AN13709

Linux Boot Time Optimizations for i.MX8M Family

Rev. 0 — 18 August 2022 Application note

Document information

Information	Content
Keywords	i.MX8MN, Reduce Boot Time, Linux, Falcon Mode
Abstract	This document guides how to reduce Linux boot time for i.MX8M Nano board.

1 Introduction

This document guides how to reduce Linux boot time for i.MX8M Nano board.

Note: *The same workflow is applied to the entire i.MX8M family, but for each SoC, specific code updates is required.*

The objectives of this document are as follows:

- Measurement and evaluation of default boot time
- Bootloader optimizations
- Linux Kernel and User Space optimizations

1.1 Software environment

Linux BSP release 5.10.72-2.2.0 is used in the optimization process. The **imx-image-core** Yocto image is used during the experiments.

1.2 Hardware setup and equipment

- Development kit NXP i.MX 8MN EVK LPDDR4.
- Micro SD card: SanDisk Ultra 32 GB Micro SDHC I Class 10 was used for the current experiment.
- Micro-USB cable for debug port.
- Logic analyzer with the following minimum requirements for time measurements: Four channels and 10 MS/s. Saleae Logic 8 is used for the current experiment.

2 General description

This section describes an overview of the typical modifications that can be done to achieve shorter boot times.

2.1 Reducing bootloader time

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

2.2 Reducing Linux kernel boot time

- **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) in order 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

- **Change running order in initialization Systemd scripts**: It saves about 600 ms. Launch PSPLASH process as soon as possible, taking into account its dependencies.

3 Measurements

The scope of the measurements is considered to be between the board POR (Power-On Reset) and the start of the PSPLASH process.

The setup used for the following measurements is described in the Boot Time Measurements Methodology document.

The measured intervals are as follows:

Table 1. Measured intervals

Time point	Interval between pulses	Location of the pulse	Boot stages	
BootROM	nRST -> before ddr_ init()	board/freescale/imx8mn_evk/spl.c/board_init_ f()	SPL	Timeline
DDR initialization	before ddr_init() -> after ddr_init()	board/freescale/imx8mn_evk/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/imx8mn_evk.h		
Kernel Image Load	before loadimage -> after loadimage	include/configs/imx8mn_evk.h		
Kernel Boot Until PSPLASH Image	after loadimage -> psplash	psplash.c	Kernel	

4 Bootloader optimizations

4.1 Default boot mode

Figure 1 describes the default boot sequence. After power-on or reset, i.MX8M executes 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 finds a boot device from where it loads a bootloader image, which can include the following component: U-Boot SPL, ATF, U-Boot, and so on.

Figure 1. Default boot sequence

Because a typical U-Boot image does not fit inside internal SRAM, it was split into two parts: **Secondary Program Loader (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 will, in turn, jump to U-Boot proper.

Arm Trusted Firmware (ATF), included recently in i.MX8* 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 starts 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, etc.). Then, it loads and prepares the 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.

4.2 Falcon mode

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

Figure 2 shows the Falcon mode booting sequence.

Figure 2. Falcon mode boot sequence

To implement this mode, perform the following actions:

- Activate some specific configurations for Falcon.
- Prepare the Flattened Device Tree (FDT) in advance.
- Configure ATF to jump to Kernel.
- Write the ATF, Kernel image and FDT on SD card in a raw section, outside partitions.
- Modify SPL to load the ATF and the Kernel, then jump to ATF.

4.3 Implementation

The next modifications are made on the host computer, where the Yocto environment was installed.

Note: *On Windows the `dd` command (from Linux) is available by installing MSYS2. It is a collection of GNU/Linux Tools compiled for Windows.*

1. Remove the SPL BootROM support from the configuration file.

   Location: <yocto_build_dir>/tmp/work/imx8mn_lpddr4_evk-poky-linux/u-boot-imx/<specified_git_folder>/git/configs/imx8mn_evk_defconfig

```
# CONFIG_SPL_BOOTROM_SUPPORT is not set
```

2. Add support for raw SD read, enable `spl export` command, add support for legacy Image, and make specific configurations for Falcon mode.

