Linux - BL808

Preparation before compilation

- A host PC (or virtual machine) for Ubuntu 20.04
- Host memory recommended to be
- greater than 2GB host

  hard drive recommended to be greater than 40GB
- A BL808 development board (flash larger than 8M)
- A USB to TTL serial port (baud rate supports 2000000Mpbs and above)
-

The BL808 Linux SDK is developed and tested on Ubuntu 20.04, so it is recommended to use the Ubuntu 20.04 host environment for the following development steps to minimize some problems caused by inconsistencies in the development environment. Of course, Ubuntu 16, 18, etc. are also possible, but you may need to self-install some missing tools or components during the compilation process. You can also use Oracle VM VirtualBox open source virtual machine software to install the corresponding virtual machine for compilation.

This tutorial is based on the Ubuntu 20.04 virtual machine; if you are developing on a physical machine, you can skip the Oracle VM VirtualBox download and installation section.

Oracle VM VirtualBox Download and Virtual Machine Installation

Official Website:

https://www.virtualbox.org/虚拟机的具体

安装教程可自百度

After the virtual machine is installed, it is recommended to install the enhancements. Installing the enhancements will enable the shared clipboard function and shared folder function. Shared Pasteboard allows you to paste and copy content between PCs and virtual machines. Folder sharing can create a folder under the local host, and then VirtualBOX will map the folder to the specified folder under the virtual machine.

In this way, we can access sub-folders and files under the folder. Both of these functions can greatly improve our development efficiency.

Method of installing enhancements

Click [Device] -[Install Enhancement] on the menu bar at the top of the virtual machine screen. Then you can see that the enhanced iso file is loaded automatically in the CD-ROM drive of the virtual machine.

(VBoxGuestAdditions.iso), and click the [Run] button. Then follow the steps in the virtual machine guide.

Setting up Self-Developed Folders

After installing the enhancements, you can drag and drop the BL808's Linux SDK files directly into the virtual machine.

Simply unzip the SDK zip file.

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

Compiler environment configuration

If the installation network environment is not friendly enough, it is recommended to set the source of the package to a domestic open source mirror, such as CSC, Tsinghua, A.A., etc. After switching the software source, you can install it.

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

# Replace all the contents of the file

with the following.

#  The source code repository is commented out by default and can be uncommented if needed.
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
# Pre-release software sources, not recommended for use
# 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
```

To `install` it, `execute` the following `command`

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

If there is an error or missing tool during the compilation process, install it according to the instructions.

BL808 Linux SDK Source Code

Enter the bl808_linux_all entry, the following file is available, briefly describe the entry as follows.

```
$ cd bl808_linux_all
$ ls -al
.
├── bl808_dts           # kernel dts file
├── bl_mcu_sdk_bl808 #  bl_mcu_sdk used to compile output low load bin
├── build.sh            # Compile script
├── linux-5.10.4-808 #  linux kernel source code
├── opensbi-0.6-808 #  opensbi source code
├ out                   # Related bin file output destination
├ toolchain             # Toolchain needed for compilation execution
└── README.md           # Description File
```

compile

The bl808_linux_all folder already contains the necessary files such as the relevant toolchain and the compilation scripts, so there is no need to set the relevant toolchain and environment variables.

```
$ . /build.sh --help          # View supported
$ . /build.sh                 compilation commands
opensbi                       # Compile opensbi
$ . /build.sh kernel_config # Configure linux kernel-related
features
$ . /build.sh kernel           # Configure linux kernel-related
features
$ . /build.sh dtb              # Compile the dtb file
$ . /build.sh low_load         # Compile the low load file
$ /build.sh whole_bin          # Merge out kernel-related whole
# $ . /build.sh               # The above compilation process is performed sequentially when executing
bin files                     all, but the kernel_config operation is omitted
```

Once the compilation is complete, we can see the output file in the [**out**] entry

