. Take the i.MX 8M Nano for example, to get bootargs, perform the following steps:
   a. Open `<U-Boot dir>/include/configs/imx8mn_evk.h`.
   b. Search `mmcargs` and see how `bootargs` is assigned.

      ```
      "mmcargs=setenv bootargs ${jh_clk} ${mcore_clk} console=${console} root=
      ${mmcroot}\0 " \
      ```

   c. Try to get bootargs.
2. Boot the original U-Boot and get bootargs. Another way is to get bootargs by running the U-Boot.
   a. Boot the U-Boot. Stop the autoboot by pressing any key.
   b. Run `run mmcargs`.
   c. To get bootargs, run `echo $bootargs`.

```
Hit any key to stop autoboot:  0
u-boot=> run mmcargs
u-boot=> echo $bootargs
console=ttymxc1,115200 root=/dev/mmcblk1p2 rootwait rw
u-boot=>
```

Figure 4. bootargs value

4.3.2 How to get memory layout

Below describes the memory layout:

```
memory {

    reg = <(baseaddr1) (size1)

    (baseaddr2) (size2)

    ...

    (baseaddrN) (sizeN)>;
    device_type = “memory”;

};
```

Where:

- baseaddrX: the base address of the defined memory bank.
- sizeX: the size of the defined memory bank.
- N: number of memory banks.

Those macros define these values:

- N: If `PHYS_SDRAM_2_SIZE` is defined, N is 2. Otherwise, N is 1.
- Baseaddr: `PHYS_SDRAM` in `include/configs/imx8mn_evk.h`. This `PHYS_SDRAM` is the entry of memory.
- Size: `PHYS_SDRAM_SIZE` (or `PHYS_SDRAM_2_SIZE`) in `include/configs/imx8mn_evk.h`. This `PHYS_SDRAM_SIZE` is the size of a memory bank 1 (or 2).

Note: *Because the i.MX 8M and i.MX 93 are 64-bit SoCs, the memory layout addresses are also 64-bit. Be sure to distinguish the 64-bit from the 32-bit address notation.*

For example, for i.MX 93, there is only one memory bank. The memory start is `0x0000000080000000` and memory size is `0x0000000080000000`, so the memory layout in `dts` is:

```
memory {
        reg = <0x00000000 0x80000000 0x00000000 0x80000000>;
         device_type = "memory";
};
```

Note:

- *Since DTS files can vary significantly across different platforms, besides the necessary bootargs, the additional configurations in the Falcon DTB also differ. For instance, on the i.MX 8M platform, only bootargs is required in the Falcon DTS because the memory layout is already present in the kernel DTS. However, on the*

i.MX 93 platform, include the memory layout in the Falcon DTS since it is not present in the kernel DTS. Take care of these differences when writing the DTS for Falcon mode.

- *Besides the memory layout, U-Boot can add other contents to the DTB when booting the kernel, such as MAC addresses, PMIC configurations, board serial number, and U-Boot version. If any configurations are found missing in the kernel started under Falcon mode, it is necessary to investigate whether these configurations are included in the Falcon DTB. In the U-Boot code, the code for fixing the FDT can be found in the `arch_fixup_fdt` function (in `arch/arm/lib/bootm-fdt.c`).*

Note: *For i.MX 95 and i.MX 943 platforms, refer to i.MX93. However, testing reveals that i.MX 93 and i.MX 943 require memory node integration within the Falcon device tree.*