   Location: <yocto_build_dir>/tmp/work/imx8mn_lpddr4_evk-poky-linux/u-boot-imx/<specified_git_folder>/git/include/configs/imx8mn_evk.h

```
#define CONFIG_CMD_SPL     1 // enable spl export command
#define CONFIG_SPL_MMC_SUPPORT 1 // for reading from MMC
#define CONFIG_SPL_LEGACY_IMAGE_SUPPORT 1

/* Falcon Mode */
//#define CONFIG_SPL_OS_BOOT   1 // activate Falcon Mode
 /* (leave this line commented until you finish all the
 configurations) */

// RAM FDT address
#define CONFIG_SYS_SPL_ARGS_ADDR    0x43000000

/* Falcon Mode - MMC support */
#define CONFIG_SYS_MMCSD_RAW_MODE_ARGS_SECTOR 0x2FAF080
#define CONFIG_SYS_MMCSD_RAW_MODE_ARGS_SECTORS 0x58
#define CONFIG_SYS_MMCSD_RAW_MODE_KERNEL_SECTOR 0x2FAF0E4
```

3. Implement the `spl_start_uboot()` function that returns **0** to indicate that booting U-Boot is not the first choice;
   Location:<yocto_build_dir>/tmp/work/imx8mn_lpddr4_evk-poky-linux/u-boot-imx/<specified_git_folder>/git/board/freescale/imx8mn_evk/spl.c

```
#ifdef CONFIG_SPL_OS_BOOT
int spl_start_uboot(void) {
    return 0;
}
#endif
```

4. Modify the calculation mode of the Kernel load address in the `spl_parse_legacy_header()` function, in the else statement (when the load includes the header).
   Location: <yocto_build_dir>/tmp/work/imx8mn_lpddr4_evk-poky-linux/u-boot-imx/<specified_git_folder>/git/common/spl/spl_legacy.c
   Replace the following line:

```
spl_image->load_addr = image_get_load(header) -
          header_size;
```

With this:

```
spl_image->load_addr = image_get_ep(header) -
         header_size;
```

5. In the `mmc_load_legacy()` function, add the following line after the existing code, to load the AFT.
   Location: `<yocto_build_dir>/tmp/work/imx8mn_lpddr4_evk-poky-linux/u-boot-imx/<specified_git_folder>/git/common/spl/spl_mmc.c`

```
/* existing code */
count = blk_dread(mmc_get_blk_desc(mmc), sector,
 image_size_sectors,
           (void *)(ulong)spl_image->load_addr);
/* end of existing code */
unsigned long count1 = blk_dread(mmc_get_blk_desc(mmc),
 0x2FBDAE0, 0x71,
(void*)(ulong)0x00960000);  //write ATF from SD to RAM
```

6. In the `board_init_r()` function, modify the SPL to jump to ATF (in jump to Linux case).
   Location: `<yocto_build_dir>/tmp/work/imx8mn_lpddr4_evk-poky-linux/u-boot-imx/<specified_git_folder>/git/common/spl/spl.c`