```
.
├── fw_jump.bin              # opensbi firmware
├── hw.dtb.5M                #  dts  compiled  output
├── Image.lz4                dtb  file  #  compressed
├ squashfs_test.img          kernel  image  file  #
├── low_load_bl808_d0.bin #  low_load C906 firmware
├── low_load_bl808_m0.bin #  low_load E907 firmware
├── merge_7_5Mbin.py          # script to merge opensbi,dtb,kernel,squashfs to whole img
└── whole_img_linux.bin       # The created whole img image
```

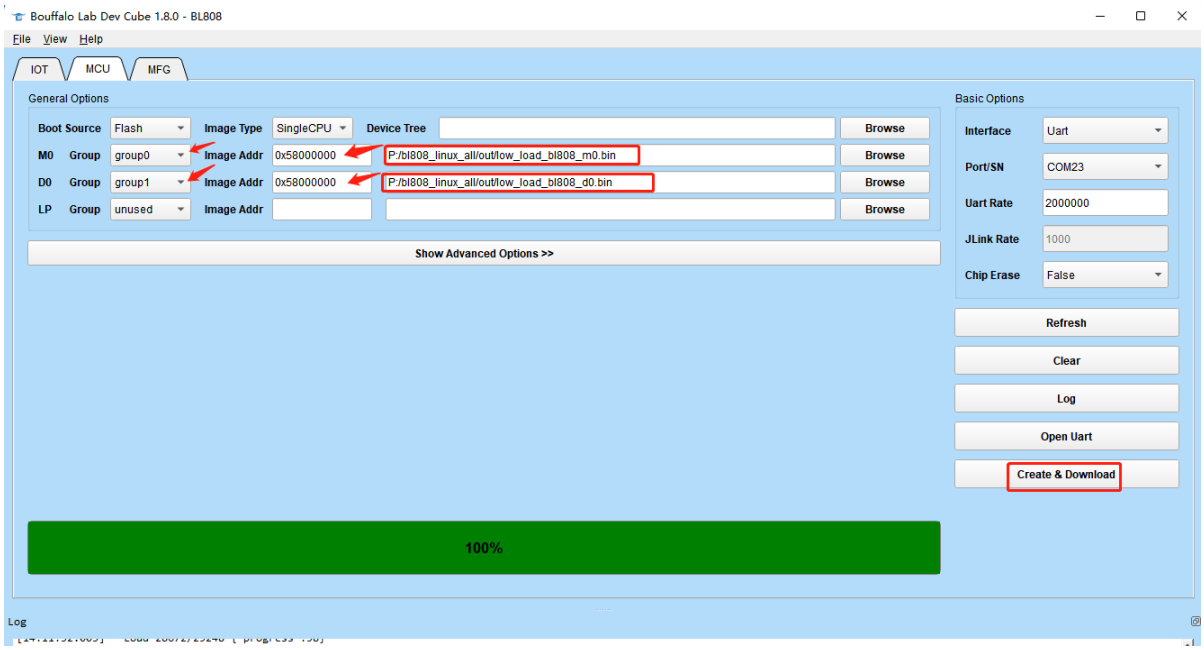
burn

After compiling, you can burn the corresponding file to the BL808 for operation.

Download the BouffaloLab Dev Cube software from https://www.bouffalolab.com/download Download the BouffaloLab Dev Cube burn-in tool software by default.

The burn-in tool supports either Ubuntu or Windows platforms, and the window platform is used as an example.

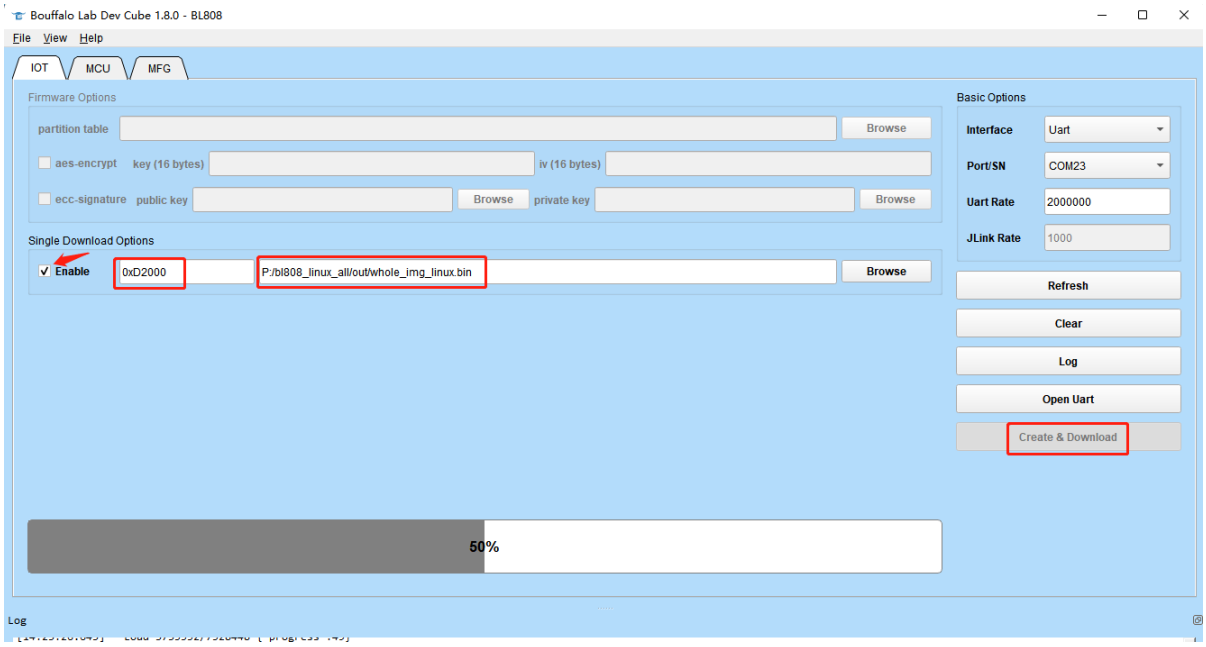
Unzip it and double-click on it to run it. `BLDevCube.exe` Select [BL808] to open the software, click on the [MCU] tab in the upper panel, and set the burn-in settings as shown in the figure below.



Once configured, put the core in boot key, press rst key; then click the Create& Download button to start burn-in mode and press and hold once downloading the two low_load_xxxx.bin

The low_load-related usage and functions will be described in the subsequent development notes.

After burning the low_load firmware, we also need to burn the kernel-related whole img. Click the [IOT] tab bar above to switch to the IOT screen. The burn configuration is shown in the figure.



Click again Create& Download button to begin downloading the starter core. Mirroring to the core.

Once the burn is rst complete, you can click whole_img_linux.bin

The default burn-in port will output logs related to E907 operations (UART0: IO14 TX, IO15 RX); logs for Linux kernel operations and shell terminals will be output via UART3 (IO5 RX, IO8 TX); the terminal is shown below after kernel boot.

