Linux - BL808

编译前的准备

- 一台 Ubuntu 20.04 的 host PC（or 虚拟机）
- host 内存建议大于 2GB
- host 硬盘建议大于 40GB
- 一块 BL808 开发板（flash 大于 8M）
- 一个 USB 转 TTL 串口（波特率支持 2000000Mpbs 及以上）
-

BL808 的 Linux SDK 是在 Ubuntu 20.04 开发测试的，因此推荐使用 Ubuntu 20.04 的主机环境进行以下开发步骤，以减少开发环境带来的不一致导致的一些问题。当然了 Ubuntu 16、18 等版本也是可以的只是在编译过程中可能需要自行安装缺少的一些工具或组件。也可以使用 Oracle VM VirtualBox 开源虚拟机软件安装相应的虚拟机进行编译。

本教程基于 Ubuntu 20.04 虚拟机进行的；如使用实体机器开发，可跳过 Oracle VM VirtualBox 下载安装部分。

Oracle VM VirtualBox 下载以及虚拟机安装

官网下载地址：https://www.virtualbox.org/

虚拟机的具体安装教程可自行百度

虚拟机安装好之后，建议安装增强功能，安装增强功能可以实现共享剪切板功能、共享文件夹功能。共享粘贴板可在 PC 和虚拟机之间互相粘贴复制内容。文件夹共享可以在本地主机下创建一个文件夹，然后 VirtualBOX 将该文件夹映射到虚拟机下指定的文件夹中，这样我们就可以访问该文件夹下的子文件夹以及文件了。这两个功能都能极大地提高我们的开发效率。

安装增强功能的方法

点击虚拟机界面上面菜单栏的【设备】--【安装增强功能】。然后可以看到在虚拟机的光驱中自动加载了增强 iso 的文件(VBoxGuestAdditions.iso)，点击【Run】运行按钮。接着按虚拟机所指引步骤进行即可。

共享文件夹的设置自行百度

在安装好增强功能之后，可以直接将 BL808 的 Linux SDK 文件拖拽放入虚拟机内。解压 SDK 压缩包即可。

```
$ tar -zxvf bl808_linux_all.tar.gz
```

编译环境配置

编译 linux SDK 需要以下一些工具包，若安装网络环境不够友好，建议将软件包的源设置到国内的开源镜像，如中科大、清华、阿里等。切换软件源之后在安装。

```
$ sudo vi /etc/apt/sources.list

# 将文件内容全部替换成如下内容：

# 默认注释了源码仓库，如有需要可自行取消注释
deb https://mirrors.ustc.edu.cn/ubuntu/ focal main restricted universe multiverse
# deb-src https://mirrors.ustc.edu.cn/ubuntu/ focal main restricted universe multiverse
deb https://mirrors.ustc.edu.cn/ubuntu/ focal-security main restricted universe multiverse
# deb-src https://mirrors.ustc.edu.cn/ubuntu/ focal-security main restricted universe multiverse
deb https://mirrors.ustc.edu.cn/ubuntu/ focal-updates main restricted universe multiverse
# deb-src https://mirrors.ustc.edu.cn/ubuntu/ focal-updates main restricted universe multiverse
deb https://mirrors.ustc.edu.cn/ubuntu/ focal-backports main restricted universe multiverse
# deb-src https://mirrors.ustc.edu.cn/ubuntu/ focal-backports main restricted universe multiverse
# 预发布软件源，不建议启用
# deb https://mirrors.ustc.edu.cn/ubuntu/ focal-proposed main restricted universe multiverse
# deb-src https://mirrors.ustc.edu.cn/ubuntu/ focal-proposed main restricted universe multiverse
```

执行以下命令安装即可

```
$ sudo apt update
$ sudo apt upgrade
$ sudo apt install gcc flex bison libncurses-dev python3 make device-tree-compiler
```

若在编译过程中出现报错或者缺少工具，根据提示安装好即可。

BL808 Linux SDK 源码

进入 bl808_linux_all 目录，有如下文件，简述有关目录如下：

```
$ cd bl808_linux_all
$ ls -al
.
├── bl808_dts         # kernel dts 文件
├── bl_mcu_sdk_bl808  # bl_mcu_sdk 用于编译输出 low load bin
├── build.sh          # 编译脚本
├── linux-5.10.4-808  # linux kernel 源码
├── opensbi-0.6-808   # opensbi 源码
├── out               # 相关 bin 文件输出目录
├── toolchain         # 编译执行过程中需要的工具链
└── README.md         # 说明文件
```