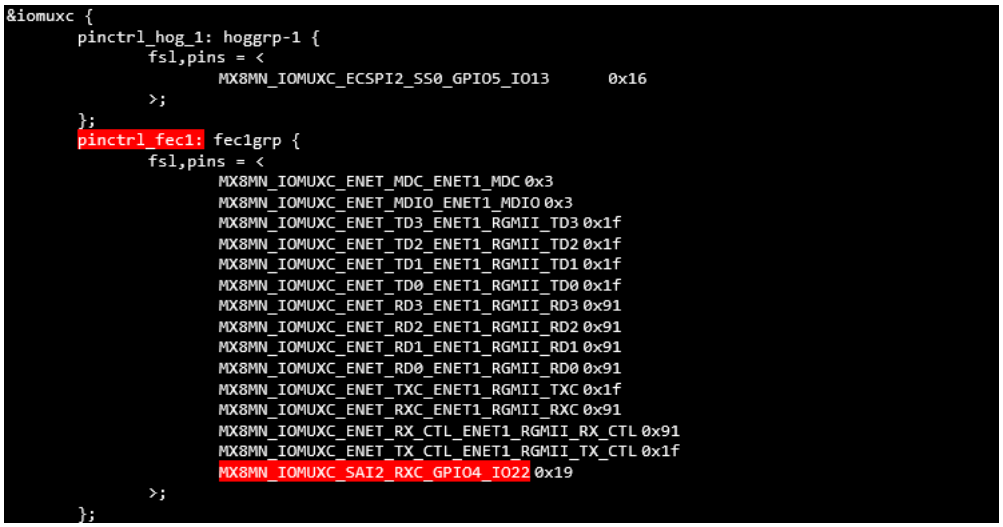
```
#ifdef CONFIG_SPL_OS_BOOT
       case IH_OS_LINUX:
       debug("Jumping to Linux\n");
#if defined(CONFIG_SYS_SPL_ARGS_ADDR)
      spl_fixup_fdt((void *)CONFIG_SYS_SPL_ARGS_ADDR);
#endif
       spl_board_prepare_for_linux();
       typedef void __noreturn (*image_entry_noargs_t)
(void);

       image_entry_noargs_t image_entry =
 (image_entry_noargs_t)0x00960000;

       image_entry();
#endif
```

Note: *When `CONFIG_SPL_OS_BOOT` is defined, SPL calls the `dram_init_banksize()` function which causes an error resulting in a CPU reset. This is happening due to the usage of uninitialized functions, which can be avoided by allocating memory for the `gd->bd` structure before using it, in the `<yocto_build_dir>/tmp/work/imx8mn_lpddr4_evk-poky-linux/u-boot-imx/<specified_git_folder>/git/arch/arm/mach-imx/imx8m/soc.c`) source file.*

```
gd->bd = (struct bd_info*)malloc(sizeof(struct bd_info));
```

7. To bring it up in the operational state in which Ethernet MAC can interact with PHY, reset the Ethernet PHY in U-boot SPL or Linux. The following instructions detail how to reset the PHY from U-Boot SPL:
   a. Check at which pin is connected the FEC device. This is described in the associated DTS file of i.MX8MN EVK board.

Location: <yocto_build_dir>/tmp/work/imx8mn_lpddr4_evk-poky-linux/u-boot-imx<specified_git_folder>/git/arch/arm/dts/imx8mn-evk.dtsi



According to board Schematics, the reset pin of Ethernet PHY is connected to group 4, pin 22 (marked in red).

b. Declare a macro containing the pair of GPIO group and GPIO pin in the file. Add a macro to use a pad as a GPIO.
Location: <yocto_build_dir>/tmp/work/imx8mn_lpddr4_evk-poky-linux/u-boot-imx/<specified_git_folder>/git/common/spl/spl.c

```
#define RESET_ETH_GPIO IMX_GPIO_NR(4,22)
#define USDHC_GPIO_PAD_CTRL (PAD_CTL_HYS | PAD_CTL_DSE1)
```

c. Reset the Ethernet PHY by adding the following lines in the board_init_r() function.
Location: <yocto_build_dir>/tmp/work/imx8mn_lpddr4_evk-poky-linux/u-boot-imx/<specified_git_folder>/git/common/spl/spl.c

```
  /*reset eth*/
  gpio_request(RESET_ETH_GPIO, "reset_eth_gpio");
  gpio_direction_output(RESET_ETH_GPIO, 0);
  mdelay(1);
  gpio_direction_output(RESET_ETH_GPIO, 1);
  mdelay(1);
```

8. Before creating a patch, make sure that all the new modifications are correct. Rebuild the U-Boot image using the following bitbake commands on the host:

```
$ bitbake -f -c configure u-boot-imx
$ bitbake -f -c compile u-boot-imx
$ bitbake u-boot-imx imx-boot
```

9. After the setup is done, create a patch which contains the U-Boot modifications:

```
$ git add --all
$ git commit -s
```

```
$ git format-patch HEAD~1
```