```
dynamic memory init success,heap size = 26 Kbyte
C906 start...
kernel 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 Xuantiec910
Platform HART Features : RV64ACDFIMSUVX
Platform Max HARTs     : 1
Current Hart           : 0
Firmware Base          : 0x3eff0000
Firmware Size          :56KB
Runtime SBI Version    : 0.2

MIDELEG : 0x0000000000000222
MEDELEG :
0x0000000000000000b1ff
[    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 0x0000000050000000-0x0000000053ffffffff]
[    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:     RCUrestricting 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: 0xffffffffffffffffff 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
```

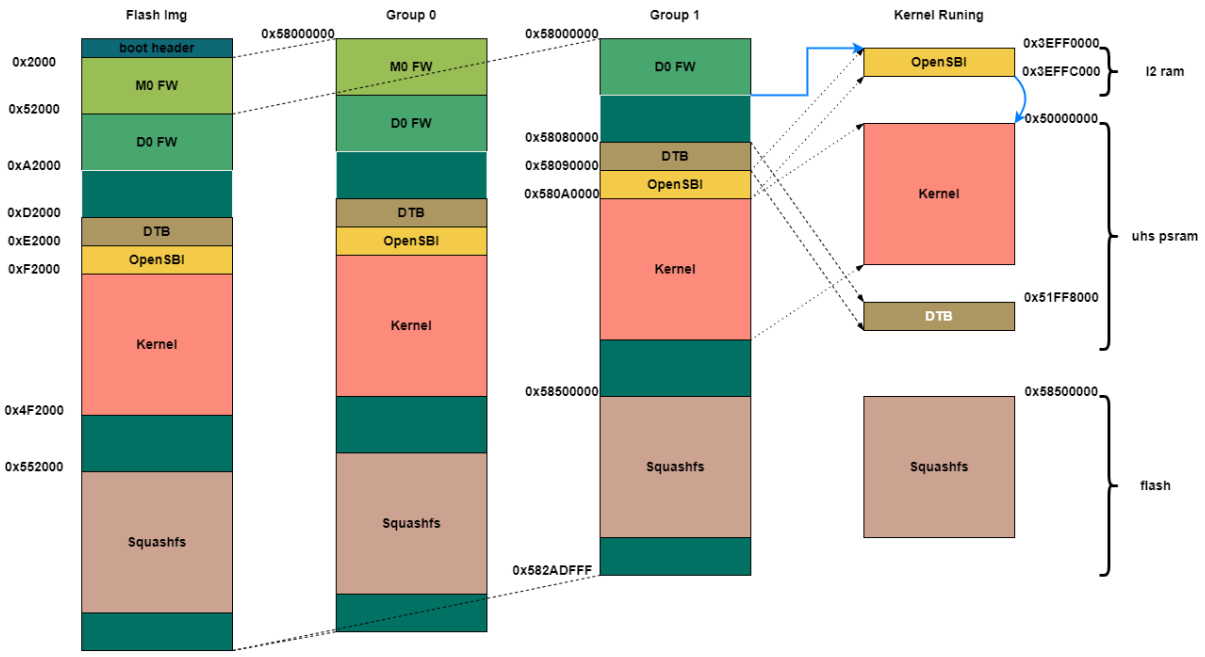
In the terminal, type `root` You can enter the system. Once you are in the system, you can use some standard Linux system commands to operate.

At this point, a basic linux kernel is up and running.

Kernel boot process

Flash Mirror Layout

To understand the entire kernel boot process, let's first understand the layout of each image in flash. The complete flash image is shown on the left, and contains the boot header, M0 FW, D0 FW, DTB, OpenSBI, Kernel, Squashfs.



After the chip is booted, the ROM program on the chip will first complete the hardware-related settings (such as power-up, clock setup, etc.), and after that, it will first start up M0, which is the operating mode of the chip. (GP) Shipping, in `low_load m0` firmware, complete hardware initialization such as psram, and prepare the kernel environment for D0 (C906) to run; D0 will wait until M0 is ready before continuing.

`low_load d0` The firmware will load and decompress OpenSBI, DTB, and Kernel from the corresponding addresses in the flash to the corresponding pSRAM and set the PMP

After loading, it will jump to OpenSBI operation, and after OpenSBI operation, it will boot the Kernel.

LowLoader

Low Load related source code at **bl_mcu_sdk_bl808→examples→low_load** destination

OpenSBI

OpenSBI is currently using v0.6, and the main source code for bl808 is available under **opensbi-0.6-bl808→platform→thead→c910**.

Linux Kernel

The Linux Kernel is currently using a version based on the 5.10.4 kernel + T-head patch, and the main purpose of this project is to add UART-related source code to the drivers for kernel shell interactions. The kernel-related device tree `bl808.dts` file is in the The device tree is defined under the destination, which mainly defines devices such as memory, cpu, serial port, file system, etc.