在调试器下理解ARMv8

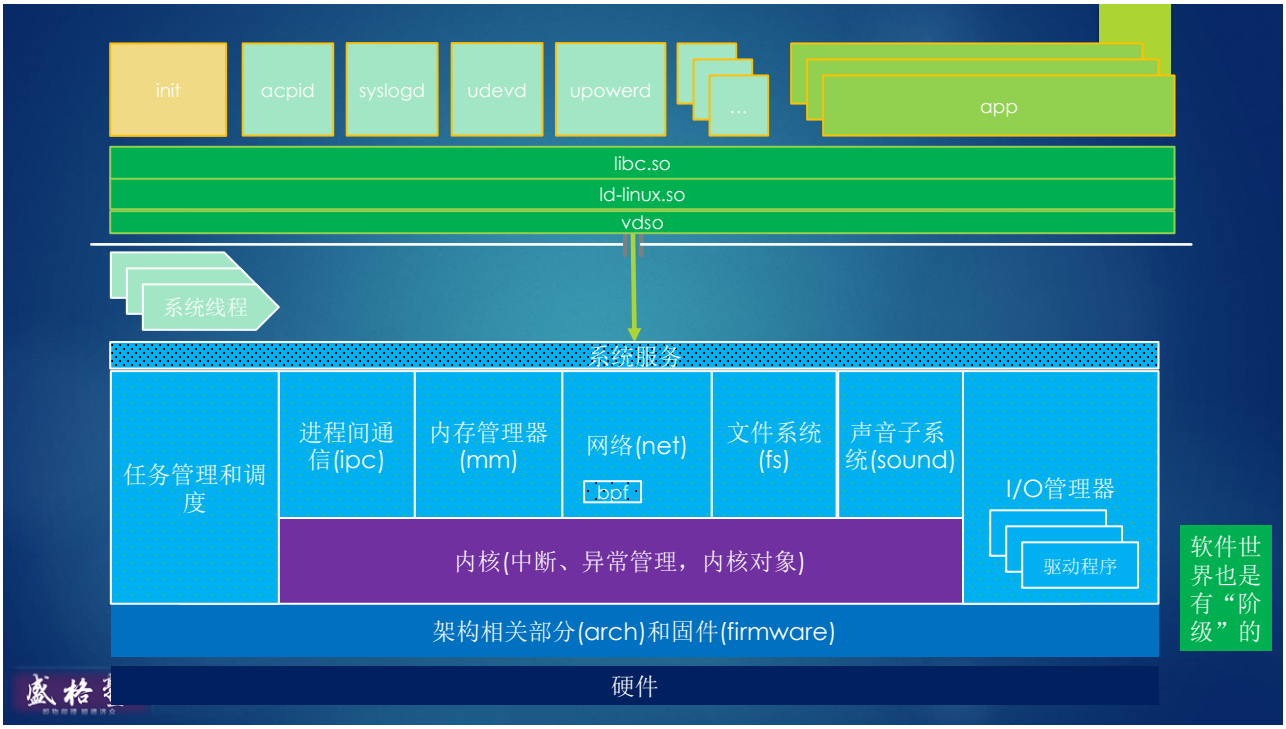
——特权级别和系统编程

RAYMOND ZHANG, JASON

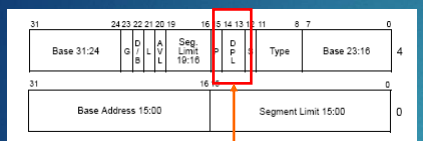
盛格塾

特权级别 → 内核模块 → 系统寄存器 → ATF → U-BOOT

盛格塾



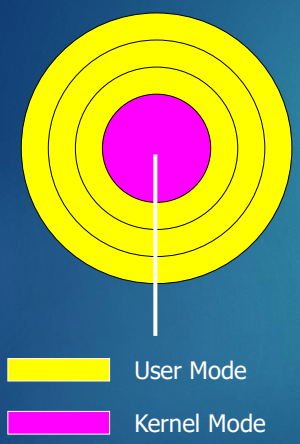
X86的描述符特权级别(DPL)

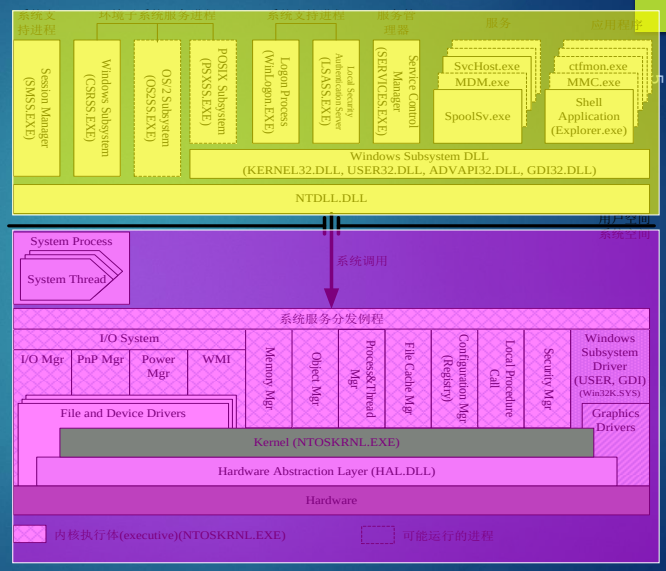


DPL – Descriptor Privilege Level

- 2个比特位，4种级别，通常所说的Ring 0 – 3， 0最高
- 正在执行的代码所在代码段的特权级别就是该代码的特权级别
- 当一段代码调用位于其它段的函数或访问其它段的数据时，CPU会检查发起访问者是否有足够的权限。如果没有通过检查，则产生异常

对应关系





页表中的特权级别

U/S – User/Supervisor

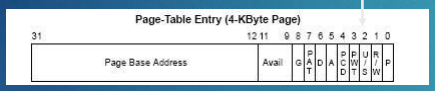


Table 4-20. Format of a Page-Table Entry that Maps a

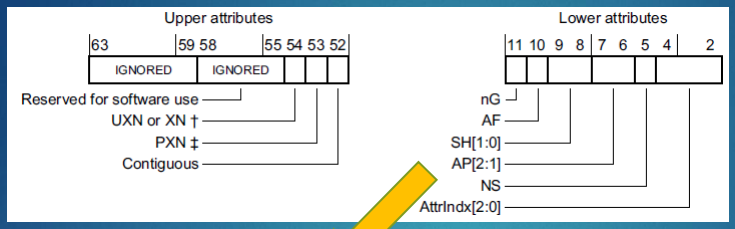
Bit Position(s)	Contents
0 (P)	Present; must be 1 to map a 4-KByte page
1 (R/W)	Read/write; if 0, writes may not be allowed to the 4-KByte page reference
2 (U/S)	User/supervisor; if 0, user-mode accesses are not allowed to the 4-KByte p 4.6)

- 现代操作系统种，段的粒度太大
- 页中描述特权，一个比特位，一般为bit 2，U/S，0代表高特权

内核页和用户页

```
0: kd> !pte fffff803`075c6220
                                           VA fffff803075c6220
PXE at FFFFF6FB7DBEDF80    PPE at FFFFF6FB7DBF0060    PDE at FFFFF6FB7E00C1D0    PTE at FFFFF6FC0183AE30
contains 00000000003D4063  contains 00000000003D5063  contains 00000002F4A009E3  contains 000000000000000

pfn 3d4      ---DA--KWEV  pfn 3d5      ---DA--KWEV  pfn 2f4a00   -GLDA--KWEV  LARGE PAGE pfn 2f4bc6

0: kd> !pte 7ffefdcf0000
                                           VA 00007ffefdcf0000
PXE at FFFFF6FB7DBED7F8    PPE at FFFFF6FB7DAFFFD8    PDE at FFFFF6FB5FFFBF70    PTE at FFFFF6BFFF7EE780
contains 2D1000011FCAB867  contains 018000011FD7B867  contains 26E000011FFA1867  contains BD70000106E5C02

pfn 11fcab   ---DA--UWEV  pfn 11fd7b   ---DA--UWEV  pfn 11ffa1   ---DA--UWEV  pfn 106e5c   ----A--U
```

ARMv8亦然



页属性分为高低两堆

AP[2:1]	Access from higher Exception level	Access from EL0
00	Read/write	None
01	Read/write	Read/write
10	Read-only	None
11	Read-only	Read-only

AP – Access Permission

The ARMv8-A architecture defines a set of Exception levels, **EL0 to EL3**, where:

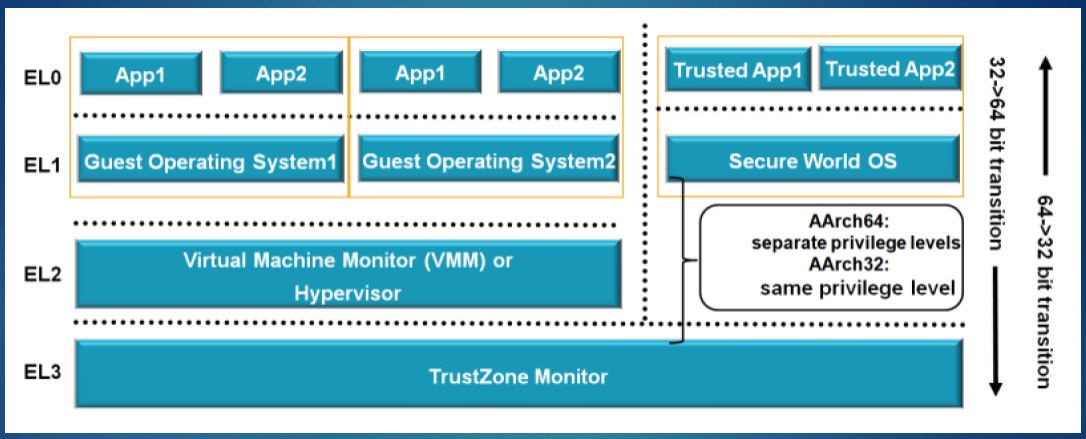
- If ELn is the Exception level, increased values of n indicate increased software execution **privilege**.
- Execution at EL0 is called unprivileged execution.

来自ARMv8 ARM D1: *The AArch64 System Level Programmers' Model*

使用模式

- EL0 Applications.
- EL1 OS kernel and associated functions that are typically described as privileged.
- EL2 Hypervisor.
- EL3 Secure monitor.

盛格塾



盛格塾

All implementations must include EL0 and EL1. EL2 and EL3 are optional.
A PE is not required to implement a contiguous set of Exception levels. For example, it is permissible for an implementation to include only EL0, EL1, and EL3.

盛格塾

Secure state

When in this state, the PE can access both the Secure physical address space and the Non-secure physical address space.