Copy the resulting patch to the following location: `imx-yocto-bsp/sources/meta-imx/meta-bsp/recipes-bsp/u-boot/u-boot-imx`, then add the name of the patch into the source location identifier in the Yocto recipe for U-Boot (`imx-yocto-bsp/sources/meta-imx/meta-bsp/recipes-bsp/u-boot/u-boot-imx_2021.04.bb`):

```
SRC_URI = "... \
       file://<patch_name>.patch \
      "
```

To check that the patch works after the modifications, apply the following commands. Then, the files should contain the latest changes.

```
$ bitbake -f -c cleansstate u-boot-imx
$ bitbake u-boot-imx imx-boot
```

10. Generate the bootloader image with SPL only.
    Create a new target for building the new image, in the `<yocto_build_dir>/tmp/work/imx8mn_lpddr4_evk-poky-linux/imx-boot/<specified_git_folder>/git/iMX8M/soc.mak` file.
    If the board is configured with HDMI, add these lines in `ifeq($(HDMI),yes)` condition.

```
flash_evk_falcon: $(MKIMG) signed_hdmi_imx8m.bin u-boot-spl-ddr.bin
        ./mkimage_imx8 -fit -signed_hdmi
 signed_hdmi_imx8m.bin -loader u-boot-spl-ddr.bin
 $(SPL_LOAD_ADDR) -out $(OUTIMG)
```

    Else, make a new target that will not include HDMI and implement it:

```
flash_evk_falcon: flash_evk_falcon_no_hdmi

flash_evk_falcon_no_hdmi: $(MKIMG) u-boot-spl-ddr.bin
      ./mkimage_imx8 -version $(VERSION) -loader u-boot-spl-ddr.bin
$(SPL_LOAD_ADDR) -out $(OUTIMG)
```

    To regenerate `flash.bin` (imx-boot image), run `make SOC=iMX8MN flash_evk_falcon`.

11. Create a patch that will include the modifications above:

```
$ git add soc.mak
$ git commit -s
$ git format-patch HEAD~1
```

    Copy the resulted patch to `~/imx-yocto-bsp/sources/meta-imx/meta-bsp/recipes-bsp/imx-mkimage/files`, and then add the name of the patch into the source location identifier in the Yocto recipe for `imx-mkimage` (`~/imx-yocto-bsp/sources/meta-imx/meta-bsp/recipes-bsp/imx-mkimage/imx-mkimage_git.inc`):

```
SRC_URI = "... \
      file://<patch_name>.patch \
      "
```

To check that the patch works after the modifications, apply the following commands. Then, the file should contain the latest changes:

```
$ bitbake -f -c cleansstate imx-boot
$ bitbake imx-boot
$ make SOC=iMX8MN flash_evk_falcon
```

12. Write the bootloader image on SD card.
Note: *Storage location may vary. Adjust the* `of` *parameter to point to the SD card location.*

```
dd if=flash.bin of=/dev/sdb bs=1k seek=32 conv=fsync
```

13. Configure the ATF to jump to Kernel image.
In the `bl31_early_platform_setup2()` function, set the program counter member of `bl33_image_ep_info` structure to kernel address and give the address of the FDT as an argument.
Location: `<yocto_build_dir>/tmp/work/cortexa53-crypto-mx8mn-poky-linux/imx-atf/<specified_git_folder>/git/plat/imx/imx8m/imx8mn/imx8mn_bl31_setup.c`

```
  //bl33_image_ep_info.pc = PLAT_NS_IMAGE_OFFSET;
bl33_image_ep_info.pc = 0x40400000; // RAM kernel address
bl33_image_ep_info.spsr = get_spsr_for_bl33_entry();
// RAM FDT address
bl33_image_ep_info.args.arg0 = (u_register_t)0x43000000;
bl33_image_ep_info.args.arg1 = 0U;
bl33_image_ep_info.args.arg2 = 0U;
bl33_image_ep_info.args.arg3 = 0U;
SET_SECURITY_STATE(bl33_image_ep_info.h.attr, NON_SECURE);
```