编译

在 bl808_linux_all 文件夹中已经自带了相关的编译工具链等必要的文件，并且提供了，编译脚本，因此不需要再设置相关的工具链和环境变量，只用安装如下步骤运行编译脚本即可。

```
$ ./build.sh --help         # 查看支持的编译命令
$ ./build.sh opensbi        # 执行编译 opensbi
$ ./build.sh kernel_config  # 配置 linux kernel 相关 feature
$ ./build.sh kernel         # 配置 linux kernel 相关 feature
$ ./build.sh dtb            # 编译 dtb 文件
$ ./build.sh low_load       # 编译 low load 文件
$ ./build.sh whole_bin      # 合并出 kernel 相关 whole bin 文件

# $ ./build.sh all          # 执行 all 时会按顺序执行上述编译流程，不过会省略掉 kernel_config 操作
```

编译完成之后，我们就可以在 **【out】** 目录中看到相关的输出文件

```
.
├── fw_jump.bin            # opensbi 固件
├── hw.dtb.5M              # dts 编译输出的 dtb 文件
├── Image.lz4              # 压缩后的 kernel 镜像文件
├── squashfs_test.img      # squashfs 文件系统镜像
├── low_load_bl808_d0.bin  # low_load C906 固件
├── low_load_bl808_m0.bin  # low_load E907 固件
├── merge_7_5Mbin.py       # 合并 opensbi dtb kernel squashfs 到 whole img 的脚本
└── whole_img_linux.bin    # 生成的 whole img 镜像
```

