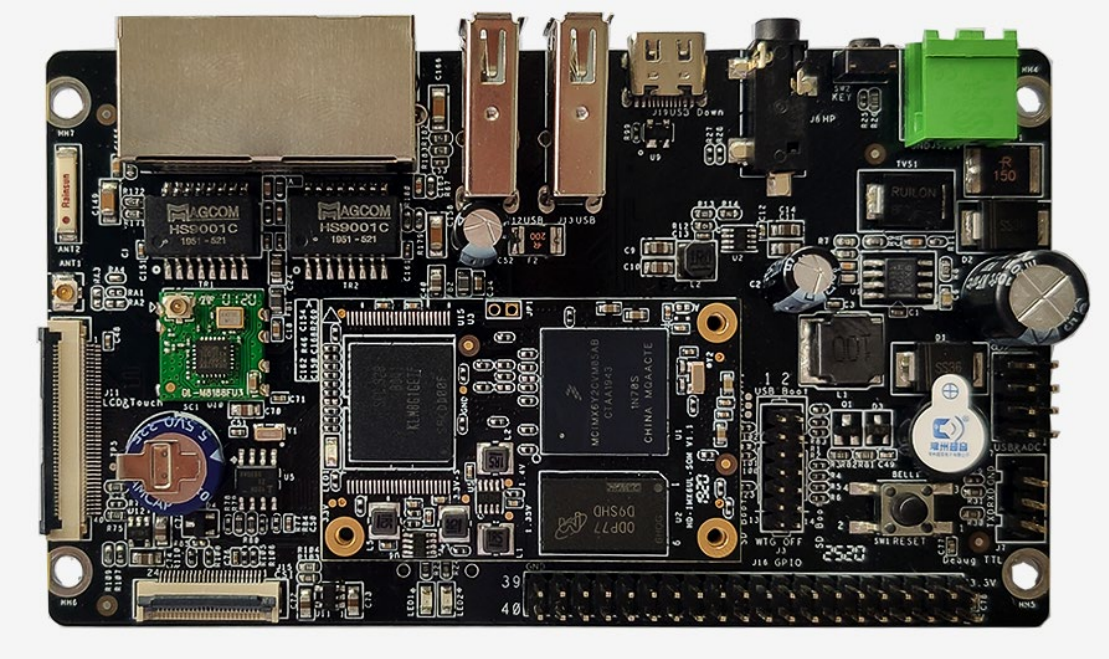
武汉万象奥科电子有限公司

用户手册

HD-IMX6ULL-MB 开发板

v1.0.4



目录

1 产品介绍

1.1 产品简介

HD-IMX6ULL-MB 系列开发板是武汉芯路遥科技有限公司与武汉万象奥科电子有限公司合作推出的一款开发板。芯路遥团队拥有专业耐心的技术支持团队，帮助您快速入门 i.MX6ULL 开发。万象奥科拥有强大的博士团队，可针对企业客户提供定制化项目开发。

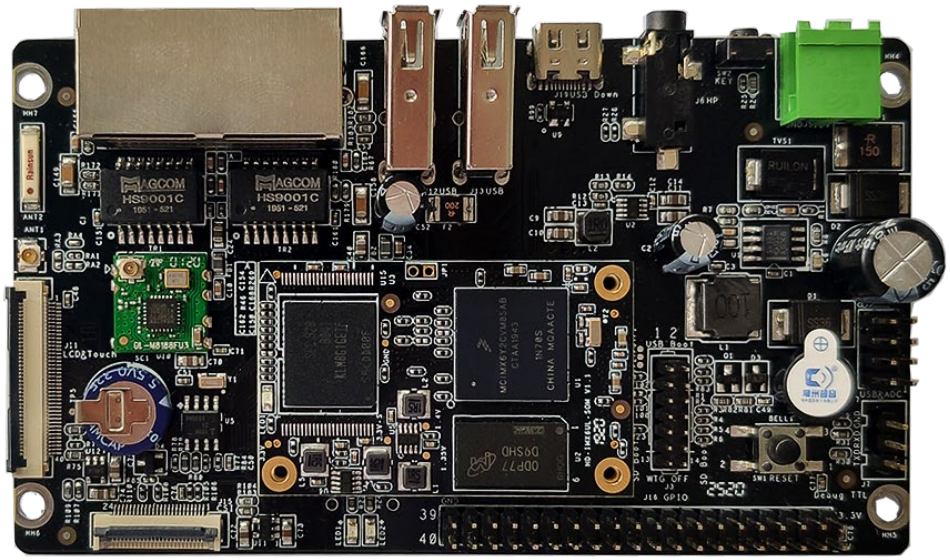
此开发板基于 NXP iMX6ULL 系列 Cortex-A7 高性能处理器设计，支持 RS232、RS485、双路 CAN 总线接口、双路以太网接口、HDMI 接口、摄像头接口等，适用于快速开发一系列具有创新性的产品如人机界面工业 4.0 扫描仪、车载终端以及便携式医疗设备。

HD-IMX6ULL-MB 开发板硬件资源参数如下：

产品名称	HD-IMX6ULL-MB 开发板
操作系统	Linux
处理器	i.MX6ULL Cortex-A7
主频	528MHz
内存	512MB
Flash	8G eMMC
摄像头	1 路，CSI
Wi-Fi	板载，2.4G
LCD 最高分辨率	1366 * 768
触摸屏	支持，电容屏接口
音频接口	1 路输出，无需声卡 ，支持外扩声卡
USB	1 路 USB OTG，3 路 USB 2.0
串口	5 路（1 路调试）
CAN	2 路
以太网	2 路
SD 卡接口	1 路
I2C	1 路
RTC	支持
机械尺寸	48.1mm * 30mm

1.2 硬件介绍

1.2.1 HD-IMX6ULL-MB 资源介绍



HD-IMX6ULL-MB 开发板机械尺寸为 117mm*67mm。核心板和底板固定处进行防呆设计，避免用户插错损坏开发板；核心板放置了固定孔，保证开发板在震动环境的稳定性。

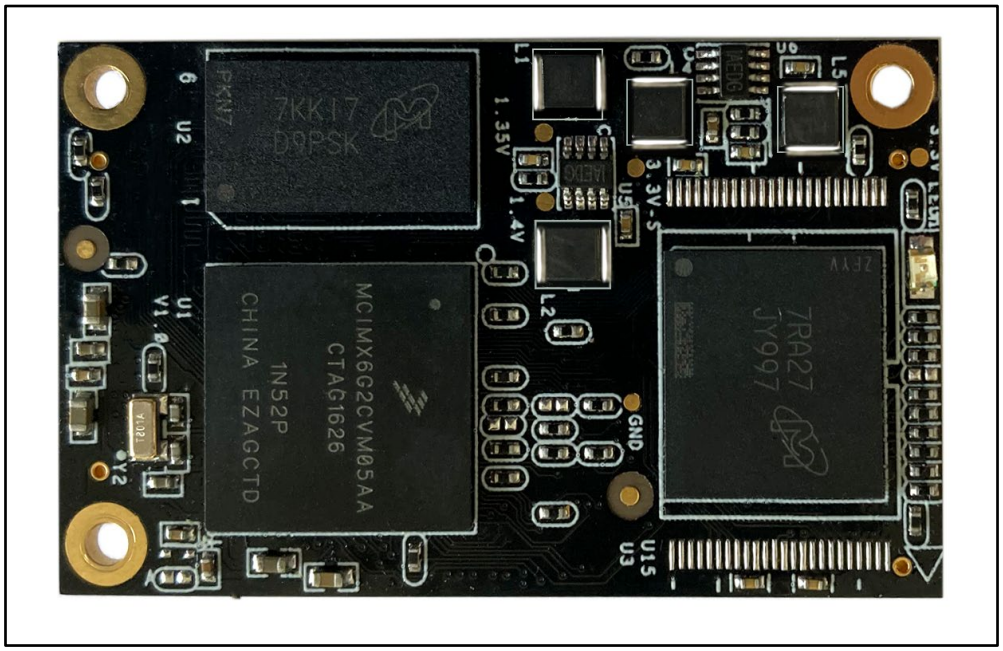
HD-IMX6ULL-MB 开发板板载资源如下：

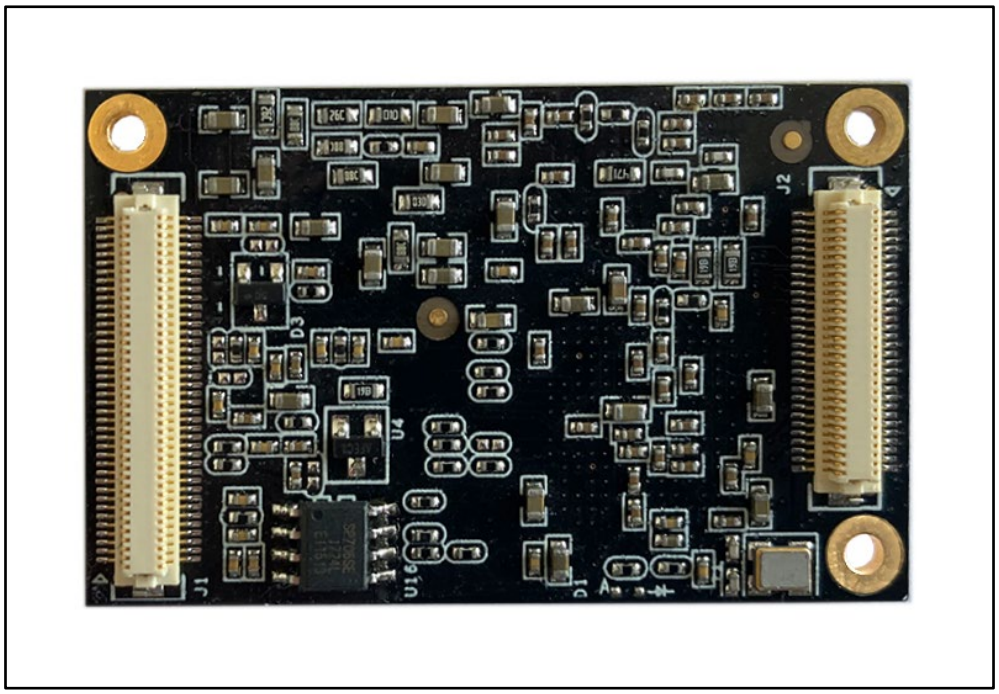
- 电源插座
- 电源指示灯
- RUN 指示灯
- 一个 Debug 串口
- 一个 USB DEVICE 接口
- 2 个 USB HOST 接口
- 2 路百兆以太网(带隔离)
- 1 个 RGB LCD 屏幕接口
- 1 个 CAMERA 摄像头接口
- 一路音频输出接口
- 1 个 RTC 时钟模块
- 1 个蜂鸣器

- 1 个 RESET 复位按键
- 1 个用户按键
- 板载 USB Wifi

针对板上外设，都给出了功能齐全的驱动程序和测试 DEMO，方便用户直接使用。

1.2.2 核心板资源简介





HD-IMX6ULL-MB 开发板使用核心板型号为 HD-IMX6ULL-512F8GL。该核心板机械尺寸为 48.1mm*30mm，采用板对板连接器，可以应对各种复杂环境。