14. Recompile the ATF sources.
Before making a patch, make sure that the sources compile.

```
$ bitbake imx-atf
```

You can find the ATF binary (`bl31.bin`) in the `build/imx8mn/release` directory.
Create a patch which includes the new modifications.

```
$ git add plat/imx/imx8m/imx8mn/imx8mn_bl31_setup.c
$ git commit -s
$ git format-patch HEAD~1
```

Copy the resulting patch to the following location: `imx-yocto-bsp/sources/meta-imx/meta-bsp/recipes-bsp/imx-atf/files`, append a new recipe to the existing one (create `imx-atf_2.4.bbappend` file), and add these lines in `imx-yocto-bsp/sources/meta-imx/meta-bsp/recipes-bsp/imx-atf/imx-atf_2.4.bbappend`.

```
FILESEXTRAPATHS_prepend := "${THISDIR}/files:"
SRC_URI += "file://0001-jump-to-kernel.patch \
     "
```

To check that the patch works after the modifications, apply the following commands. Then, the file should contain the new changes:

```
$ bitbake -f -c cleansstate imx-atf
```

```
$ bitbake imx-atf
```

Note: *To build the ATF manually, use the Yocto meta-toolchain.*

```
$ bitbake meta-toolchain
$ cd imx-yocto-bsp/build-wayland/tmp/deploy/sdk
$ ./fsl-imx-wayland-glibc-x86_64-meta-toolchain-cortexa53-
crypto-imx8mn-lpddr4-evk-toolchain-5.10-hardknott.sh
$ source /opt/fsl-imx-wayland/5.10-hardknott/environment-
setup-cortexa53-crypto-poky-linux; unset LDFLAGS
$ cd <yocto_build_dir>/tmp/work/cortexa53-crypto-mx8mn-poky-
linux/imx-atf/<specified_git_folder>/git
$ make PLAT=imx8mn bl31
```

15. Write the ATF on SD card.
    Note: *Storage location may vary. To point to the SD card location, adjust the* `of` *parameter.*

```
dd if=bl31.bin of=/dev/sdb bs=512 seek=50060000 conv=fsync
```

16. Choose the entry point and load addresses of the Kernel image.
    Before booting the kernel, U-Boot relocates the kernel image to an address which is multiple of 2 MB. To bypass this relocation, the entry point address has to be set in advance using the same criteria (address has to be a multiple of 2 MB). The address from where the kernel will be executed is `0x40400000`. Considering that the uImage has a 64-byte header, the load address will be `0x403fffc0`.

17. Build the Kernel legacy `uImage` file from Image.
    `uImage` is a special image file that adds a 64-byte header before Image, where loader information is specified (load address, entry point, OS type, and so on).
    a. Change the directory where the Kernel image is deployed after build.

```
$ cd <yocto_build_dir>/tmp/work/imx8mn_lpddr4_evk-poky-
linux/linux-imx/<specified_git_folder>/build/arch/arm64/
boot
```

    b. Generate uImage using U-Boot's mkimage command:

```
$ mkimage -A arm -O linux -T kernel -C none -a 0x403FFFC0
 -e 0x40400000 -n "Linux kernel" -d Image  uImage
Image Name:   Linux kernel
Created:      Wed May  4 11:02:52 2022
Image Type:   ARM Linux Kernel Image (uncompressed)
Data Size:    30118400 Bytes = 29412.50 KiB = 28.72 MiB
Load Address: 403fffc0
Entry Point:  40400000
```

    Where:
    - A [*architecture*]: To set architecture.
    - O [*os*]: To set operating system.
    - T [*image type*]: To set image type.
    - C [*compression type*]: To set compression type.
    - n [*image name*]: To set image name to **image name**.
    - d [*image data file*]: To use image data from **image data file**.
    - a [*load address*]: To set load address with a hex number.
    - e [*entry point*]: To set entry point with a hex number.