Incorrect memory nodes in Falcon DTS can trigger kernel panic. For example:

```
[    0.000000] SError Interrupt on CPU0, code 0x00000000be000011 -- SError
[    0.000000] CPU: 0 UID: 0 PID: 0 Comm: swapper Not tainted 6.12.20-
gdfaf2136deb2 #19
[    0.000000] Hardware name: NXP i.MX95 15X15 board Falcon Mode Device Tree
 (DT)
[    0.000000] pstate: 800000c9 (Nzcv daIF -PAN -UAO -TCO -DIT -SSBS BTYPE=--)
[    0.000000] pc : new_slab+0x1e4/0x468
[    0.000000] lr : new_slab+0x174/0x468
[    0.000000] sp : ffff8000821e3cc0
[    0.000000] x29: ffff8000821e3cc0 x28: 0000000000000000 x27: 0000000000000000
[    0.000000] x26: ffff8000823cad90 x25: ffff0000000000c0 x24: 0000000000010082
[    0.000000] x23: ffff000000000100 x22: fffffdffc0000000 x21: 0000000000000004
[    0.000000] x20: ffff000000000100 x19: ffff800081ea2780 x18: 000000000000000d
[    0.000000] x17: 0000000000000000 x16: ffff8000823c1160 x15: ffff00007fba40c8
[    0.000000] x14: 0000000000000000 x13: 0000000000000001 x12: 00000000f0000000
[    0.000000] x11: 0000000000000068 x10: 0000000000000100 x9 : 0000000000000000
[    0.000000] x8 : 0000000000000038 x7 : 0000000000000000 x6 : 0000000000000000
[    0.000000] x5 : ffff7ffffdc28000 x4 : 0000000000000001 x3 : 0000000000000000
[    0.000000] x2 : 0000000000000000 x1 : 0000000000000000 x0 : 0000000000000020
[    0.000000] Kernel panic - not syncing: Asynchronous SError Interrupt
[    0.000000] CPU: 0 UID: 0 PID: 0 Comm: swapper Not tainted 6.12.20-
gdfaf2136deb2 #19
[    0.000000] Hardware name: NXP i.MX95 15X15 board Falcon Mode Device Tree
 (DT)
[    0.000000] Call trace:
[    0.000000]  dump_backtrace+0x94/0xec
[    0.000000]  show_stack+0x18/0x24
[    0.000000]  dump_stack_lvl+0x38/0x90
[    0.000000]  dump_stack+0x18/0x24
[    0.000000]  panic+0x388/0x3e8
[    0.000000]  nmi_panic+0x40/0x8c
[    0.000000]  arm64_serror_panic+0x64/0x70
[    0.000000]  arm64_is_fatal_ras_serror+0x3c/0xb4
[    0.000000]  do_serror+0x60/0x78
[    0.000000]  el1h_64_error_handler+0x30/0x48
[    0.000000]  el1h_64_error+0x64/0x68
[    0.000000]  new_slab+0x1e4/0x468
[    0.000000]  do_kmem_cache_create+0x2fc/0x588
[    0.000000]  create_boot_cache+0x98/0x118
[    0.000000]  kmem_cache_init+0xf4/0x1c0
[    0.000000]  mm_core_init+0x90/0xfc
[    0.000000]  start_kernel+0x2dc/0x710
[    0.000000]  __primary_switched+0x80/0x88
[    0.000000] ---[ end Kernel panic - not syncing: Asynchronous SError
 Interrupt ]---
```

Therefore, suggestions for adding memory nodes to Falcon DTS are:

1. User can refrain from adding memory nodes in Falcon DTS initially, while carefully distinguishing potential boot issues during Falcon mode operation.
2. If the above kernel oops error persists regardless of whether the memory node is added, it suggests that U-Boot fixup operation has modified the memory node before launching the kernel. In this case, apply the `0001-u-boot-Add-fdt-fixup-arch-command.patch` patch from the software package to U-Boot. This patch introduces the `fdt fixup arch` command. Then, follow the procedure below to obtain the correctly fixed-up memory node.
   a) Apply the patch to U-Boot and boot.
   b) Input commands:

```
run loadfdt
fdt addr $fdt_addr_r
fdt fixup arch
fdt print
```

   c) Check memory node in output dts.

4.4 Build the Linux kernel image and Falcon DTB into flash.bin

Usually, we use the script in `imx-mkimage` to generate `flash.bin`.

For i.MX 8MN, `flash.bin` consists of an image header, an SPL image, and a `U-Boot.itb` file in the FIT format. Here, we are planning to add **kernel image** and **`kernel dtb`** to `kernel.itb` and replace `U-Boot.itb` with the `kernel.itb`.

Figure 5 shows the change to `flash.bin` structure.