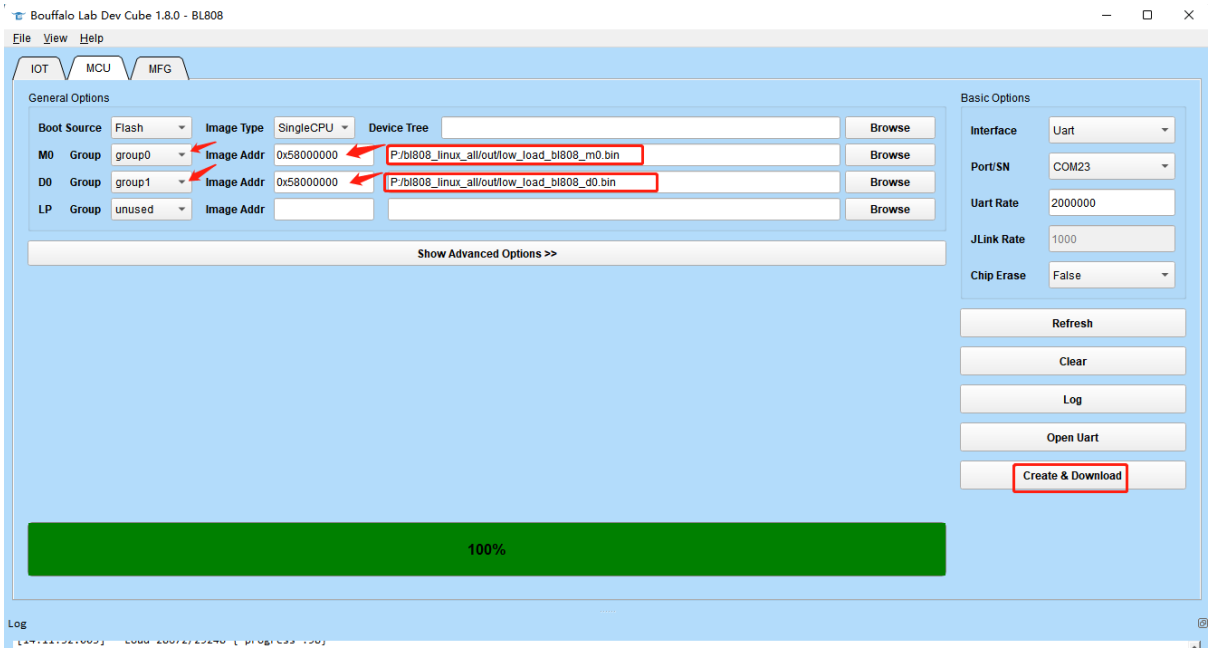
烧写

编译完成之后，即可将相应的文件烧写到 BL808 中去运行，接下来简要介绍一下烧写的方法。

从 `https://dev.bouffalolab.com/download` 页面下载 BouffaloLab Dev Cube 烧写工具软件；

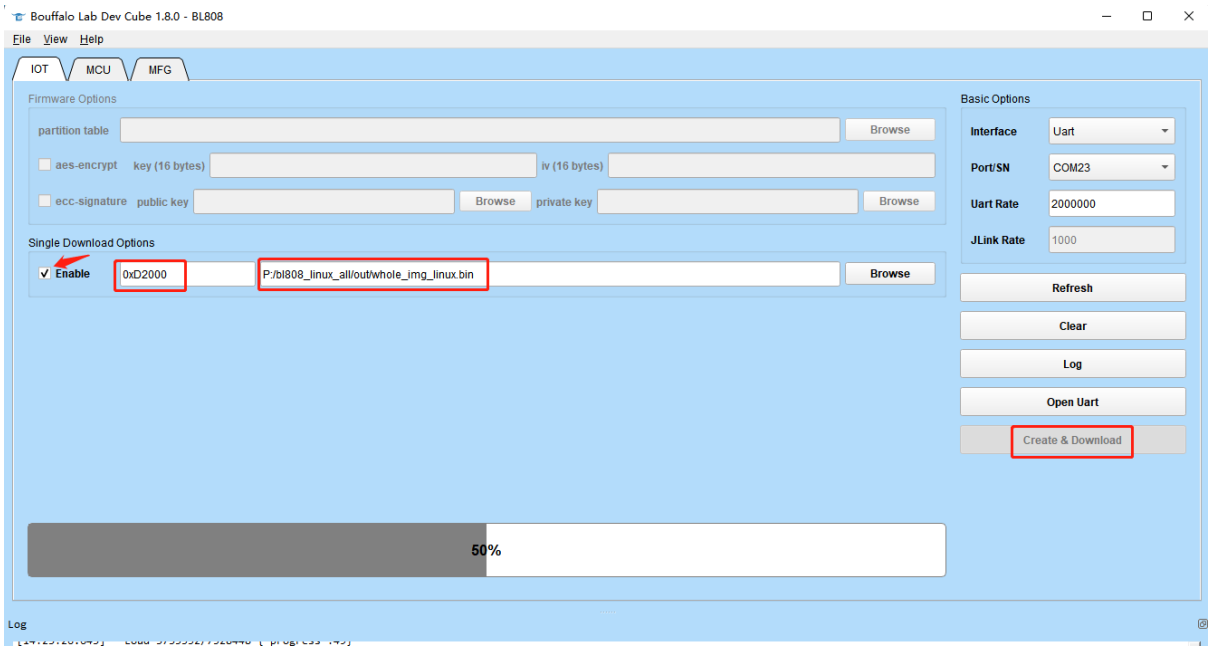
烧写工具支持 Ubuntu 或者 Windows 平台运行，这里以 window 平台为例。

解压后双击运行 `BLDevCube.exe` ，选择 【BL808】打开软件，点击上方的【MCU】标签栏，烧写设置如下图所示：



配置好后，使芯片进入烧写模式，按住 `boot` 键，按一下 `rst` 键；再点击 `Create& Download` 按钮开始下载两个 `low_load_xxxx.bin` 固件到芯片中，在后续的开发说明中会介绍 low_load 相关的作用和功能。

