Little Kernel Customer Document

Version: 1.1
Release date: 2016-06-236/23/2016 4:14:00 PM

Document Revision History

Revision	Date	Author	Description
0.9	2011-11-30	Hong-Rong Hsu	Refine uboot portion
0.91	2011-12-01	Infinity Chen	Refine pre-loader portion
0.92	2012-07-27	Hong-Rong Hsu	Replace uboot by Little Kernel
0.93	2013-06-19	Chieh-jay Liu	Refine Little Kernel portion
0.94	2013-10-28	Chieh-jay Liu	Refine Little Kernel portion
0.95	2014-10-14	Yi-Lung Tsai	Refine pre-loader portion
0.96	2015-02-09	Sten Liao	Refine Little Kernel portion
0.97	2015-04-24	Sten Liao	Refine Little Kernel portion
1.0	2016-03-10	Jim Du	Refine Little Kernel portion
1.1	2016-06-23	Jim Du	Refine Little Kernel portion

Table of Contents

1 Introduction

1.1 Software/Hardware Environment

1.1.1 Software Environment

The boot loader described in this document is able to prepare an essential execution environment and bring up the Linux operating system and Android framework. It can be divided into 1st boot loader, MTK in-house pre-loader, and 2nd boot loader, Little Kernel. This document only describes LK. Preloader is mentioned in another document.

1.1.2 Hardware Environment

The hardware environment described in this document is based on but not limit to phone chipset platform.

1.2 Functionality

The main functionality of LKis to initialize the essential execution environment, including setup processor and memory frequency, debug message port, bootable storages and so on. After execution environment is setup, following software is loaded to memory and executed. Except for software loading, an external tool also can handshake with PL and LKand indicate the device to enter different operation mode, such as USB download mode and META mode. Without tool handshaking, the LK also could be indicated to enter these modes via any combination or customized keys. For detail, please refer to key customization document.

1.2.1 Little Kernel Functionality

Little kernel is a non-GPL licensed loader and it's also a Linux loader. With Little kernel, we can easily boot up Linux Kernel because it has already handled all the Linux kernel boot argument and device tree for us.

Beside, you can find some auxiliary function in Little kernel source tree. It's strongly recommended to search the existing function first in Little kernel source tree before you start to implement any function on your own. (For example: file system, openssl, libc … etc)

1.2.1.1 The changes in LK from native LK

1. Add MTK related BSP drivers.
2. Add lk/app/mt_boot for mediatek boot up flow.
   lk directory: <lkdir>
   AOSP: vendor/mediatek/proprietary/bootable/bootloader/lk
3. Port laest Mediatek ARM based chipsets for LK. (cache line 32B)
4. Port zlib library for decoding boot logo.
5. Integrate LK build system into MTK build system.
6. Integrate customization folder into
   <lkdir> /target/<project>/
   ex: vendor/mediatek/proprietary/bootable/bootloader/lk/target/flora01v1
7. Gic v3 & 3-layer tlb

1.3 Hardware Background

When the device is up, the boot loaders are loaded into different memory regions for execution. The execution location of LK is 0x46000000 in external DRAM (the LK location is defined in target folder rules.mk, MEMBASE := 0x46000000), When boot loaders execute, several hardware modules are initialized to create an execution environment. Related hardware modules used in boot loaders will be described in following sections.

1.3.1 Hardware Components in Little Kernel

LK is the second loader and brought up by pre-loader.
Basically, the module which have been initialized in pre-loader is no need to be re-configure in LK

But some modules will reset again in LK just for reconfiguring the hardware registers to create a clean environment. Timer module is an example. In LK, timer will be reset again to clean the hardware count to reset the time. All the hardware modules initialized in LK are listed as below:

(1) Timer module:
- A. It's for resetting the hardware register to reset the time.
- B. It generates timer tick for schedule context-switch.

(2) Serial module:
- A. This module is for LK to configure its serial input / output system. After initializing this module, we can use the serial functions provided in LK such as "printf (..) .."

(3) I2C module:

(4) PMIC module:
- A. For power-off charging.

(5) PWM module:
- A. For backlight.

(6) RTC module:
- A. For record some reset scheme.

(7) LED module:
- A. With this module, device can notify the user of the current charging state

(8) Power Charging module:
- A. This module is response for power off charging, lower charging in the system.

