实验 5 鸿蒙 LiteOS-a 内核移植——串口移植

22920212204396　　黄子安

一、实验目的

1、移植串口，使其可以传送一个字符进而传输字符串

二、实验环境

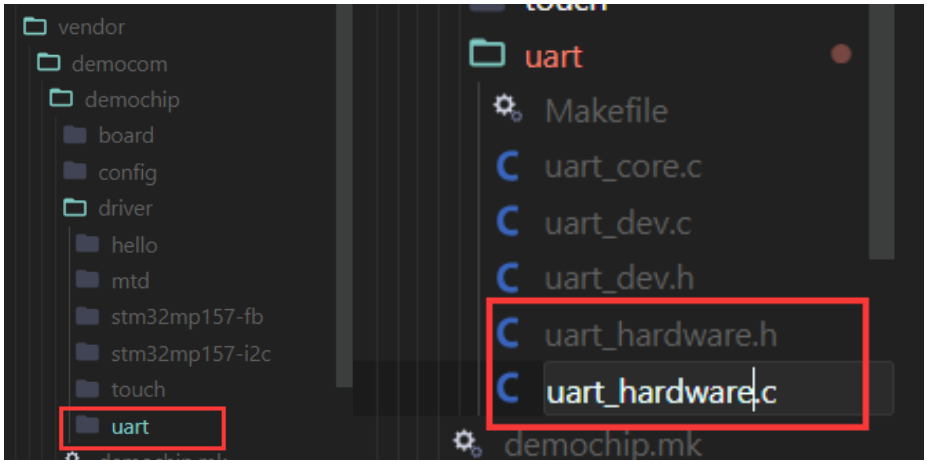
1、物理机 Windows11
2、虚拟机 Ubuntu18.04
3、开发板 Imx6ull

三、实验内容

我们要实现的是 Liteos-a 的最小系统移植，因此不需要初始化串口，u-boot 已经完成了相关的初始化，此外也不需要动态配置串口，固定使用波特率 15200 即可，该操作已经在 u-boot 中完成设置

本次实验视频动作偏快，代码之间的跳转非常多，因此建议在虚拟机中安装一个 vscode 或者在本地物理机使用 ssh 和虚拟机建立连接

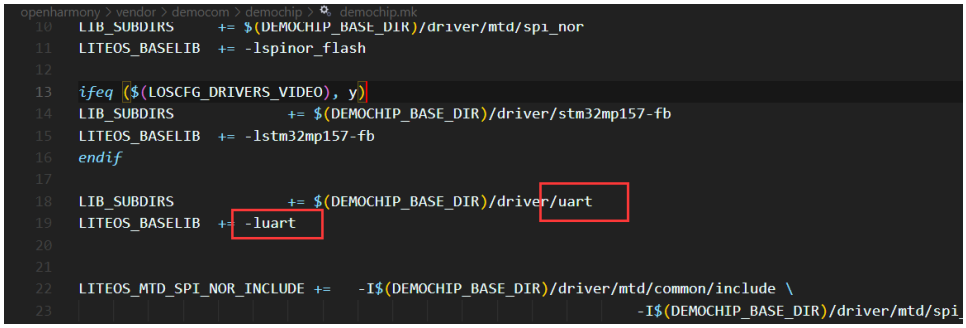
首先为了移植看起来比较舒服一些，将文件名从 stm32mp157-uart 修改成 uart，之后将里面的两个文件名也修改下



查看包含这个文件夹的 makefile 文件，可以看到在 demochip 中有三个

```
book@hza-virtual-machine:~/openharmony$ grep "stm32mp157-uart" * -nr
vendor/democom/demochip/board/Makefile:7:LOCAL_FLAGS :=  -I$(LITEOSTOPDIR)/../../vendor/st/stm32mp157/driver/stm32mp157-uart
vendor/democom/demochip/demochip.mk:18:LIB_SUBDIRS            += $(DEMOCHIP_BASE_DIR)/driver/stm32mp157-uart
vendor/democom/demochip/demochip.mk:19:LITEOS_BASELIB  += -lstm32mp157-uart
vendor/st/stm32mp157/board/Makefile:7:LOCAL_FLAGS :=  -I$(LITEOSTOPDIR)/../../vendor/st/stm32mp157/driver/stm32mp157-uart
vendor/st/stm32mp157/stm32mp157.mk:18:LIB_SUBDIRS            += $(STM32MP157_BASE_DIR)/driver/stm32mp157-uart
vendor/st/stm32mp157/stm32mp157.mk:19:LITEOS_BASELIB  += -lstm32mp157-uart
```

依次修改这 3 个 makefile 文件，这里视频中出现了一个问题，对于第一个 makefile 文件没有修改完文件目录，还是在 st 目录下没改成 democom，不过视频后续在编译内核的时候发现了这个问题



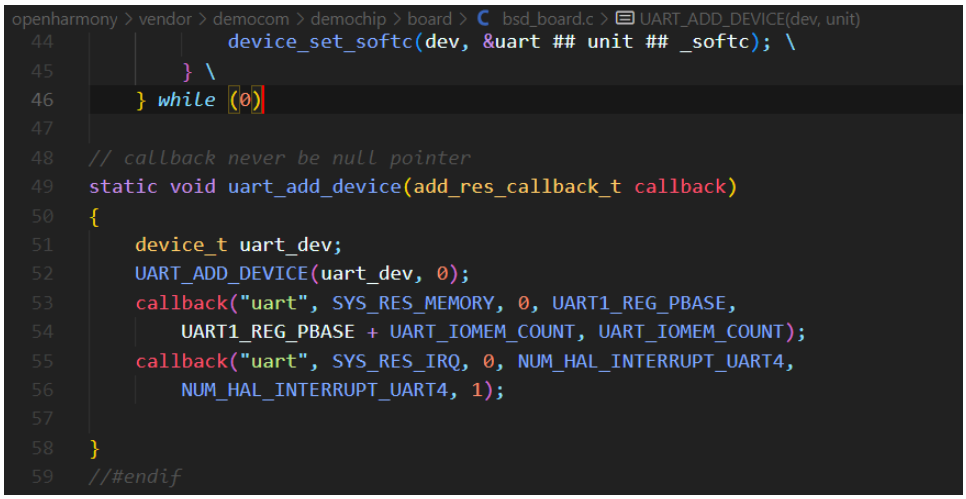
之后修改串口的物理基地址，根据相关文件查询可以知道物理基地址为 0x2020000（这里视频里明显韦东山老师手误了，说的 4 个打了 5 个，不过后续也发现）

0203_4000	0203_7FFF
0203_0000	0203_3FFF
0202_C000	0202_FFFF
0202_8000	0202_BFFF
0202_4000	0202_7FFF
0202_0000	0202_3FFF
0201_C000	0201_FFFF