烧写好 low_load 固件后，我们还需要烧写，kernel 相关的 whole img。点击上方的【IOT】标签栏，切换到 IOT 界面。烧写配置如图所示：



再次点击 `Create& Download` 按钮开始下载 `whole_img_linux.bin` 镜像到芯片中。

烧写完成后，即可点击 `rst` 启动芯片。

默认的烧写串口将会输出 E907 运行的相关 log（UART0：IO14 TX， IO15 RX）；

Linux kernel 运行的 log 和 shell 终端将通过 UART3（IO5 RX，IO8 TX） 输出；

kernel 启动后终端如下所示：

```
dynamic memory init success,heap size = 26 Kbyte
C906 start...
mtimer clk:1000000
linux load start...
```

```
len:0x00376c50
vm linux load done!
dtb load done!
opensbi load done!

load time: 426357 us

OpenSBI v0.6
   ____                    _____ ____ _____
  / __ \                  / ____|  _ \_   _|
 | |  | |_ __   ___ _ __ | (___ | |_) || |
 | |  | | '_ \ / _ \ '_ \ \___ \|  _ < | |
 | |__| | |_) |  __/ | | |____) | |_) || |_
  \____/| .__/ \___|_| |_|_____/|____/_____|
        | |
        |_|

Platform Name          : T-HEAD Xuantie c910
Platform HART Features : RV64ACDFIMSUVX
Platform Max HARTs     : 1
Current Hart           : 0
Firmware Base          : 0x3eff0000
Firmware Size          : 56 KB
Runtime SBI Version    : 0.2

MIDELEG : 0x0000000000000222
MEDELEG : 0x000000000000b1ff
[    0.000000] Linux version 5.10.4 (sw@pve103) (riscv64-unknown-linux-gnu-gcc (Xuantie-900 linux-5.10.4 glibc gcc Toolchain V2.2.4 B-
[    0.000000] earlycon: sbi0 at I/O port 0x0 (options '')
[    0.000000] printk: bootconsole [sbi0] enabled
[    0.000000] Zone ranges:
[    0.000000]   DMA32    [mem 0x0000000050000000-0x0000000053ffffff]
[    0.000000]   Normal   empty
[    0.000000] Movable zone start for each node
[    0.000000] Early memory node ranges
[    0.000000]   node   0: [mem 0x0000000050000000-0x0000000053ffffff]
[    0.000000] Initmem setup node 0 [mem 0x0000000050000000-0x0000000053ffffff]
[    0.000000] On node 0 totalpages: 16384
[    0.000000]   DMA32 zone: 224 pages used for memmap
[    0.000000]   DMA32 zone: 0 pages reserved
[    0.000000]   DMA32 zone: 16384 pages, LIFO batch:3
[    0.000000] software IO TLB: Cannot allocate buffer
[    0.000000] SBI specification v0.2 detected
[    0.000000] SBI implementation ID=0x1 Version=0x6
[    0.000000] SBI v0.2 TIME extension detected
[    0.000000] SBI v0.2 IPI extension detected
[    0.000000] SBI v0.2 RFENCE extension detected
[    0.000000] riscv: ISA extensions acdfimsuv
[    0.000000] riscv: ELF capabilities acdfimv
[    0.000000] percpu: Embedded 17 pages/cpu s32600 r8192 d28840 u69632
[    0.000000] pcpu-alloc: s32600 r8192 d28840 u69632 alloc=17*4096
[    0.000000] pcpu-alloc: [0] 0
[    0.000000] Built 1 zonelists, mobility grouping off.  Total pages: 16160
[    0.000000] Kernel command line: console=ttyS0,2000000 loglevel=8 earlyprintk earlycon=sbi root=/dev/mtdblock0 ro rootfstype=squash
[    0.000000] Dentry cache hash table entries: 8192 (order: 4, 65536 bytes, linear)
[    0.000000] Inode-cache hash table entries: 4096 (order: 3, 32768 bytes, linear)
[    0.000000] Sorting __ex_table...
[    0.000000] mem auto-init: stack:off, heap alloc:off, heap free:off
[    0.000000] Memory: 53200K/65536K available (3960K kernel code, 2846K rwdata, 2048K rodata, 159K init, 288K bss, 12336K reserved, 0
[    0.000000] SLUB: HWalign=64, Order=0-3, MinObjects=0, CPUs=1, Nodes=1
[    0.000000] rcu: Hierarchical RCU implementation.
[    0.000000] rcu:     RCU restricting CPUs from NR_CPUS=8 to nr_cpu_ids=1.
[    0.000000]  Tracing variant of Tasks RCU enabled.
[    0.000000] rcu: RCU calculated value of scheduler-enlistment delay is 25 jiffies.
[    0.000000] rcu: Adjusting geometry for rcu_fanout_leaf=16, nr_cpu_ids=1
[    0.000000] NR_IRQS: 64, nr_irqs: 64, preallocated irqs: 0
[    0.000000] riscv-intc: 64 local interrupts mapped
[    0.000000] plic: interrupt-controller@e0000000: mapped 64 interrupts with 1 handlers for 2 contexts.
[    0.000000] random: get_random_bytes called from start_kernel+0x298/0x3a6 with crng_init=0
[    0.000000] riscv_timer_init_dt: Registering clocksource cpuid [0] hartid [0]
[    0.000000] clocksource: riscv_clocksource: mask: 0xffffffffffffffff max_cycles: 0x1d854df40, max_idle_ns: 3526361616960 ns
[    0.000019] sched_clock: 64 bits at 1000kHz, resolution 1000ns, wraps every 2199023255500ns
[    0.000818] Console: colour dummy device 80x25
[    0.001080] Calibrating delay loop (skipped), value calculated using timer frequency.. 2.00 BogoMIPS (lpj=4000)
[    0.001568] pid_max: default: 32768 minimum: 301
[    0.002187] Mount-cache hash table entries: 512 (order: 0, 4096 bytes, linear)
[    0.002512] Mountpoint-cache hash table entries: 512 (order: 0, 4096 bytes, linear)
[    0.007158] ASID allocator initialised with 65536 entries
[    0.007762] rcu: Hierarchical SRCU implementation.
[    0.009098] smp: Bringing up secondary CPUs ...
[    0.009261] smp: Brought up 1 node, 1 CPU
[    0.010349] devtmpfs: initialized
[    0.013166] clocksource: jiffies: mask: 0xffffffff max_cycles: 0xffffffff, max_idle_ns: 7645041785100000 ns
[    0.013641] futex hash table entries: 256 (order: 2, 16384 bytes, linear)
[    0.015213] NET: Registered protocol family 16
[    0.016367] DMA: preallocated 128 KiB GFP_KERNEL pool for atomic allocations
[    0.017282] DMA: preallocated 128 KiB GFP_KERNEL|GFP_DMA32 pool for atomic allocations
```

```
[    0.018442] i2c-core: driver [dummy] registered
[    0.039296] SCSI subsystem initialized
[    0.041703] clocksource: Switched to clocksource riscv_clocksource
[    0.065219] NET: Registered protocol family 2
[    0.067180] tcp_listen_portaddr_hash hash table entries: 256 (order: 0, 4096 bytes, linear)
[    0.067600] TCP established hash table entries: 512 (order: 0, 4096 bytes, linear)
[    0.068030] TCP bind hash table entries: 512 (order: 1, 8192 bytes, linear)
[    0.068434] TCP: Hash tables configured (established 512 bind 512)
[    0.069147] UDP hash table entries: 256 (order: 1, 8192 bytes, linear)
[    0.069568] UDP-Lite hash table entries: 256 (order: 1, 8192 bytes, linear)
[    0.070318] NET: Registered protocol family 1
[    0.072524] workingset: timestamp_bits=62 max_order=14 bucket_order=0
[    0.087316] squashfs: version 4.0 (2009/01/31) Phillip Lougher
[    0.088968] NET: Registered protocol family 38
[    0.089239] Block layer SCSI generic (bsg) driver version 0.4 loaded (major 252)
[    0.089659] io scheduler mq-deadline registered
[    0.089813] io scheduler kyber registered
[    0.095356] 30002000.serial: ttyS0 at MMIO 0x30002000 (irq = 1, base_baud = 2000000) is a BFLB UART
[    0.095823] printk: console [ttyS0] enabled
[    0.095823] printk: console [ttyS0] enabled
[    0.096275] printk: bootconsole [sbi0] disabled
[    0.096275] printk: bootconsole [sbi0] disabled
[    0.126769] brd: module loaded
[    0.152765] loop: module loaded
[    0.154292] physmap-flash 58500000.xip_flash: physmap platform flash device: [mem 0x58500000-0x588fffff]
[    0.155755] 1 fixed-partitions partitions found on MTD device xip-flash.0
[    0.156188] Creating 1 MTD partitions on "xip-flash.0":
[    0.156527] 0x000000000000-0x000000280000 : "rootfs"
[    0.161240] mousedev: PS/2 mouse device common for all mice
[    0.162183] i2c /dev entries driver
[    0.162725] i2c-core: driver [i2c-slave-eeprom] registered
[    0.164126] [perf] T-HEAD C900 PMU probed
[    0.166787] NET: Registered protocol family 10
[    0.169125] Segment Routing with IPv6
[    0.169643] sit: IPv6, IPv4 and MPLS over IPv4 tunneling driver
[    0.171529] NET: Registered protocol family 17
[    0.171895] Key type dns_resolver registered
[    0.172518] debug_vm_pgtable: [debug_vm_pgtable         ]: Validating architecture page table helpers
[    0.185412] VFS: Mounted root (squashfs filesystem) readonly on device 31:0.
[    0.192759] devtmpfs: mounted
[    0.193766] Freeing unused kernel memory: 156K
[    0.218718] Run /sbin/init as init process
[    0.218989]   with arguments:
[    0.219183]     /sbin/init
[    0.219362]     earlyprintk
[    0.219545]   with environment:
[    0.219748]     HOME=/
[    0.219907]     TERM=linux
*********************************
 Exec rcS
*********************************
********mount all********
mount: according to /proc/mounts, porc is already mounted on /proc
mount: according to /proc/mounts, devtmpfs is already mounted on /dev
mount: mounting devpts on /dev/pts failed: No such file or directory
This may take some time ...
mount: mounting sysfs on /sys failed: Device or resource busy
--------Start Local Services--------
*********************************
*********************************

Linux login: root
```