(9) LCD module:
- A. With this module, device can display the logo or any notification message on the LCD panel

(10)NAND module:
- A. Since the LK also needs to read the image (e.g. kernel or ramdisk) from flash, it's necessary to initialize the NAND related function in LK

(11)eMMC module:

Support eMMC boot.

(12) USB modeuls:

Support fastboot

2 Design

2.1 Architecture

Architecture – Normal Boot up Procedure

1) An on-chip immortal Boot ROM contains a small program, which is executed automatically after system reset
2) Enter Boot Mode or Download Mode, BROM will check UART/USB CMD from tool
3) Verify 1st boot loader (pre-loader) from boot media
4) Load boot loader from boot media to ISRAM
5) The 1st boot loader initializes CPU, clock, and memory then loads ATF(ARM Trusted Firmware) and 2nd boot loader (Little Kernel) from boot media into DRAM. Jump to ATF then execute 2nd bootloader
6) The 2nd boot loader tries to initialize other board peripherals and devices such as LCM, backlight, GPIO.
7) The 2nd bootloader loads the Linux kernel from boot media and decompress the Linux kernel
8) The 2nd bootloader loads the ramdisk from boot media
9) The 2nd bootloader sets ups boot environment, prepares boot parameters and start Linux kernel

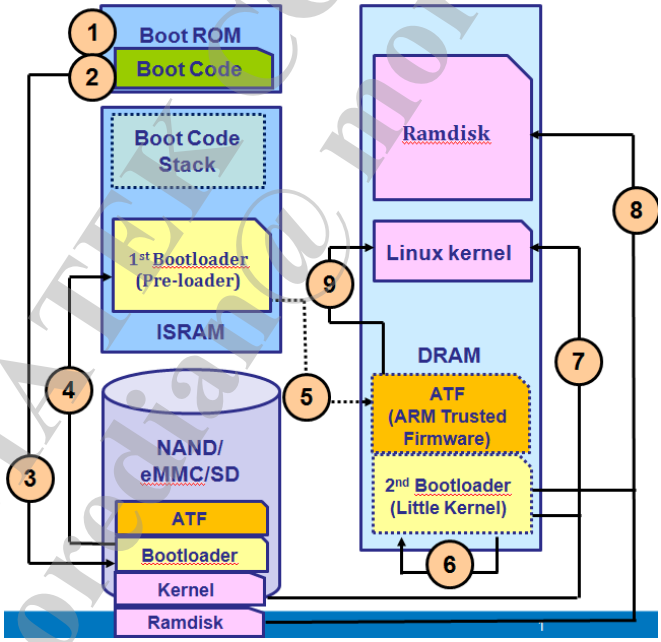


Figure 1 – Normal Boot up Procedure

2.1.1 Architecture - Normal Download Procedure

1) Boot ROM is activated when the device is powered on
2) BROM check pre-loader signature
3) BROM load pre-loader in ISRAM
4) Pre-loader executes in ISRAM and SYNC with FLASHTOOL , FLASHTOOL Via USB copy Pre-DA to ISRAM
5) FLASHTOOL Via USB copy DA binary to DRAM
6) DA SYNC with FLASHTOOL copy Pre-Loader to storage
7) DA download the LK to storage
8) DA download the Linux Kernel to storage
9) DA download other images to storage