18. Prepare the Flattened Device Tree and write it on SD card.
When booting in Falcon Mode, an important step is to prepare the device tree. Usually, U-Boot does FDT fixups when booting Linux. It means that to the initial device tree, U-Boot adds Kernel arguments.
These arguments can be found in one of the configuration files: `<yocto_build_dir>/tmp/work/imx8mn_lpddr4_evk-poky-linux/u-boot-imx/<specified_git_folder>/git/include/configs/imx8mn_evk.h`, under the name `bootargs`, where the console parameters are specified and whichh tells the kernel where to find the root file system.
To skip U-Boot, the FDT has to be prepared in advance by using `spl export` command. It should be called under the normal boot. The command it is equivalent to go through `bootm` until device tree fixup is done. The device tree in memory is the one needed for falcon mode. This image has to be saved to SD at the location pointed by macro `CONFIG_SYS_MMCSD_RAW_MODE_ARGS_SECTOR`, with maximum size specified by macro `CONFIG_SYS_MMCSD_RAW_MODE_ARGS_SECTORS`.
Steps to prepare the FTD:
- Boot the board into U-Boot and stop it right before entering in autoboot sequence.
- Load the FDT into RAM;

```
u-boot=> run loadfdt
41608 bytes read in 2 ms (19.8 MiB/s)
```

- Load the Kernel uImage into RAM.

```
u-boot=> fatload mmc ${mmcdev}:${mmcpart} ${loadaddr}
 uImage
30118464 bytes read in 320 ms (89.8 MiB/s)
```

- Write the Kernel Image to SDHC at the specified offset sector;

```
u-boot=> mmc write ${loadaddr} 0x2FAF0E4 0xe5c9
MMC write: dev # 1, block # 50000100, count 58825 ... 58825
 blocks written: OK
```

- Prepare FDT

```
u-boot=> spl export fdt ${loadaddr} - ${fdt_addr_r}
## Booting kernel from Legacy Image at 40400000 ...
   Image Name:   Linux kernel
   Created:      2022-04-08  13:08:28 UTC
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    30118400 Bytes = 28.7 MiB
   Load Address: 403fffc0
   Entry Point:  40400000
   Verifying Checksum ... OK
## Flattened Device Tree blob at 43000000
   Booting using the fdt blob at 0x43000000
   Loading Kernel Image
   Using Device Tree in place at 0000000043000000, end
 000000004300d287
subcommand not supported
subcommand not supported
  Using Device Tree in place at 0000000043000000, end
 0000000043010287
Argument image is now in RAM: 0x0000000043000000
```

- Write the prepared FDT to SD card.

```
u-boot=> mmc write 0x43000000 0x2FAF080 0x58
```

Note:
It is possible to rewrite the FDT on SD card. To save it locally on PC, use the following command:

```
dd if=/dev/sdb of=imx8mn-evk.dtb bs=512 skip=50000000
 count=88 conv=fsync
```

Note: *Storage location may vary. To point to the SD card location, adjust the* `of` *parameter.*
After the modifications, the resulted SD card looks like Figure 3.

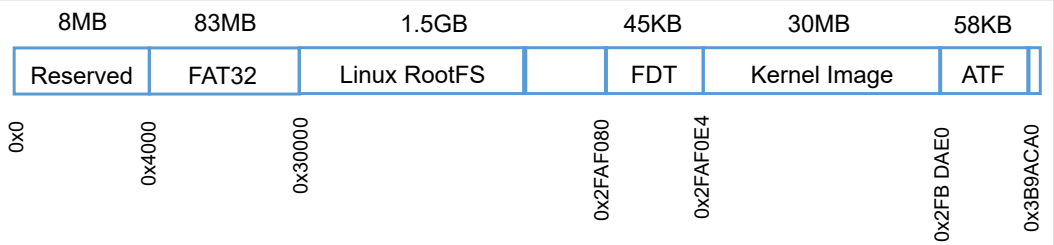


Figure 3. SD card structure

19. Uncomment the definition of `CONFIG_SPL_OS_BOOT` from Step 2.

20. Recompile and write the bootloader on SD card.

```
$ bitbake -f -c configure u-boot-imx
$ bitbake -f -c compile u-boot-imx
$ cd <yocto_build_dir>/tmp/work/imx8mn_lpddr4_evk-poky-linux/
imx-boot/<specified_git_folder>/git/
$ make SOC=iMX8MN flash_evk_falcon
```