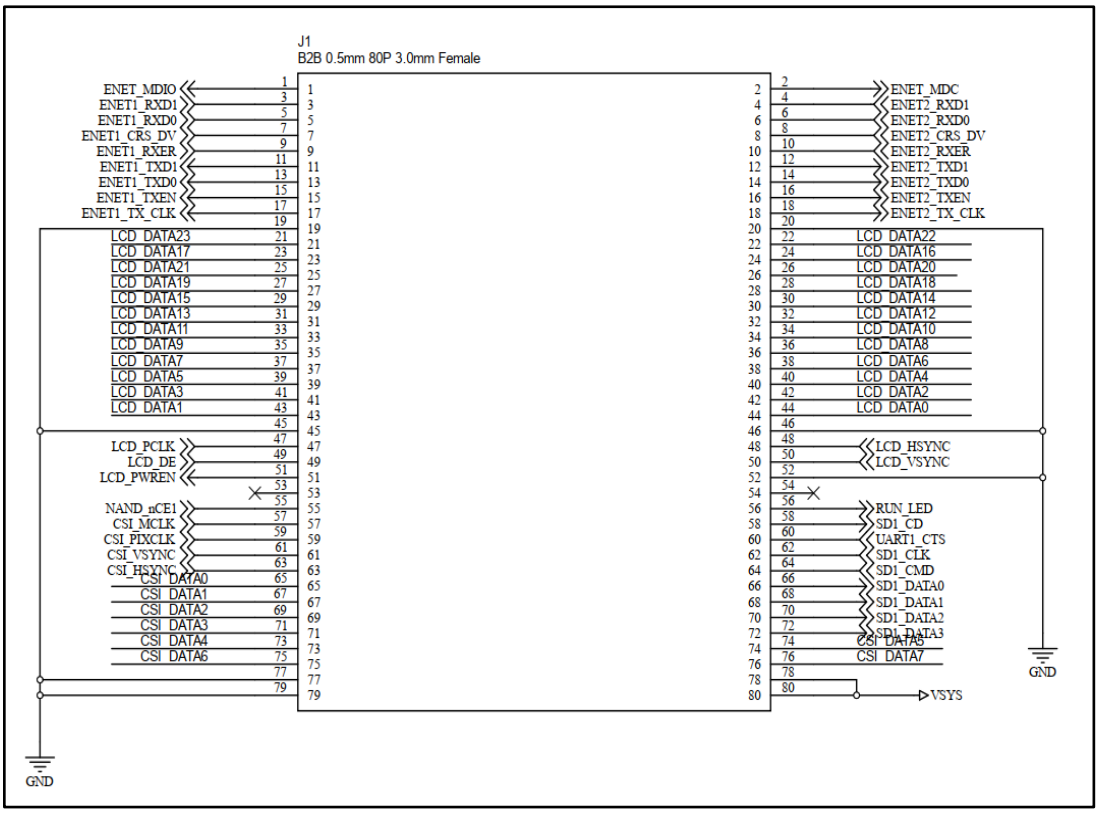
核心板 CPU 使用 MCIMX6Y2CVM05AB(工业级，528M)，DDR3L 内存容量为 512MB， eMMC 容量为 8GB。

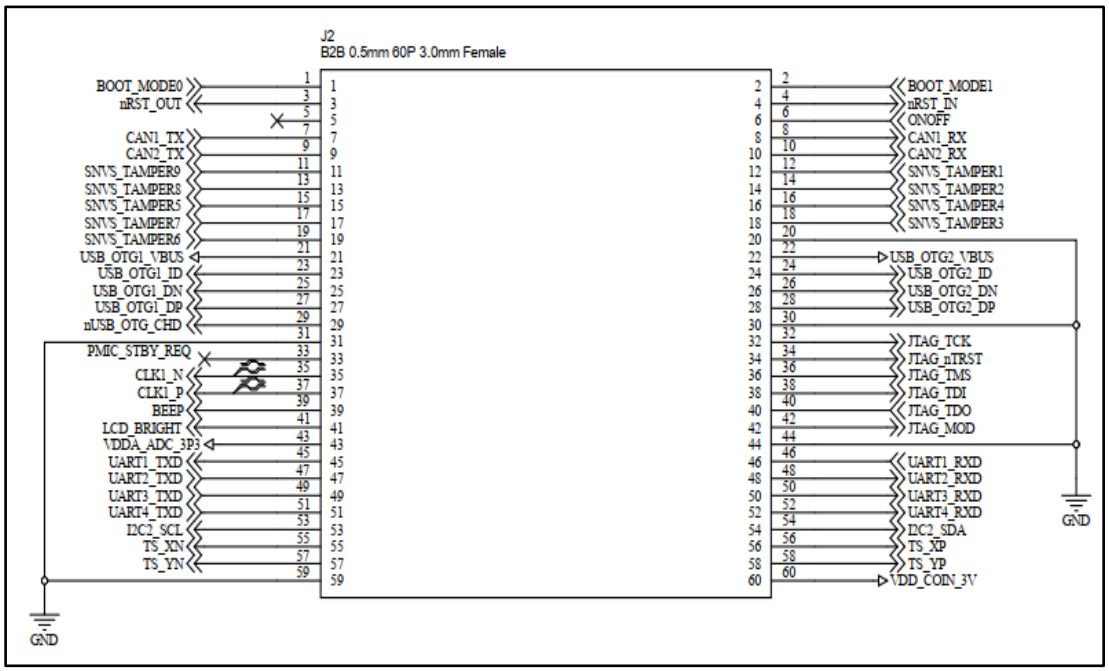
对于工业客户，可提供纯工业级版本。

1.3 底板原理图分析

1.3.1 核心板接口

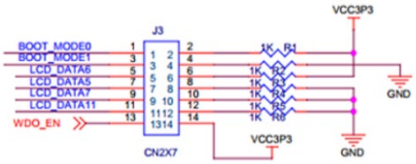
J1、J2 是底板上的转接板接口，J1 为 2*40PIN 的板对板母座，J2 为 2*30PIN 的板对板母座，这些引脚引出 IO 以及电源等。





1.3.2 启动选择

J3 用来选择启动方式。下图描述了开发板具体启动模式：



序号 \ 模式	eMMC 启动 NAND 启动	SD 卡启动	升级 eMMC	升级 SD 卡
1 2				
3 4				
5 6				
7 8				
9 10				
11 12				

序号 \ WDG	使能 WDG	关闭 WDG
13 14		

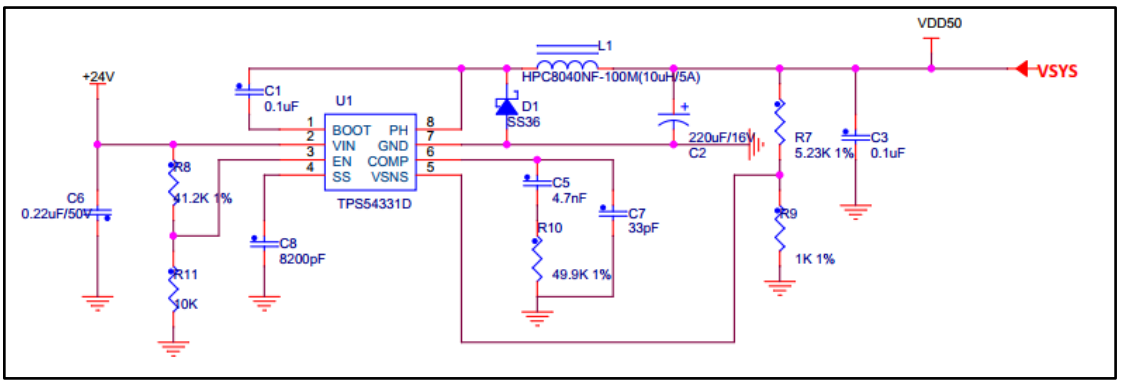
：跳线帽连接

：跳线帽断开

注：由于核心板上默认关闭了 WDG 功能。所以 WDG 功能无法使用跳线帽开启，如需使用 WDG 功能可与我们联系。

1.3.3 电源

HD-IMX6ULL-MB 使用 7V~28V 电压供电，电源输入电压经过板载 TPS54331 转换为 5V 供核心板使用，这个芯片最高输入电压为 28V，最大输出电流为 3A。



1.3.4 SD 卡电路

SD 卡连接至 SD1 控制器,在 linux 系统中会被识别 mmcblk0。

nCD_SD 为 SD 插入检测引脚，有卡插入为低电平，连接至 GPIO1_IO19。

SDCMD 连接至 MX6UL_PAD_SD1_CMD__USDHC1_CMD。

SDCLK 连接至 MX6UL_PAD_SD1_CLK__USDHC1_CLK。

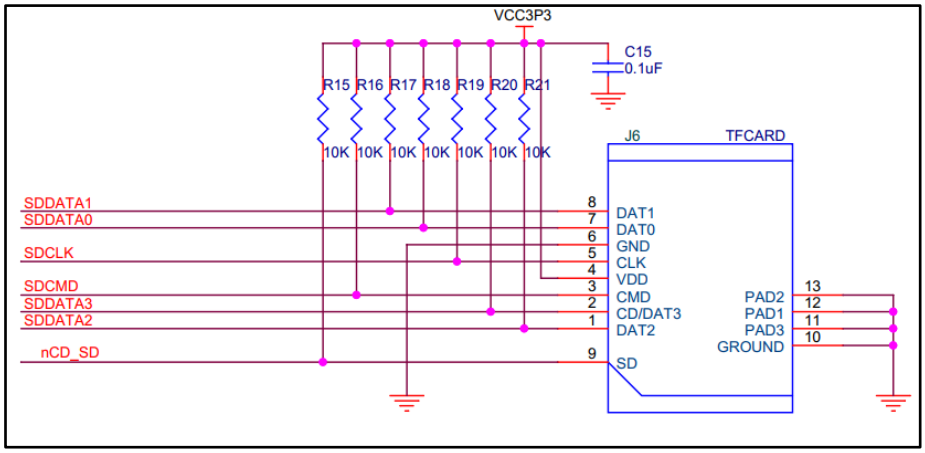
SDDATA0 连接至 MX6UL_PAD_SD1_DATA0__USDHC1_DATA0。

SDDATA1 连接至 MX6UL_PAD_SD1_DATA1__USDHC1_DATA1。

SDDATA2 连接至 MX6UL_PAD_SD1_DATA2__USDHC1_DATA2。

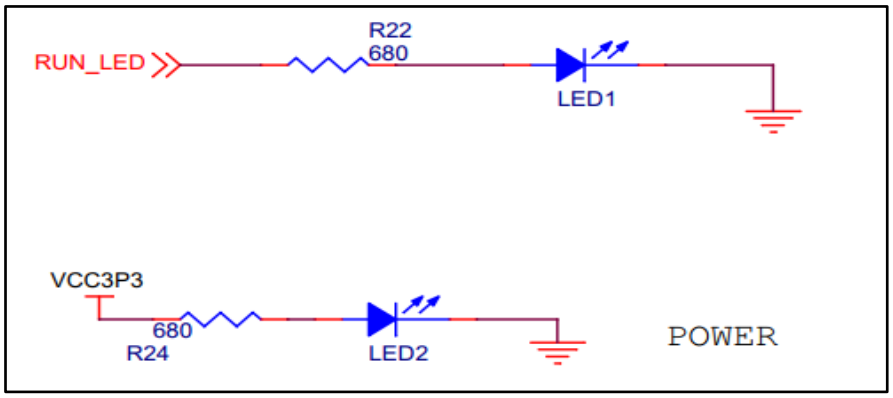
SDDATA3 连接至 MX6UL_PAD_SD1_DATA3__USDHC1_DATA3。

