Sturmflut's blog About

MediaTek details: Little Kernel

Jul 5, 2015

NOTE: This is a continuation of the previous article in the series. The information was obtained from various sources and through reverse engineering, don't take it as a reference!

UPDATE 04.05.2015: I've come across this github repository, which contains the missing source code parts! The article is now much more detailed than before!

The Preloader has done its work, but you might have noticed that some parts of the puzzle are still missing.

1. Who cares about the charging animation when the phone is off and the USB charger is attached? The Preloader doesn't.
2. Who shows the Boot Menu and then actually executes the correct boot mode, e.g. Recovery instead of normal boot? The Preloader passes on some boot arguments, but it always loads and executes the same flash partition at the end.
3. Who shows the boot logo during normal boot? The Preloader doesn't.
4. Who implements all those cool Android features (Boot menu, Recovery, Fastboot)? The Preloader certainly doesn't.

There must be some crucial piece missing between the Preloader and the actual Operating System kernel. On MediaTek SoCs, this component is called "Little Kernel" (LK). It is a kind of second stage bootloader, because the Preloader has to be small enough to fit into the On-chip SRAM when loaded by the Boot ROM.

Tho following sections do not necessarily correspond to sections in the code, but the order is correct.

Early thread init

Turns out the Little Kernel has a multithreaded design with priority scheduling! The very first step is to create the necessary data structures and encapsulate the running context into the first thread and

Architecture and platform init

Yet again the first step is be to bring up all necessary hardware, most importantly the caches, the MMU, the flash storage, power management circuits, the GPIO pins and the display. The interesting thing here is that the Little Kernel comes with a minimal display driver for the used platform in `mediatek/platform/${platform}/lk/mt_disp_drv.c`, in case of the MT6582 it's a small driver for the ARM Mali GPU that just enables the display, allocates a full-screen framebuffer in RGB565 format and hard-wires the GPU to display it.

The next step after hardware bring-up is to load the logo from the `LOGO` partition. Since there is no color space conversion available at the driver level, the logo has to be stored in "raw" format. The `mediatek/custom/common/lk/logo/tool/bmp_to_raw` tool can be used to convert BMP images. Note that the logo is only loaded in this step, but not yet displayed.

Chose the boot mode

Execution of LK may start for various reasons: The device is off and the charger connected, the device was rebooted, the device is powered on normally etc. All of these modes require different code paths, and the manufacturer may also want to give the user the possibility to modify the boot mode by pressing the hardware keys. As always much of the functionality can be enabled or disabled by the manufacturer, and potential key bindings changed.

Let’s remember the available boot modes:

```
typedef enum
{
    NORMAL_BOOT = 0,
    META_BOOT = 1,
    RECOVERY_BOOT = 2,
    SW_REBOOT = 3,
    FACTORY_BOOT = 4,
    ADVMETA_BOOT = 5,
    ATE_FACTORY_BOOT = 6,
    ALARM_BOOT = 7,
#if defined (MTK_KERNEL_POWER_OFF_CHARGING)
    KERNEL_POWER_OFF_CHARGING_BOOT = 8,
    LOW_POWER_OFF_CHARGING_BOOT = 9,
#endif
    FASTBOOT = 99,
    DOWNLOAD_BOOT = 100,
    UNKNOWN_BOOT
} BOOTMODE;
```

I think I found out what most of those modes actually mean:

- `NORMAL_BOOT`: Boots the Operating System.
- `META_BOOT`: This is a special “Mobile Engineering Testing Architecture” mode for BaseBand testing during device development. It gets set by the Preloader if requested by the Flash Tool.
- `RECOVERY_BOOT`: Load the Recovery image from the `RECOVERY` partition, verify it and boot the image.
- `SW_REBOOT`: Like `NORMAL_BOOT`.
- `FACTORY_BOOT`: Boots into Factory Mode.
- `ADVMETA_BOOT`: Most likely an advanced version of `META_BOOT`. It gets set by the Preloader if requested by the Flash Tool.
- `ATE_FACTORY_BOOT`: This is a second factory mode that can be enabled during the Preloader stage and is intended for automated calibration of the BaseBand part on the production line.
- `ALARM_BOOT`: Like a `NORMAL_BOOT`.
- `KERNEL_POWER_OFF_CHARGING_BOOT` and `LOW_POWER_OFF_CHARGING_BOOT`: Boot the OS after a power off.
- `FASTBOOT`: Power on the USB port and start the Fastboot interpreter, which is part of LK.
- `DOWNLOAD_BOOT`: This is set by the Preloader if the Preloader was put into Download mode. I don’t know why LK cares about it, because LK has no Download mode, it has Fastboot.