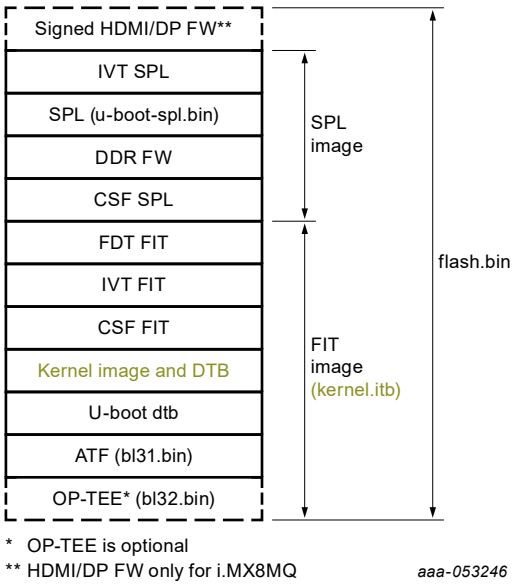

Figure 5. Change in structure

To generate `U-Boot.itb`, a description file called `U-Boot.its` is required. In the imx-image, `U-Boot.its` is generated using the script `mkimage_fit_atf.sh`. Therefore, to include a description segment for the kernel image and kernel DTB, modify `mkimage_fit_atf.sh`.

Note: *From `U-Boot.its`, we can understand that `U-Boot.itb` can contain more than U-Boot. It can also include other components, such as `bl31.bin` (ATF), `tee.bin`, `dek_blob`, `.dtb` files.*

Figure 6 shows the general diagram.

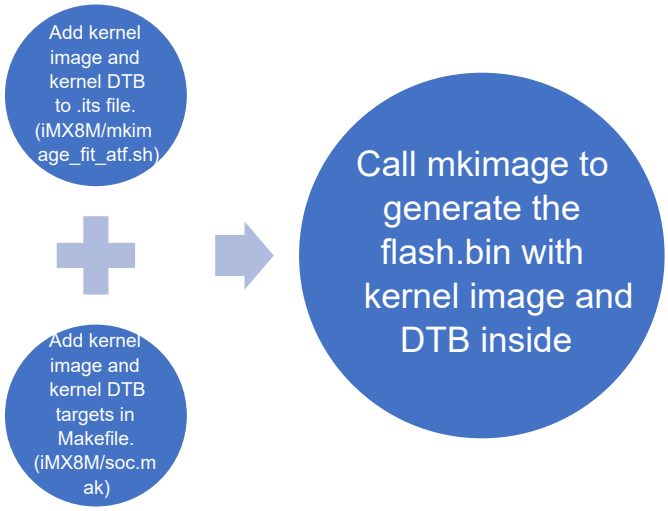

Figure 6. General diagram

See `0001-imx-mkimage-iMX8MN-Add-falcon-mode-support-to-imx-mkimage-o.patch` for more details.

Note: *Other platforms like i.MX 93 use a different image structure (not FIT image, but i.MX container) to organize the `flash.bin` file. But the operation is similar, which is to replace the `u-boot.bin` in the `flash.bin` with the kernel image and device tree blob (DTB).*

See `0001-imx-mkimage-iMX93-Add-falcon-mode-support-to-imx-mki.patch` for more details.

4.5 (Optional) Modify U-Boot for ease of use with UUU for downloading

There is a bug in the U-Boot Fastboot feature: when downloading to an SD card, the bootpartition size is fixed to 4 MB (`0x400000`). This size is not sufficient when our `flash.bin` has been increased to 30 MB and it results in a `uuu` download failure.

```
$ ./uuu.exe -b sd ./imx-boot-imx8mn-lpddr4-evk-sd.bin-flash_evk ./flash_8mn_falcon.bin
uuu (Universal Update Utility) for nxp imx chips -- libuuu_1.4.243-0-ged48c51

Success 0    Failure 1


1:2      4/ 5 [image too large for partition         ] FB: flash bootloader ./flash_8mn_falcon.bin
```

Figure 7. UUU download failure - uuu log

```
Detect USB boot. Will enter fastboot mode!
Starting download of 30668272 bytes
.....................................................................
.....................................................................
.....................................................................
...........
downloading of 30668272 bytes finished
Image too large for the partition
```

Figure 8. UUU download failure - console log

The patch `0001-uboot-adjust-bootloader-partition-size-for-uuu-download.patch` is intended to fix this issue.