具体引脚可查看”HD-IMX6ULL-512F8G 引脚定义.xlsx”文件。



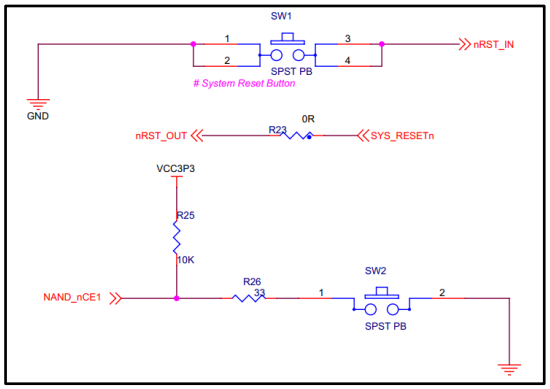
1.3.5 LED

开发板板载 2 个 LED 灯，POWER(LED2)为电源指示灯，RUN(LED1)为心跳灯。RUN 作为心跳灯，连接至 GPIO4_IO16，低电平点亮。



1.3.6 按键

RESET 为复位按键，当按键按下后，开发板复位。S4 为用户按键，连接至 SNVS_TAMPER9,默认高电平，按下为低电平。

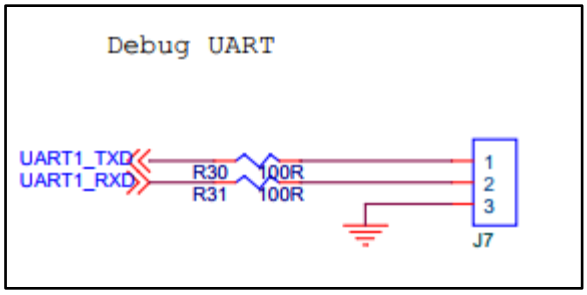


1.3.7 Debug 串口

串口 1 为调试串口。用户使用 USB 转 TTL 模块同开发板进行通信。

UART1_TXD 连接至 MX6UL_PAD_UART1_TX_DATA__UART1_DCE_TX

UART1_RXD 连接至 MX6UL_PAD_UART1_RX_DATA__UART1_DCE_RX



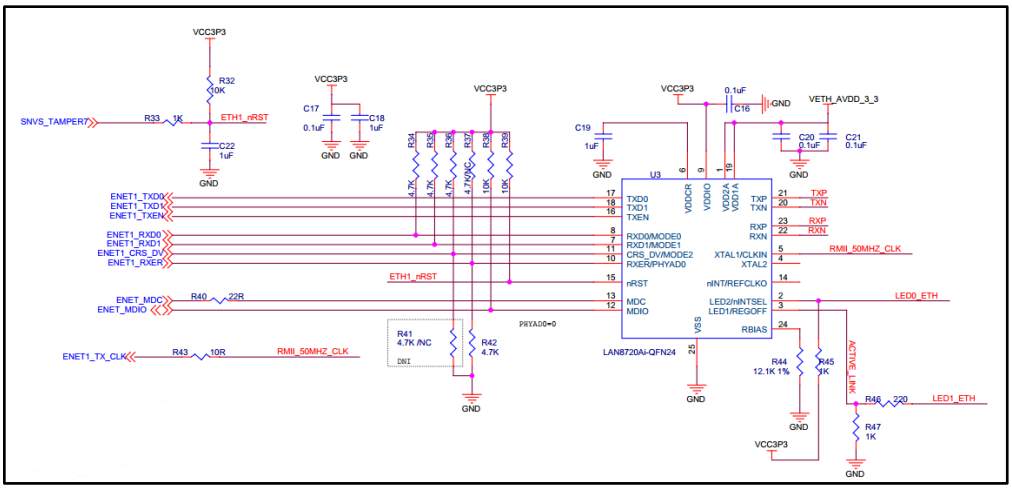
1.3.8 以太网

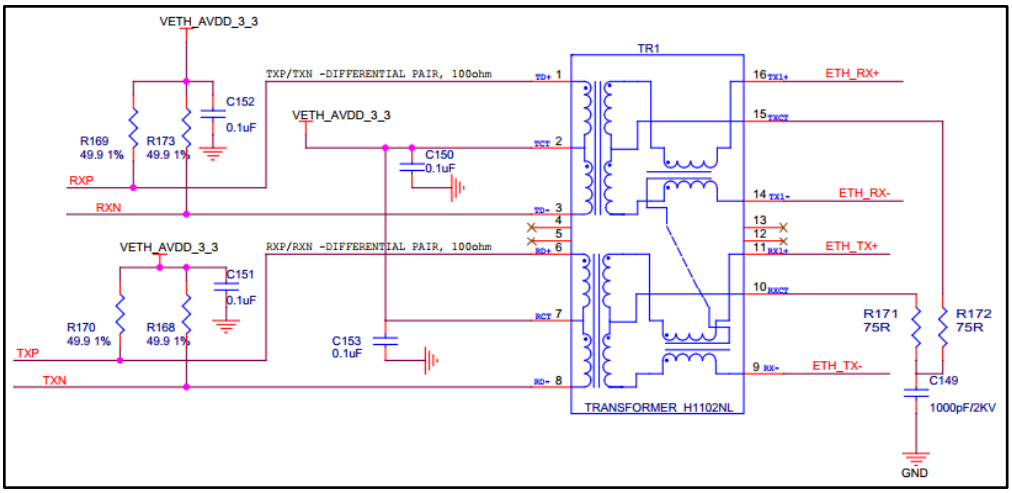
IMX6ULL 内部自带两个网络 MAC 控制器，每个网络只需增加一个 PHY 芯片，即可实现网络通信功能。HD-IMX6ULL-MB 开发板使用 LAN8720Ai 作为 PHY。LAN8720Ai 低功耗收发器是 10BASE-T/100 Base-TX 物理层(PHY)收发器，具有可变 I/O 电压，符合 IEEE 802.3-2005 标准，通过标准 RMII 接口与以太网 MAC 通信。每个器件包含一个全双工 10-BASE-T/100 Base-TX 收发器，支持 10Mbps (10BASE-T)和 100Mbps (100BASE-TX)运行。 可实施自动协商，以自动确定运行的最佳速度和双工模式。由于支持 HP Auto-MDIX，所以可以使用直连或交叉 LAN 电缆。

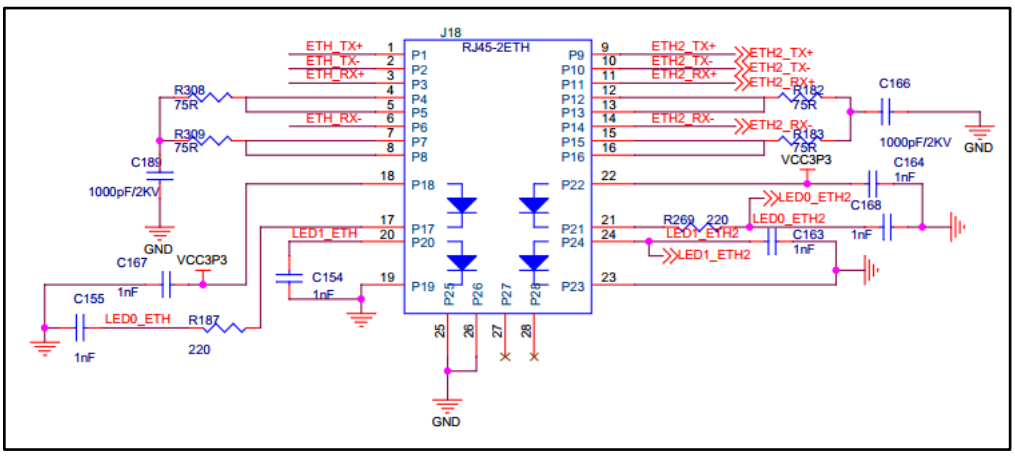
每路网口附带一路网络变压器 H1102NL，保证在工业环境使用的稳定性。

HD-IMX6ULL-MB 开发板 ENET1 PHY 地址为 0，ENET2 PHY 地址为 1。ENET1 和 ENET2 共用 ENET_MDIO 和 ENET_MDC 这两根线，这两根线连接到了 i.MX6ULL 的 GPIO1_IO06 和 GPIO1_IO07 上。

ETH1 的复位 IO 为 MX6ULL_PAD_SNVS_TAMPER7__GPIO5_IO07。ETH2 的复位 IO 为 MX6ULL_PAD_SNVS_TAMPER4__GPIO5_IO04。

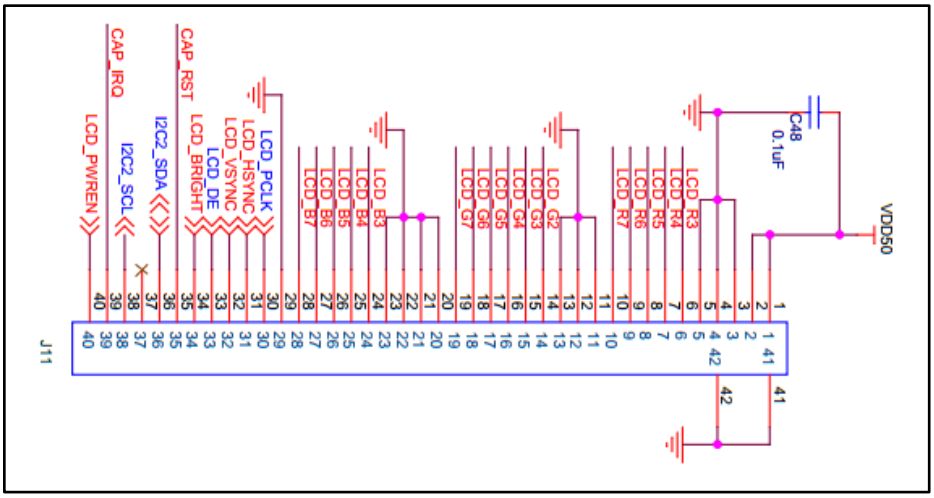






1.3.9 显示屏接口

HD-IMX6ULL-MB 显示屏接口为 RGB565 接口，支持电容触摸屏。

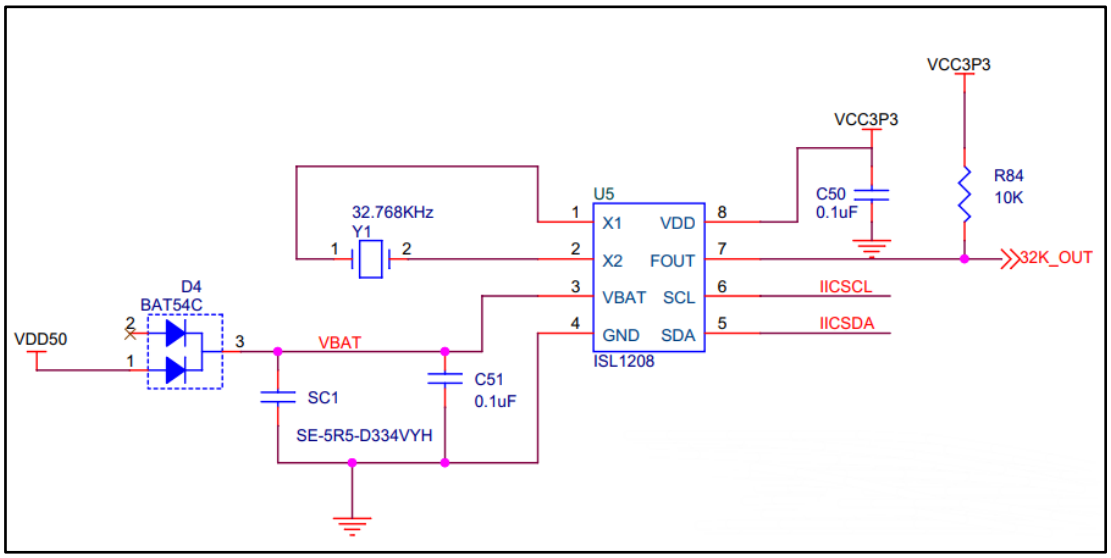


1.3.10 时钟芯片

ISL1208 是 INTERSIL 公司推出的一个 I2C 接口、低成本、低功耗的实时时钟，它带有定时与晶振补偿、支持时钟/日历等功能。振荡器采用外部 32.768kHz 晶振。

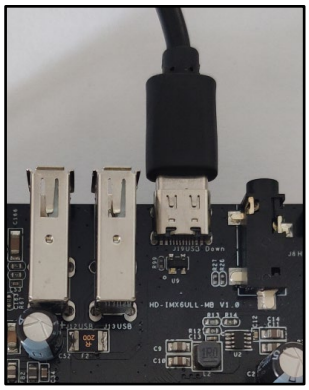
IIC2SCL 连接至 MX6UL_PAD_UART5_TX_DATA__I2C2_SCL