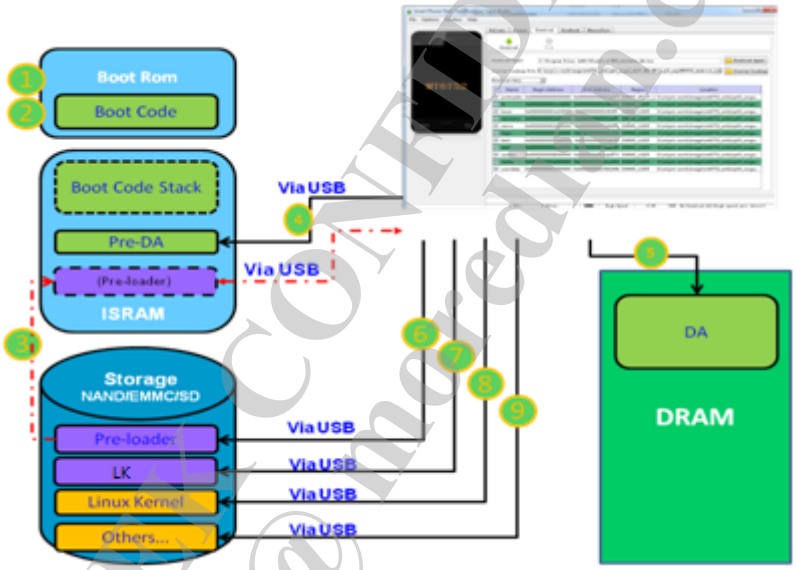


Figure 2 - Normal Download Procedure

2.1.2 Architecture - BROM Download Procedure

1) Boot ROM is activated when the device is powered on
2) Boot ROM initializes software stack, communication ports, and bootable storages
3) Boot ROM handshakes with flash tool via USB when emergency DL key is pressed
4) FLASHTOOL Via UART/USB copy Pre-DA to ISRAM
5) FLASHTOOL Via UART/USB copy DA binary to DRAM
6) DA SYNC with FLASHTOOL copy Pre-Loader to storage
7) DA download the LK to storage
8) DA download the Linux Kernel to storage
9) DA download other images to storage

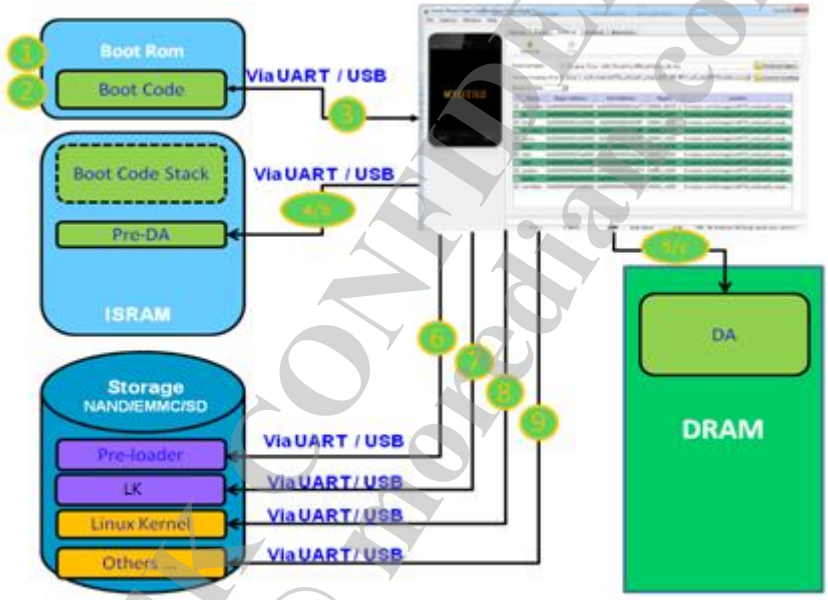


Figure 3 - BROM Download Procedure

2.2 Procedure & Flow

In this section, the function flow overview of boot up and LK is introduced.

2.2.1 Boot up flow

For the detail information about boot up flow cases, please refer to ATF related document