AIPS-1 (via SPBA)
Glob,Module ENABLE

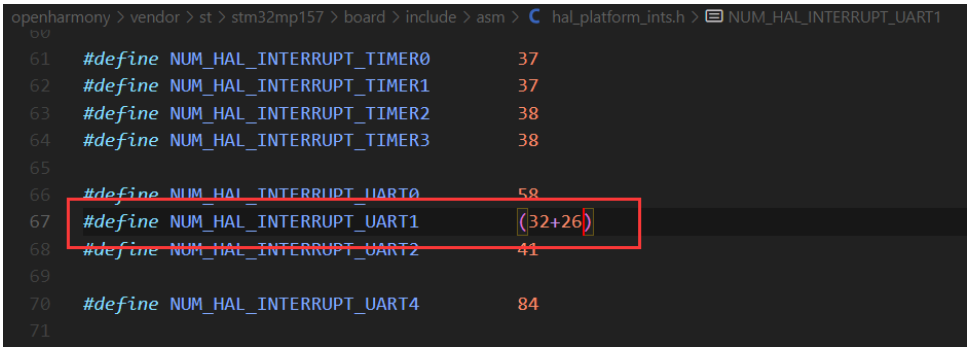
ASRC	16 KB
SAI3	16 KB
SAI2	16 KB
SAI1	16 KB
ESAI	16 KB
UART1	16 KB
Reserved for SDMA internal registers	16 KB

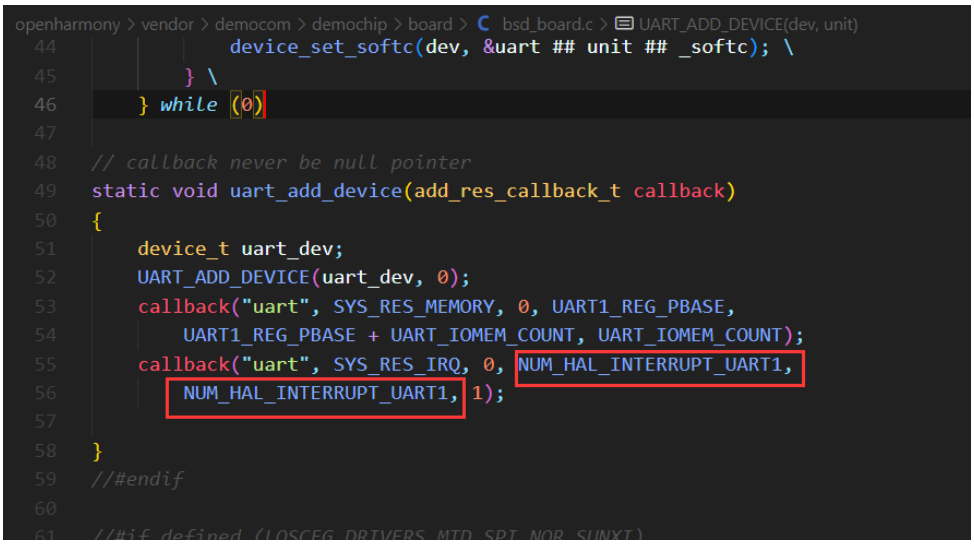
之后修改 uart_add_device 函数，将原来的串口 4 修改为串口 1



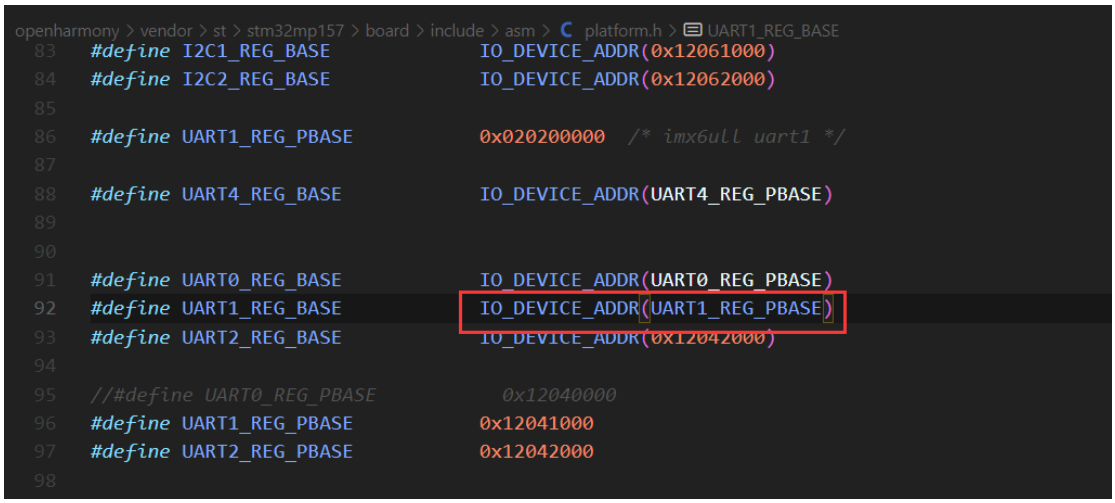
修改串口 1 的中断号，查询文档可以发现这个中断号是 32+26

24	sai3	-	SAI interrupt
25	sai3	-	SAI interrupt
26	uart1	-	UART-1 ORed interrupt
27	uart2	-	UART-2 ORed interrupt
28	uart3	-	UART-3 ORed interrupt
29	uart4	-	UART-4 ORed interrupt
30	uart5	-	UART-5 ORed interrupt