IIC2SDA 连接至 MX6UL_PAD_UART5_RX_DATA__I2C2_SDA



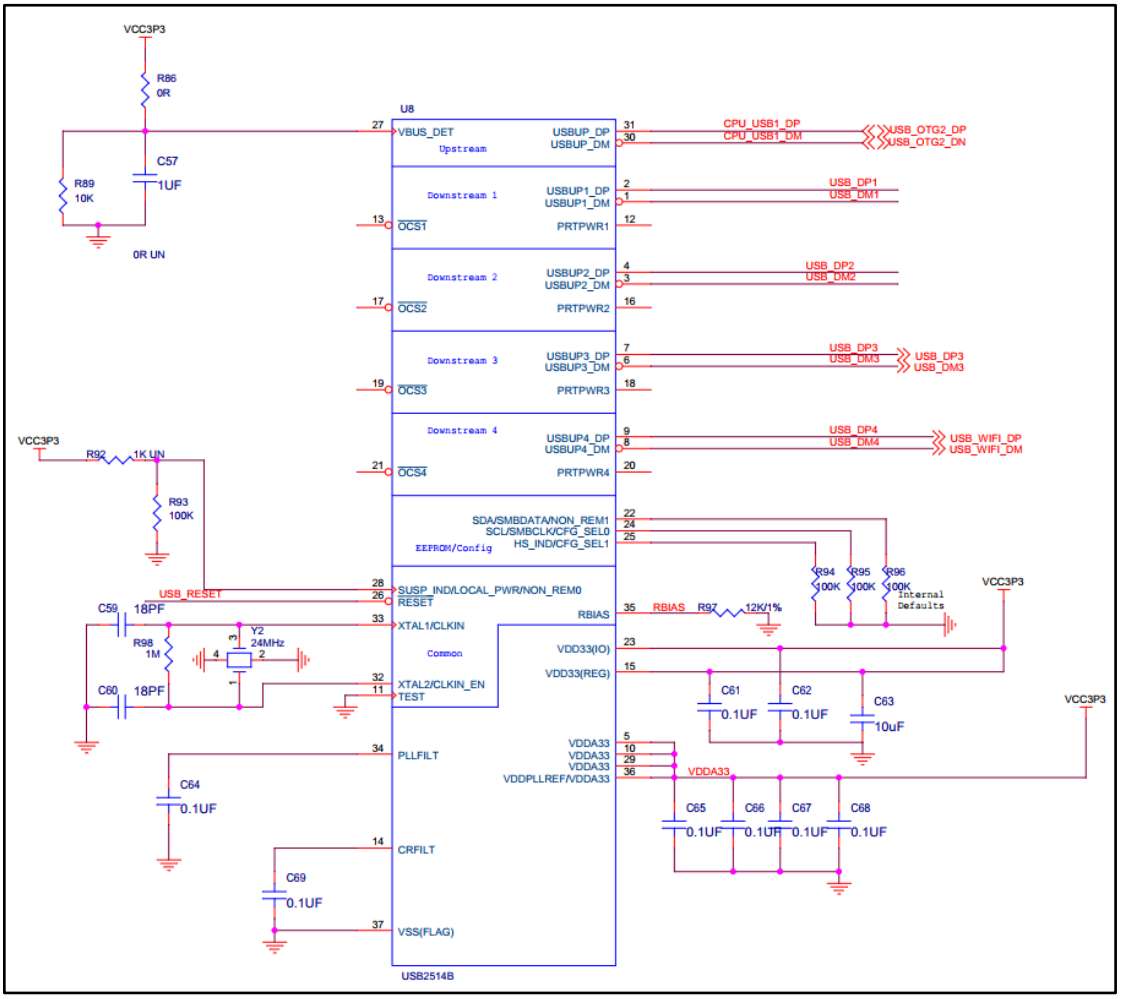
1.3.11 USB1

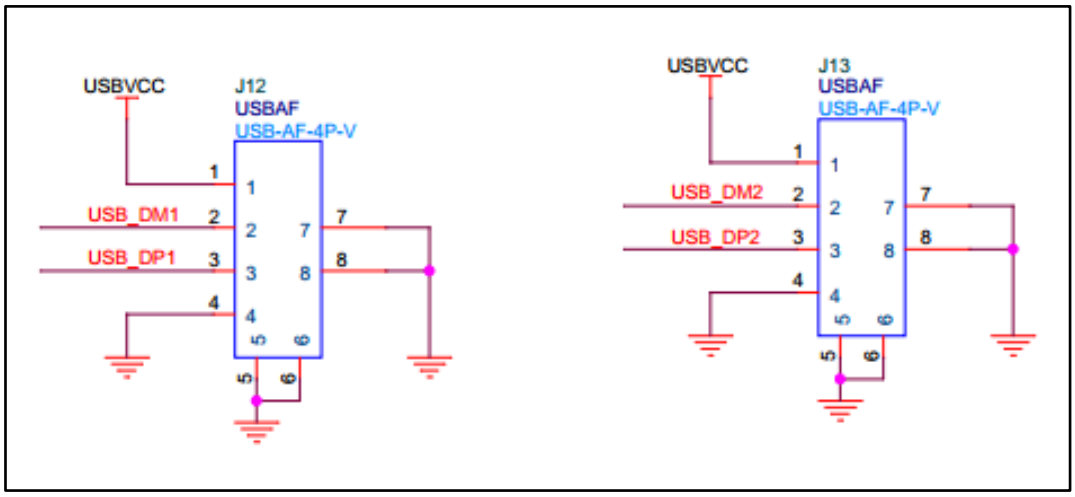
HD-IMX6ULL-MB 开发板 USB1 连接至 J19 端口，作为 USB Device 使用，一般用来同电脑连接，对开发板进行固件升级。



1.3.12 USB2

HD-IMX6ULL-MB 开发板使用一颗 USB2514B 芯片将一路 USB 扩展为 4 路 USB。其中两路作为 USB Host 引出。第 3 路通过排针引出。第 4 路用来和 USB Wifi 通信。





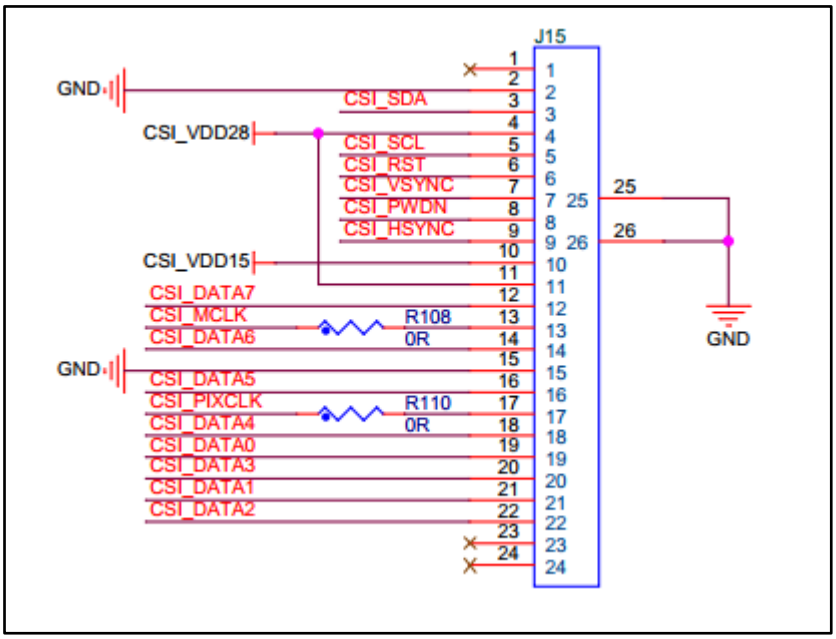
1.3.13 Camera

HD-IMX6ULL-MB 开发板预留了摄像头模块接口，可用来连接摄像头模块。

CSI_SCL 和 CSI_SDA 同 I2C2 连接。

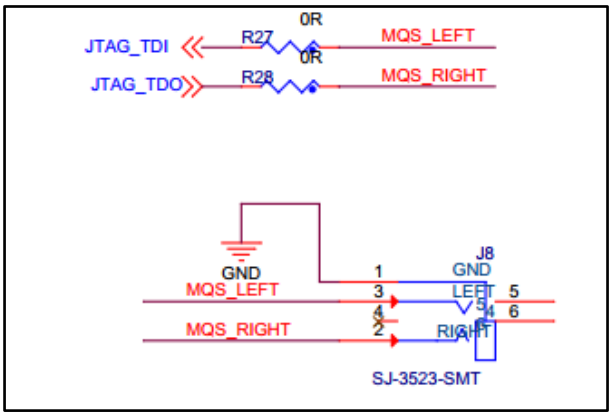
CSI_RST 为复位信号，连接至 MX6UL_PAD_SNVS_TAMPER5__GPIO5_IO05，低电平复位。

CSI_PWDN 为待机信号，连接至 MX6UL_PAD_SNVS_TAMPER5__GPIO5_IO05，高电平进入待机状态，低电平摄像头正常工作。



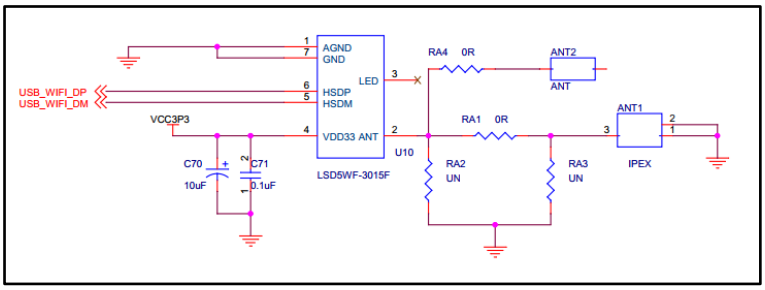
1.3.14 音频接口

HD-IMX6ULL-MB 开发板板载一个音频输出接口，可连接耳机播放音乐。



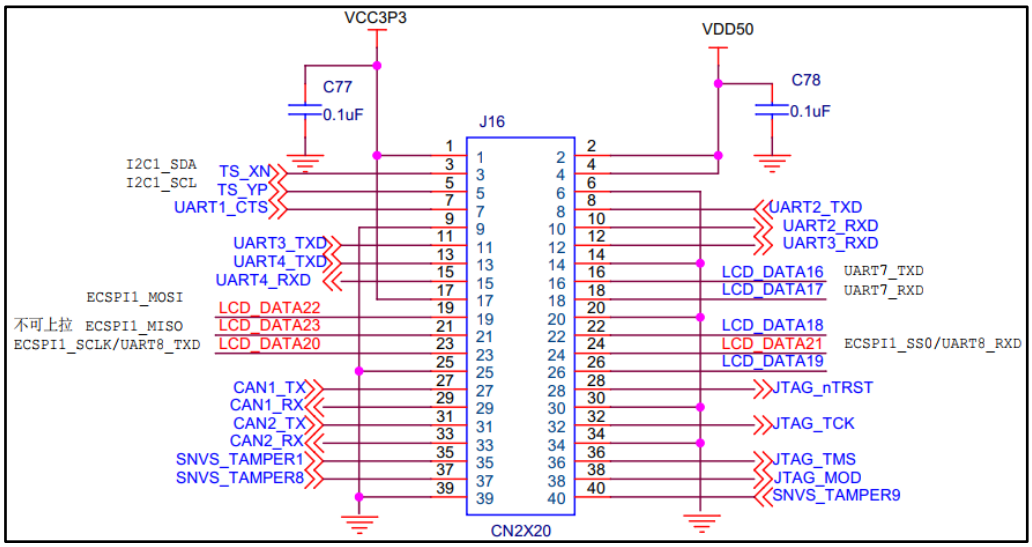
1.3.15 USB WIFI

HD-IMX6ULL-MB 开发板板载一个 RTL8188FTV USB Wifi 模块。在开发板已经板载了一个陶瓷天线，



1.3.16 扩展端口

对于富余的端口，都引出到 J16 端口方便用户使用。



sys/class 编 码	GPIO	功能名	物理引脚 BOARD编码		功能名	GPIO	sys/class 编 码
	3.3V	3.3V	1	2	5V	5V	
3	GPIO1_IO03	I2C1_SDA	3	4	5V	5V	
2	GPIO1_IO02	I2C1_SCL	5	6	GND	GND	
18	GPIO1_IO18	GPIO1_IO18	7	8	UART2_TX	GPIO1_IO20	20
	GND	GND	9	10	UART2_RX	GPIO1_IO21	21
24	GPIO1_IO24	UART3_TX	11	12	UART3_RX	GPIO1_IO25	25
28	GPIO1_IO28	UART4_TX	13	14	GND	GND	
29	GPIO1_IO29	UART4_RX	15	16	UART7_TX	GPIO3_IO21	85
	3.3V	3.3V	17	18	UART7_RX	GPIO3_IO22	86
91	GPIO3_IO27	ECSPI1_MOSI	19	20	GND	GND	
92	GPIO3_IO28	ECSPI1_MISO	21	22	GPIO3_IO23	GPIO3_IO23	87
89	GPIO3_IO25	ECSPI1_SCLK	23	24	ECSPI1_SS0	GPIO3_IO26	90
	GND	GND	25	26	GPIO3_IO24	GPIO3_IO24	88
26	GPIO1_IO26	CAN1_TX	27	28	PWM8	GPIO1_IO15	15
27	GPIO1_IO27	CAN1_RX	29	30	GND	GND	
22	GPIO1_IO22	CAN2_TX	31	32	PWM7	GPIO1_IO14	14
23	GPIO1_IO23	CAN2_RX	33	34	GND	GND	
129	GPIO5_IO01	GPIO5_IO01	35	36	GPIO1_IO11	GPIO1_IO11	11
136	GPIO5_IO08	GPIO5_IO08	37	38	GPIO1_IO10	GPIO1_IO10	10
	GND	GND	39	40	GPIO5_IO09	GPIO5_IO09	137