Write the U-Boot image on SD card.
Note: *Storage location may vary. To point to the SD card location, adjust the* `of` *parameter.*

```
dd if=flash.bin of=/dev/sdb bs=1k seek=32 conv=fsync
```

4.4 Improve raw MMC read performance in SPL

After implementing Falcon mode, it can be observed that the time spent in SPL increased significantly. It is due to the fact that the kernel image is loaded into memory slowly (large size ~30 MB). To increase the speed transfer in SPL, add support for high speed (UHS/SD):

Location: `<yocto_build_dir>/tmp/work/imx8mn_lpddr4_evk-poky-linux/u-boot-imx/<specified_git_folder>/git/configs/imx8mn_evk_defconfig`

```
CONFIG_SPL_MMC_UHS_SUPPORT=y
CONFIG_SPL_MMC_IO_VOLTAGE=y
```

Note: *You can see all the configurations available in* `<yocto_build_dir>/tmp/work/imx8mn_lpddr4_evk-poky-linux/u-boot-imx/<specified_git_folder>/git/Kconfig` ***file.***

Recompile the bootloader with these modifications and write it as in Step 20.

5 Kernel space optimizations

5.1 Adding `quiet`

To reduce the Kernel time by about a half, add `quiet` argument in Kernel `bootargs`. It suppresses some messages during the Linux start-up sequence.

The device tree have to be regenerated with the new `bootargs`, using the `spl export` command.

1. Reboot in default boot mode.
    a. Comment the definition of `CONFIG_SPL_OS_BOOT` from `<yocto_build_dir>/tmp/work/imx8mn_lpddr4_evk-poky-linux/u-boot-imx/<specified_git_folder>/git/include/configs/imx8mn_evk.h`.
    b. Recompile & rewrite the bootloader as in Step 20.
2. Enter U-Boot and edit the `bootargs` parameter by adding `quiet`.

```
u-boot=> edit bootargs
edit: console=ttymxc1,115200 root=/dev/mmcblk1p2 rootwait rw
 quiet
u-boot=> saveenv
Saving Environment to MMC... Writing to MMC(1)... OK
```

3. Regenerate and load the device tree to SDHC as in Step 18.
4. Reenter in Falcon mode following Step 19 and Step 20.

5.2 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 can analyze Kernel functions during boot with **bootgraph**, a Kernel feature that allows to graph what happens in Kernel during initializations.

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

1. Add `initcall_debug` to Kernel `bootargs`.
    a. Reboot in default boot mode as in Step 1.
    b. Enter U-Boot and edit the `bootargs` parameter by adding `initcall_debug`.

```
u-boot=> edit bootargs
edit: console=ttymxc1,115200 root=/dev/mmcblk1p2 rootwait
 rw quiet initcall_debug
u-boot=> saveenv
Saving Environment to MMC... Writing to MMC(1)... OK
```

2. Regenerate and load the device tree to SDHC as in Step 18.
3. Reenter in Falcon Mode following the Step 19 and Step 20.
4. Boot the board and get the Kernel log.

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

5. Go back on the host and create the graph using the following commands:

```
$ cd <yocto_build_dir>/tmp/work-shared/imx8mn-lpddr4-evk/
kernel-source/scripts
```

```
$ ./bootgraph.pl boot.log > boot.svg
```

You will obtain something like this and can analyze how the Kernel boot time is used.