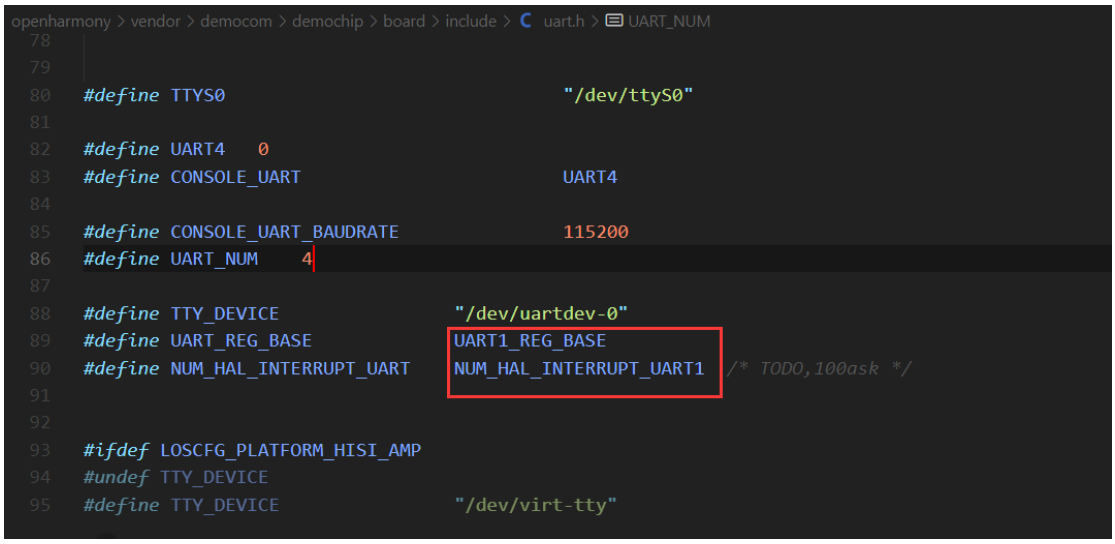




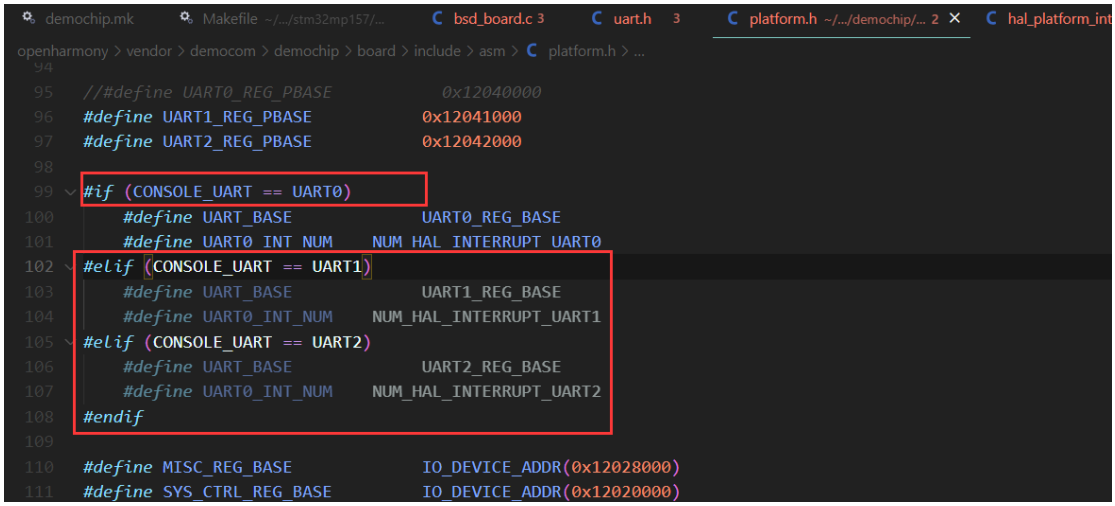
将串口 1 的物理地址映射成虚拟地址



再使用这个宏赋值给串口的基地址



删除该部分代码，之后将串口修改成串口 1



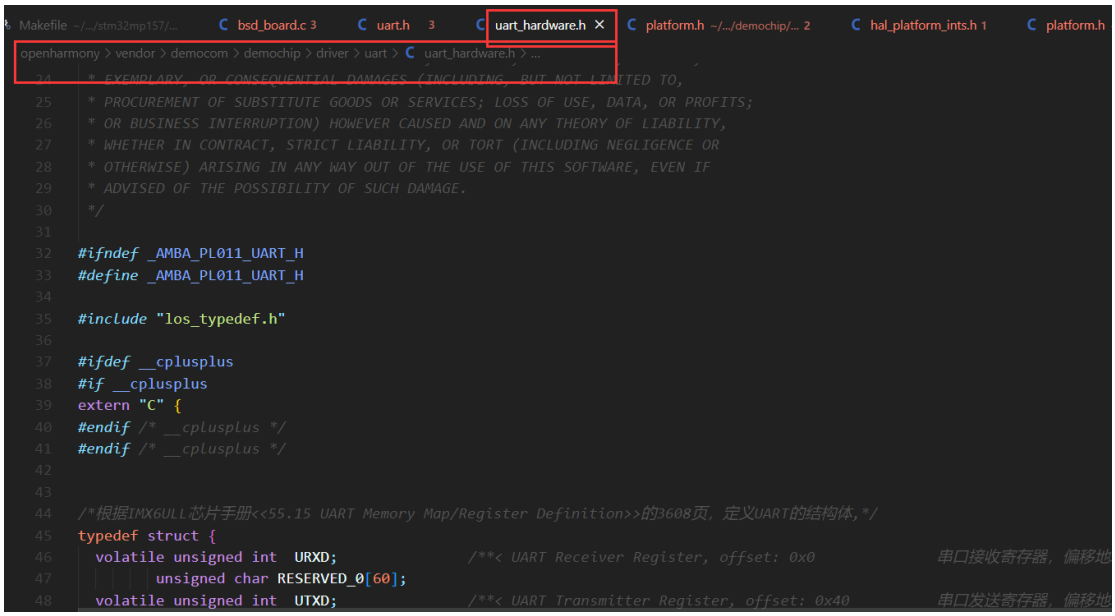
```
//#define UART0_REG_PBASE           0x12040000
#define UART1_REG_PBASE           0x12041000
#define UART2_REG_PBASE           0x12042000

#define UART_BASE             UART0_REG_BASE
#define UART0_INT_NUM     NUM_HAL_INTERRUPT_UART1

#define MISC_REG_BASE             IO_DEVICE_ADDR(0x12028000)
#define SYS_CTRL_REG_BASE         IO_DEVICE_ADDR(0x12020000)
#define CRG_REG_BASE              IO_DEVICE_ADDR(0x12010000)

#define TIMER0_ENABLE             BIT(16)
#define TIMER1_ENABLE             BIT(17)
#define TIMER2_ENABLE             BIT(18)
#define TIMER3_ENABLE             BIT(19)
```