需要注意的是，扩展 IO 第 21 脚 LCD_DATA23 不可外接上拉电阻，否则会影响系统启动。

1.4 软件特性

HD-IMX6ULL-MB 提供完善的 Linux BSP（包括 Linux 内核源码、开发工具等）和一些应用方面的例子(如 Qt Demo)。具体软件资源如下表所示：

软件资源	说明	
Linux 内核	Linux 4.9	
文件系统	根文件系统采用 ext4，在根文件系统上可挂载多种文件系统，如：sysfs、yaffs2、ubifs 等	
交叉编译器	toolchain-qt5-cortexa7t2hf-neon-toolchain-4.19-warrior.sh	
外设驱动	SD/MMC	/drivers/mmc
	LCD	/drivers/video
	触摸屏	/drivers/input/touchscreen/goodix.c
	I2C	/drivers/i2c
	UART	/drivers/tty/serial
	USB	/drivers/usb
	以太网	/drivers/net/ethernet
	CAN	/drivers/net/can
	WIFI	/drivers/net/wireless
	PWM	/drivers/pwm
	GPIO	/drivers/gpio
	RTC	/drivers/rtc
示例程序	提供 LED、串口、CAN、网络、Web 等开发例程	
Qt 程序	提供 Qt 操作串口、CAN、网络、sqlite3、OpenCV 等例程	
工具软件	如系统镜像烧写工具、串口调试工具等	

1.5 Qt Demo 界面

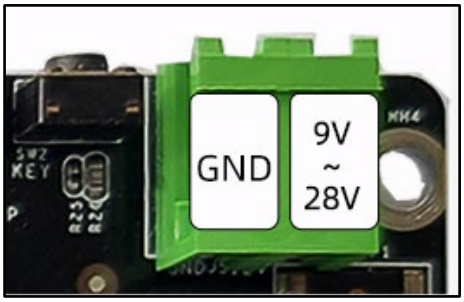

20:36:55
文件管理
视频播放
音乐播放
相册
天气
记事本
时钟
电子书
计算器


20:38:29
LED灯
蜂鸣器
背光调节
汽车仪表
InfoNES模拟器

2 快速使用说明

2.1 供电

HD-IMX6ULL-MB 开发板可使用 9~28V 电源进行供电，接口示意图如下：



2.2 启动选择

J3 用于设置启动模式，具体设置如下：

模式 / 序号	eMMC 启动 NAND 启动	SD 卡启动	升级 eMMC	升级 SD 卡
1 2				
3 4				
5 6				
7 8				
9 10				
11 12				

WDG / 序号	使能 WDG	关闭 WDG
13 14		

：跳线帽连接

：跳线帽断开

注：由于核心板上默认关闭了 WDG 功能。所以 WDG 功能无法使用跳线帽开启，如需使用 WDG 功能可与我们联系。

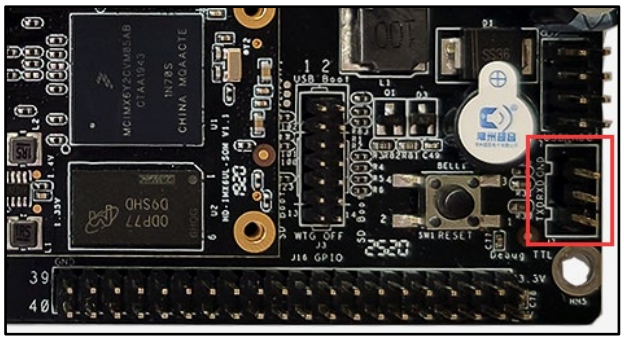
2.3 串口登录

开发板使用 COM1 口作为默认的登录调试串口，通过 COM1 口登录开发板的串口波特率为 115200，详细通讯参数设置如下表：

串口参数	值
波特率	115200
数据位	8
停止位	1
奇偶校验位	无
流控	关闭所有流控

2.3.1 串口硬件连接

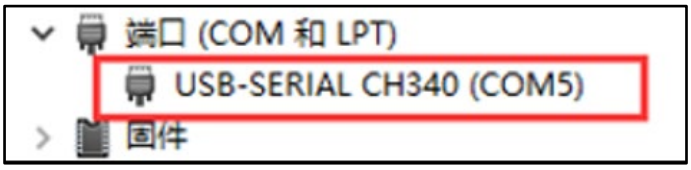
HD-IMX6ULL-MB 开发板预留了 Debug 串口。



在 windows 下，使用 USB 转 TTL 转换器连接主机和开发板时，需要安装驱动程序（在 win7、win10 下操作系统会自动搜索合适的驱动并安装）。当安装程序安装成功后，将 USB 转 TTL 转换器插入主机 USB 接口，可以在 windows 操作系统的“设备管理”对话框中检查系统是否正确识别了该串口，具体方法如下：

1) 按下快捷键“Win+X”（即同时按下 Win 和 X），在弹出的列表中选择“设备管理器(M)”

2) 在弹出的“设备管理器”对话框中，点击“端口（COM 和 LPT）”，将其展开；查看是否有对应的串口设备出现，如果有则说明系统正确识别了该转换器。如下图所示：



2.3.2 使用串口终端登录

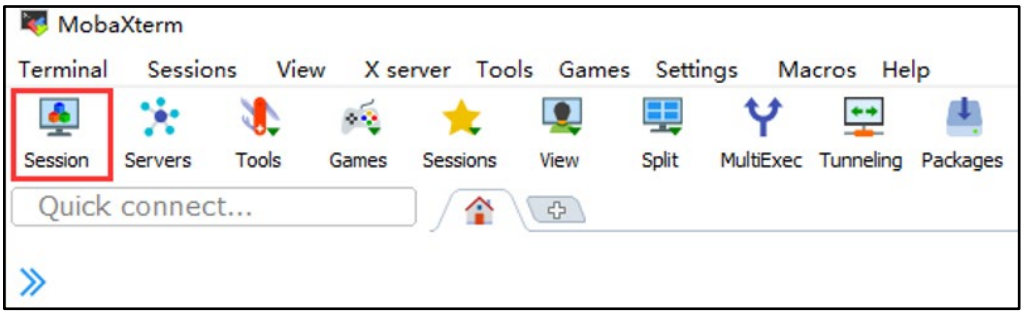
Windows 下的串口终端软件比较多，笔者推荐使用 MobaXterm。其下载链接为：https://mobaxterm.mobatek.net/download.html。

开发板资料文件夹下的“4-开发工具”目录提供了 MobaXterm 安装包：MobaXterm_Portable_v12.3.zip 和 MobaXterm_Installer_v12.3.zip。

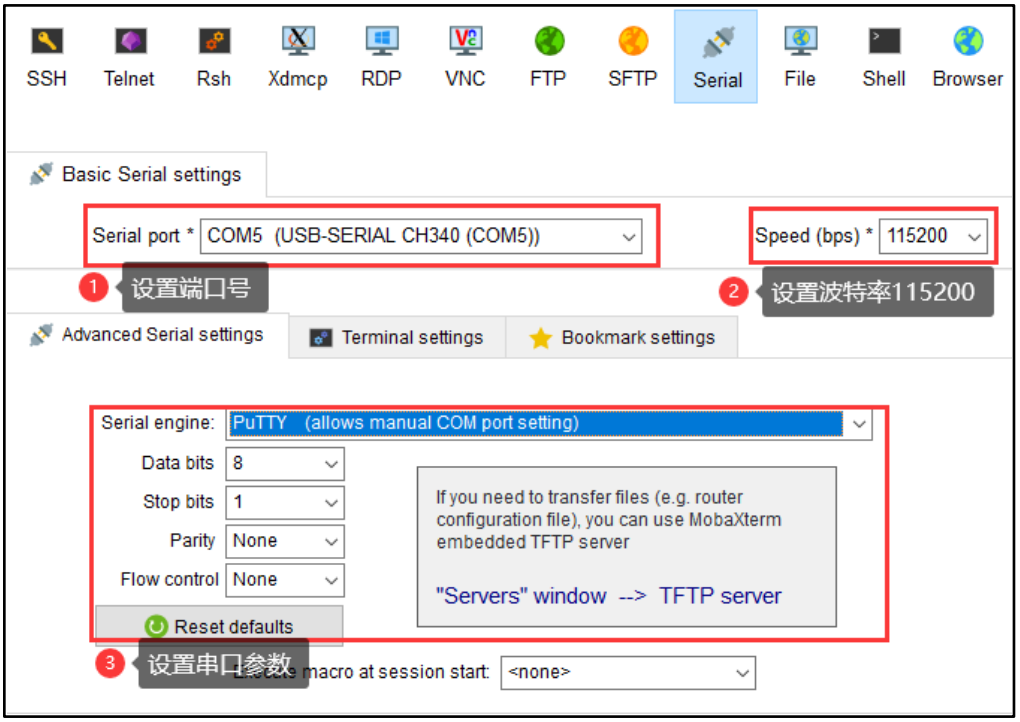
其中 MobaXterm_Portable_v12.3.zip 为免安装版本，直接解压即可使用，Portable 意为“随身携带”。

下面以 MobaXterm 软件为例具体介绍串口登录的方法。

1） 运行 MobaXterm_Personal 软件，并选择"Session"，如下图：

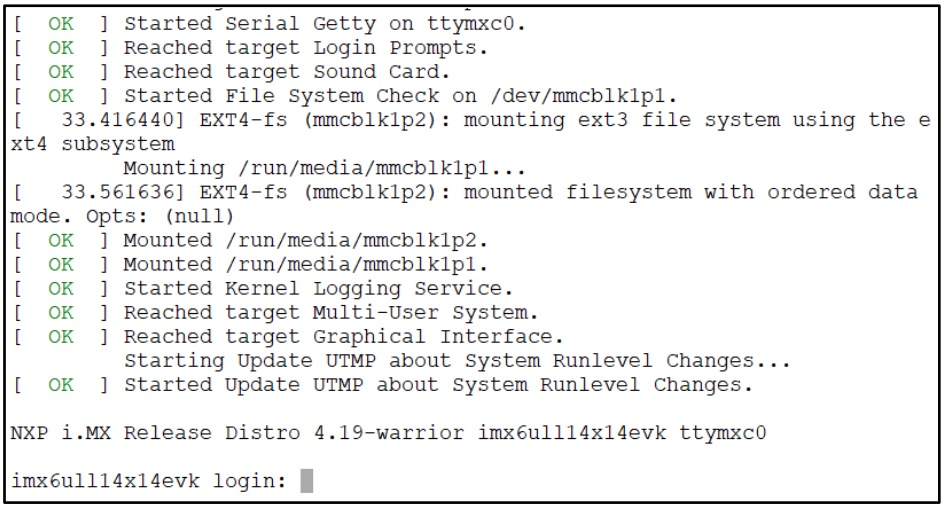


2） 设置“Serial”参数，如下图所示：



在该对话框中，“端口”需根据主机实际使用的串口号进行选择（可从操作系统的“设备管理器”中获得该信息），并对串口参数进行相应的设置。值

得注意的是：在设置串口参数时，需关闭所有的"flow control"选项。在设置完成后，点击"OK"按钮，会出现一个用于登录的 Xtrem 终端窗口。



在终端输入用户名密码即可登录，和系统进行交互。HD-IMX6ULL-MB 开发板默认用户名为 root，默认密码为空。

2.4 关机和重启

当需要关机时，为了确保数据完全写入，可执行 sync 命令完成数据同步后，再关机。

如需要重启，则使用 reboot 命令，该命令会自动完成数据同步再重启。

在 uboot 中，可执行 reset 重启开发板。

2.5 查看系统信息

2.5.1 查看内核版本

使用 uname -a 命令可以查看内核版本信息：

```
uname -a
```

```
xly-imx6ull login: root
root@xly-imx6ull:~# uname -a
Linux xly-imx6ull 4.19.35 #8 SMP PREEMPT Sun May 17 22:06:18 CST 2020 arm
v7l armv7l armv7l GNU/Linux
```