The following possibilities to modify the boot mode during LK initialisation exist:

- The Operating System can set two special bits in the memory of the real-time clock and reboot the device. If set, one of those bits will switch the LK into `FASTBOOT` mode, and the other into `RECOVERY_BOOT`. This is how the `adb reboot [bootloader|recovery]` command works.
- Various keys and key combinations will switch to `RECOVERY_BOOT`, `FACTORY_BOOT` or the Boot Menu. The manufacturer can define these keys for every device, look here and here for examples for the bq Aquaris E4.5 Ubuntu Edition.

Boot Menu

If triggered by the hardware key during startup, LK will switch to a text mode emulation and display the following Boot Menu:

```
Select Boot Mode:
[VOLUME_UP to select.  VOLUME_DOWN is OK.]

[Recovery    Mode]
[Fastboot    Mode]
[Normal      Boot]
[Normal      Boot +ftrace]
[Normal      slub debug off]
```

Option 1 will select `RECOVERY_BOOT`, option 2 `FASTBOOT` and all other options `NORMAL_BOOT`. The last two options will add the flags `trace_buf_size=11m boot_time_ftrace` or `slub_debug=-` to the kernel commandline.

The last two options are usually not enabled on production devices.

Further initialisation

Like the Preloader, LK uses the SecLib binary blob.

At this step some more hardware, like the battery controller and the RTC, is initialised, and almost all boot modes will have LK show the logo now.

The `KERNEL_POWER_OFF_CHARGING_BOOT` and `LOW_POWER_OFF_CHARGING_BOOT` modes for charging while the phone is powered off use the notification LED to indicate charging status. The code actually distinguishes between three charging levels, “low”, “medium” and “full”, but for some reason both “low” and “medium” are hardwired to the same color: Red.

LK theoretically supports Block I/O and can even mount ext2 and FAT32 file systems, but this functionality doesn’t seem to be used on production devices.

Initialize the “apps”

Since LK is multithreaded, it can run multiple things at the same time. Looking at the source code and leaving all the test code out of the picture, the following three apps would do something sensible:

- `aboot`: The default Android boot loader, includes a Fastboot interpreter.
- `mt_boot`: The MediaTek boot loader, includes a Fastboot interpreter.
- `shell`: A flexible shell, accessible via an UART. Other subsystems can add new commands.

It looks like only the `mt_boot` app runs on production devices.

Booting an Android boot image

If the boot mode is not `FASTBOOT`, the `mt_boot` thread will always immediately continue to try to boot one of the Android boot images from an SD card or the internal flash storage.

`NORMAL_BOOT`, `META_BOOT`, `ADVMETA_BOOT`, `SW_REBOOT`, `ALARM_BOOT`, `KERNEL_POWER_OFF_CHARGING_BOOT` and `LOW_POWER_OFF_CHARGING_BOOT` will all boot the regular Android boot image from the internal `BOOTIMG` partition.

`RECOVERY_BOOT` will load the Android boot image from the internal `RECOVERY` partition instead.

`FACTORY_BOOT` and `ATE_FACTORY_BOOT` will try to load an Android boot image stored in the file `factory.img` on the SD card, if that fails the `BOOTIMG` partition is used instead.

In all cases five fields will be added to the kernel command line: `lcm` for information about the liquid crystal module, `fps` for the number of frames per second the display is currently running at (times 100), `pl_t`, the Preloader boot time, `lk_t`, the Little Kernel boot time (both in milliseconds), and `boot_reason`, the boot reason (not to confuse with the boot mode).

The last step in `mt_boot` is to prepare the environment for the Linux kernel. The kernel image and the initrd are loaded to fixed, well-known memory addresses. `mt_boot` also prepares a special “tag” block in memory, a data structure that is filled with information the kernel needs later and knows how to decode. The following information is added to the tag block in the specified order:

- The magic values `2` and `0x54410001`
- A tag size (4), the magic value `0x41000802` and the boot mode number
- A platform-specific representation of the memory topology. On the MT6582, the following fields are added for every physical DRAM bank: the tag size (4), the magic value `0x54410002` and the start and end addresses.
- If the boot mode is `META_BOOT` or `ADVMETA_BOOT`: The tag size (8), the magic value `0x41000803`, the META communication type passed by the Preloader (USB or UART) and the ID of the COM port.
- A tag size (88), the magic value `0x41000804` and then 21 (yes, exactly 21) device entries. The exact format of the entries is unknown because they come from a binary blob.

- A tag size (variable), the magic value `0x54410009`, the kernel cmdline string and the length of the kernel cmdline in 32 bit words
- A tag size (8), the magic value `0x54420005`, the start address of the initrd and its size
- A tag size (24), the magic value `0x54410008`, followed by a 24 byte record with information about the framebuffer

A jump transfers control to Linux.

FASTBOOT mode

The implementation is very compact and everything important is described here.

If you know better and/or something has changed, please find me on Launchpad.net or the Freenode IRC and do get in contact!

Resources

- iq451_mt6589 github repository
- Thunder-Kernel
- Introduction of MTK Tools