For details about the patch, see AN14113SW.

5 Test steps and results

The test shows that the kernel launches from SPL.

lf-5.15.71-2.2.0 environment for building `flash.bin` and kernel image on i.MX 8M Nano and i.MX 93 and lf-6.12.20-2.0.0 environment for i.MX 943 and i.MX 95::

- Repo `imx-mkimage`
- Repo `atf-nxp`
- Repo `linux-firmware-imx`
- Repo `uboot-nxp`
- Repo `linux-lts-nxp`

5.1 Falcon mode test on i.MX 8M Nano

The test steps and results are as follows:

1. Apply the `0001-imx-mkimage-iMX8MN-Add-falcon-mode-support-to-imx-mkimage-o.patch` to `imx-mkimage`.
2. Apply the patch `0001-atf-imx8mn-Add-support-to-start-image-with-parameter.patch` to `atf-nxp`.
3. Build `atf` and copy `bl31.bin` to `imx-mkimage`.

```
make PLAT=imx8mn bl31
cp build/imx8mn/release/bl31.bin <imx-mkimage>/iMX8M/
```

4. Apply the patch `0001-linux-imx8mn-Add-falcon-dts.patch` to `linux-lts-nxp`.
5. Build linux kernel image and `dtb` and copy to `imx-mkimage`:

```
make ARCH=arm64 imx_v8_defconfig
make ARCH=arm64
cp arch/arm64/boot/Image <imx-mkimage>/iMX8M/kernel_image.bin
cp arch/arm64/boot/dts/freescale/
imx8mn-evk-falcon.dtb <imx-mkimage>/iMX8M/kernel_dtb.bin
```

6. (Optional) Apply the patch `0001-uboot-adjust-bootloader-partition-size-for-uuu-download.patch` to U-Boot.
7. Build the U-Boot.

```
make imx8mn_evk_defconfig
make
```

8. Copy `mkimage`, `u-boot-spl.bin`, and `imx8mn-evk.dtb` to <imx-mkimage/iMX8M>.

```
cp tools/mkimage <imx-mkimage>/iMX8M/mkimage_uboot
cp spl/u-boot-spl.bin <imx-mkimage>/iMX8M/
cp arch/arm/dts/imx8mn-evk.dtb <imx-mkimage>/iMX8M/
```

9. Copy the lpddr binary from `<linux-firmware-imx>/firmware/ddr/synopsys/ to <imx-mkimage>/iMX8M`.

```
cp -rf <linux-firmware-imx>/firmware/ddr/synopsys/*.bin <imx-mkimage>/iMX8M
```

10. Build the `flash.bin`.

```
make SOC=iMX8MN flash_evk_falcon
```

11. Download flash.bin to offset 32 K in the TF card (or use `uuu` to download the *flash.bin*).
12. Boot the board.

The Boot log is as below:

```
U-Boot SPL 2022.04-00001-ga0885fceac (Aug 22 2023 - 16:44:59 +0800)
DDRINFO: start DRAM init
DDRINFO: DRAM rate 3200MTS
DDRINFO:ddrphy calibration done
DDRINFO: ddrmix config done
SEC0:  RNG instantiated
Normal Boot
Trying to boot from BOOTROM
Boot Stage: Primary boot
image offset 0x8000, pagesize 0x200, ivt offset 0x0
NOTICE:  BL31: v2.6(release):automotive-13.0.0_1.1.0-rc2-1-g55a839d45
NOTICE:  BL31: Built : 07:48:37, Aug 22 2023
[    0.000000] Booting Linux on physical CPU 0x0000000000 [0x410fd034]
[    0.000000] Linux version 5.15.71-dirty (nxa08304@lsv11258.swis.cn-sha01.nxp.
com) (aarch64-poky-linux-gcc (GCC) 9.2.0, GNU ld (GNU Binutils) 2.32.0.20190204)
 #7 SMP PREEMPT Wed Aug 23 15:23:25 CST 2023
[    0.000000] Machine model: NXP i.MX8MNano EVK board
[    0.000000] efi: UEFI not found.
[    0.000000] Reserved memory: created CMA memory pool at 0x0000000058000000, s
ize 640 MiB
[    0.000000] OF: reserved mem: initialized node linux,cma, compatible id share
d-dma-pool
```