从输出信息可以看出，当前内核版本为 4.19.35。

使用/proc/version 文件，也可获得系统内核版本信息：

```
cat  /proc/version
```

```
root@xly-imx6ull:~# cat /proc/version
Linux version 4.19.35 (xinluyao@ubuntu) (gcc version 8.3.0 (GCC)) #8 SMP
PREEMPT Sun May 17 22:06:18 CST 2020
```

2.5.2 查看 CPU 信息

通过查看/proc/cpuinfo 文件，可以获得 CPU 等信息：

```
cat /proc/cpuinfo
```

```
root@xly-imx6ull:~# cat /proc/cpuinfo
processor       : 0
model name      : ARMv7 Processor rev 5 (v7l)
BogoMIPS        : 8.00
Features        : half thumb fastmult vfp edsp neon vfpv3 tls vfpv4 idiva idi
vt vfpd32 lpae
CPU implementer : 0x41
CPU architecture: 7
CPU variant     : 0x0
CPU part        : 0xc07
CPU revision    : 5

Hardware        : Freescale i.MX6 UltraLite (Device Tree)
Revision        : 0000
Serial          : 393399d263e0b904
```

其中 BogoMIPS 参数可以用来衡量处理器的运算能力，表示 CPU 每秒钟可以处理的指令数，单位为百万。

2.5.3 查看 CPU 主频

查看 CPU 支持的频率：

```
cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_available_frequencies
```

```
root@xly-imx6ull:~# cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_available_f
requencies
198000 396000 528000
```

查看 CPU 当前频率：

```
cat /sys/devices/system/cpu/cpu0/cpufreq/cpuinfo_cur_freq
```

```
root@xly-imx6ull:~# cat /sys/devices/system/cpu/cpu0/cpufreq/cpuinfo_cur_freq

528000
```

如上图，表示当前 CPU 主频为 528MHz。当系统监测到 CPU 负载较低时，会自动降低运行频率，以降低功耗，CPU 负载大时，会适当提高运行频率。

2.5.4 查看内存使用情况

使用 free 命令可以查看内存的使用情况：

```
free -m
```

```
root@xly-imx6ull:~# free -m
              total        used        free      shared  buff/cache   available
Mem:            491          99         357           8          34         326
Swap:             0           0           0
```

2.5.5 查看磁盘使用情况

使用 df -m 命令可以查看磁盘的使用情况：

```
df -m
```

```
root@xly-imx6ull:~# df -m
Filesystem                        1M-blocks  Used Available Use% Mounted on
192.168.1.118:/srv/imx6ull/rootfs    114835 35859     73101  33% /
devtmpfs                                86     1        86   1% /dev
tmpfs                                  246     0       246   0% /dev/shm
tmpfs                                  246     9       238   4% /run
tmpfs                                  246     0       246   0% /sys/fs/cgroup
tmpfs                                  246     0       246   0% /tmp
tmpfs                                  246     1       246   1% /var/volatile
/dev/mmcblk1p2                        6887   143      6388   3% /run/media/mmcblk1
p2
/dev/mmcblk1p1                         127     8       119   6% /run/media/mmcblk1
p1
```

2.5.6 查看 FLASH 存储器容量

/proc/partitions 文件包含了存储器的分区信息，查看分区信息可以了解板子的存储容量：

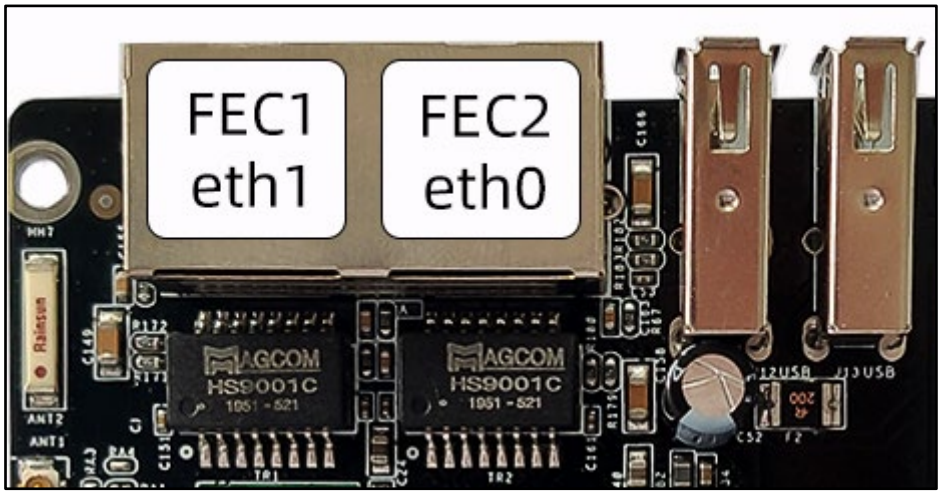
```
cat /proc/partitions
```

其中 mmcblk 开头的都属于 eMMC 存储器的 blocks 的数量，eMMC 每个 block 大小为 512Bytes，把它们所有的 blocks 加起来就可以算出容量。

```
root@xly-imx6ull:~# cat /proc/partitions
major minor  #blocks  name

   1        0      65536 ram0
   1        1      65536 ram1
   1        2      65536 ram2
   1        3      65536 ram3
   1        4      65536 ram4
   1        5      65536 ram5
   1        6      65536 ram6
   1        7      65536 ram7
   1        8      65536 ram8
   1        9      65536 ram9
   1       10      65536 ram10
   1       11      65536 ram11
   1       12      65536 ram12
   1       13      65536 ram13
   1       14      65536 ram14
   1       15      65536 ram15
 179        0    7438336 mmcblk1
 179        1     131072 mmcblk1p1
 179        2    7297024 mmcblk1p2
```

2.6 网络设置

HD-IMX6ULL-MB 有两个以太网卡，如下图：



IMX6ULL 芯片有两个 FEC，FEC1(ethernet@2188000)，FEC2(ethernet@20b4000)，由于 FEC2 地址在 FEC1 地址前，所以 FEC2 在系统中为 eth0,而 FEC1 在系统中识别为 eth1。HD-IMX6ULL-MB 在 uboot 中使用 FEC2 作为默认网口。

请大家在进行以下操作前，确保网线是连接在 **FEC2(eth0)** 上。

2.6.1 注意事项

开发板和主机通讯时，开发板和主机必须连接在一个路由器上，它们的 IP 地址必须在同一网段内。如：假设主机 IP 地址为 192.168.1.118，子网掩码为 255.255.255.0，则将开发板的 IP 地址设为 192.168.1.x，如设置为 192.168.1.12。另外在配置完成主机 IP 和开发板 IP 后，可以在主机上运行 ping 命令，查看网络是否畅通。

2.6.2 查看 PC IP 地址

设置网卡 IP 前，需要先查看 PC ip 地址，确保为开发板和 PC 处于同一网段。

```
ip addr show
```

```
xinluyao@ubuntu:~$ ip addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:0c:29:50:27:20 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.118/24 brd 192.168.1.255 scope global dynamic noprefixroute ens33
       valid_lft 36338sec preferred_lft 36338sec
    inet6 fe80::362e:1a02:5429:8e5c/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

2.6.3 设置网卡 IP

从 PC 的 IP 可知，该网卡的网关为 192.168.1.1，则我们可以将 eth0 的 ip 地址设置为 192.168.1.160，执行如下命令：

```
ifconfig  eth0  192.168.1.160  netmask  255.255.255.0
```

设置完毕后，可以使用 ip addr show 命令查看 eth0 的配置：

```
ifconfig addr show
```

```
root@xly-6ull-emmc:~# ip addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default q
len 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: can0: <NOARP,ECHO> mtu 16 qdisc noop state DOWN group default qlen 10
    link/can
3: can1: <NOARP,ECHO> mtu 16 qdisc noop state DOWN group default qlen 10
    link/can
4: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group
 default qlen 1000
    link/ether 16:e5:0f:30:42:4f brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.160/24 brd 192.168.1.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::14e5:fff:fe30:424f/64 scope link
       valid_lft forever preferred_lft forever
```

其中接口 eth0/1 为以太网卡，lo 表示本地回环接口。需要注意，某些比较旧版本的文件系统可能没有 ip 命令，可以使用 ifconfig 命令操作。

2.6.4 设置默认网关

使用 route 命令可以用来添加、删除网关或查看网关配置：

1） 添加默认网关：

使用如下命令给网卡添加默认网关：

```
route add default gw  网关 IP 地址网络接口名
```

例如要将 eth0 的默认网关设置为 192.168.1.1，执行如下命令：

```
route add default gw 192.168.1.1 eth0
```

2） 删除网关：

使用如下命令删除网卡上已经配置的默认网关：

```
route del default gw  网关 IP 地址网络接口名
```

例如要将 eth0 上配置的 IP 为 192.168.1.1 的默认网关删除，执行如下命令：

```
route del default gw 192.168.1.1 eth0
```

3） 查看网关配置：

使用 route -n 命令可以查看当前的网关配置：

```
route -n
```

```
root@xly-imx6ull:~# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         192.168.1.1     0.0.0.0         UG    0      0        0 eth0
192.168.1.0     0.0.0.0         255.255.255.0   U     0      0        0 eth0
```

2.6.5 关闭/启动网卡

使用 ifconfig 命令可以关闭和启动网卡，关闭网卡的命令格式为：

```
ifconfig 网络接口名 down
```

启动网卡的命令格式为：

```
ifconfig 网络接口名 up
```

如要关闭 eth0，执行如下命令：

```
ifconfig eth0 down
```

要启动 eth0，执行如下命令：

```
ifconfig eth0 up
```

2.6.6 设置 DNS

如果需要使用域名访问互联网，则需要配置 DNS 服务器。配置方法如下：

编辑“/etc/resolv.conf”文件（如果不存在则创建一个新的文件），并在其中添加一个或者多个 DNS 服务器的 IP 地址，例如要将 114.114.114.114 作为首选的 DNS 服务器，将 8.8.8.8 作为备选的 DNS 服务器，则在/etc/resolv.conf 添加如下内容：

```
nameserver 114.114.114.114
nameserver 8.8.8.8
```

注：此方法设置的 resolv.conf 文件下次重启会被覆盖，设置会失效。

2.6.7 设置固定 IP

开发板默认在开机时，会自动联网，从路由器获取一个 IP 地址，这个获取到的 IP 地址是随机的，当需要把开发板作为服务器时，就需要将 IP 地址设置为固定 IP 地址。

HD-IMX6ULL-MB 开发板采用了 systemd，所以网络服务被 systemd 接管，所以固定 IP 设置方法也有所改变。具体方法如下：

在"/etc/systemd/network"目录下，新建一个名为 10-static-eth0.network 文件。并在其中填入如下内容：