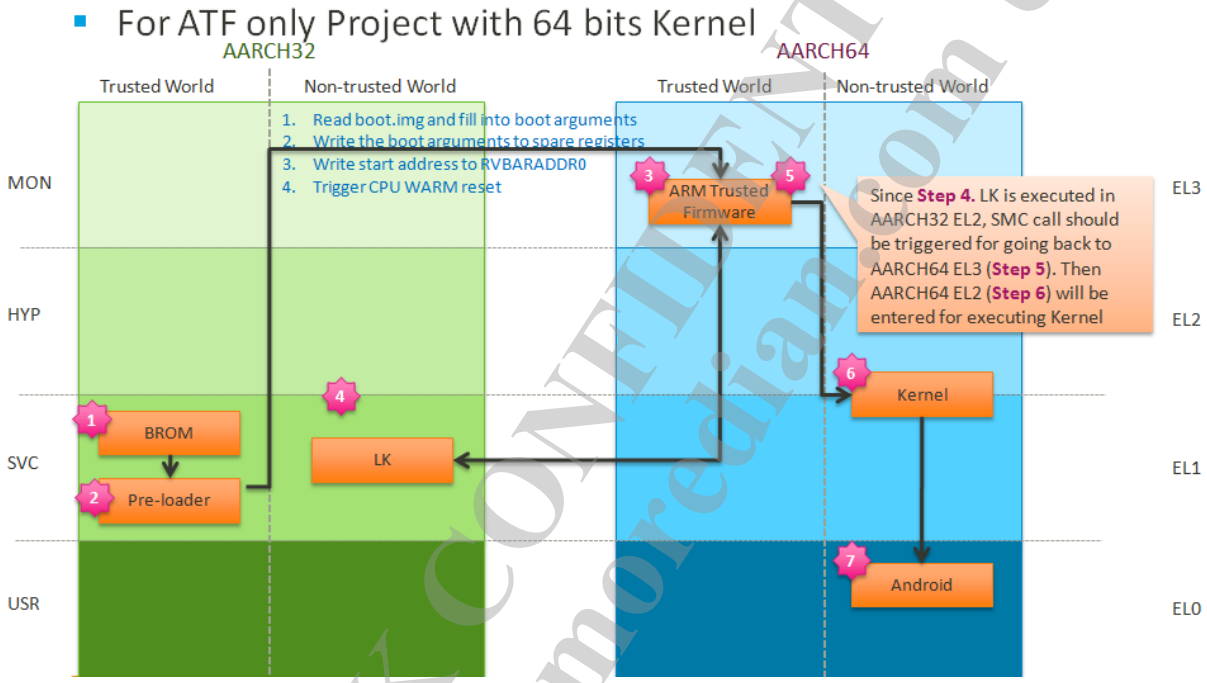
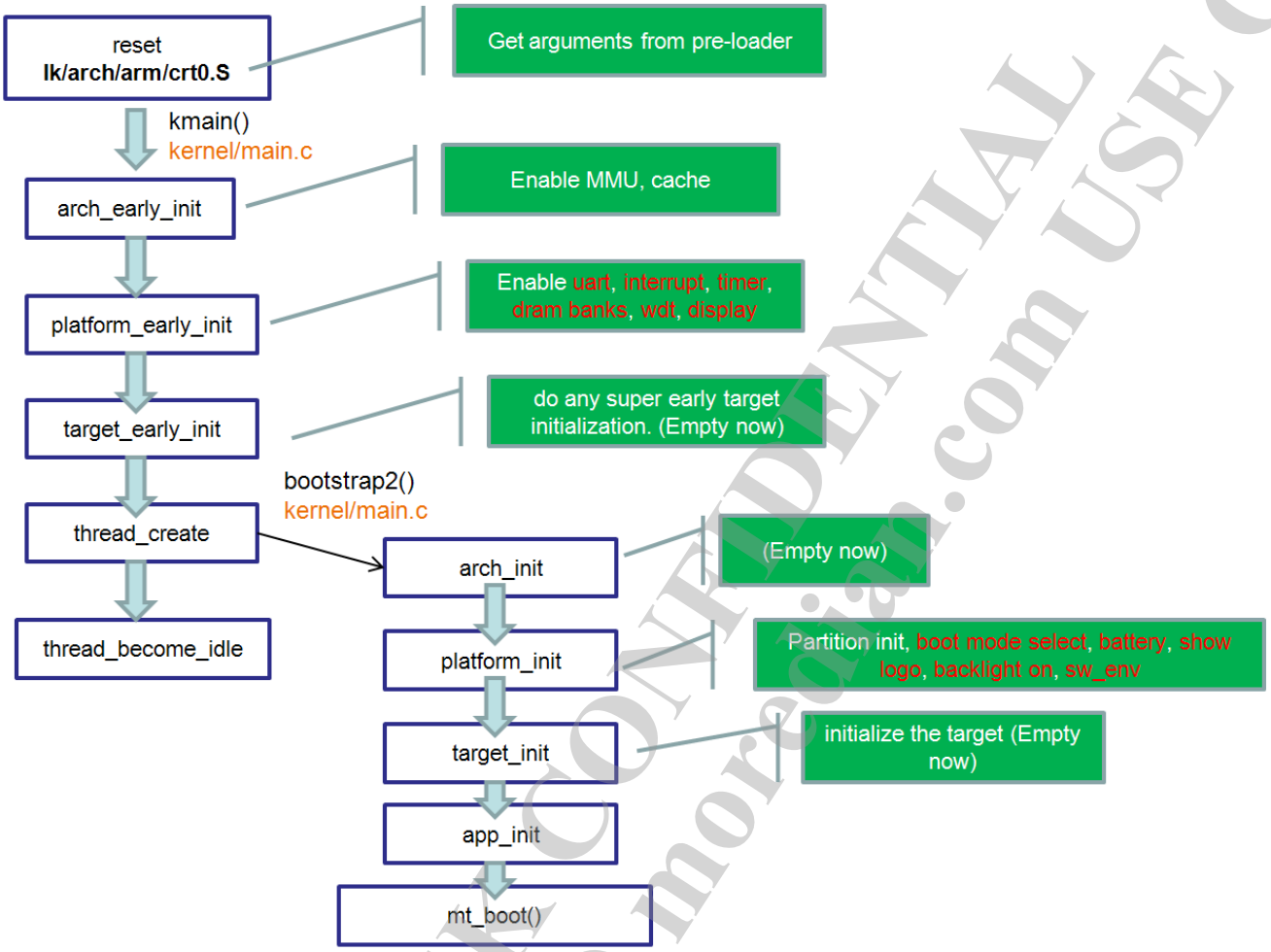