Figure 9. Successful boot log

5.2 Falcon mode test on i.MX 93

The test steps and results are as follows:

1. Apply the patch `0001-imx-mkimage-iMX93-Add-falcon-mode-support-to-imx-mki.patch` to `imx-mkimage`.
2. Apply the patch `0001-atf-imx93-Add-support-to-start-image-with-parameters.patch` to `atf-nxp`.
3. Build `atf` and copy `bl31.bin` to `imx-mkimage`.

```
make PLAT=imx93 bl31
cp build/imx93/release/bl31.bin <imx-mkimage>/iMX9/
```

4. Apply the patch `0002-linux-imx93-Add-falcon-dts.patch` to `linux-lts-nxp`.

5. Build the Linux kernel image and `dtb` and copy to `imx-mkimage`:

```
make ARCH=arm64 imx_v8_defconfig
make ARCH=arm64
cp arch/arm64/boot/Image <imx-mkimage>/iMX9/kernel_image.bin
cp arch/arm64/boot/dts/freescale/imx93-11x11-evk-falcon.dtb <imx-mkimage>/
iMX9/kernel_dtb.bin
```

6. (Optional) Apply patch `0001-uboot-adjust-bootloader-partition-size-for-uuu-download.patch` to `U-Boot`.
7. Build the U-Boot.

```
make imx93_11x11_evk_defconfig
make
```

8. Copy `mkimage` and `u-boot-spl.bin` to `<imx-mkimage/iMX9>`.

```
cp spl/u-boot-spl.bin <imx-mkimage>/iMX9/
```

9. Copy the `lpddr` binary from `<linux-firmware-imx>/firmware/ddr/synopsys/` to `<imx-mkimage>/iMX9`.

```
cp -rf <linux-firmware-imx>/firmware/ddr/synopsys/*.bin <imx-mkimage>/iMX9
```

10. Build the `flash.bin`.

```
make SOC=iMX9 flash_singleboot_falcon
```

11. Download the `flash.bin` to offset 32 K in the TF card (or use `uuu` to download the `flash.bin`).
12. Boot the board.

5.3 Falcon mode test on i.MX 95

The test steps and results are as follows:

1. Apply the patch `0001-imx-mkimage-iMX95-Add-falcon-mode-support-to-imx-mki.patch`.
2. Apply the patch `0001-atf-imx95-Add-support-to-start-image-with-parameters.patch`.
3. Build `atf` and copy `bl31.bin` to `imx-mkimage`.

```
make PLAT=imx95 bl31
cp build/imx95/release/bl31.bin <imx-mkimage>/iMX95/
```

4. Apply the patch `0001-linux-imx95-Add-falcon-dts.patch` to `linux-lts-nxp`.
5. Build the Linux kernel and `dtb` and copy to `imx-mkimage`:

```
make ARCH=arm64 imx_v8_defconfig
make ARCH=arm64
cp arch/arm64/boot/Image /iMX95/kernel_image.bin
cp arch/arm64/boot/dts/freescale/imx95-15x15-evk-falcon.dtb <imx-mkimage>/
iMX95/
```

6. (Optional) Apply the patch `0001-uboot-adjust-bootloader-partition-size-for-uuu-download` to U-Boot.
7. Build the U-Boot.

```
make imx95_15x15_evk_defconfig
```

```
make
```

8. Copy `mkimage` and `u-boot-spl.bin` to `<imx-mkimage/iMX95>`.

```
cp spl/u-boot-spl.bin <imx-mkimage>/iMX95/
```

9. Copy the lpddr binary from `<linux-firmware-imx>/firmware/ddr/synopsys/` to `<imxmkimage>/iMX95`.

```
cp -rf <linux-firmware-imx>/firmware/ddr/synopsys/*.bin <imx-mkimage>/iMX95
```

10. Build the flash.bin.

```
make SOC=iMX95 REV=B0 OEI=YES LPDDR_TYPE=lpddr4x KERNEL_DTB=imx95-15x15-evk-falcon.dtb flash_all_falcon
```

11. Download the `flash.bin` to offset 32 kB in the TF card (or use UUU to download the `flash.bin`).
12. Boot the board.