```
[Match]
Name=eth0

[Network]
Address=192.168.1.160
Gateway=192.168.1.1
DNS 8.8.8.8
```

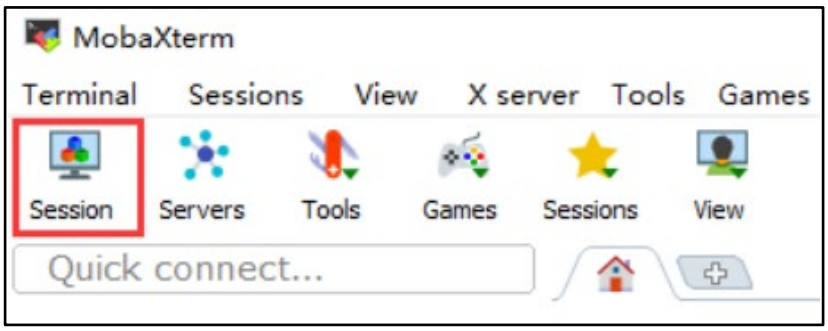
编辑好后，直接保存文件，再次重启，开发板 IP 会被设置为 192.168.1.160。

2.7 网络登录

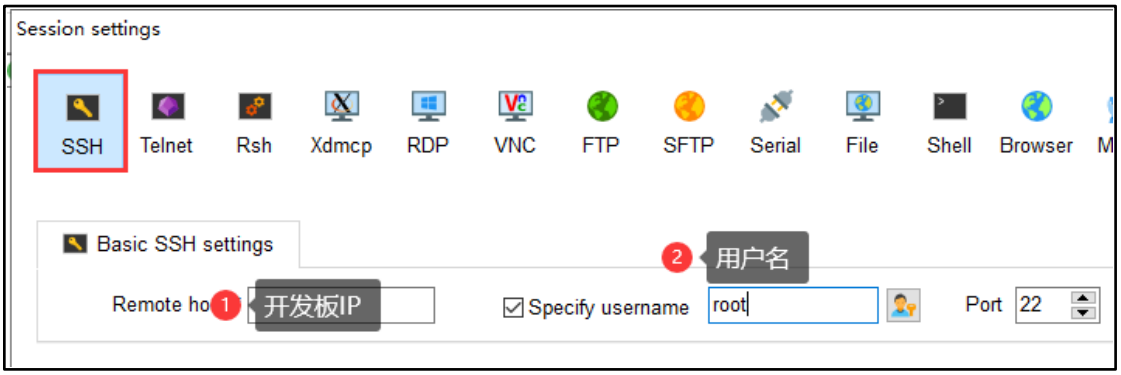
当设置好开发板 IP 地址，且主机和开发板之间网络通讯正常后，即可以采用网络登录到开发板上。此处仍以 XTerm 软件为例，介绍网络登录的方法。

2.7.1 使用 XTerm 登录

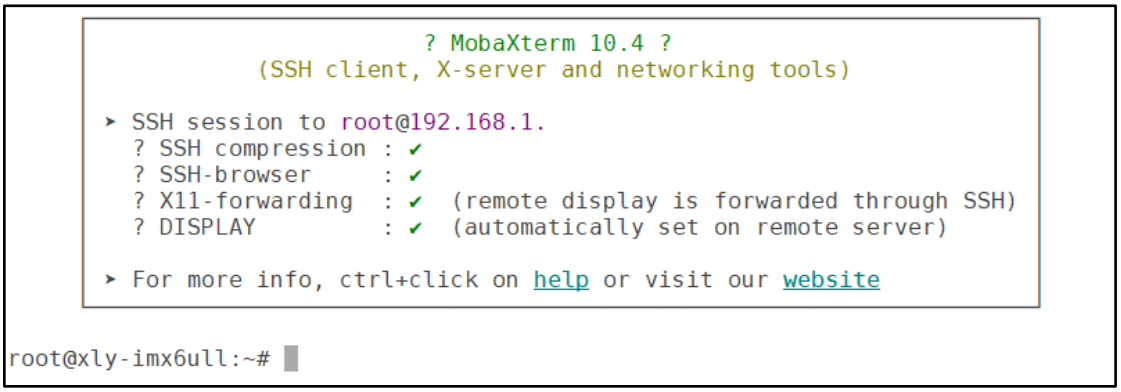
1） 运行 Xterm 软件，选择“Session”，如下图所示：



2） 点击“SSH”并进行设置，如下图所示：



在该对话框中，选择“协议”为“SSH”，“主机名”为开发板的 IP 地址，端口号使用默认值 22。在设置完成后，点击“OK”按钮，会出现一个命令行终端。如下图所示：



2.8 USB 鼠标与 USB 键盘使用

将 USB 鼠标插入到开发板 USB HOST 接口上，Linux 操作系统会检测到 USB 鼠标，并在控制台终端上打印 USB 鼠标的相关信息，例如：

```
root@xinluyao-imx6ull:~# usb 1-1.2: new low-speed USB device number 3 using ci_hdr
c
input: USB OPTICAL MOUSE  as /devices/platform/soc/2100000.aips-bus/2184200.usb/ci
_hdrc.1/usb1/1-1/1-1.2/1-1.2:1.0/0003:275D:0BA6.0001/input/input3
hid-generic 0003:275D:0BA6.0001: input: USB HID v1.11 Mouse [USB OPTICAL MOUSE ] o
n usb-ci_hdrc.1-1.2/input0
usbcore: registered new interface driver usbhid
usbhid: USB HID core driver
```

将 USB 键盘插入到开发板 USB HOST 接口上，Linux 操作系统会检测到 USB 键盘，并在控制台终端上打印 USB 键盘的相关信息，例如：

```
root@xinluyao-imx6ull:~# usb 1-1.2: new low-speed USB device number 4 using ci_hdr
c
input: USB USB Keyboard as /devices/platform/soc/2100000.aips-bus/2184200.usb/ci_h
drc.1/usb1/1-1/1-1.2/1-1.2:1.0/0003:1A2C:20C0.0002/input/input4
hid-generic 0003:1A2C:20C0.0002: input: USB HID v1.10 Keyboard [USB USB Keyboard]
on usb-ci_hdrc.1-1.2/input0
input: USB USB Keyboard as /devices/platform/soc/2100000.aips-bus/2184200.usb/ci_h
drc.1/usb1/1-1/1-1.2/1-1.2:1.1/0003:1A2C:20C0.0003/input/input5
hid-generic 0003:1A2C:20C0.0003: input: USB HID v1.10 Device [USB USB Keyboard] on
 usb-ci_hdrc.1-1.2/input1
```

2.9 TF 卡使用

将 TF 卡插入到开发板 TF 卡插槽中，Linux 操作系统会检测到 TF 卡，并在控制台终端上打印 TF 卡的相关信息，例如：

```
root@xinluyao-imx6ull:~# mmc0: host does not support reading read-only switch, ass
uming write-enable
mmc0: new high speed SDHC card at address 0001
mmcblk0: mmc0:0001 SD8GB 7.35 GiB
```

在此例中，从打印信息可以看出 Linux 操作系统检测到一个容量为 8GB 的 TF 卡，其对应的设备名为 mmcblk0。

可以使用 fdisk -l 查看 SD 卡信息。

```
Disk /dev/mmcblk0: 7.4 GiB, 7948206080 bytes, 15523840 sectors
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: gpt
Disk identifier: 2988B0A4-1200-453B-850B-334871B23B81

Device         Start      End  Sectors  Size Type
/dev/mmcblk0p1  2048 15521791 15519744  7.4G Microsoft basic data
```

系统会为 TF 卡会出现在/run/media 目录下，目录名为 mmcblk0pn

（其中 n 表示不同的分区，n=1、2、3……），TF 卡的每个分区就挂载在这些目录下。TF 卡挂载成功后，即可对 TF 卡进行文件查看、文件拷贝等操作。以下以文件拷贝操作为例，介绍 TF 卡的使用。

将 TF 卡根目录上的 test 文件拷贝到/home 目录下，可执行如下命令：

```
cp /run/media/mmcblk0p1/test  /home/
```

将/home 目录下的 test1 文件拷贝到 TF 卡根目录上，可执行如下命令：

```
cp /home/test1  /run/media/mmcblk0p1/
```

2.10 U 盘使用

将格式为 FAT32 的 U 盘插入到开发板 USB HOST 接口上，Linux 操作系统会检测到 U 盘，并在控制台终端上打印 U 盘的相关信息，例如：

```
root@xly-imx6ull:/run/media# [ 1163.496197] usb 1-1.3: new high-speed USB device nu
mber 6 using ci_hdrc
[ 1163.664834] usb-storage 1-1.3:1.0: USB Mass Storage device detected
[ 1163.683414] scsi host0: usb-storage 1-1.3:1.0
[ 1164.728524] scsi 0:0:0:0: Direct-Access              USB FLASH DRIVE  PMAP PQ: 0
 ANSI: 6
[ 1168.189853] sd 0:0:0:0: [sda] 30277632 512-byte logical blocks: (15.5 GB/14.4 Gi
B)
[ 1168.199937] sd 0:0:0:0: [sda] Write Protect is off
[ 1168.215554] sd 0:0:0:0: [sda] No Caching mode page found
[ 1168.221126] sd 0:0:0:0: [sda] Assuming drive cache: write through
[ 1168.252311]  sda: sda1
[ 1168.271634] sd 0:0:0:0: [sda] Attached SCSI removable disk
```

在此例中，从打印信息可以看出系统检测到一个容量为 15.5GB 的 U 盘，其对应的设备名为 sda。可进行如下操作：

2.10.1 查看信息

可以使用 fdisk -l 查看 U 盘信息。

```
root@xly-imx6ull:/run/media# fdisk -l /dev/sda
Disk /dev/sda: 14.4 GiB, 15502147584 bytes, 30277632 sectors
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: 0xeec10345

Device     Boot Start      End  Sectors  Size Id Type
/dev/sda1  *    16128 30277631 30261504 14.4G  c W95 FAT32 (LBA)
```

由上可以知道，U 盘中有一个分区，大小为 14.4GB。

2.10.2 挂载 mount

U 盘中有分区为/dev/sda1，可使用如下命令挂载

```
mount /dev/sda1 /mnt
```

挂载成功，即可对 U 盘进行操作。操作/mnt 目录就是操作 U 盘中内容。

2.10.3 卸载 umount

U 盘使用完毕后，在拔出 U 盘前，需执行 umount 命令卸载 U 盘所有的分区,命令如下：

```
umount /mnt/
```

umount 会确保所有缓存的数据都被正确的写入 U 盘。在 umount 成功后，即可拔出 U 盘。

在调用 umount 前，必须确保 U 盘上的文件没有被其他程序所占用，且用户当前的工作目录不在 U 盘的挂载目录上，否则调用 umount 会提示失败。

2.11 TFTP 与 PC 互传文件

通过串口或网络登录到开发板上，并使用 tftp 命令从开发板下载文件：

```
cd $HOME
tftp -r imx6ull-xly-emmc-lcd.dtb -g 192.168.1.118
```

其中“-r”参数为需要从 PC 机下载的文件名，“-g”参数为 PC 机的 IP 地址。上述命令行的含义是从 IP 地址为 192.168.1.118 的 PC 机下载 imx6ull-xly-emmc-lcd.dtb 文件，下载的文件将保存在 HOME 目录下。

如果需要将开发板中的文件上传到 PC 机上，执行如下步骤：

1） 通过串口或网络登录到开发板上，并进入文件所在目录

2） 执行如下命令：

```
tftp -r imx6ull-xly-emmc-lcd.dtb -p 192.168.1.118
```

其中“-r”参数为开发板中需要上传到 PC 机上的文件的文件名，“-p”参数为 PC 机的 IP 地址。上传完成后，在 PC 机 TFTP 服务器设置的文件传输目录中可以找到上传的文件。

2.12 LED 测试

HD-IMX6ULL-MB 开发板上有两个用户 LED 灯，其中 1 个用于心跳指示，设备名为“run”;另外一个由用户控制，设备名为“user”。

/sys/class/leds/user 目录下的文件用于控制这个 LED 灯，此目录下包含 brightness、trigger 等文件。brightness 对应 LED 的亮度，trigger 对应 LED 灯的触发方式。用户可以通过 echo、cat 等命令来修改或查看这些文件，从而达到控制 LED 灯的目的。

① 查看 LED 灯的 trigger 模式