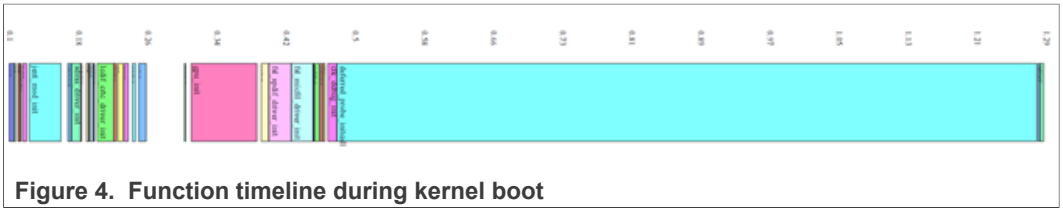


Figure 4. Function timeline during kernel boot

6. To disable a driver or a feature, update the Kernel configuration.
   For example, we disabled the debug from Kernel (that reduce the size of the image) and the UBI file system.
   a. In `sources/meta-imx/meta-bsp/recipies-kernel/linux/files`, create a fragment configuration file `frag.cfg` in which you include these lines:

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

   b. For this to be merged with the default configuration file, we have to append a new recipe. Create `linux-imx_5.10.bbappend` in `sources/meta-imx/meta-bsp/recipies-kernel/linux` and add these lines:

```
FILESEXTRAPATHS_prepend := "${THISDIR}/files:"
SRC_URI += "file://frag.cfg \
      "
DELTA_KERNEL_DEFCONFIG = "frag.cfg"
```

   c. Recompile the Kernel with the new configuration, convert into `uImage` with `mkimage` and write it to SD.

```
$ bitbake -c cleansstate virtual/kernel
$ bitbake -f -c compile virtual/kernel
```

   d. Convert the Kernel image into uImage as in Step 17.
   e. Write the `uImage` Kernel to SD card.

Note: *Storage location may vary. To point to the SD card location, adjust the `of` parameter.*

```
dd if=uImage of=/dev/sdb bs=512 seek=50000100 conv=fsync
```

6 User space optimizations

The easiest way to reduce the time spent in user-space is to reorder the sequence in which applications are run. To start the `psplash` service earlier, change the dependencies with which `Systemd` operates.

On board, open `/lib/systemd/system/psplash-start.service` file and change the unit dependencies by making `psplash` started before `local-fs-pre.target`.

```
[Unit]
Description=Start Psplash Boot Screen
#Wants=systemd-vconsole-setup.service
#After=systemd-vconsole-setup.service systemd-udev-
trigger.service systemd-udevd.service
Before=local-fs-pre.target
DefaultDependencies=no
```

If the command system-analyze is called with blame argument, `Systemd` also provides a utility called systemd-analyze which prints the services and their starting time.

```
$ systemd-analyze blame
```

To disable a service, you can use the `systemctl disable` command. Some services (especially the ones provided by `systemd`) might need the `systemctl mask` command to disable them. However, take care when disabling services since the system may depend on them to operate properly.

7 Results

Table 2. Initial Linux boot time measurements

	SPL			U-BOOT				KERNEL	
No.	BOOTROM	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	Kernel boot until PSPLASH image	Total time
	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)
1	260	253	285	594	906	3651	329	5768	12046
2	299	253	285	594	1016	3852	328	5920	12547
3	258	252	284	587	901	3730	328	5902	12242
4	257	253	284	587	896	3726	328	5846	12177
5	261	253	284	587	896	3726	328	5975	12310
Average time (ms)	**805**			**5579**				**5883**	**12264**

Table 3. Optimized Linux boot time measurements

	SPL				KERNEL	
No.	BOOTROM	DDR initialization	SPL initializations	Kernel Image Load	Kernel Boot Until PSPLASH Image	Total time
	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)
1	262	252	128	460	2661	3763
2	264	253	130	461	2772	3880
3	249	252	129	460	2924	4014
4	255	252	129	460	2717	3813
5	253	253	131	461	2594	3692
Average time (ms)	**1099**				**2734**	**3832**

8 References

- *i.MX 8M Nano Applications Processor Reference Manual* (document IMX8MNRM)
- Presentation: Understanding U-Boot Falcon Mode, Michael Opdenacker, June 3rd 2021
- U-Boot Source Code - Falcon README

9 Revision history

Rev.	Date	Description
0	18 August 2022	Initial release

10 Legal information

10.1 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.

10.2 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 http://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 data sheet 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.

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.

10.3 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

For more information, please visit: http://www.nxp.com
For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 18 August 2022
Document identifier: AN13709