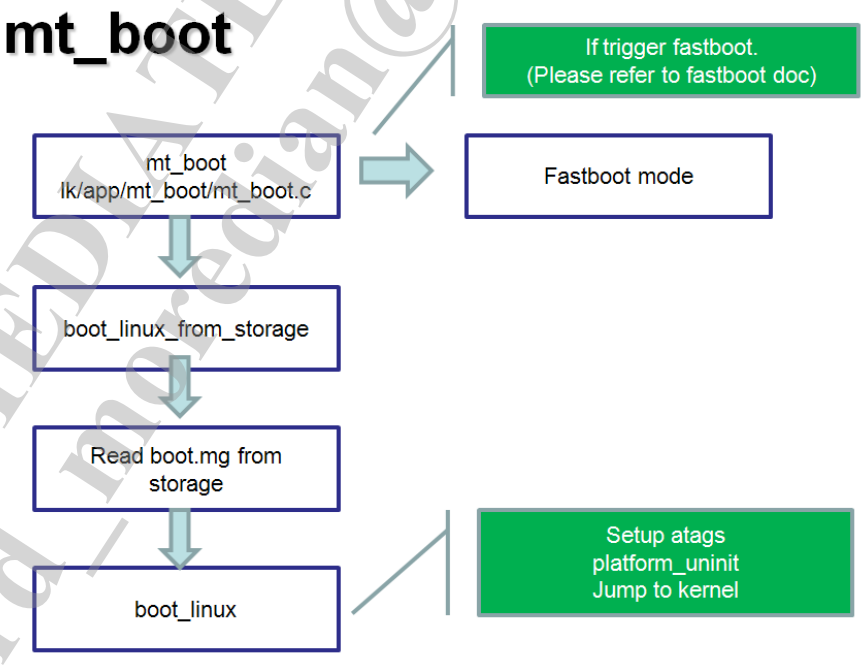