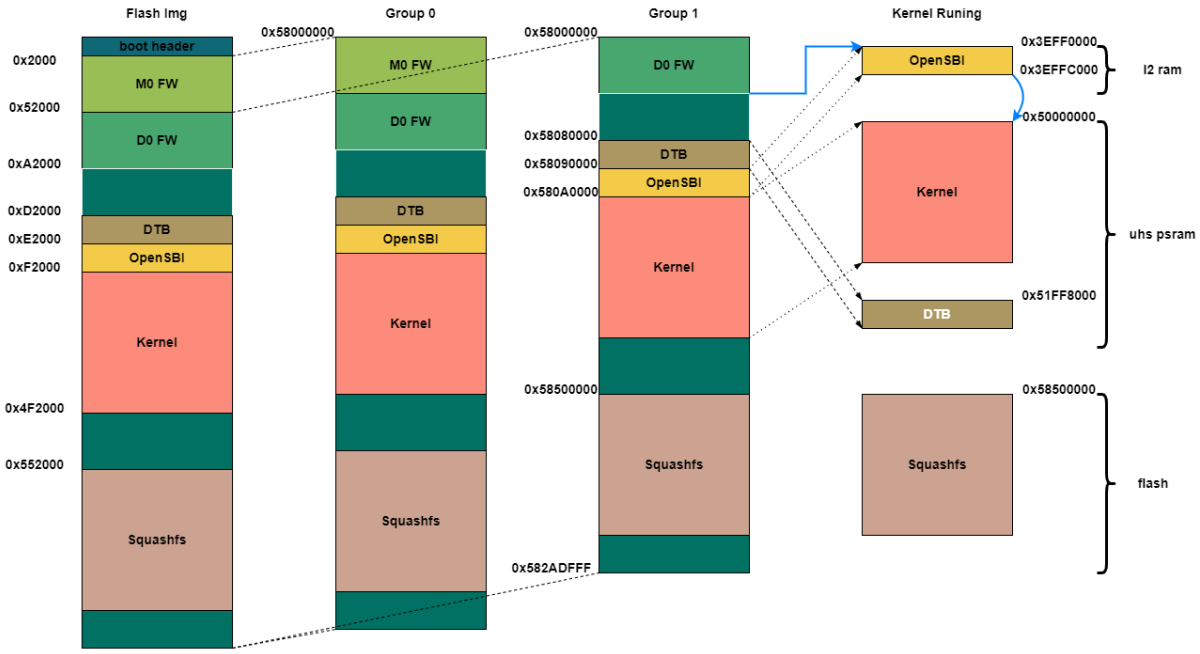
在终端中输入 `root` 即可进入系统。进入系统后即可使用一些标准的 Linux 系统命令进行操作了。