这样子修改后这里串口的基地址就是串口 1 的基地址，串口的中断号就是串口 1 的中断号

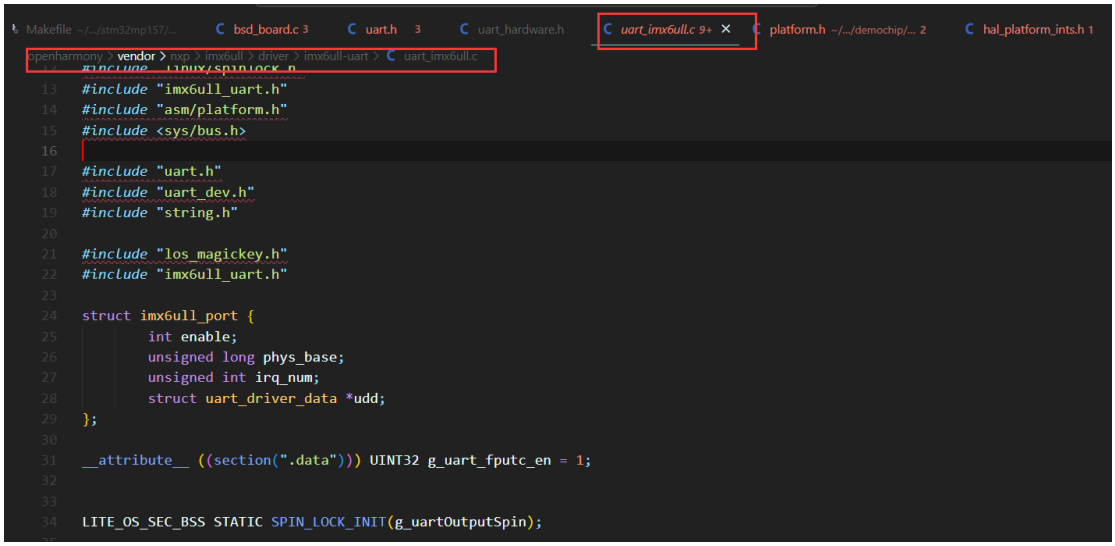
```
#define UART4    0
#define CONSOLE_UART                    UART4

#define CONSOLE_UART_BAUDRATE           115200
#define UART_NUM     4

#define TTY_DEVICE              "/dev/uartdev-0"
#define UART_REG_BASE           UART1_REG_BASE
#define NUM_HAL_INTERRUPT_UART  NUM_HAL_INTERRUPT_UART1  /* TODO,100ask */


#ifdef LOSCFG_PLATFORM_HISI_AMP
#undef TTY_DEVICE
#define TTY_DEVICE              "/dev/virt-tty"
#endif
```

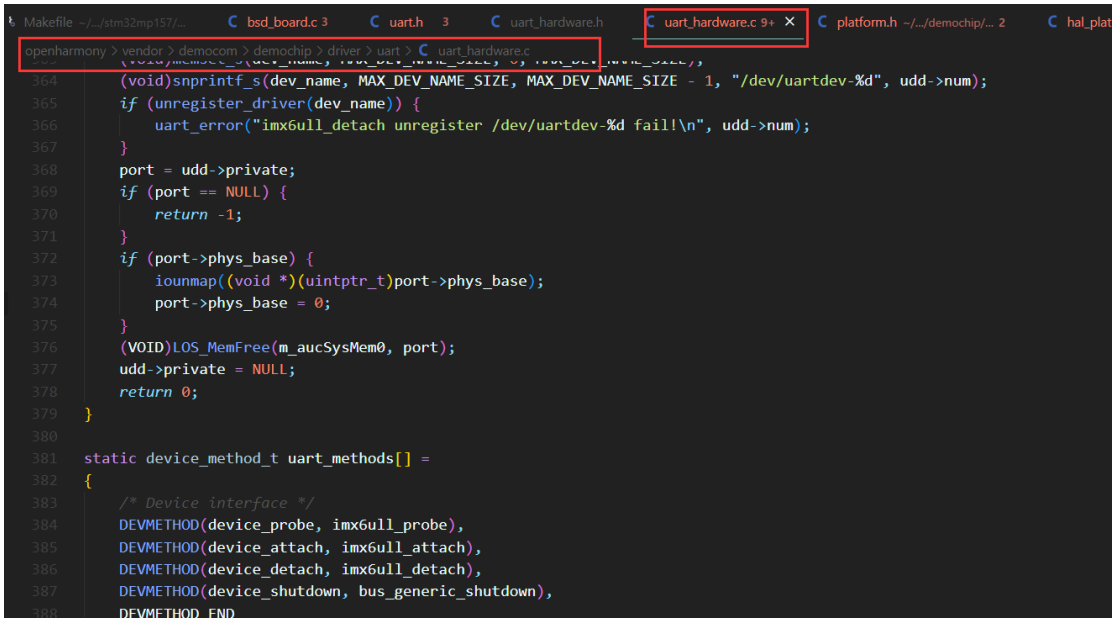
使用 imx6ull_uart.h 文件替换整个 uart_hardware.h 文件

```
#define _AMBA_PL011_UART_H

#include "los_typedef.h"

#ifdef __cplusplus
#if __cplusplus
extern "C" {
#endif /* __cplusplus */
#endif /* __cplusplus */


/* 根据IMX6ULL芯片手册<<55.15 UART Memory Map/Register Definition>>的3608页，定义UART的结构体 */
typedef struct {
    volatile unsigned int  URXD;             /**< UART Receiver Register, offset: 0x0 */
    unsigned char RESERVED_0[60];
    volatile unsigned int  UTXD;             /**< UART Transmitter Register, offset: 0x40 */
    unsigned char RESERVED_1[60];
    volatile unsigned int  UCR1;             /**< UART Control Register 1, offset: 0x80 */
    volatile unsigned int  UCR2;             /**< UART Control Register 2, offset: 0x84 */
    volatile unsigned int  UCR3;             /**< UART Control Register 3, offset: 0x88 */
    volatile unsigned int  UCR4;             /**< UART Control Register 4, offset: 0x8C */
    volatile unsigned int  UFCR;             /**< UART FIFO Control Register, offset: 0x90 */
    volatile unsigned int  USR1;             /**< UART Status Register 1, offset: 0x94 */
    volatile unsigned int  USR2;             /**< UART Status Register 2, offset: 0x98 */
    volatile unsigned int  UESC;             /**< UART Escape Character Register, offset: 0x9C */
```