EL3仅存在于安全状态

Non-secure state

When in this state, the PE: Can access only the Non-secure physical address space. Cannot access the Secure system control resources

EL2仅存在于非安全状态

盛格塾

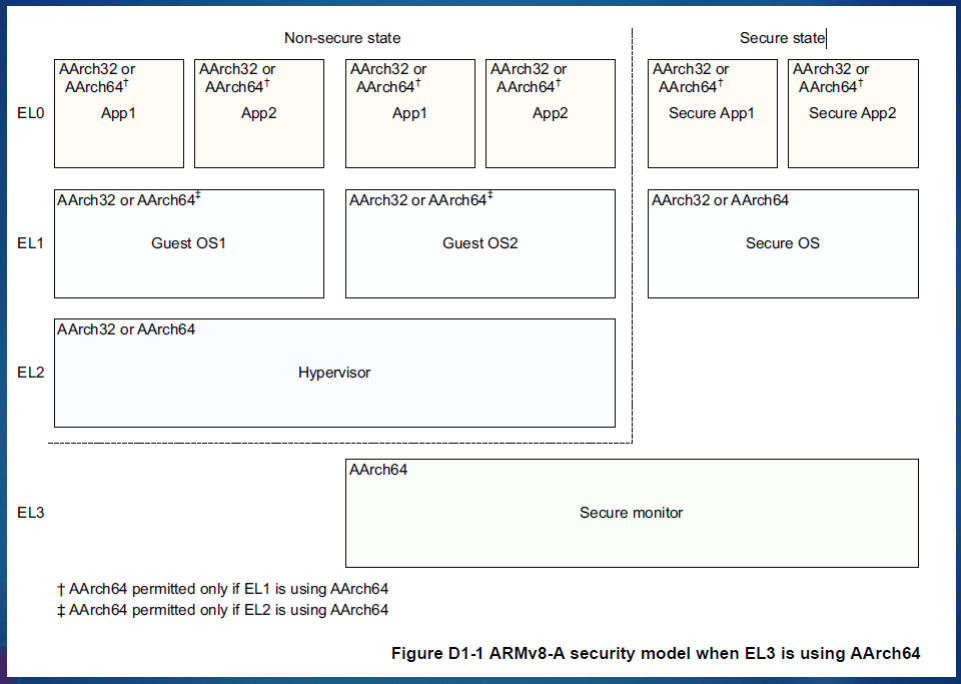
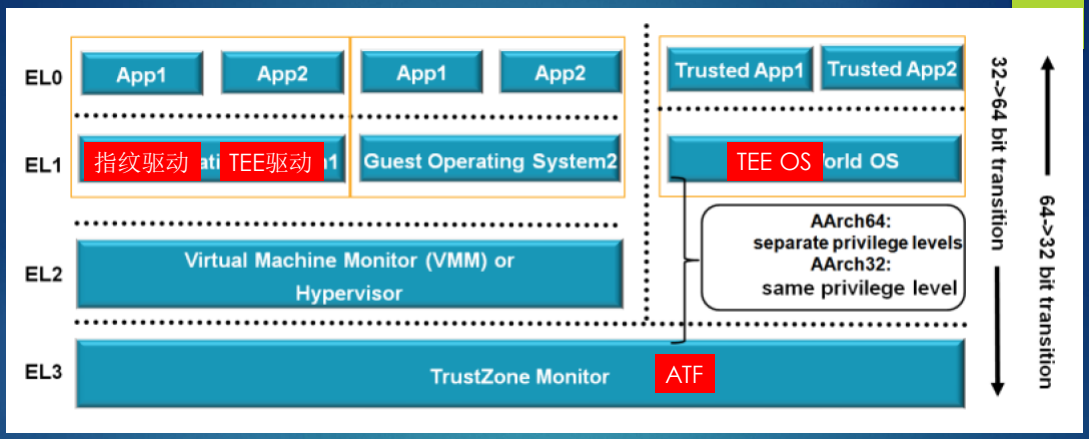
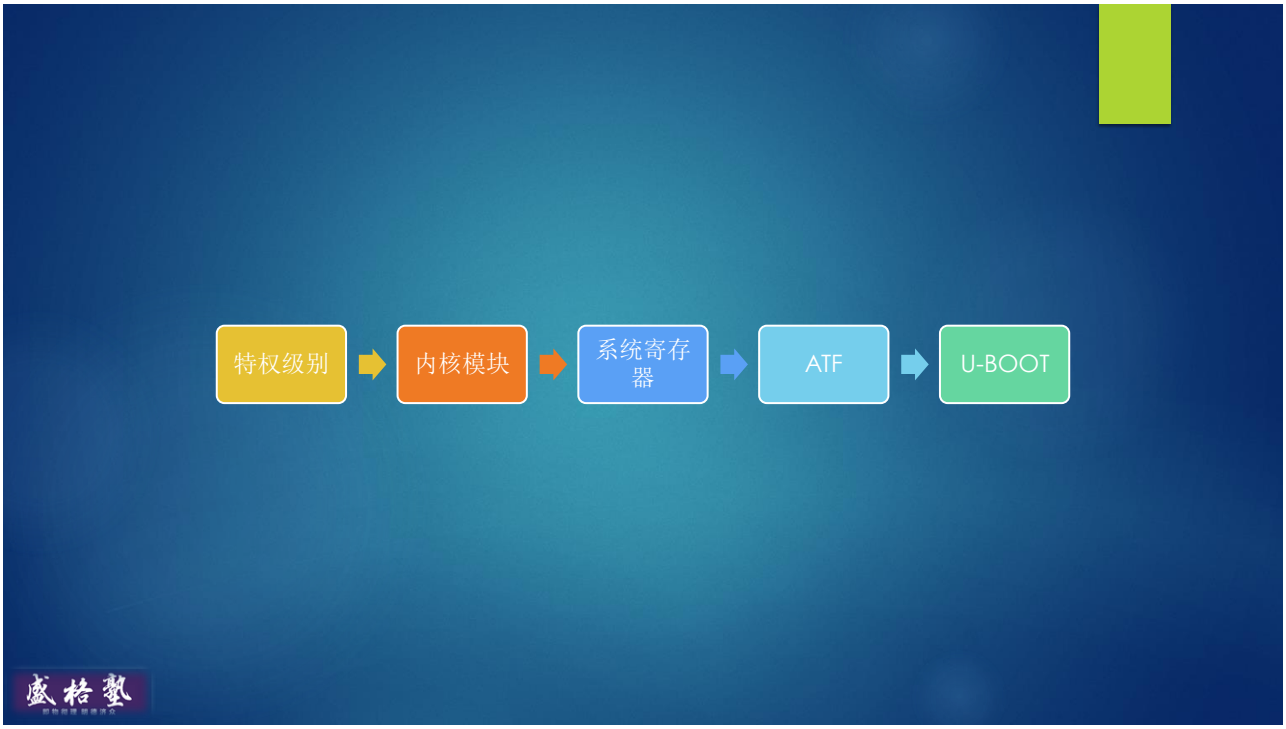
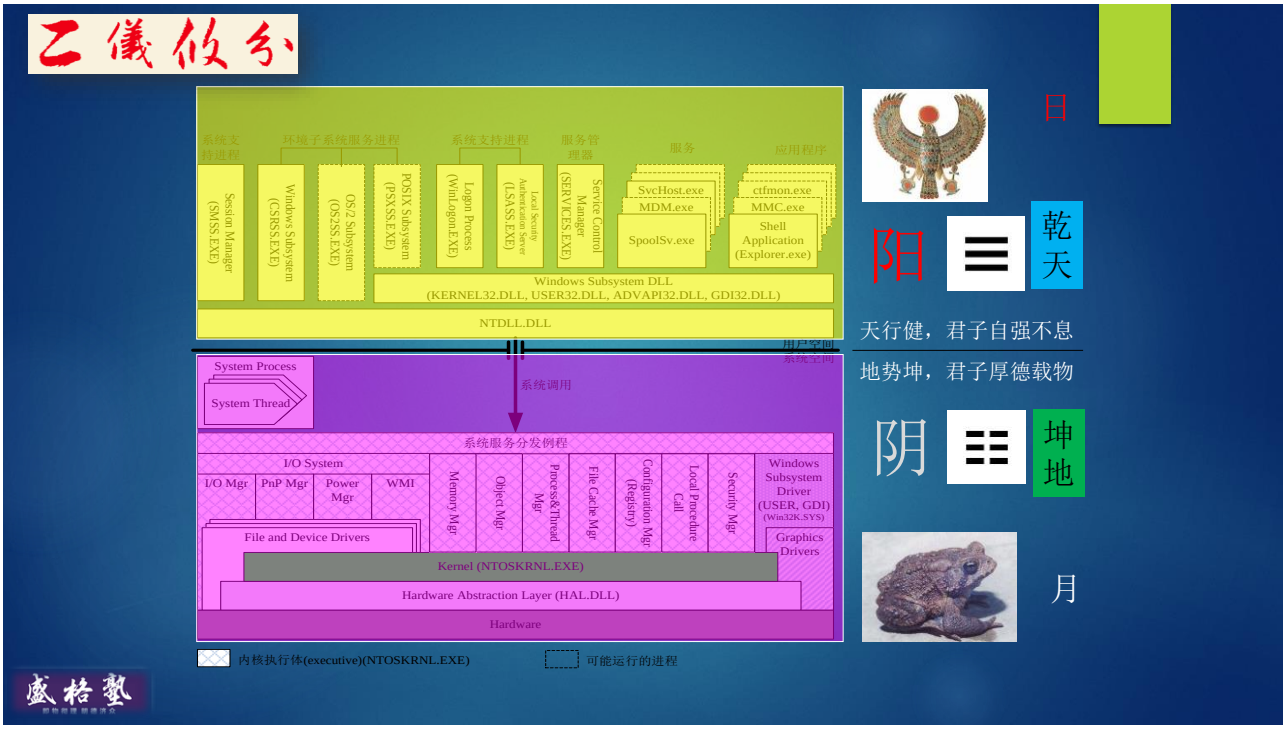


Figure D1-1 ARMv8-A security model when EL3 is using AArch64



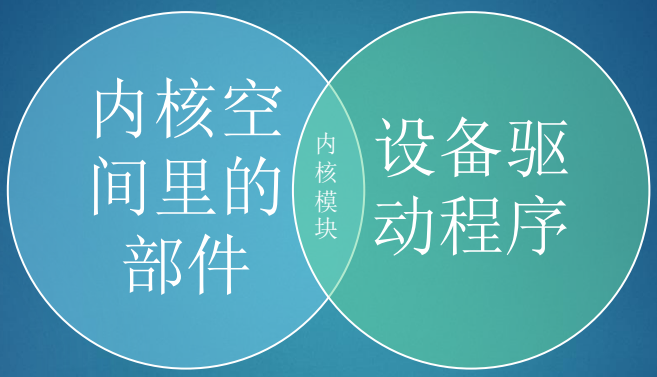
ATF - ARM Trusted Firmware
TEE – Trusted Execution Environment