5.4 Falcon mode test on i.MX 943

The test steps and results are as follows:

1. Apply the patch `0001-imx-mkimage-iMX943-Add-falcon-mode-support-to-imx-mki.patch`.
2. Apply the patch `0001-atf-imx943-Add-support-to-start-image-with-parameters.patch`.
3. Build `atf` and copy `bl31.bin` to `imx-mkimage`.

```
make PLAT=imx94 bl31
cp build/imx94/release/bl31.bin <imx-mkimage>/iMX94/
```

4. Apply the patch `0001-linux-imx943-Add-falcon-dts.patch` to `linux-lts-nxp`.
5. Build the Linux kernel and `dtb` and copy to `imx-mkimage`:

```
make ARCH=arm64 imx_v8_defconfig
make ARCH=arm64
cp arch/arm64/boot/Image /iMX94/kernel_image.bin
cp arch/arm64/boot/dts/freescale/imx943-evk-falcon.dtb <imx-mkimage>/iMX94/
```

6. (Optional) Apply the patch `0001-uboot-adjust-bootloader-partition-size-for-uuu-download` to U-Boot.
7. Build the U-Boot.

```
make imx943_evk_defconfig
make
```

8. Copy `mkimage` and `u-boot-spl.bin` to `<imx-mkimage/iMX94>`.

```
cp spl/u-boot-spl.bin <imx-mkimage>/iMX94/
```

9. Copy the lpddr binary from `<linux-firmware-imx>/firmware/ddr/synopsys/` to `<imxmkimage>/iMX94`.

```
cp -rf <linux-firmware-imx>/firmware/ddr/synopsys/*.bin <imx-mkimage>/iMX94
```

10. Build the flash.bin.

```
make SOC=iMX94 REV=B0 OEI=YES LPDDR_TYPE=lpddr4 KERNEL_DTB=imx943-evk-
falcon.dtb flash_all_falcon
```

11. Download the `flash.bin` to offset 32 kB in the TF card (or use UUU to download the `flash.bin`).
12. Boot the board.

6 Optimization

The advantage of Falcon mode is its quick boot-up time. Falcon mode skips the U-Boot boot-up time by directly booting the kernel, saving approximately 3 to 4 seconds, which is the time U-Boot takes to run.

To speed up the boot process, optimize not only the bootloader but also the kernel and user space.

For optimizations on kernel and user space, refer to “Chapter 5” and “Chapter 6” in the *Linux Boot Time Optimizations for i.MX8M Family* (document AN13709).

7 Limitations

The solution we implement Falcon mode in this document also has some limitations:

1. eMMC card uses the user partition to boot. As `flash.bin` is too large, it cannot be put to the bootpartition of the eMMC card. So, use the user partition to boot. To boot from the eMMC user partition, disable the bootpartition functionality in EXT_CSD register 179, while ensuring that the FAST_BOOT fuse remains unprogrammed.
   Note: *Another method is to use the U-Boot command bootpart-resize to resize the bootpartition. However, if this command does not work on your eMMC card, consult with your eMMC provider first.*
2. Speed. Only ROMAPI is used to load images. The maximum speed mode of eMMC that ROM can support is DDR50 (can be supported by changing the boot switch), which needs consideration. However, the DDR50 is 100 MB/s, which is enough for the fast boot.

Note: *In Linux Boot Time Optimizations for i.MX8M Family (document AN13709), another method for implementing Falcon mode has been discussed. Users can review and choose the best one that suits their needs.*

8 Software package description

A software package is attached with this application note.

The software package provides patches for i.MX8 MN and i.MX 93 based on lf-5.15.71-2.2.0, and patches for i.MX 943 and i.MX 95 based on lf-6.1.20-2.0.0.

Table 1 describes the files in AN14113SW.