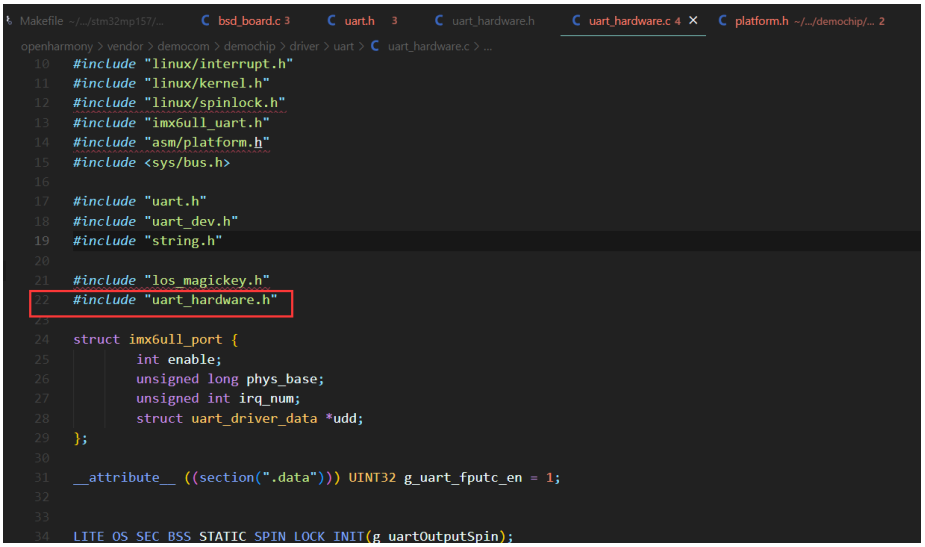


使用 imx6ull_uart.c 文件替换整个 uart_hardware.c 文件



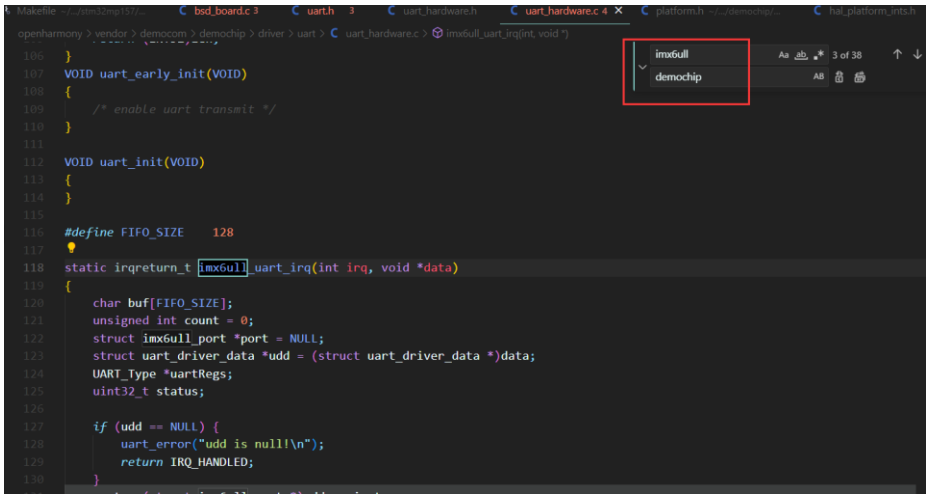


修改第 22 行的包含文件，将其修改成 uart_hadware.h



```
openharmony > vendor > democom > demochip > driver > uart > uart_hardware.c > ...
10  #include "linux/interrupt.h"
11  #include "linux/kernel.h"
12  #include "linux/spinlock.h"
13  #include "imx6ull_uart.h"
14  #include "asm/platform.h"
15  #include <sys/bus.h>
16
17  #include "uart.h"
18  #include "uart_dev.h"
19  #include "string.h"
20
21  #include "los_magickey.h"
22  #include "uart_hardware.h"
23
24  struct imx6ull_port {
25          int enable;
26          unsigned long phys_base;
27          unsigned int irq_num;
28          struct uart_driver_data *udd;
29  };
30
31  __attribute__ ((section(".data"))) UINT32 g_uart_fputc_en = 1;
32
33
34  LITE_OS_SEC_BSS_STATIC SPIN_LOCK_INIT(g_uartOutputSpin);
```

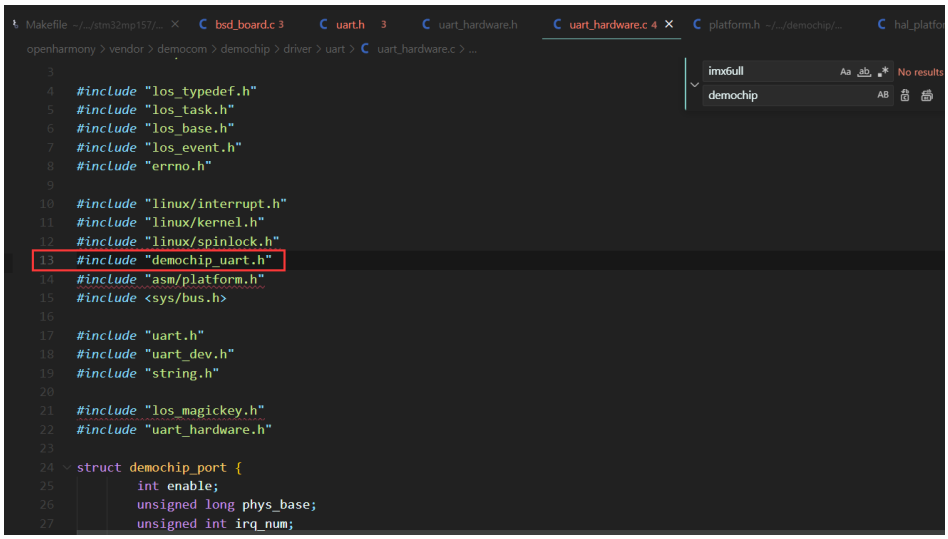
使用全局替换的方式将 imx6ull 替换成 demochip



```
openharmony > vendor > democom > demochip > driver > uart > uart_hardware.c > imx6ull_uart_irq(int, void *)
106 }
107 VOID uart_early_init(VOID)
108 {
109     /* enable uart transmit */
110 }
111
112 VOID uart_init(VOID)
113 {
114 }
115
116 #define FIFO_SIZE    128
117
118 static irqreturn_t imx6ull_uart_irq(int irq, void *data)
119 {
120     char buf[FIFO_SIZE];
121     unsigned int count = 0;
122     struct imx6ull_port *port = NULL;
123     struct uart_driver_data *udd = (struct uart_driver_data *)data;
124     UART_Type *uartRegs;
125     uint32_t status;
126
127     if (udd == NULL) {
128         uart_error("udd is null!\n");
129         return IRQ_HANDLED;
130     }
```