特权级别
内核模块
系统寄存器
ATF
U-BOOT
盛格塾


二儀攸分
系统支持进程
环境子系统服务进程
系统支持进程
服务管理器
服务
应用程序
Session Manager (SMSS.EXE)
Windows Subsystem (CSRSS.EXE)
OS/2 Subsystem (OS2SS.EXE)
POSIX Subsystem (PSXSS.EXE)
Logon Process (WinLogon.EXE)
Local Security Authentication Server (LSASS.EXE)
Service Control Manager (SERVICES.EXE)
SvcHost.exe
MDM.exe
SpoolSv.exe
ctfmon.exe
MMC.exe
Shell Application (Explorer.exe)
Windows Subsystem DLL
(KERNEL32.DLL, USER32.DLL, ADVAPI32.DLL, GDI32.DLL)
NTDLL.DLL
用户空间
系统空间
System Process
System Thread
系统调用
系统服务分发例程
I/O System
I/O Mgr
PnP Mgr
Power Mgr
WMI
Memory Mgr
Object Mgr
Process&Thread Mgr
File Cache Mgr
Configuration Mgr (Registry)
Local Procedure Call
Security Mgr
Windows Subsystem Driver (USER, GDI) (Win32K.SYS)
File and Device Drivers
Graphics Drivers
Kernel (NTOSKRNL.EXE)
Hardware Abstraction Layer (HAL.DLL)
Hardware
内核执行体(executive)(NTOSKRNL.EXE)
可能运行的进程
日
阳
乾
天
天行健，君子自强不息
地势坤，君子厚德载物
阴
坤
地
月
盛格塾

```
#0  udp_sendmsg (iocb=0xdb805e28, sk=0xede0b400, msg=0xdb805e78, len=35)
    at net/ipv4/udp.c:827
#1  0xc15fa360 in inet_sendmsg (iocb=0xdb805e28, sock=<optimized out>,
    msg=<optimized out>, size=35) at net/ipv4/af_inet.c:770
#2  0xc158300b in __sock_sendmsg_nosec (size=35, msg=0xdb805e78,
    sock=0xcf40f600, iocb=0xdb805e28) at net/socket.c:631
#3  __sock_sendmsg (size=35, msg=0xdb805e78, sock=0xcf40f600, iocb=0xdb805e28)
    at net/socket.c:639
#4  __sock_sendmsg (size=35, msg=0xdb805e78, sock=0xcf40f600, iocb=0xdb805e28)
    at net/socket.c:3485
#5  sock_sendmsg (sock=0xcf40f600, msg=0xdb805e78, size=35) at net/socket.c:650
#6  0xc1583792 in SYSC_sendto (fd=<optimized out>, buff=0xa18fb930, len=35,
    flags=16448, addr=0x0, addr_len=0) at net/socket.c:1796
#7  0xc1583dad in SyS_sendto (fd=50, buff=-1584416464, len=35, flags=16384,
    addr=0, addr_len=0) at net/socket.c:1763
#8  0xc1583deb in SYSC_send (flags=<optimized out>, len=<optimized out>,
    buff=<optimized out>, fd=<optimized out>) at net/socket.c:1811
#9  SyS_send (fd=50, buff=-1584416464, len=35, flags=16384)
    at net/socket.c:1809
#10 0xc1584864 in SYSC_socketcall (args=0xa18fb7f0, call=9)
    at net/socket.c:2516
#11 SyS_socketcall (call=9, args=-1584416784) at net/socket.c:2460
#12 <signal handler called>
#13 0xb77b4424 in ?? ()
#14 0x00000002 in ?? ()
#15 0x00000001 in ?? ()
#16 0x0100007f in ?? ()
#17 0x00000000 in ?? ()
```

界定



Modules, or loadable modules, 在Linux系统中有特定含义，特制运行在内核空间中的可动态加载模块

很多内核模块的功能是设备驱动程序

有些设备驱动程序编译进内核，也有些运行在用户态

刘姥姥

main.c ✖ Makefile ✖

```c
/*
 Example of a minimal character device driver
*/
#include <linux/module.h>

static int __init llaolao_init(void)
{
    int n = 0x1937;
    printk(KERN_INFO "Hi, I am llaolao at address 0x%p stack 0x%p.\n",
        llaolao_init, &n);

    return 0;
}

static void __exit llaolao_exit(void)
{
    printk("Exiting from 0x%p... Bye, GEDU friends\n", llaolao_exit);
}

module_init(llaolao_init);
module_exit(llaolao_exit);

MODULE_AUTHOR("GEDU lab");
MODULE_DESCRIPTION("LKM example - llaolao");
MODULE_LICENSE("GPL");
```

- Hello world
- 一进LINUX大观园
- A minimal LKM
- A bit more than that ☺

<linux/module.h>

- All Linux kernel modules must include <linux/module.h>
- Module.h defines a number of macros that are used by the kernel build system to add necessary metadata to compiled module files
- Defines the central 'struct module'

```c
struct module
{
    enum module_state state;

    /* Member of list of modules */
    struct list_head list;

    /* Unique handle for this module */
    char name[MODULE_NAME_LEN];

    /* Sysfs stuff. */
    struct module_kobject mkobj;
    struct module_attribute *modinfo_attrs;
    const char *version;
    const char *srcversion;
    struct kobject *holders_dir;

    /* Exported symbols */
    const struct kernel_symbol *syms;
    const unsigned long *crcs;
    unsigned int num_syms;

    /* Kernel parameters. */
    struct kernel_param *kp;
    unsigned int num_kp;

    /* GPL-only exported symbols. */
    unsigned int num_gpl_syms;
    const struct kernel_symbol *gpl_syms;
    const unsigned long *gpl_crcs;
```

- 内核模块的“户籍档案”
- 颇为庞大的结构体
- 以链表形式相互关联，模块列表
- 只是LKM，不包括内核本身

盛格塾

Makefile

main.c | Makefile

```makefile
WFLAGS := -Wall -Wstrict-prototypes -Wno-trigraphs

LDFLAGS = -Map /var/tmp/lllmap.txt

EXTRA_CFLAGS := $(WFLAGS)

# EXTRA_CFLAGS += -g -Wa,-adhln=$(<:.c=.lst)

EXTRA_CFLAGS += -D_DEBUG -g3  -fno-stack-protector

MODULE = llaolao

obj-m := $(MODULE).o

all:
    make -C /lib/modules/$(shell uname -r)/build SUBDIRS=$(shell pwd) modules

$(MODULE)-objs := main.o

clean:
    rm -rf *.o *~ .*.cmd *.ko *.mod.c *.order *.symvers .tmp_versions built-in.o
```

新的内核使用 M=

必须 TAB 不是空格

- 姑且用之，以后慢慢理解

盛格塾

$ make

```
ge@gewubox:~/work/llaolao$ make
make -C /lib/modules/3.12.2/build SUBDIRS=/home/ge/work/llaolao modules
make[1]: Entering directory `/home/ge/work/linux-3.12.2'
  CC [M]  /home/ge/work/llaolao/main.o
  LD [M]  /home/ge/work/llaolao/llaolao.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/ge/work/llaolao/llaolao.mod.o
  LD [M]  /home/ge/work/llaolao/llaolao.ko
make[1]: Leaving directory `/home/ge/work/linux-3.12.2'
```

- $ make –C /lib/modules/x.y.z/build SUBDIRS=<> modules

加载和运行

```
ge@gewubox:~/work/llaolao$ sudo insmod llaolao.ko
```

```
insmod(8)                                                    insmod(8)

NAME
       insmod – simple program to insert a module into the Linux Kernel

SYNOPSIS
       insmod [filename]  [module options ...]

DESCRIPTION
       insmod  is  a  trivial program to insert a module into the kernel: if
       the filename is a hyphen, the module is taken  from  standard  input.
       Most users will want to use modprobe(8) instead, which is more clever
       and can handle module dependencies.

       Only the most general of error messages are reported: as the work  of
       trying  to  link  the module is now done inside the kernel, the dmesg
       usually gives more information about errors.
```

dmesg

[1016.454855] Hi, I am llaolao at address: **ffffff8000bd0000**

symbol: 0xllaolao_init+0x0/0x1000 [llaolao]

stack: **0xffffffc0f40d3c4c**

first 16 bytes: fd:7b:ba:a9:e0:26:83:52:04:02:80:52:fd:03:00:91

盛格塾

lsmod

- 列出所有加载模块
- "Module" denotes the name of the module.
- "Size" denotes the size of the module (not memory used).
- "Used by" denotes each module's use count and a list of referring modules. The "Used by" list is sometimes incomplete.
- If the module controls its own unloading via a can_unload routine then the use count displayed by lsmod is always -1, irrespective of the real use count.

```
Module                  Size  Used by
vboxsf                 38472  1
rfcomm                 57808  0
bnep                   18960  2
bluetooth             327609  10 rfcomm,bnep
parport_pc             31968  0
ppdev                  17363  0
snd_intel8x0           37241  2
snd_ac97_codec        109846  1 snd_intel8x0
ac97_bus               12642  1 snd_ac97_codec
snd_pcm                85369  2 snd_intel8x0,snd_ac97_codec
snd_seq_midi           13132  0
snd_rawmidi            25155  1 snd_seq_midi
crc32_pclmul           12967  0
snd_seq_midi_event     14475  1 snd_seq_midi
snd_seq                55403  2 snd_seq_midi,snd_seq_midi_event
vboxguest             238260  6 vboxsf
aesni_intel            18156  0
snd_timer              28639  2 snd_pcm,snd_seq
ablk_helper            13357  1 aesni_intel
cryptd                 15579  1 ablk_helper
snd_seq_device         14137  3 snd_seq_midi,snd_rawmidi,snd_seq
joydev                 17101  0
lrw                    13098  1 aesni_intel
snd                    60898  12 snd_intel8x0,snd_ac97_codec,snd_pcm,snd_seq_midi,snd_rawmidi,snd_seq,snd_timer
psmouse                90880  0
aes_i586               16995  1 aesni_intel
xts                    12749  1 aesni_intel
gf128mul               14503  2 lrw,xts
soundcore              12600  1 snd
microcode              18928  0
i2c_piix4              17723  0
video                  18742  0
mac_hid                13037  0
snd_page_alloc         14230  2 snd_intel8x0,snd_pcm
serio_raw              13230  0
lp                     13299  0
parport                40803  3 parport_pc,ppdev,lp
hid_generic            12492  0
usbhid                 47035  0
hid                    87353  2 hid_generic,usbhid
ahci                   25579  2
libahci                26541  1 ahci
e1000                 128446  0
```

盛格塾

卸载模块

```
$ sudo rmmod llaolao
```

```
[ 2911.263615] Exiting from 0xf0984000... Bye, GEDU friends
```