到这里一个基本的 linux kernel 就成功运行起来了。

Kernel 启动流程

Flash镜像排布

要想了解整个 kernel 的启动流程，我们先了解一下各个镜像在 flash 中的排布，完整的 flash 镜像如图左所示，分别包含了 boot header、M0 FW、D0 FW、DTB、OpenSBI、Kernel、Squashfs；



芯片启动后，首先会执行片上 ROM 程序完成芯片硬件相关的设置（如 power-up、clock setup 等），执行完会先启动 M0 即运行 `low_load m0` 固件，完成 psram 等硬件初始化，给 D0（C906）运行 kernel 准备环境；D0 会一直等待 M0 准备完毕后在继续运行，在 `low_load d0` 固件中会从 flash 相应的地址将 OpenSBI、DTB、Kernel 分别加载解压到相应的 pSRAM 上，并设置 PMP 等操作；加载完毕后，就会跳转到 OpenSBI 运行，OpenSBI 运行完毕后就会引导 Kernel 启动。

LowLoader

Low Load 相关源码在 **bl_mcu_sdk_bl808→examples→low_load** 目录下

OpenSBI

OpenSBI 目前使用的是 v0.6 版本，bl808 相关的主要源码在 **opensbi-0.6-bl808→platform→thead→c910** 目录下

Linux Kernel

Linux Kernel 目前使用的是基于 5.10.4 内核+ T-head patch 的版本，目前主要是在 drivers 中添加了 UART 相关的源码，用于 kernel shell 交互。与 kernel 相关的设备树文件在 `bl808_dts` 目录下，主要定义了 memory 、cpu、串口、文件系统等设备。