imx6ull — 3 of 38
demochip

删掉这个头文件包含



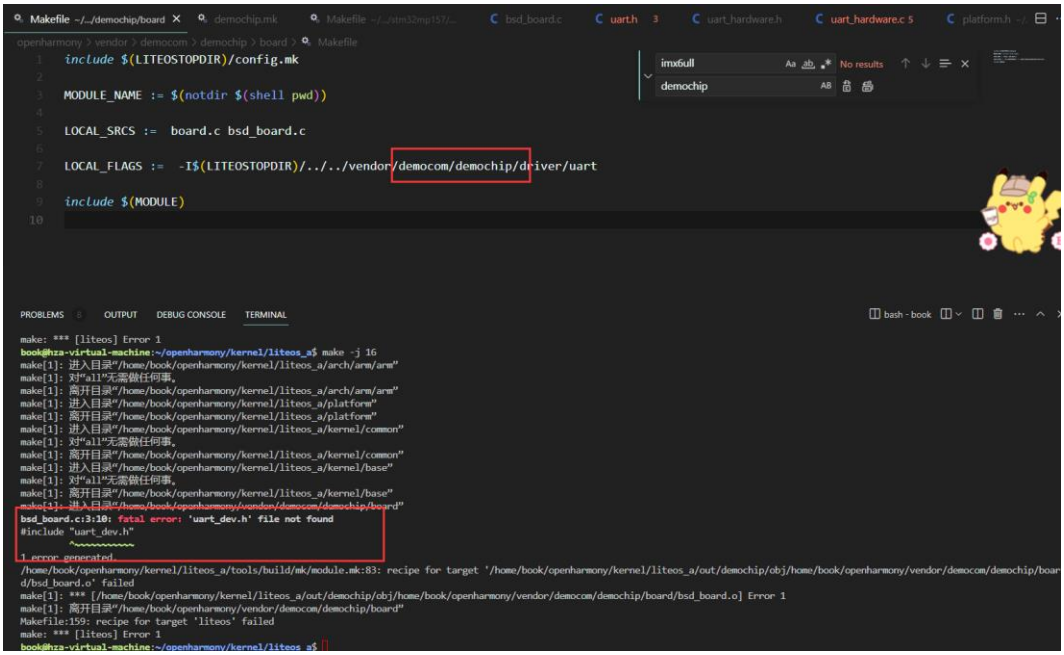
```
openharmony > vendor > democom > demochip > driver > uart > uart_hardware.c > ...
4   #include "los_typedef.h"
5   #include "los_task.h"
6   #include "los_base.h"
7   #include "los_event.h"
8   #include "errno.h"
9
10  #include "linux/interrupt.h"
11  #include "linux/kernel.h"
12  #include "linux/spinlock.h"
13  #include "demochip_uart.h"
14  #include "asm/platform.h"
15  #include <sys/bus.h>
16
17  #include "uart.h"
18  #include "uart_dev.h"
19  #include "string.h"
20
21  #include "los_magickey.h"
22  #include "uart_hardware.h"
23
24  struct demochip_port {
25          int enable;
26          unsigned long phys_base;
27          unsigned int irq_num;
```

imx6ull — No results
demochip

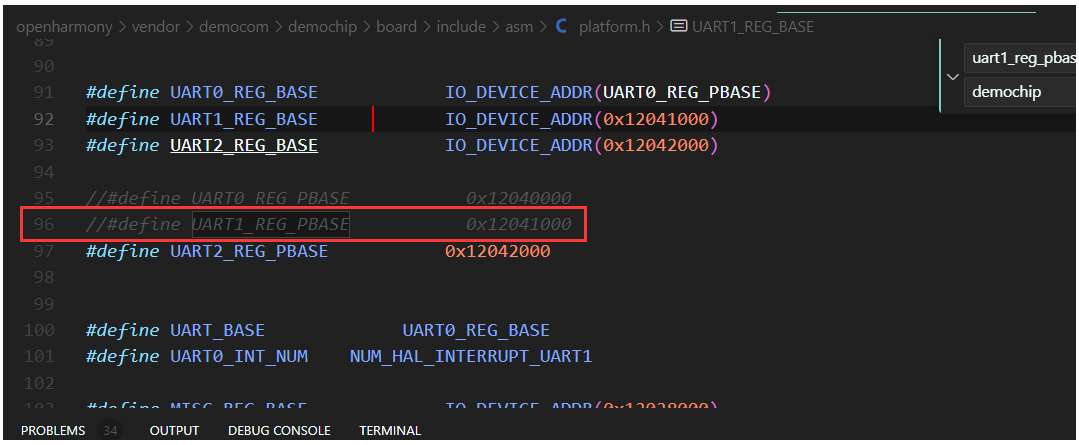
之后将对应的配置文件复制到指定目录，之后执行 make clean 和 make -j 16 命令编译内核

```
book@hza-virtual-machine:~/openharmony$ cd kernel/liteos_a
book@hza-virtual-machine:~/openharmony/kernel/liteos_a$ cp tools/build/config/debug/demochip_clang.config  .config
book@hza-virtual-machine:~/openharmony/kernel/liteos_a$
```