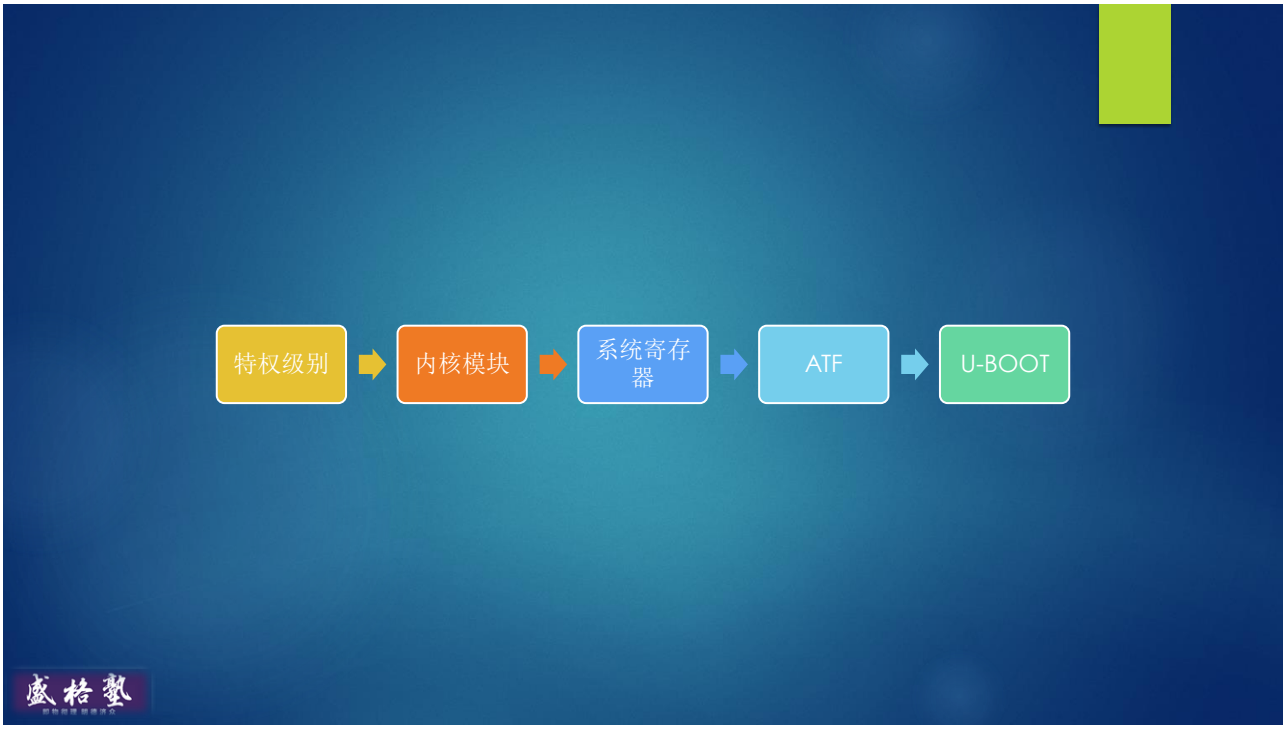
- module_exit函数会被调用

sections

```
ge@gewubox:~/work/llaolao$ sudo cat /sys/module/llaolao/sections/.
./                          .note.gnu.build-id
../                         .rodata.str1.4
.exit.text                  .strtab
.gnu.linkonce.this_module   .symtab
.init.text
ge@gewubox:~/work/llaolao$ sudo cat /sys/module/llaolao/sections/.strtab
0xf0836250
ge@gewubox:~/work/llaolao$ sudo cat /sys/module/llaolao/sections/.symtab
0xf0836000
ge@gewubox:~/work/llaolao$ sudo cat /sys/module/llaolao/sections/.init.text
0xf0835000
ge@gewubox:~/work/llaolao$ sudo cat /sys/module/llaolao/sections/.gnu.linkonce
.this_module
0xf0981000
```

```
geduer@gdk8:~/llaolao$ sudo cat /sys/module/llaolao/sections/.init.text
0xffffff8000bd0000
```

- 每个节在内存中的起始地址（线性地址）



系统寄存器——给系统软件使用的寄存器

以操作系统内核为核心的软件，包括驱动程序和系统固件

一般形式

<register_name>_ELx, where x is 0, 1, 2, or 3

- 后缀_ELx用来指定可以访问这个寄存器的最低特权模式

MIDR_EL1

Bits Name Function

[31:24] Implementer Indicates the implementer code. This value is:

`0x41` ARM Limited.

[23:20] Variant Indicates the variant number of the processor. This is the major revision number *n* in the r*n* part of the r*n*p*n* description of the product revision status. This value is:

`0` Major revision number.

[19:16] Architecture Indicates the architecture code. This value is:

`0xF` Defined by CPUID scheme.

[15:4] Primary part number Indicates the primary part number. This value is:

`0xD08` Cortex-A72 processor.

[3:0] Revision Indicates the minor revision number of the processor. This is the minor revision number *n* in the p*n* part of the r*n*p*n* description of the product revision status. This value is:

`3` Minor revision number.

0.000000] Boot CPU: AArch64 Processor [410fd034]

ARM

0.000000] Boot CPU: AArch64 Processor [41 0 f d03 4]

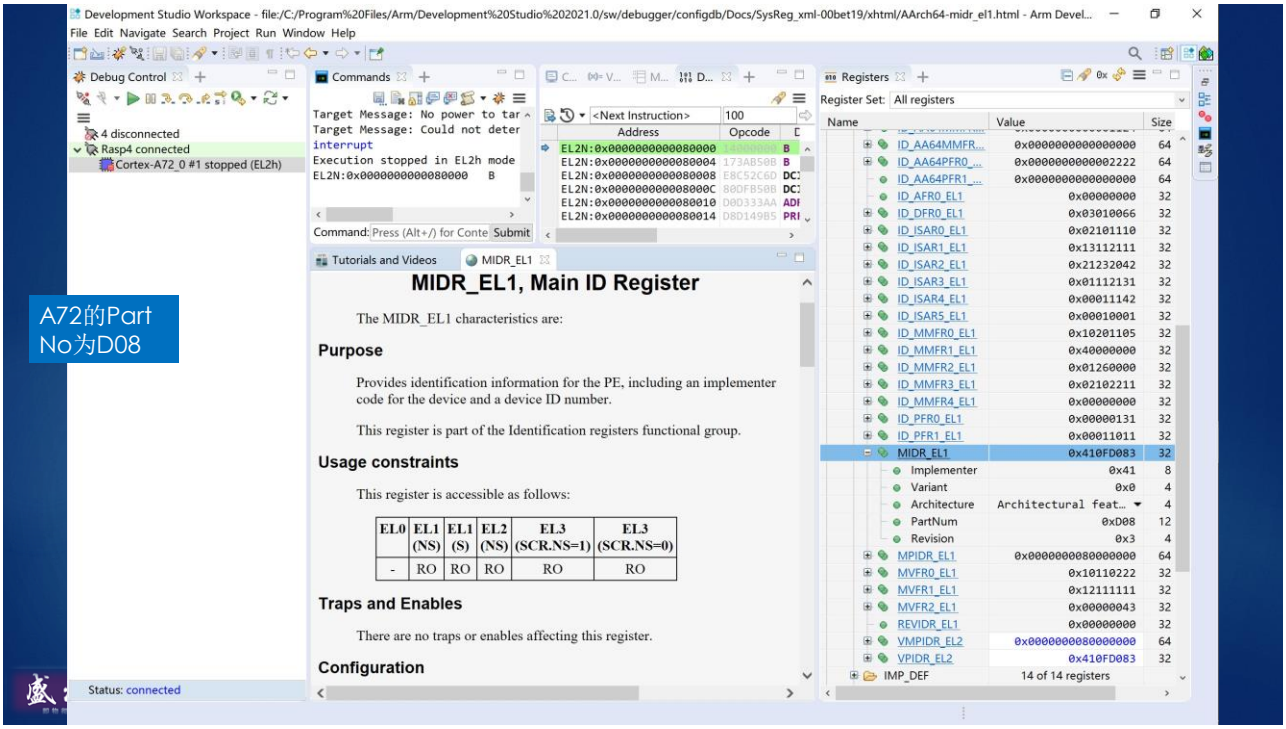
大版本号

A53

小版本号

架构代号

```
geduer@gdk8:~/llaolao$ dmesg
[    0.000000] Booting Linux on physical CPU 0x0
[    0.000000] Initializing cgroup subsys cpuset
[    0.000000] Initializing cgroup subsys cpu
[    0.000000] Initializing cgroup subsys cpuacct
[    0.000000] Linux version 4.4.179-yanzi (yanzi@yanzi-ws)
[    0.000000] Boot CPU: AArch64 Processor [410fd034]
```



A72的Part No为D08

三种访问方式

To access the MIDR_EL1 in AArch64 state, read the register with:

- MRS <Xt>, MIDR_EL1; Read Main ID Register

To access the MIDR in AArch32 state, read the CP15 register with:

- MRC p15, 0, <Rt>, c0, c0, 0; Read Main ID Register

The MIDR can be accessed through the memory-mapped interface and the external debug interface,

- offset 0xD00.

盛格塾

/sysreg.h

```
#define sys_reg(op0, op1, crn, crm, op2) \
        (((op0) << Op0_shift) | ((op1) << Op1_shift) | \
         ((crn) << CRn_shift) | ((crm) << CRm_shift) | \
         ((op2) << Op2_shift))

#define sys_insn sys_reg
```

arch > arm64 > include > asm > C sysreg.h

```c
/*
 * System registers, organised loosely by encoding but grouped together
 * where the architected name contains an index. e.g. ID_MMFR<n>_EL1.
 */
#define SYS_OSDTRRX_EL1          sys_reg(2, 0, 0, 0, 2)
#define SYS_MDCCINT_EL1          sys_reg(2, 0, 0, 2, 0)
#define SYS_MDSCR_EL1            sys_reg(2, 0, 0, 2, 2)
#define SYS_OSDTRTX_EL1          sys_reg(2, 0, 0, 3, 2)
#define SYS_OSECCR_EL1           sys_reg(2, 0, 0, 6, 2)
#define SYS_DBGBVRn_EL1(n)       sys_reg(2, 0, 0, n, 4)
#define SYS_DBGBCRn_EL1(n)       sys_reg(2, 0, 0, n, 5)
#define SYS_DBGWVRn_EL1(n)       sys_reg(2, 0, 0, n, 6)
#define SYS_DBGWCRn_EL1(n)       sys_reg(2, 0, 0, n, 7)
#define SYS_MDRAR_EL1            sys_reg(2, 0, 1, 0, 0)
#define SYS_OSLAR_EL1            sys_reg(2, 0, 1, 0, 4)
#define SYS_OSLSR_EL1            sys_reg(2, 0, 1, 1, 4)
#define SYS_OSDLR_EL1            sys_reg(2, 0, 1, 3, 4)
#define SYS_DBGPRCR_EL1          sys_reg(2, 0, 1, 4, 4)
#define SYS_DBGCLAIMSET_EL1      sys_reg(2, 0, 7, 8, 6)
#define SYS_DBGCLAIMCLR_EL1      sys_reg(2, 0, 7, 9, 6)
#define SYS_DBGAUTHSTATUS_EL1        sys_reg(2, 0, 7, 14, 6)
#define SYS_MDCCSR_EL0           sys_reg(2, 3, 0, 1, 0)
#define SYS_DBGDTR_EL0           sys_reg(2, 3, 0, 4, 0)
#define SYS_DBGDTRRX_EL0         sys_reg(2, 3, 0, 5, 0)
#define SYS_DBGDTRTX_EL0         sys_reg(2, 3, 0, 5, 0)
#define SYS_DBGVCR32_EL2         sys_reg(2, 4, 0, 7, 0)
```

盛格塾

```
#include <asm/hwcap.h>
#include <linux/module.h>

#define rd_arm_reg(id) ({                                   \
        unsigned long __val;                                \
        asm("mrs %0, "#id : "=r" (__val));                  \
        printk("%-20s: 0x%016lx\n", #id, __val);            \
    })
void ge_arm_sysregs(void)
{
    rd_arm_reg(ID_AA64ISAR0_EL1);
    rd_arm_reg(ID_AA64ISAR1_EL1);
    rd_arm_reg(ID_AA64MMFR0_EL1);
    rd_arm_reg(ID_AA64MMFR1_EL1);
    rd_arm_reg(ID_AA64PFR0_EL1);
    rd_arm_reg(ID_AA64PFR1_EL1);
    rd_arm_reg(ID_AA64DFR0_EL1);
    rd_arm_reg(ID_AA64DFR1_EL1);

    rd_arm_reg(MIDR_EL1);
    rd_arm_reg(MPIDR_EL1);
    rd_arm_reg(REVIDR_EL1);
```

```
[14115.743840] proc_lll_write called legnth 0x7, 0000007ff8385a98
[14115.743858] ID_AA64ISAR0_EL1     : 0x0000000000011120
[14115.743864] ID_AA64ISAR1_EL1     : 0x0000000000000000
[14115.743870] ID_AA64MMFR0_EL1     : 0x0000000000001122
[14115.743875] ID_AA64MMFR1_EL1     : 0x0000000000000000
[14115.743880] ID_AA64PFR0_EL1      : 0x0000000000002222
[14115.743886] ID_AA64PFR1_EL1      : 0x0000000000000000
[14115.743891] ID_AA64DFR0_EL1      : 0x0000000010305106
[14115.743896] ID_AA64DFR1_EL1      : 0x0000000000000000
[14115.743901] MIDR_EL1             : 0x00000000410fd034
[14115.743907] MPIDR_EL1            : 0x0000000080000000
[14115.743912] REVIDR_EL1           : 0x0000000000000180
[14115.743918] ID_MMFR0_EL1         : 0x0000000010201105
```

PMCCNTR_EL0

Holds the value of the processor Cycle Counter, CCNT, that counts processor clock cycles

The PMCR.{LC, D} bits configure whether PMCCNTR increments every clock cycle, or once every 64 clock cycles.