Figure 4 – Boot up flow

2.2.2 Little Kernel Procedure & Flow

Note that the source of lk is in
<lkdir>
AOSP: alps/vendor/mediatek/proprietary/bootable/bootloader/lk





3 Interface

This section describes important scenarios' interfaces and data structures.

3.1 Power on Scenario

3.1.1 LK Power on Scenario – Long Press to Boot

Introduction:

When booting to Little kernel, battery will check if the power key is pressed. If current boot reason is USB charger instead of pressing power key, battery module will wait user to press power key to boot (Please note that: if the charger is plugged in, the device would automatically power up. That's because the RTC PWBB is auto latched by HW)

Data Structure or Important Register:

Power Key's Keypad Code: 0x08

Function:

Prototype		
bool mtk_detect_key (u32 key)		
Parameters		
In	u32 key	Key code to detect
Return Value		
Return this key is pressed or not		
Description		
Call this function to check if specific key is pressed		
Location of Source Tree		
<lkdir>/platform/<chipset>/factory.c		

3.2 Boot Charging Scenario

3.3 Mode Selection Scenario

3.3.1 LK Mode Selection Scenario

Introduction:

LK supports three main kinds of boot modes

1) Factory mode:
   A) For manufactory testing when MP (For more detail information, please refer to Factory related document)
2) META mode:
   A) For functionality testing when MP (For more detail information, please refer to META related document)
3) Recovery mode:
   A) For SD card image upgrade (It's an image upgrade scheme provided by Android. User can upgrade Android system image from SD card. For more detail information, please refer to Recovery related document)
4) Advanced META mode:
   A) For functionality testing when MP (Unlike META mode, this mode is co-exist with normal Android boot. Usually used to test multimedia functions.)
5) ATE Factory boot:
   A) Comment with ATE tool (PC-side). Testing features via sending command from ATE tool.
6) Alarm boot:
   A) Boot-up reason is RTC alarm.
7) Fastboot boot:
   A) Enter fastboot mode.
8) Donwload boot:
   A) It is a feature that support show logo when downloading.
8) SW reboot:
   A) The boot reason is reboot. (ex. adb reboot)

When power on, LK needs to check current boot mode

Data Structure or Important Register:

1) boot type definitions

```
/* boot type definitions */
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;
```

* For more detail information, please refer to <lkdir>/platform/<chipset>/include/platform/boot_mode.h

2) Global variable to record the current boot mode (default is normal boot)

```
BOOTMODE g_boot_mode = NORMAL_BOOT;
```

Function:

Prototype
void boot_mode_select (void)
Parameters
None
Return Value
None
Description
Check the current boot mode and configure the correct boot mode
Location of Source Tree
<lkdir>\platform\<chipset>\boot_mode.c

4 Customization

4.1 Power on Customization

4.1.1 LK Power on Customization – Bypass "Long Press to Boot"

You can bypass the scenario of pressing power key to boot by letting the function "`mtk_detect_key(0x8)`" always return TRUE

4.2 Mode Selection Customization

4.2.1 LK Mode Selection Customization – Add New Boot Mode

Currently, LK supports eight modes: (1) META, (2) Recovery, (3) Factory (4) Advanced META …etc, and LK will send the boot mode information to Linux for further processing.
Because of key number limitation (some phone may not have so much key), we provide a TEXT menu (press a special key to enter) at LK for user to choose which mode they want to enter. (This is default on, you can disable the TEXT menu by marking "HAVE_LK_TEXT_MENU" in <lkdir>/platform/<chipset>/include/platform/mt_reg_base.h")

Phone case:
#If defined HAVE_LK_TEXT_MENU
PRESS "pwr + vol_up" key to enter TEXT MENU and choose
PRESS "pwr + vol_down" key to enter FACTORY mode
#else
All hard key trigger.

You can add your new boot mode by following steps
1) Add new boot mode definition

For example:

```
/* boot type definitions */
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;
```

* For more detail information, please refer to
<lkdir>/platform/<chipset>/include/platform/boot_mode.h

2) Add your check function in the following: (* you can change the check order).

For example (**TEXT MENU**)

In “boot_mode_menu_select” (**boot_mode_menu.c**)

```
while(1)
{
   if(mtk_detect_key(MT65XX_VOLUMEUP_KEY))//VOL_UP
   {
        if(select == 0)
        {
            select = 1;

            video_set_cursor(video_get_rows()/2, 0);
            video_printf(title_msg);
            video_printf("1. [Recovery    Mode]          \n");
            video_printf("2. [Fastboot    Mode]       <<==\n");
            video_printf("3. [Normal      Boot]          \n");
        }
        else if(select == 1)
        {
            select = 2;

            video_set_cursor(video_get_rows()/2, 0);
            video_printf(title_msg);
            video_printf("1. [Recovery    Mode]          \n");
            video_printf("2. [Fastboot    Mode]          \n");
            video_printf("3. [Normal      Boot]       <<==\n");
        }
```

(Add new boot mode at TEXT-MENU)

```
if(select == 0)
{
    g_boot_mode = RECOVERY_BOOT;
}
else if(select == 1)
{
    g_boot_mode = FASTBOOT;
}
else if(select == 2)
{
    g_boot_mode = NORMAL_BOOT;
}
```

(Assign boot-mode by choice)

For example (**no TEXT MENU**)

In “boot_mode_select” (**boot_mode.c**)

We dived the boot_mode_select() into two sections.
Section1:

put conditions here to filer some cases that bypass key detection or give some limitation when do key dectection.