这里会出现第一次编译出错，原因就是前面提到的 makefile 的文件没有修改全

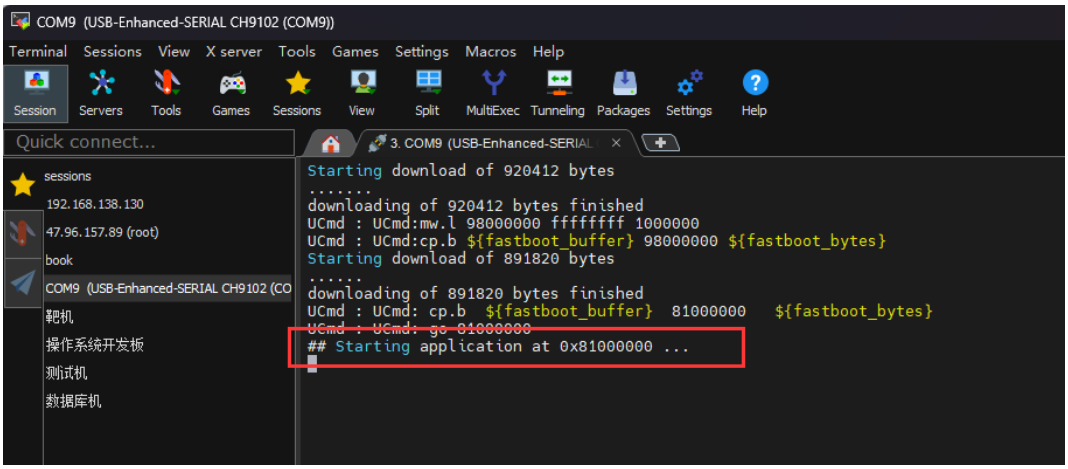


再次编译内核，出现另一个报错，发现是宏重复定义了，对其进行修改

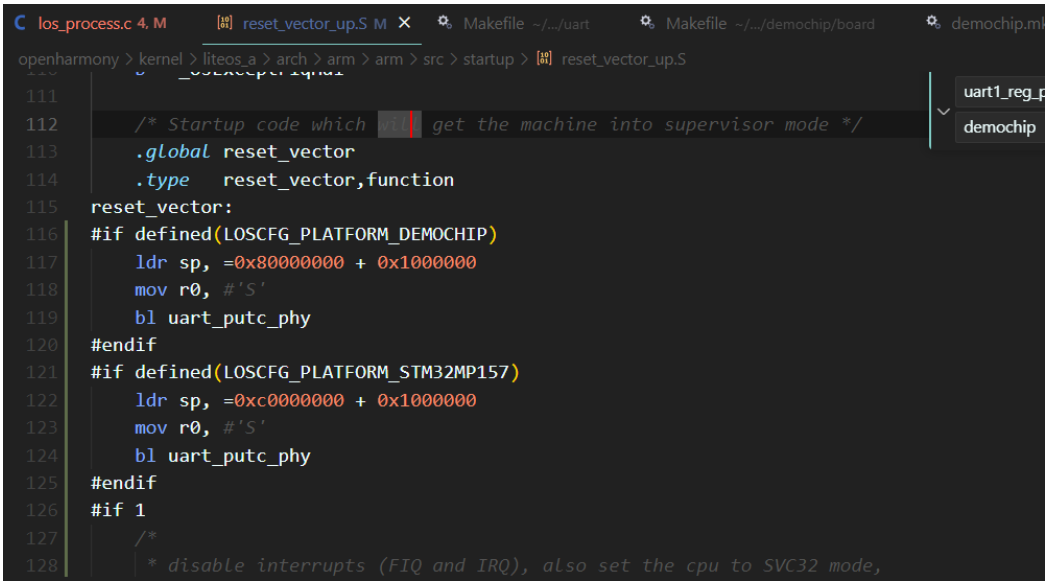


最后再次编译内核发现没有报错，链接成功生成 liteos.bin 文件，将其复制到烧写工具的 files 文件中之后烧写到开发板中

运行后发现内核没有启动，根据视频里的方法进行调试

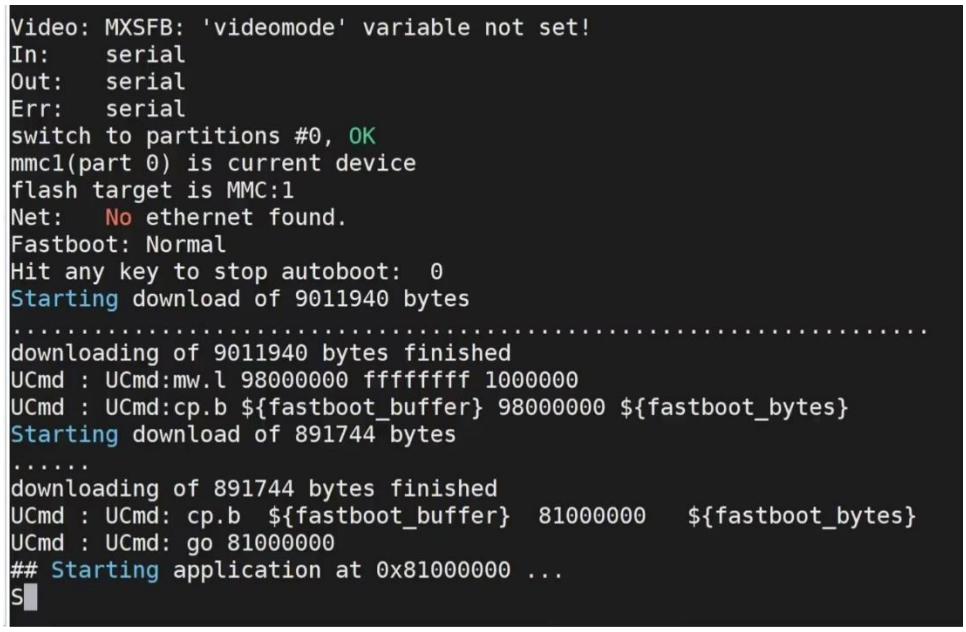


在内核启动前的汇编程序中增加一段调试代码，如果编译平台是 DEMOCHIP 的话就执行下 uart_putc_phy 函数，其作用是输出一个字符

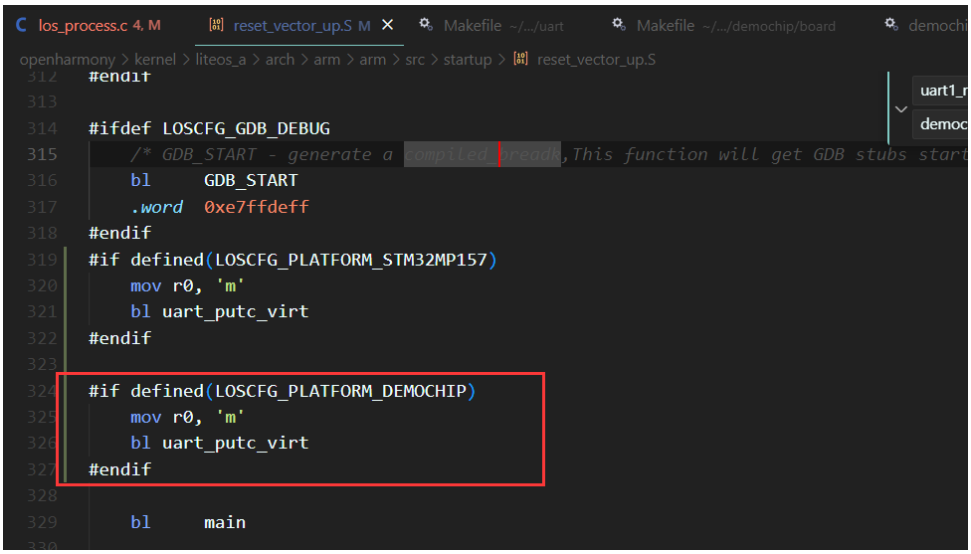


这里调试的时候需要使用物理地址，因为这个时候内核刚启动，MMU 还没有启动，所以还没有虚拟地址的概念，因此需要使用物理地址

运行后发现输出一个 S，说明该部分没有问题



之后再根据视频继续调试



再实现一下这个 uart_putc_virt,这个时候 MMU 已经启动了因此可以使用虚拟地址，这里韦东山老师发现多输了一个 0，同步进行修改即可

```
43  void uart_putc_virt(char c)
44  {
45      UART_Type *uartRegs = (UART_Type *)UART_REG_BASE;
46
47      while (!((uartRegs->USR2) & (1<<3))); /*等待上个字节发送完毕*/
48      uartRegs->UTXD = (unsigned char)c;
49  }
50
```