Thread ID寄存器在WOA中的用法

Register	Role
TPIDR_EL0	Reserved.
TPIDRRO_EL0	Contains CPU number for current processor.
TPIDR_EL1	Points to KPCR (处理器控制区) structure for current processor.

* https://docs.microsoft.com/en-us/cpp/build/arm64-windows-abi-conventions?view=msvc-160

盛格塾

Chapter D7
AArch64 System Register Descriptions

This chapter defines the AArch64 System registers. It contains the following sections:

- *About the AArch64 System registers* on page D7-1888.
- *General system control registers* on page D7-1895.
- *Debug registers* on page D7-2147.
- *Performance Monitors registers* on page D7-2215.
- *Generic Timer registers* on page D7-2255.

盛格塾

特权级别 → 内核模块 → 系统寄存器 → ATF → U-BOOT

盛格塾

Embedded boot sequence

Cortex-A72/A53s

CPU-ROM

- CPU0 runs from CPU-ROM. CPU0 simply jump to eMMC offset 0 to execute ATF code.
- Other CPUs run from CPU-ROM, and WFE in CPU-ROM.

ATF

- CPU0 setups an executing environment for utilizing secure boot, ARM Trustzone, and the implementation of secure world software within the Armv8 architecture. CPU0 also performs some low level initialization on DDR, GIC.

u-boot

- CPU0 initializes some devices and prepare environment to load and boot Linux kernel. Finally it loads Linux kernel image to memory and jump to the entry point of Linux kernel.

Linux Kernel

- Linux kernel initializes CPUs and other devices, and setups all OS services such as scheduler, MMU, process, interrupt, device driver, file systems, TCP/IP. Finally, it mounts a specified file system as root file system and run the first application "/sbin/init".

Root File System

- Root file system provides common used applications and libraries. "/sbin/init" is the first running application. Other applications are executed according to the /etc/rc.S. Finally, Linux shell appears.

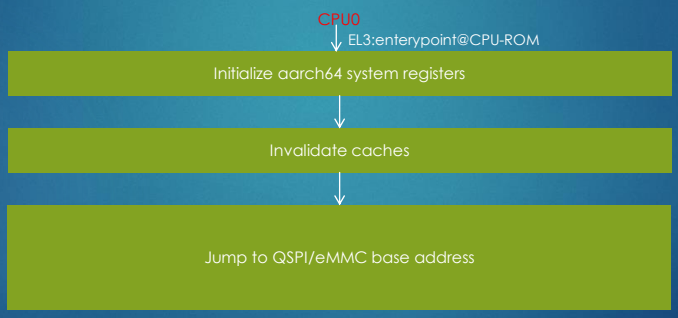
盛格塾

Aarch64 boot sequence

- CPU-ROM boot code is in ROM and could not be changed. The data region is in RAM and writeable.
- Other CPU cores boot from ROM, and run WFE (Wait For Event) instruction to sleep
- In later ATF/U-boot stages, CPU0 writes run_address at penholder addresses and wakes up CPU cores, then other CPU cores will continue to execute from boot_address

主CPU启动

CPU-ROM boot code is in ROM and could not be changed. The source code is not available to customers
The boot sequence is simple. Finally, CPU0 XIP to execute ATF BL1 code in QSPI flash/eMMC.



ATF

- ARM Trusted Firmware

SWS (F:) > bench > Jetson > Linux_for_Tegra > source > public > arm-trusted-firmware

名称	修改日期	类型	大小
bl1	2021-07-17 13:47	文件夹	
bl2	2021-07-17 13:47	文件夹	
bl2u	2021-07-17 13:47	文件夹	
bl31	2021-07-17 13:47	文件夹	
bl32	2021-07-17 13:47	文件夹	
common	2021-07-17 13:47	文件夹	
docs	2021-07-17 13:47	文件夹	
drivers	2021-07-17 13:46	文件夹	
fdts	2021-07-17 13:47	文件夹	
include	2021-07-17 13:47	文件夹	
lib	2021-07-17 13:47	文件夹	
make_helpers	2021-07-17 13:47	文件夹	
plat	2021-07-17 13:47	文件夹	
services	2021-07-17 13:47	文件夹	
tools	2021-07-17 13:47	文件夹	
.checkpatch.conf	2021-07-09 23:49	CONF 文件	4 KB
acknowledgements.md	2021-07-09 23:49	MD 文件	1 KB
contributing.md	2021-07-09 23:49	MD 文件	7 KB
dco.txt	2021-07-09 23:49	文本文档	2 KB
license.md	2021-07-09 23:49	MD 文件	2 KB
maintainers.md	2021-07-09 23:49	MD 文件	3 KB
Makefile	2021-07-09 23:49	文件	25 KB
readme.md	2021-07-09 23:49	MD 文件	9 KB

L4T R32.5.2

NVIDIA L4T 32.5.2 is identical to L4T 32.5.1 but contains additional security fixes. It supports all Jetson modules: Jetson AGX Xavier series, Jetson Xavier NX, Jetson TX2 series, Jetson TX1, and Jetson Nano. All Jetson developer kits are also supported.

For more details on security fixes included, please refer to NVIDIA security bulletin.

	Jetson AGX Xavier, Xavier NX and TX2	Jetson Nano, Nano 2GB and TX1
DRIVERS	L4T Driver Package (BSP)	L4T Driver Package (BSP)
	Sample Root Filesystem	Sample Root Filesystem
SOURCES	L4T Driver Package (BSP) Sources	L4T Driver Package (BSP) Sources
	Cboot Sources T186 Cboot Sources T194	
	Sample Root Filesystem Sources	
DOCS	Release Notes	

https://developer.nvidia.com/embedded/linux-tegra-r3251

安全启动流程

49

来自Nvidia Jetson文档

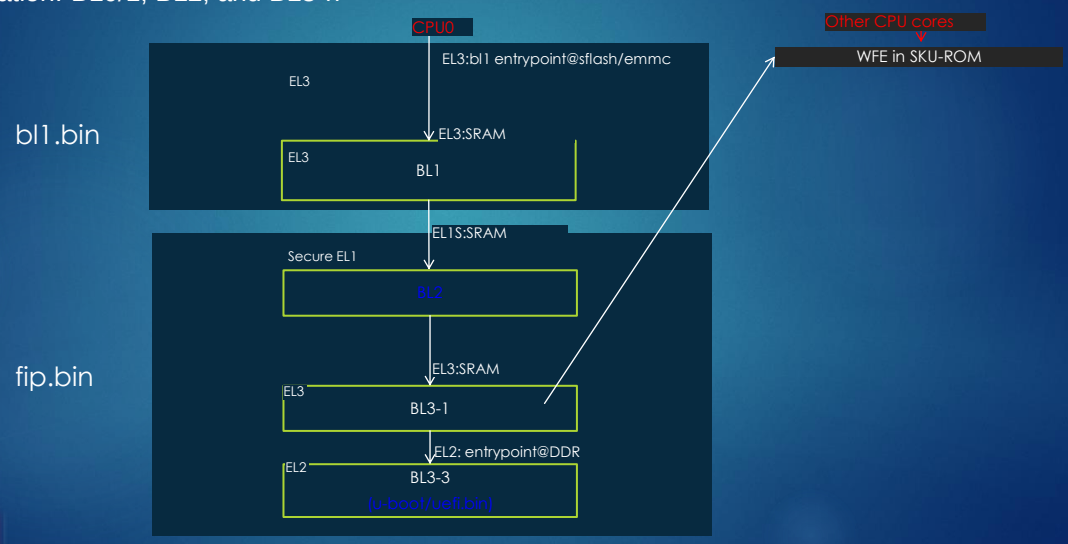
Aarch64 ATF

- ATF (ARM Trusted Firmware) is an architecture and software package developed by ARM and the open source community with the intent of providing a consistent execution framework for utilizing secure boot, ARM Trustzone, and the implementation of secure world software within the Armv8 architecture
- Let us take ARM Trusted Firmware v1.1 as reference
  - https://github.com/ARM-software/arm-trusted-firmware/releases
- Though ATF is mainly designed for secure purpose, non-secure boot also needs this stage
- ATF boot process provides several sub-stages of firmware initialization: BL1, BL2, BL3-1, BL3-2 and BL3-3
- After build, bl1.bin and fip.bin are generated. bl0.bin includes bl0.bin and bl1.bin

  fip.bin includes bl2.bin, bl31.bin, bl32.bin bl33.bin (u-boot.bin)

ATF Boot Sequence Overview

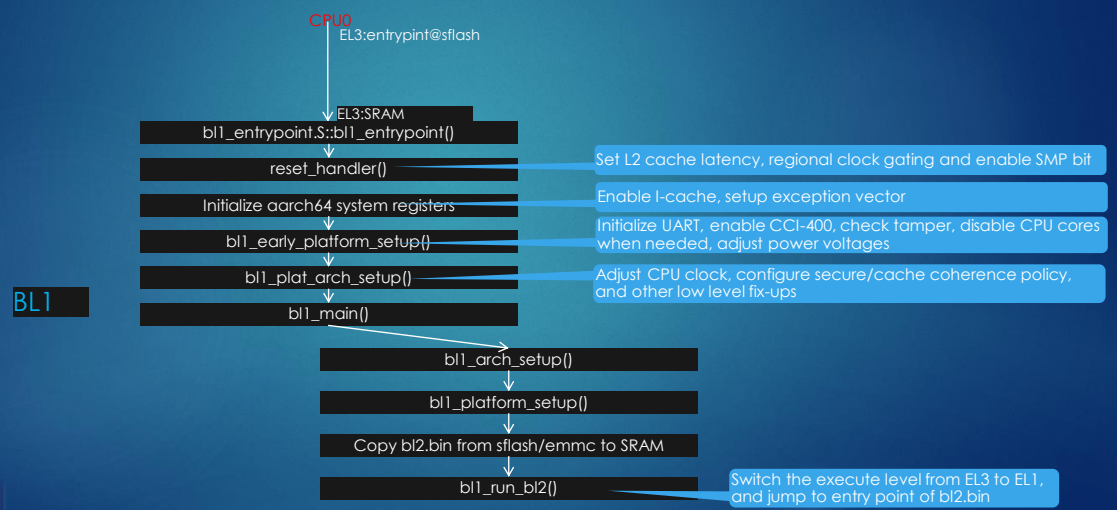
The boot process for the ARM Trusted Firmware (ATF) provides several sub-stages of firmware initialization: BL0/1, BL2, and BL3-n




```
Bootrom start
  Boot Media: eMMC
  Decrypt auxiliary code ...OK

  lsadc voltage min: 000000FE, max: 000000FF, aver: 000000FE, index: 00000000

  Entry boot auxiliary code

  Auxiliary code - v1.00
  DDR code - V1.1.2 20160205
  Build: Mar 24 2016 - 17:09:44
  Reg Version:  v134
  Reg Time:     2016/03/18 09:44:55
  Reg Name:     hi3798cv2dmb_hi3798cv200_ddr3_2gbyte_8bitx4_4layers.reg

  Boot auxiliary code success
  Bootrom success

  LOADER:  Switched to aarch64 mode
  LOADER:  Entering ARM TRUSTED FIRMWARE
  LOADER:  CPU0 executes at 0x000ce000
```