Table 1. Files in software package

Filename	Description	Comments
5.15.71-2.2.0/imx8mn/0001-imx-mkimage-iMX8MN-Add-falcon-mode-support-to-imx-mkimage-o.patch	Add kernel image and kernel DTB to *flash.bin* as a loadable image. Add build options to build *flash.bin* with kernel image and DTB inside for i.MX 8M Nano.	Based on version lf-5.15.71-2.2.0
5.15.71-2.2.0/imx93/0001-imx-mkimage-iMX93-Add-falcon-mode-support-to-imx-mki.patch	Add kernel image and kernel DTB to *flash.bin* as a loadable image. Add build options to build *flash.bin* with kernel image and DTB inside for i.MX 93.	Based on version lf-5.15.71-2.2.0

Table 1. Files in software package*...continued*

Filename	Description	Comments
5.15.71-2.2.0/imx8mn/0001-atf-imx8mn-Add-support-to-start-image-with-parameter.patch	Allow *atf* to start the image with parameters for i.MX 8M Nano.	Based on version lf-5.15.71-2.2.0
5.15.71-2.2.0/imx93/0001-atf-imx93-Add-support-to-start-image-with-parameters.patch	Allow *atf* to start the image with parameters for i.MX 93.	Based on version lf-5.15.71-2.2.0
5.15.71-2.2.0/imx8mn/0001-linux-imx8mn-Add-falcon-dts.patch	Add Falcon mode *dts* to Linux for i.MX 8M Nano.	Based on version lf-5.15.71-2.2.0
5.15.71-2.2.0/imx93/0002-linux-imx93-Add-falcon-dts.patch	Add Falcon mode *dts* to Linux for i.MX 93.	Based on version lf-5.15.71-2.2.0
5.15.71-2.2.0/common/0001-yocto-change-boot-partition-size-to-50MB.patch	Enlarge the bootpartition to 50 MB.	Based on version lf-5.15.71-2.2.0
5.15.71-2.2.0/common/0001-uboot-adjust-bootloader-partition-size-for-uuu-download.patch	Optional patch to fix bootpartition size in *fastboot*.	Based on version lf-5.15.71-2.2.0

Based on lf-6.1.20-2.2.0, similar patches are provided, the details of which are omitted here for brevity.

9 Acronyms

Table 2 lists the acronyms used in this document.

Table 2. Acronyms

Acronym	Description
ATF	Arm Trusted Firmware
DTB	Device Tree Blob
DTS	Device Tree Source
FIT	Flattened Image Tree
OS	Operating System
ROM	Read-Only Memory
SoC	System on Chip
SPL	Secondary Program Loader

10 Reference

- *i.MX Porting Guide* (document IMXBSPPG)
- *Linux Boot Time Optimizations for i.MX8M Family* (document AN13709)
- Understanding U-Boot Falcon Mode

11 Note about the source code in the document

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

12 Revision history

Table 3 summarizes the revisions to this document.

Table 3. Revision history

Revision number	Release date	Description
AN14113 v.3.0	13 July 2026	Updated document for i.MX 943, i.MX 95
AN14113 v.2.0	22 February 2024	Section 5 and Section 7 are updated.
AN14113 v.1.0	13 November 2023	Initial version

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.

Contents

1 Introduction 2
2 Introduction to U-Boot SPL 2
3 Falcon mode in U-Boot 3
4 Load and boot Linux kernel from SPL 3
4.1 Modify Yocto source code to enlarge bootpartition size to 50 MB 4
4.2 Modify ATF code to allow launching an image with parameters in ATF 4
4.3 Create a falcon dts in the kernel 6
4.3.1 How to get bootargs from U-Boot env 7
4.3.2 How to get memory layout 8
4.4 Build the Linux kernel image and Falcon DTB into flash.bin 10
4.5 (Optional) Modify U-Boot for ease of use with UUU for downloading 11
5 Test steps and results 12
5.1 Falcon mode test on i.MX 8M Nano 12
5.2 Falcon mode test on i.MX 93 13
5.3 Falcon mode test on i.MX 95 14
5.4 Falcon mode test on i.MX 943 15
6 Optimization 16
7 Limitations 16
8 Software package description 16
9 Acronyms 17
10 Reference 17
11 Note about the source code in the document 17
12 Revision history 18
Legal information 19