Section 2:

we put key dectection here to detect key which is pressed

The following example shows a feature implementation that we want to do key detections except factory key detection when user reboots with “power key + volumn down”.,

```
/*We put conditions here to filer some cases that can not do key detection*/

/*Check RTC to know if system want to reboot to Fastboot*/
  if(Check_RTC_PDN1_bit13())
  {
    dprintf(INFO,"[FASTBOOT] reboot to boot loader\n");
    g_boot_mode = FASTBOOT;
    Set_Clr_RTC_PDN1_bit13(false);
         return TRUE;
  }
   /*If forbidden mode is factory, cancel the factory key detection*/
    if(g_boot_arg->sec_limit.magic_num == 0x4C4C4C4C)
    {
        if(g_boot_arg->sec_limit.forbid_mode == F_FACTORY_MODE)
        {
            //Forbid to enter factory mode
            printf("%s Forbidden\n",MODULE_NAME);
            factory_forbidden=1;
        }
    }
```

```
    /*If boot reason is power key + volumn down, then
     disable factory mode dectection*/
    if(mtk_detect_pmic_just_rst())
    {
          factory_forbidden=1;
    }
    /*Check RTC to know if system want to reboot to Recovery*/
    if(Check_RTC_Recovery_Mode())
    {
          g_boot_mode = RECOVERY_BOOT;
          return TRUE;
    }
    /*If MISC Write has not completed  in recovery mode
     and interrupted by system reboot, go to recovery mode to
    finish remain tasks*/
    if(unshield_recovery_detection())
    {
        return TRUE;
    }
/*we put key dectection here to detect key which is pressed*/
while(get_timer(begin)<50){
    if(mtk_detect_key(MT_CAMERA_KEY))
    {
          dprintf(INFO,"[FASTBOOT]Key Detect\n");
          g_boot_mode = FASTBOOT;
          return TRUE;
    }
    if(!factory_forbidden){
                if(mtk_detect_key(MT65XX_FACTORY_KEY))
                {
                      printf("%s Detect key\n",MODULE_NAME);
                      printf("%s Enable factory mode\n",MODULE_NAME);
                      g_boot_mode = FACTORY_BOOT;
                      return TRUE;
                }
    }
       if(mtk_detect_key(MT65XX_RECOVERY_KEY))
       {
          printf("%s Detect cal key\n",MODULE_NAME);
          printf("%s Enable recovery mode\n",MODULE_NAME);
          g_boot_mode = RECOVERY_BOOT;
                      return TRUE;
       }
}
```

3) LK will create device tree and pass this boot mode number to Linux Kernel.

```
unsigned *target_atag_boot(unsigned *ptr)
{
    *ptr++ = tag_size(tag_boot);
    *ptr++ = ATAG_BOOT;
    *ptr++ = g_boot_mode;

    return ptr;
}
```

.

4.3 LK to Kernel Boot argument Customization

4.3.1 Kernel Tags – ATAG_MEM

Sometimes, you would probably change the DRAM type and change the memory banks. It is automatically generate by the EMI auto_gen so you don't need to modify anything in LK.

4.3.2 Kernel Tags – KERNEL / RAMDISK Location

In YuSu project, you can easily change the RAMDISK location of DRAM by simply change the following definition
(Please refer to device/mediatek/{project}/BoardConfig.mk)

BOARD_KERNEL_OFFSET = 0x00080000
BOARD_RAMDISK_OFFSET = 0x04000000
BOARD_TAGS_OFFSET = 0xE000000

4.3.3 Boot Parameters

Unlike kernel tags, you can also carry some information in boot parameters and pass to Linux Kernel by simply modify the string in COMMANDLINE_TO_KERNEL
(Please refer to "<lkdir>/platform/<chipset>/include/platform/mt_reg_base.h")

For example:

```
#define COMMANDLINE_TO_KERNEL "console=tty0 console=ttyMT3,921600n1
root=/dev/ram"
```