Bl0 and Bl1 boot sequence

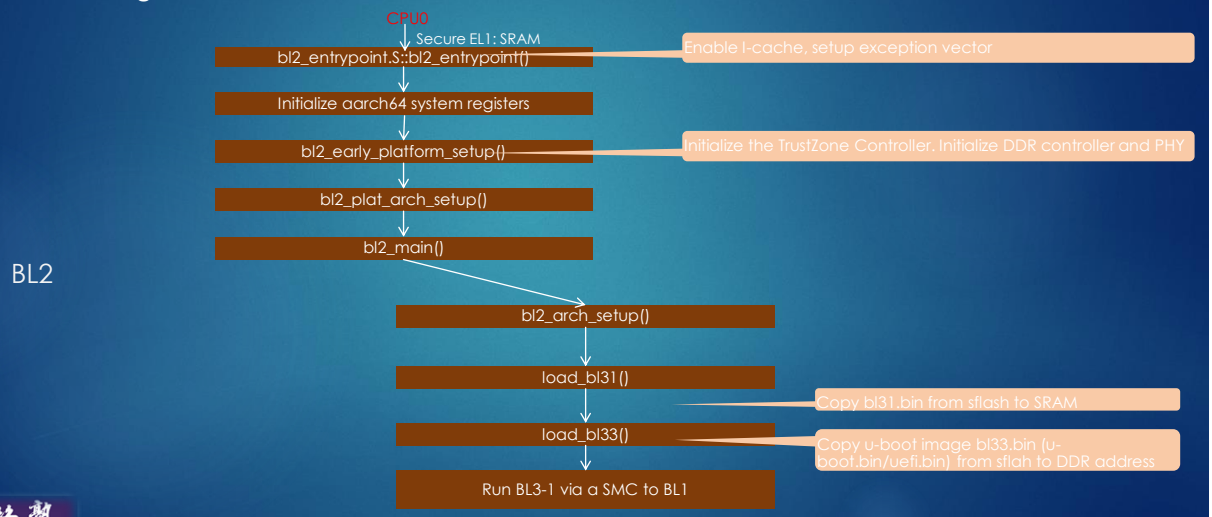
CPU0 begins to execute at flash offset 0. BL1 performs some low level configure and fix-up




```
INFO:    BL1: 0xe1000 - 0xe7000 [size = 24576]
NOTICE:  Booting Trusted Firmware
NOTICE:  BL1: v1.3(debug):v1.3-372-g1ba9c60
NOTICE:  BL1: Built : 17:51:33, Apr 30 2017
INFO:    BL1: RAM 0xe1000 - 0xe7000
INFO:    BL1: Loading BL2
INFO:    Loading image id=1 at address 0xe9000
INFO:    Image id=1 loaded at address 0xe9000, size = 0x5008
NOTICE:  BL1: Booting BL2
```

Bl2 boot sequence

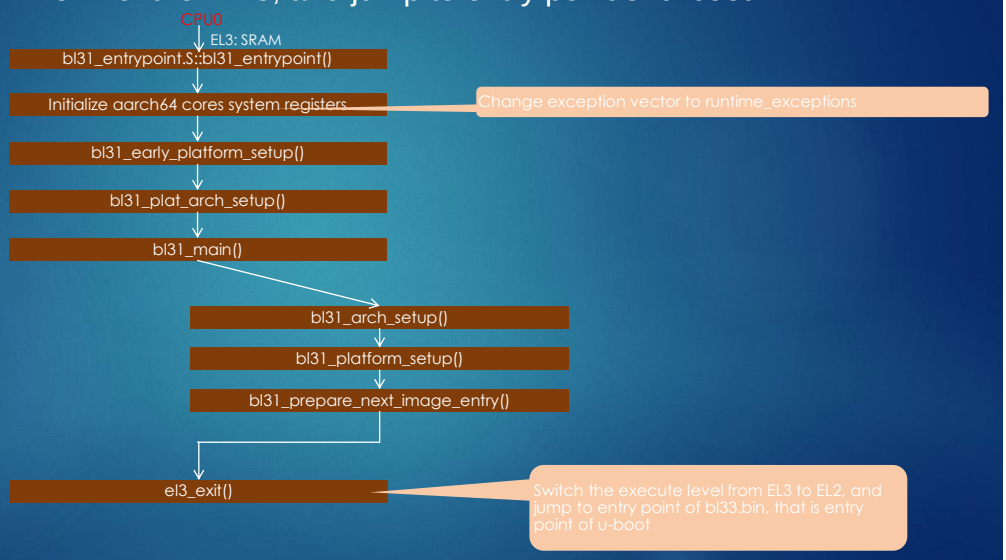
Bl2 setups environment for EL1S, initialize DDR controller and PHY, and load BL3-n images




```
NOTICE:  BL1: Booting BL2
  INFO:    Entry point address = 0xe9000
  INFO:    SPSR = 0x3c5
  NOTICE:  BL2: v1.3(debug):v1.3-372-g1ba9c60
  NOTICE:  BL2: Built : 17:51:33, Apr 30 2017
  INFO:    BL2: Loading BL31
  INFO:    Loading image id=3 at address 0x129000
  INFO:    Image id=3 loaded at address 0x129000, size = 0x8038
  INFO:    BL2: Loading BL33
  INFO:    Loading image id=5 at address 0x37000000
  INFO:    Image id=5 loaded at address 0x37000000, size = 0x58f17
  NOTICE:  BL1: Booting BL31
```

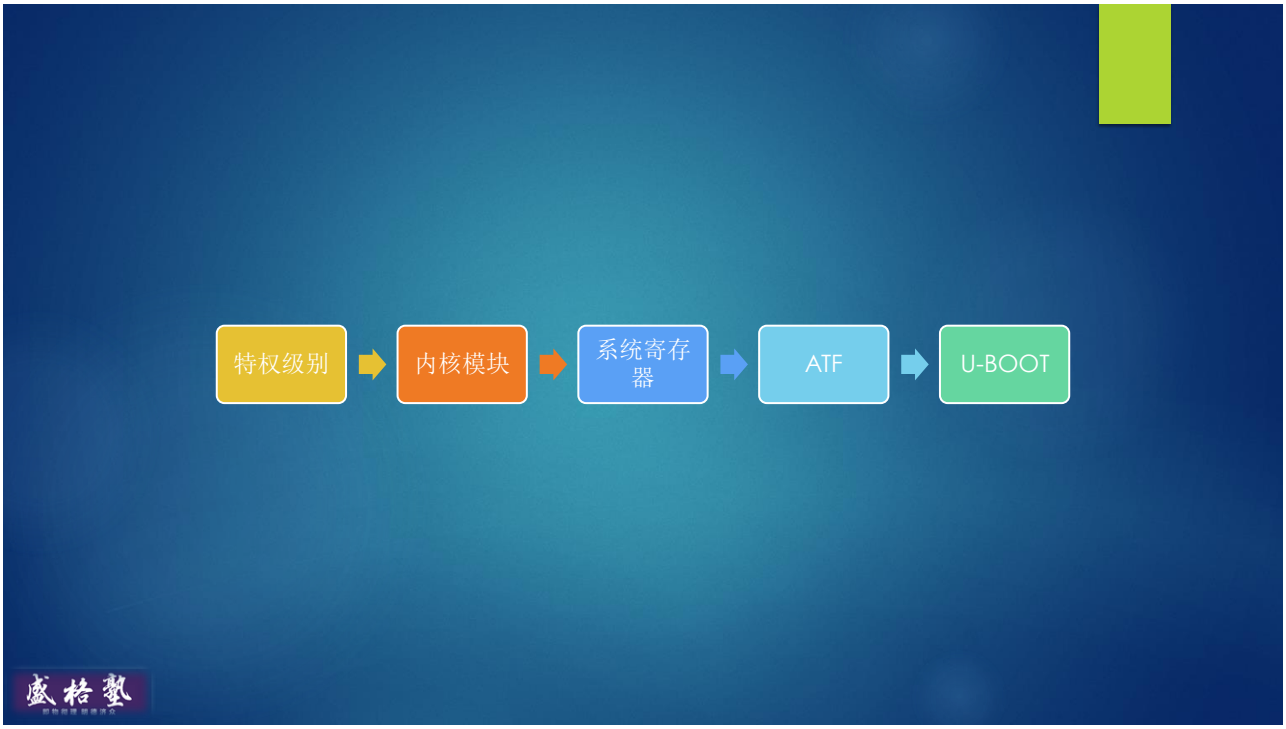
Bl3-1 boot sequence

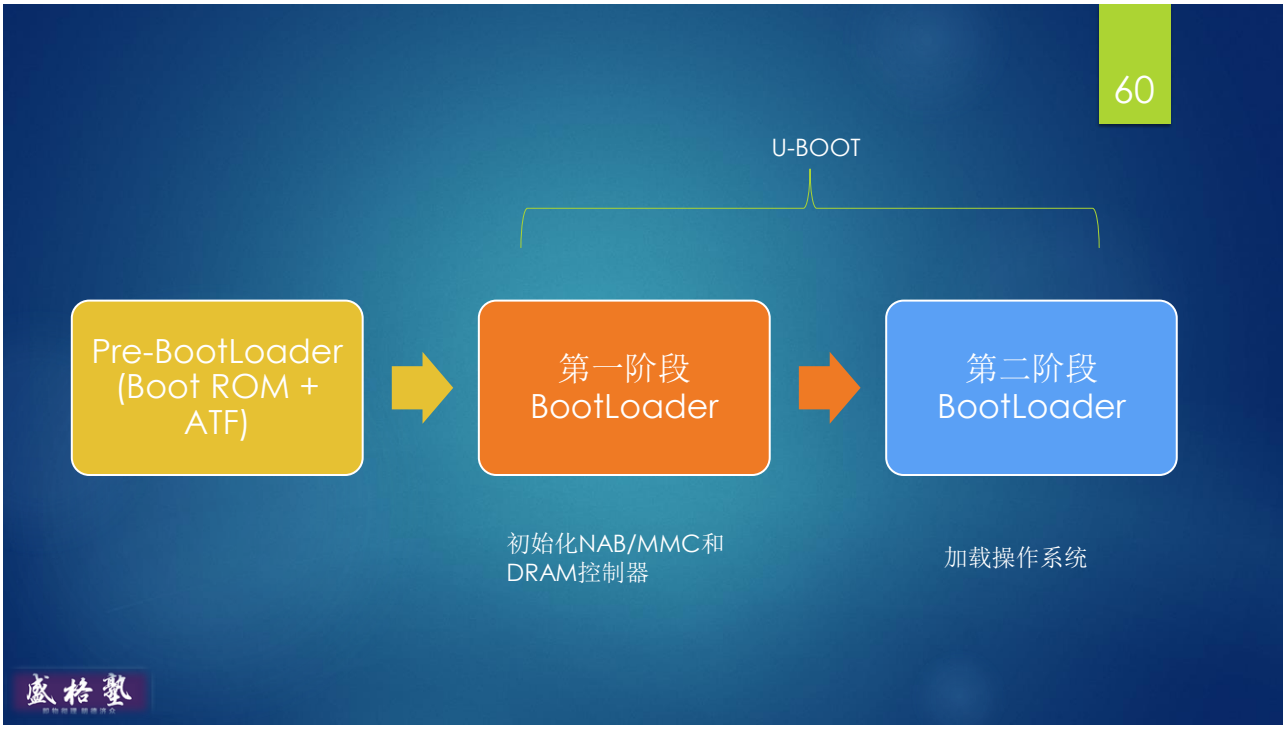
BL3-1 setups environment for EL3, and jump to entry point of u-boot.




```
NOTICE:  BL1: Booting BL31
INFO:    Entry point address = 0x129000
INFO:    SPSR = 0x3cd
INFO:    Boot bl33 from 0x37000000 for 364311 Bytes
NOTICE:  BL31: v1.3(debug):v1.3-372-g1ba9c60
NOTICE:  BL31: Built : 17:51:33, Apr 30 2017
INFO:    BL31: Initializing runtime services
INFO:    BL31: Preparing for EL3 exit to normal world
INFO:    Entry point address = 0x37000000
INFO:    SPSR = 0x3c9
```