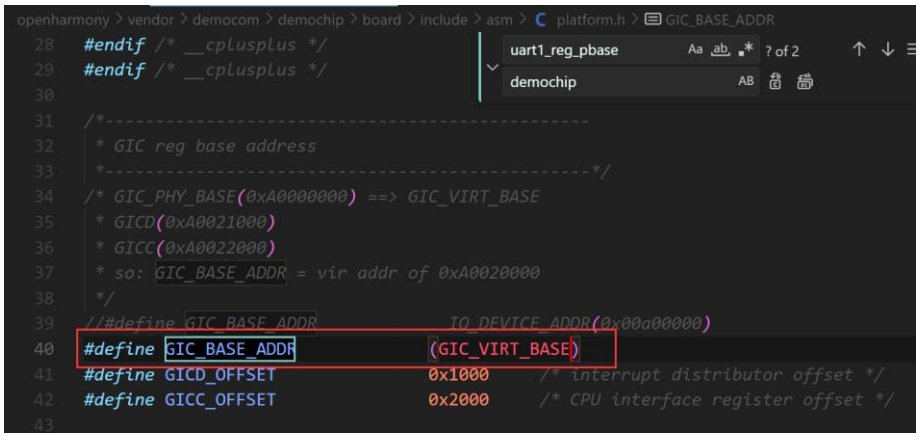
```
74  #define GPIO9_REG_BASE              (GPIO_REG_BASE + 0x9000)
75
76  #define SPI0_REG_BASE               IO_DEVICE_ADDR(0x12070000)
77  #define SPI1_REG_BASE               IO_DEVICE_ADDR(0x12071000)
78
79  #define I2C0_REG_PBASE              0x12060000
80  #define I2C1_REG_PBASE              0x12061000
81  #define I2C2_REG_PBASE              0x12062000
82  #define I2C0_REG_BASE               IO_DEVICE_ADDR(0x12060000)
83  #define I2C1_REG_BASE               IO_DEVICE_ADDR(0x12061000)
84  #define I2C2_REG_BASE               IO_DEVICE_ADDR(0x12062000)
85
86  #define UART1_REG_PBASE             0x020200000 /* imx6ull uart1 */
87
88  #define UART4_REG_BASE              IO_DEVICE_ADDR(UART4_REG_PBASE)
89
```

```
74  #define GPIO9_REG_BASE              (GPIO_REG_BASE + 0x9000)
75
76  #define SPI0_REG_BASE               IO_DEVICE_ADDR(0x12070000)
77  #define SPI1_REG_BASE               IO_DEVICE_ADDR(0x12071000)
78
79  #define I2C0_REG_PBASE              0x12060000
80  #define I2C1_REG_PBASE              0x12061000
81  #define I2C2_REG_PBASE              0x12062000
82  #define I2C0_REG_BASE               IO_DEVICE_ADDR(0x12060000)
83  #define I2C1_REG_BASE               IO_DEVICE_ADDR(0x12061000)
84  #define I2C2_REG_BASE               IO_DEVICE_ADDR(0x12062000)
85
86  #define UART1_REG_PBASE             0x02020000 /* imx6ull uart1 */
87
88  #define UART4_REG_BASE              IO_DEVICE_ADDR(UART4_REG_PBASE)
89
```

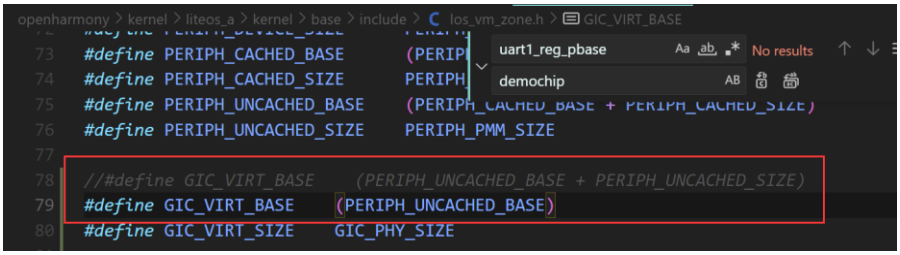
这里运行后出现了 Sm 和 main 的提示，说明串口可以使用，根据视频进行研究可以发现是中断控制器的地址没有进行修改

```
Out:   serial
Err:   serial
switch to partitions #0, OK
mmc1(part 0) is current device
flash target is MMC:1
Net:   No ethernet found.
Fastboot: Normal
Hit any key to stop autoboot:  0
Starting download of 9011940 bytes
.................................................................
downloading of 9011940 bytes finished
UCmd : UCmd:mw.l 98000000 ffffffff 1000000
UCmd : UCmd:cp.b ${fastboot_buffer} 98000000 ${fastboot_bytes}
Starting download of 891744 bytes
......
downloading of 891744 bytes finished
UCmd : UCmd: cp.b  ${fastboot_buffer}  81000000   ${fastboot_bytes
UCmd : UCmd: go 81000000
## Starting application at 0x81000000 ...
Sm
*****************Main*****************
```

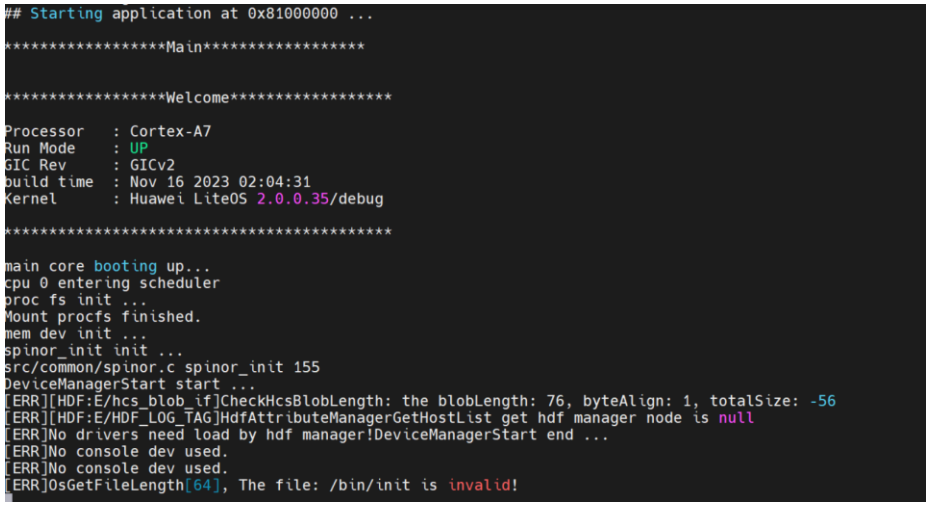
对中断控制器相关的内容进行修改，删除掉这里的加 0x20000 偏移



删除掉该宏后面加上的东西



之后再重新编译与烧写内核，运行后可以看到成功输出了信息，后面文件系统没有移植，因此相关部分没有运行成功，会在后续实验中完成



四、实验心得

本次实验和之前代码阅读实验共享中断有些类似，但是更加全面了解了串口有关的驱动程序，实验如果直接在虚拟机中操作文件互相跳转过于麻烦，因此学会了配置 vscode 的 ssh 连接，方便了开发虚拟机中的文件

此外还学到了一些调试的方法，之前阅读代码的时候最不敢碰的就是汇编代码，这里韦东山老师使用了汇编调 C 函数的方法展示了一种调试的手段

另外本次实验也进一步解了操作系统对于 IO 的管理和 IO 的内存映射关系，通过代码的角度对这些之前在计组里学到的东西有些新的认知。