5 MMU mechanism

5.1 Default initial mappings areas

```
struct mmu_initial_mapping mmu_initial_mappings[] = {
    {
        .phys = (uint64_t)0,
        .virt = (uint32_t)0,
        .size = 0x40000000,
        .flags = MMU_MEMORY_TYPE_STRONGLY_ORDERED |
MMU_MEMORY_AP_P_RW_U_NA,
        .name = "mcusys"
    },
    {
        .phys = (uint64_t)MEMBASE,
        .virt = (uint32_t)MEMBASE,
        .size = ROUNDUP(MEMSIZE, SECTION_SIZE),
        .flags = MMU_MEMORY_TYPE_NORMAL_WRITE_BACK |
MMU_MEMORY_AP_P_RW_U_NA,
        .name = "ram"
    },
    {
        .phys = (uint64_t)CFG_BOOTIMG_LOAD_ADDR,
        .virt = (uint32_t)CFG_BOOTIMG_LOAD_ADDR,
        .size = 64*MB,
        .flags = MMU_MEMORY_TYPE_NORMAL_WRITE_BACK |
MMU_MEMORY_AP_P_RW_U_NA,
        .name = "bootimg"
    },
    {
        .phys = (uint64_t)SCRATCH_ADDR,
        .virt = (uint32_t)SCRATCH_ADDR,
        .size = SCRATCH_SIZE + 16*MB,
        .flags = MMU_MEMORY_TYPE_NORMAL_WRITE_BACK |
MMU_MEMORY_AP_P_RW_U_NA,
        .name = "download"
    },
    /* null entry to terminate the list */
    { 0 }
};
```

5.2 Run time register mmu mapping

Run time malloc memories also need to perform mmu mapping,otherwise cause data abort when access the malloc memory. User should RoundDown start address to align PAGE_SIZE, and RoundUp size to align PAGE_SIZE.

```
uint32_t start = ROUNDDOWN((uint32_t)buff_header, PAGE_SIZE);
uint32_t logsize = ROUNDUP(((uint32_t)buff_header - start + LOG_STORE_SIZE), PAGE_SIZE);
arch_mmu_map((uint64_t) start, start,
    MMU_MEMORY_TYPE_NORMAL_WRITE_BACK | MMU_MEMORY_AP_P_RW_U_NA, logsize);
```

6 Build

6.1 Source Code Structure

6.1.1 LK Folder Structure

the source of lk is in <lkdir>
AOSP: alps/vendor/mediatek/proprietary/bootable/bootloader/lk

[AOSP Style]

- app/
  - Applications in lk like fastboot, thread testing, string test. (thread base)
- alps/out
  - Collect complier output files (.o .lib .elf) and header dependency files (.d). Disassembly files.
- dev/
  - Common device interface like usb, frame buffer, keys and lcm.
  - Device customer folders.
- lib/
  - Include general memory, debug, and c libraries.
- make/
  - lk build scripts
- platform/
  - Include chip dependent drivers
  - MediaTek drivers are in "mediatek/platform/<$CHIP>/lk/"
- project/
  - Project specific build script. You can choose what apps you want to include.
- scripts/
  - ICE cmm files
- target/
  - Define some target based definition.
  - Target specific init flow.

6.1.2 LK Customization Source Tree

Folder : <lkdir>/target/<project>/

File	Description
cust_display.c	
cust_msdc.c	eMMC/SD card customization
init.c	1. Customize project specific boot flow. 2. Customize fastboot trigger event.

File	Description
power_off.c	Some customization setting for power off
cust_leds.c	Some customization setting for led
fastboot_oem_commands.c	Customize fastboot oem commands.
partition.c	Customize partition information
cust_leds.h	Some customization setting for led
cust_nand.h	Some customization setting for nand
mt_partition.h	Customize partition name and some structer used by partition module
board.h	Customize project specific lk definitions.
cust_display.h	Some customization setting for display.
cust_msdc.h	The customization definition header for MSDC
debugconfig.h	Define UART debug
cust_battery.h	Customize battery/charging parameter based on customer's battery.
cust_key.h	Customization for key setting
cust_usb.h	Some customization setting for USB

6.2 Build Command

6.2.1 Tool Chain of and LK

Current tool chain used is "arm-linux-androideabi- gcc"

6.2.2 LK Build Command

```
source build/envseup.sh
lunch <your project name>
make -j4 lk
```