U-Boot 2017.05-rc2-00130-gd2255b0 (Apr 30 2017 - 17:51:28 +0200)poplar


特权级别
内核模块
系统寄存器
ATF
U-BOOT
盛格塾


60
U-BOOT
Pre-BootLoader (Boot ROM + ATF)
第一阶段 BootLoader
第二阶段 BootLoader
初始化NAB/MMC和DRAM控制器
加载操作系统
盛格塾

61

WAVE DIGITECH EASY IT SOLUTIONS

Types of Boot Loaders

- What are different types of boot loaders ?
  - Boot-ROM (or Pre-Boot Loader)
    - Small code which loads First stage boot loader
  - First Stage Boot Loader
    - Small Piece of code that initialize the NAND/MMC & DRAM controller.
  - Second Stage Boot Loader
    - Primary function of the second-stage boot loader is to Loading the kernel into RAM or jumping directly to the start of the kernel.

E-mail: info@wavedigitech.com; http://www.wavedigitech.com Phone : 91-9632839173

盛格塾

62

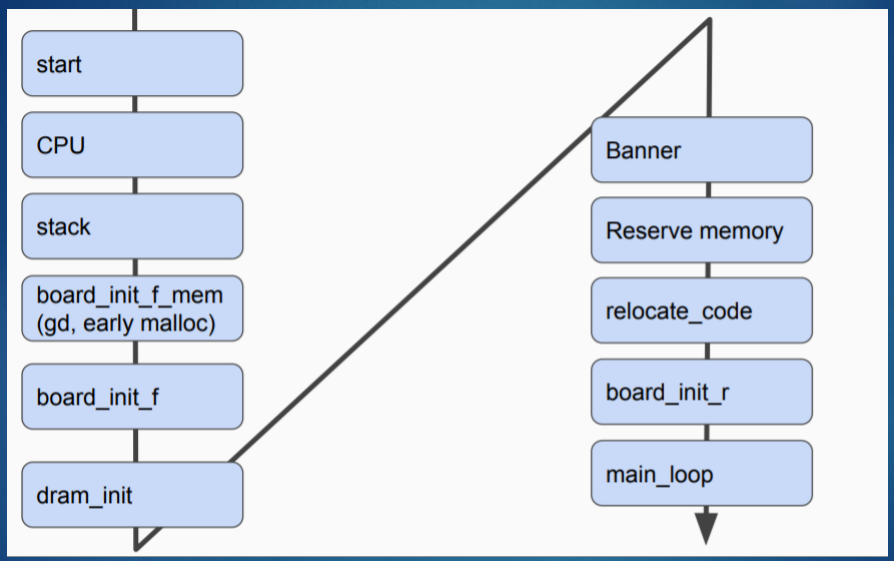
U-boot

denx software engineering

- Das U-Boot -- the Universal Boot Loader
- Free Software: full source code under GPL
- Oct 22, 1999: fadsrom - Dan Malek => PPCBoot rev. 1.1
- www.denx.de

盛格塾

63



U-Boot proper on ARM

U-Boot startup sequence

Simon Glass, sjg@chromium.org

64

构建U-Boot

- export ARCH=arm
- export PATH=~/arm_eabi/bin/:$PATH
- export CROSS_COMPILE=arm-eabi-

- make distclean
- make <board>_config
- make

Output :

- u-boot is an ELF file of the code
- u-boot.bin is the binary image. (with ELF headers striped out) used for download and debugging
- u-boot.srec is the S-Record image.
- u-boot.map has the global addresses for symbols, etc.

调试U-Boot

65

- Include/configs/logannite.h
  - #define DEBUG
- Gcc –g
- Objdump –D u-boot > u-boot.dis

盛格塾

调试在ROM中运行的阶段

66

```
bash[0]$ ${CROSS_COMPILE}gdb u-boot

(gdb) target remote bdi:2001
Remote debugging using bdi:2001
0xfffffffc in ?? ()
(gdb) b cpu_init_f
Breakpoint 1 at 0xfffd3310: file cpu_init.c, line 136.
(gdb) c
Continuing.

Breakpoint 1, cpu_init_f () at cpu_init.c:136
136             asm volatile("  bl      0f"             ::: "lr");
(gdb) s
137             asm volatile("0:        mflr    3"              :::
"r3");
(gdb)
138             asm volatile("  addi    4, 0, 14"       ::: "r4");
(gdb)
```

cpu_init_f is the first C function called from the code in start.C.

盛格塾

搬家

67

- U-boot会搬迁到DRAM中运行
  - 从ROM搬到RAM

struct global_data (register r8)→'relocaddr' contains start address of u-boot in RAM

Relocation address = <MAXMEM> - CFG_MONITOR_LEN
= 16 MB – 192 KB
= 0x1000000 - 0x30000 = 0xFD0000

盛格塾

搬迁后调试

68

```
(gdb) symbol-file
Discard symbol table from `/home/dzu/denx/cvs-trees/u-boot/u-boot'? (y or n) y
No symbol file now.
(gdb) add-symbol-file u-boot 0xfd0000
add symbol table from file "u-boot" at
        .text_addr = 0xfd0000
(y or n) y
Reading symbols from u-boot...done.
(gdb) b board_init_r
Breakpoint 2 at 0xfd99ac: file board.c, line 533.
(gdb) c
Continuing.

Breakpoint 2, board_init_r (id=0xfbb1f0, dest_addr=16495088) at board.c:533
533     {
(gdb)
```

board_init_r is the first C routine running in the newly relocated C friendly RAM environment.

盛格塾

基本命令

69

- 信息命令
- 内存命令
- 闪存命令
- 执行控制命令
- 网络命令
- 环境变量命令
- 文件系统命令
- 特殊命令

盛格塾

Log命令

70

- log info - show pointer details
- log log reset - clear contents
- log log show - show contents
- log log append - append to the logbuffer
- setenv stdout log - redirect standard output to log buffer

盛格塾

内嵌调试器命令

71

- ds - disassemble memory
- as - assemble memory
- break - set or clear a breakpoint
- continue - continue from a breakpoint
- step - single step execution.
- next - single step execution, stepping over subroutines.
- where - Print the running stack.
- rdump - Show registers.

盛格塾

硬件诊断命令

72

- cache - Cache test
- watchdog - Watchdog timer test
- i2c - I2C test
- rtc - RTC test
- memory - Memory test
- cpu - CPU test
- uart - UART test
- ethernet - ETHERNET test
- spi - SPI test
- usb - USB test
- spr - Special register test
- sysmon - SYSMON test
- dsp - DSP test

盛格塾

使用fdisk观察镜像文件

73

```
gedu@gedu-VirtualBox:~/gearm$ fdisk -l 2020-02-13-raspbian-buster.img
Disk 2020-02-13-raspbian-buster.img: 3.54 GiB, 3787456512 bytes, 7397376 sector
s
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0xea7d04d6

Device                          Boot  Start     End Sectors  Size Id Type
2020-02-13-raspbian-buster.img1        8192  532479  524288  256M  c W95 FAT32
2020-02-13-raspbian-buster.img2      532480 7397375 6864896  3.3G 83 Linux
```

盛格塾

挂接主文件系统

74

sudo mount -t ext4 -v -o offset=272629760 2020-02-13-raspbian-buster.img /mnt/raspbian
mount: /dev/loop5 mounted on /mnt/raspbian.

2: kd> ?? 532480*512
int 0n272629760

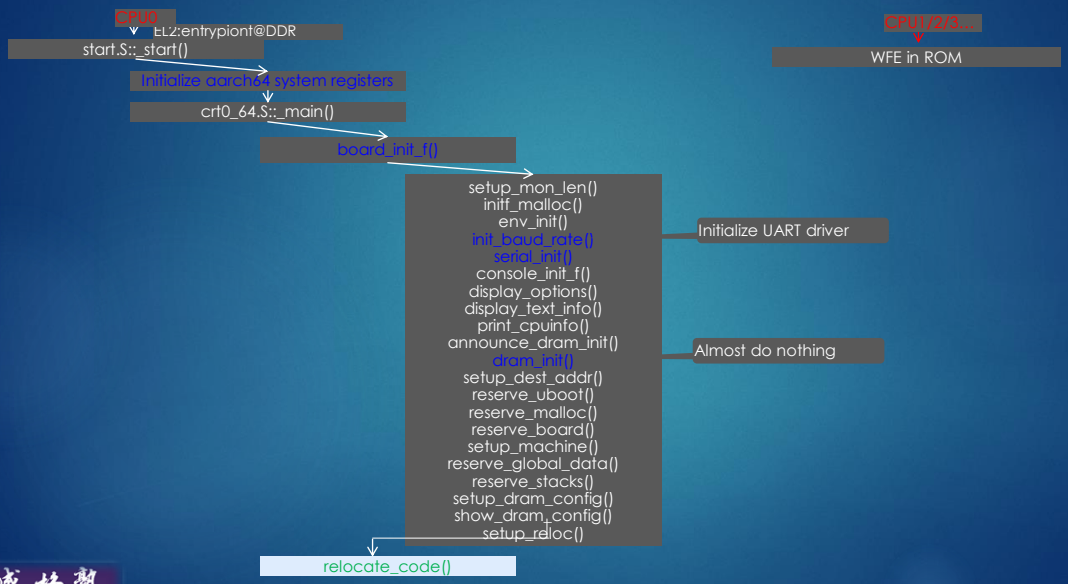
- 偏移位置一般以扇区为单位，乘以512转为字节数

盛格塾

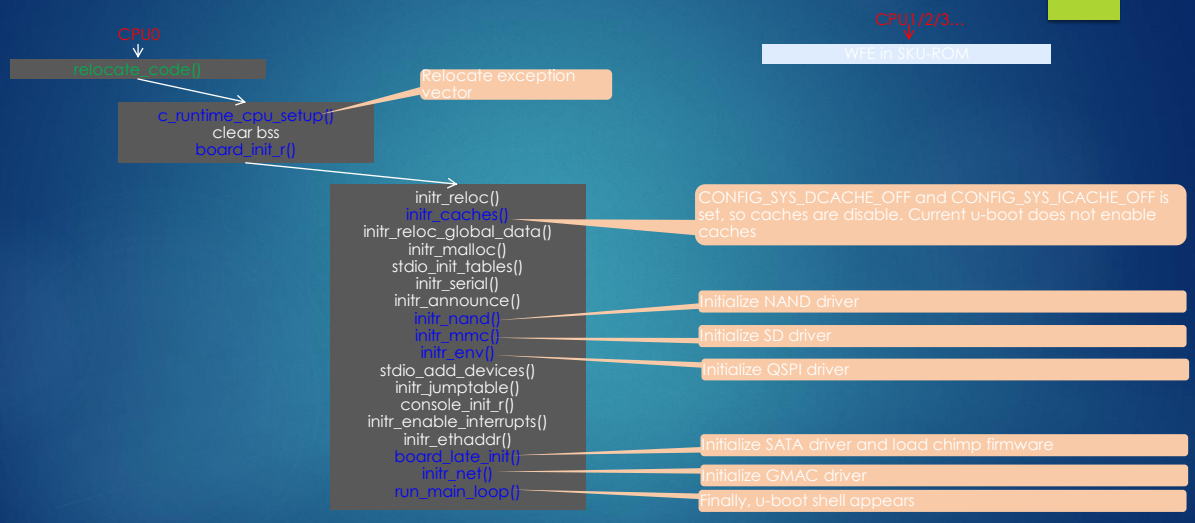
aarch64 U-boot

u-boot does not initialize DDR because it has been initialized in ATF stage
In u-boot, CPU0's execute level is EL2. Execute level will change to EL1N before booting Linux Kernel
Mostly in u-boot MMU, caches, interrupt are all disabled
Flattened device tree utility commands are supported

U-boot boot sequence (1 of 2)



U-boot boot sequence (2 of 2)



Device drivers in U-boot

UART, NAND, QSPI, SD, GMAC and SATA drivers are initialized during u-boot boot

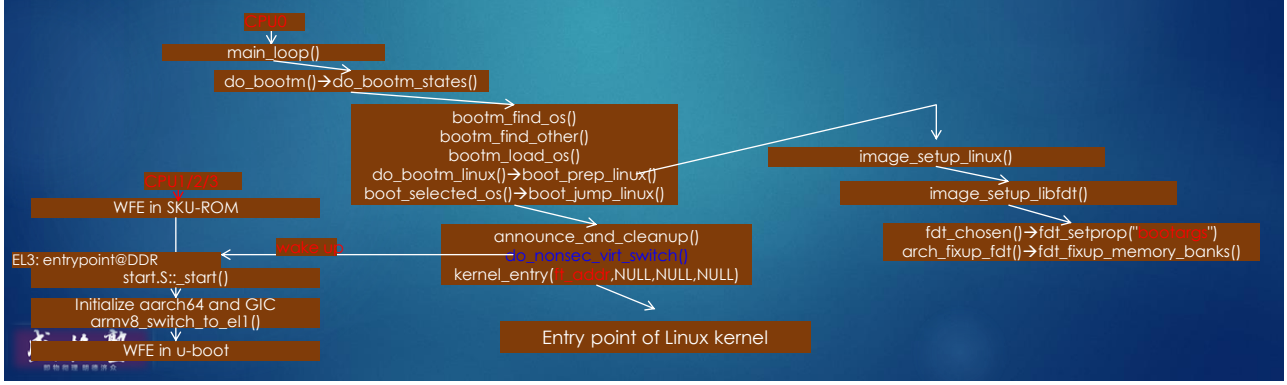
- Try nand/sf/mmc/ping/scsi commands in u-boot shell to use NAND/QSPI/SD/GMAC/SATA devices

Some other device drivers are supported in U-boot, though they are not initialized during u-boot boot

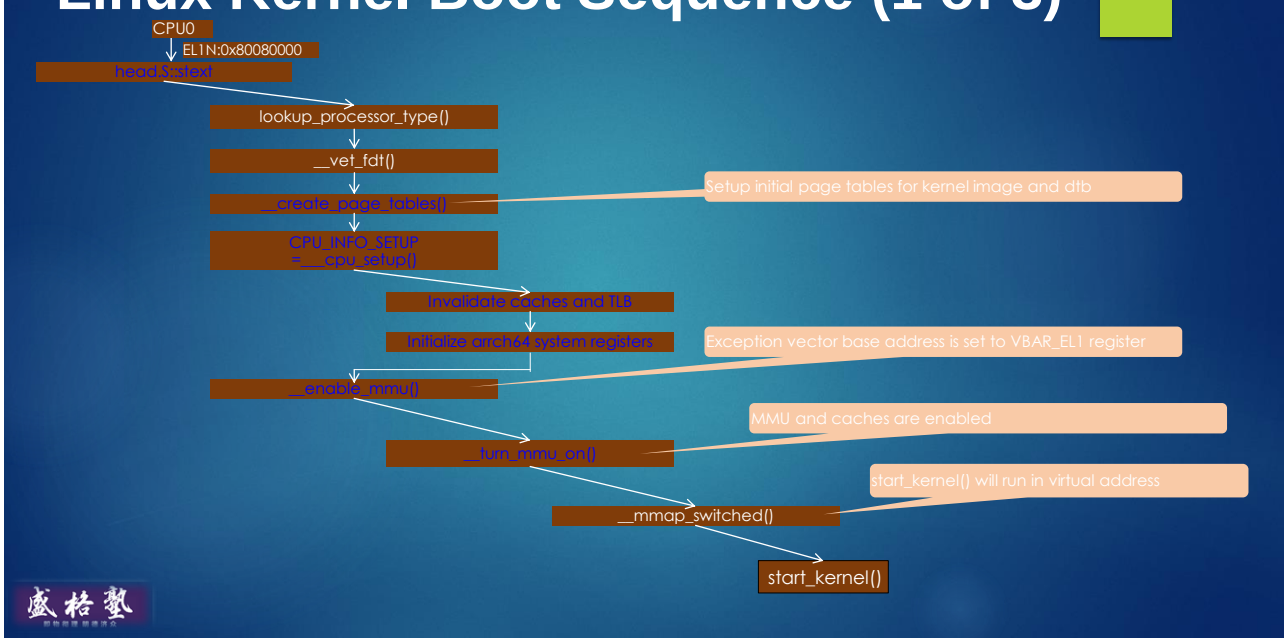
- USB2.0
  - uboot64/drivers/usb/ehci-usb.c
  - Run “usb start” command in u-boot shell to initialize USB2.0 driver
  - Use usb commands to access USB device
- SPI
  - uboot64/drivers/spi/pl022_spi.c
  - Use sspi commands to access SPI device
- I2C
  - uboot64/drivers/spi/brcm_i2c.c
  - Use i2c commands to access I2C device

Boot Linux Kernel in u-boot

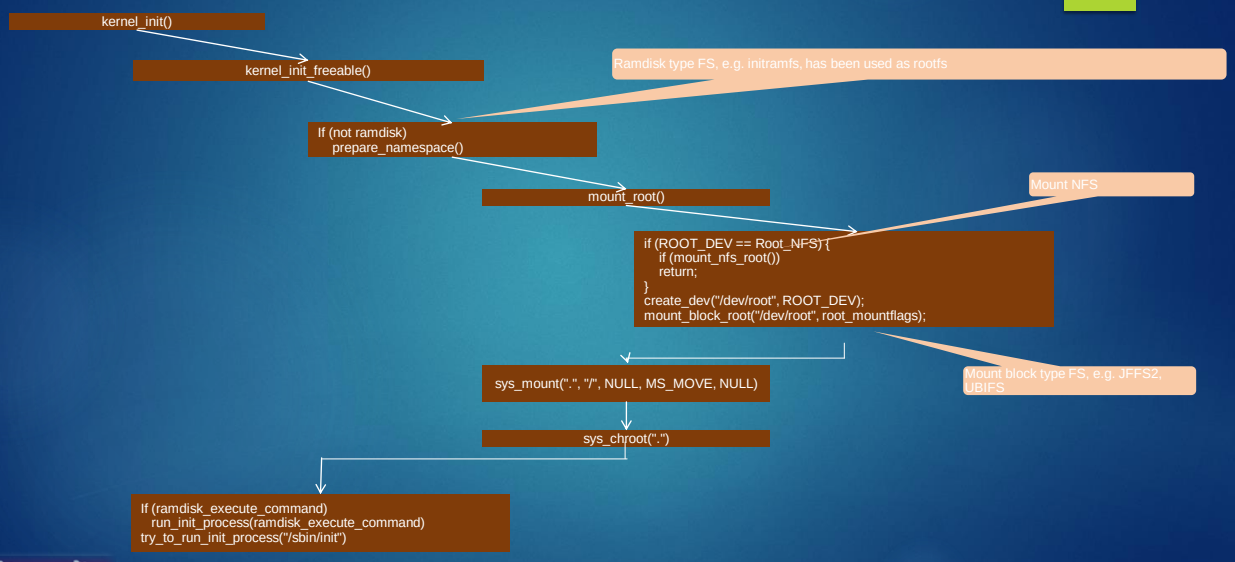
- Set Linux kernel boot parameters
  u-boot>setenv bootargs " loglevel=7 console=ttyS0,115200n8 ubi.mtd=nrootfs root=ubi0 rootfstype=ubifs pci=pcie_bus_safe"
- Load uImage and dtb into memory
  sf read 0x8007FFC0 0x200000 0xE00000
  sf read 0x84000000 0xc00000 0x10000
- Run bootm to boot Linux kernel
  u-boot> bootm 0x8007FFC0 - 0x84000000



Linux Kernel Boot Sequence (1 of 3)



Mount Root File System in Linux Kernel



案例探讨——挂接根分区时Panic

[1.548169] Kernel panic - not syncing: VFS: Unable to mount root fs on unknown-block(0,0)

```
[    1.544198] VFS: Cannot open root device "(null)" or unknown-block(0,0): error -6
[    1.544794] rockchip-dmc dmc: Cannot get the regulator "center"
[    1.545575] Please append a correct "root=" boot option; here are the available partitions:
[    1.545951] rockchip-saradc ff280000.saradc: failed to get regulator, -517
[    1.546656] rk-vcodec ff360000.rkvdec: vcodec regulator not ready, retry
[    1.547685] 0100          4096 ram0  (driver?)
[    1.548169] Kernel panic - not syncing: VFS: Unable to mount root fs on unknown-block(0,0)
[    1.548473] rockchip-iodomain ff100000.syscon:io-domains: Looking up vccio1-supply from device tree
[    1.549801] CPU: 2 PID: 1 Comm: swapper/0 Not tainted 4.4.179 #174
[    1.550378] Hardware name: Rockchip RK3328 EVB avb (DT)
[    1.550874] Call trace:
[    1.551141] [<ffffff8008088268>] dump_backtrace+0x0/0x220
[    1.551673] [<ffffff80080884ac>] show_stack+0x24/0x30
[    1.552164] [<ffffff80083b346c>] dump_stack+0x94/0xbc
[    1.552648] [<ffffff8008160c10>] panic+0xe4/0x238
[    1.553113] [<ffffff8008e211d0>] mount_block_root+0x22c/0x298
[    1.553663] [<ffffff8008e213c8>] mount_root+0x70/0x80
[    1.554146] [<ffffff8008e21578>] prepare_namespace+0x1a0/0x1b0
[    1.554701] [<ffffff8008e20dd0>] kernel_init_freeable+0x1e8/0x220
[    1.555280] [<ffffff8008a1edc0>] kernel_init+0x18/0x100
[    1.555785] [<ffffff8008082ef0>] ret_from_fork+0x10/0x20
```

资源

- ARM® Architecture Reference Manual
- **ARMv8, for ARMv8-A architecture profile**

切问而近思

欢迎关注格友公众号

盛格塾