```
cd /sys/class/leds/user
cat trigger
```

```
root@xly-imx6ull:/sys/class/leds/user# cat trigger
[none] rfkill-any rfkill-none kbd-scrolllock kbd-numlock kbd-capslock kbd-kanalock
kbd-shiftlock kbd-altgrlock kbd-ctrllock kbd-altlock kbd-shiftllock kbd-shiftrlock
kbd-ctrlllock kbd-ctrlrlock mmc0 mmc1 timer oneshot heartbeat backlight gpio
```

可以看到目前[none]被选中，LED 由用户控制。

② 点亮 LED

向 brightness 写 1 可以点亮 LED 灯。

```
echo 1 > brightness
```

③ 熄灭 LED

向 brightness 写 0 可以熄灭 LED 灯。

```
echo 0 > brightness
```

另外笔者在/home/root/hardware_test 目录下提供了各种硬件的测试程序。其中 led 目录下为 led 测试程序。该程序可控制 LED 每 2 秒钟闪烁一次。在命令行下运行如下命令执行测试脚本：

```
cd /home/root/hardware_test/led
./led.sh
```

2.13 蜂鸣器测试

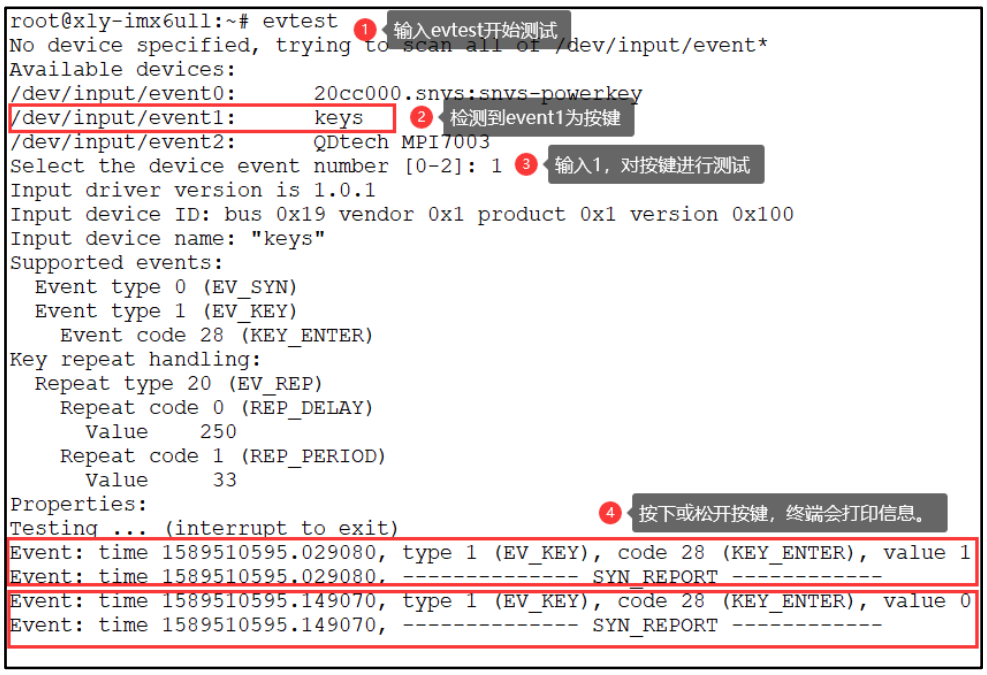
在开发板/home/root/hardware_test 目录下，运行程序可以对蜂鸣器进行测试。该程序可控制蜂鸣器，每 2 秒钟鸣叫一次。为执行该测试，可在命令行下运行如下命令：

```
cd /home/root/hardware_test/beep
./beep.sh
```

请在蜂鸣器关闭时退出测试程序，否则蜂鸣器会一直鸣叫。

2.14 按键测试

在命令行运行 evtest，可进行按键测试，具体操作如下：



2.15 液晶背光设置

可以通过修改/sys/class/backlight/backlight/brightness 文件的值对液晶背光进行调整。该值的取值范围为 0 ~ 100，共 100 个背光级别。当设置为 100 时背光最亮。

默认把背光值设置为 60，命令如下：

```
echo 60 > /sys/class/backlight/backlight/brightness
```

2.16 GPIO 输入输出

Linux 中，最常见的读写 GPIO 的方式就是 GPIO sysfs interface。通过操作/sys/class/gpio 目录下的 export、unexport、GPIO{N}/direction，GPIO{N}/value(N 用实际引脚号替代)等文件实现。sysfs interface 使用比较广泛，但由于其存在一些问题，比如不能并发获取 sysfs 属性等。所以从 linux4.8 开始，gpiod 取代了 sysfs interface。

HD-IMX6ULL-MB 开发板使用 4.9 内核，理所当然使用了 gpiod。使用 gpiod 时，GPIO 访问控制是通过操作字符设备文件（比如/dev/gpiodchip0）实现的，并通过 libgpiod 提供一些命令工具。

2.16.1 gpiodetect

gpiodetect 用来查看 GPIO 组信息

```
root@xly-imx6ull:~# gpiodetect
gpiochip0 [209c000.gpio] (32 lines)
gpiochip1 [20a0000.gpio] (32 lines)
gpiochip2 [20a4000.gpio] (32 lines)
gpiochip3 [20a8000.gpio] (32 lines)
gpiochip4 [20ac000.gpio] (32 lines)
```

IMX6ULL 上一共有 5 个 GPIO 组，gpiochip0~gpiochip4，分别对应 GPIO1~GPIO5。

2.16.2 gpioinfo

gpioinfo 查询指定组内的 GPIO 信息。如我们要查询 GPIO3(即 gpiochip2)。

```
root@xly-imx6ull:~# gpioinfo gpiochip2
gpiochip2 - 32 lines:
        line   0:      unnamed       unused   input  active-high
        line   1:      unnamed       unused   input  active-high
        line   2:      unnamed       unused   input  active-high
        line   3:      unnamed       unused   input  active-high
        line   4:      unnamed "lcd_pwr_en"  output  active-high [used]
        line   5:      unnamed       unused   input  active-high
        line   6:      unnamed       unused   input  active-high
        line   7:      unnamed       unused   input  active-high
        line   8:      unnamed       unused   input  active-high
        line   9:      unnamed       unused   input  active-high
        line  10:      unnamed       unused   input  active-high
        line  11:      unnamed       unused   input  active-high
        line  12:      unnamed       unused   input  active-high
        line  13:      unnamed       unused   input  active-high
        line  14:      unnamed       unused   input  active-high
        line  15:      unnamed       unused   input  active-high
        line  16:      unnamed       unused   input  active-high
        line  17:      unnamed       unused   input  active-high
        line  18:      unnamed       unused   input  active-high
        line  19:      unnamed       unused   input  active-high
        line  20:      unnamed       unused   input  active-high
        line  21:      unnamed       unused   input  active-high
        line  22:      unnamed       unused   input  active-high
        line  23:      unnamed       unused  output  active-high
```

图中可以看到 GPIO3_IO04 用于 LCD_PWR_EN 功能，这个与实际情况一致。其余引脚没有使用。

2.16.3 gpioset

当需要设置引脚输出电平时，使用 gpioset 功能。我们使用预留接口中的 LCD_D18(GPIO3_IO23)进行测试。

1) 输出高电平

```
gpioset gpiochip2 23=1  或  gpioset 2 23=1
```

2) 输出低电平

```
gpioset gpiochip2 23=0  或  gpioset 2 23=0
```

2.16.4 gpioget

当需要设置引脚输出电平时，使用 gpioget 功能。我们使用预留接口中的 LCD_D18(GPIO3_IO23)进行测试。

当给引脚接低电平时，读取数据如下：

```
root@xly-imx6ull:~# gpioget 2 23
1
```

当给引脚接低电平时，读取数据如下：

```
root@xly-imx6ull:~# gpioget 2 23
0
```

2.17 串口测试

HD-IMX6ULL-MB 开发板提供了多个串口供用户使用：其中 COM1 为调试串口，其余串口为通讯串口。在开发板/home/root/hardware_test/serial 目录下，运行 serialtest 程序可以对串口进行数据收发测试。

该程序在运行时，需要提供一个命令行参数，该参数既可以是需要打开的串口名，如：“COM2”、“COM3”、“COM4”等；也可以是需要打开的设备名，如/dev/ttymxc1、/det/ttymxc2、/dev/ttymxc3。

如需要通过 COM4 口进行数据收发，在命令行下执行如下命令：

```
cd /home/root/hardware_test/serial/
./serial_test COM4
```

该程序运行流程如下：

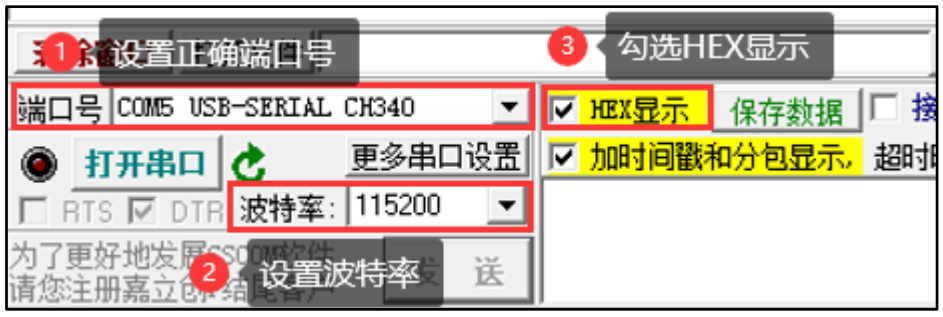
1) 打开串口（串口通讯参数为：波特率 115200，8 位数据位，1 位停止位，无数据校验位）；
2) 通过串口发送一个 20 字节的数据；
3) 从串口接收数据；
4) 重复步骤 2~3，实现数据的循环发送和接收。

下面介绍具体操作步骤，在 PC 上使用 USB 转串口工具和开发板连接，对开发板进行测试：

1) 使用 USB 转串口连接电脑和开发板串口。

2） 在 Windows 打开 ssom32 软件(该软件位于资料“4-开发工具/Windows/SSCOM32”文件夹)。

3） 选择合适的端口号，并设置串口通讯参数。参数设置为波特率 115200，8 位数据位，1 位停止位，无数据校验位。由于测试中我们是直接发送 HEX 数据，所以需要勾选“HEX 显示”选项。



4） 点击“打开串口”按钮：

2.17.1 发送测试

在开发板上运行 serial_test 测试程序。在命令行下执行如下命令：

```
cd ~/hardware_test/serial
~/hardware_test/serial
./serial_test COM4
```

此处假定测试开发板 COM4 口。运行 serial_test 后，serial_test 会通过串口发送数据，此时可以在 sscom32 看到 serial_test 发送的数据，如下图所示：

2.17.2 接收数据

测试开发板接收数据是否正常。方法如下：

在 sscom32 上，输入需要发送的数据，如果数据是二进制字节数据（即非 ASCII 字符）需勾选“HEX 发送”选项。设置完成后，点击“发送”按钮。如下图所示：

此例中发送的二进制字节数据为：“0x11 0x22 0x33 0x44 0x55 0x66 0x77 0x88 0x99 0x00 0xaa 0xbb 0xcc 0xdd 0xee 0xff”。此时在开发板命令行终端上，可以接收到 sscom32 发送的数据，如下图所示：

```
root@xinluyao-imx6ull:~/hardware_test/serial# ./serial_test COM4
serial read: length=16, packet:11 22 33 44 55 66 77 88 99 00 AA BB CC DD EE FF
```

2.18 CAN 测试

HD-IMX6ULL-MB 提供了两路 can 口供用户使用：can1 和 can2。在开发板/home/root/hardware_test/can 目录下，运行 can_test 程序可以对 can 口进行数据收发测试。

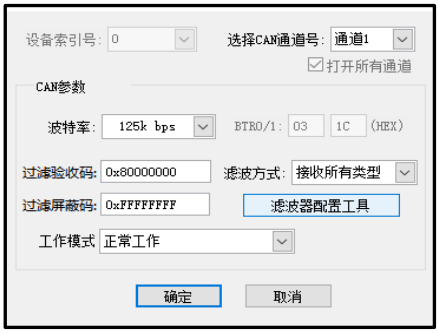
该程序在运行时，需要提供一个命令行参数，该参数是需要打开的 CAN 口名，如：“can1”、“can2”等。如需要通过 can1 口进行数据收发，在命令行下执行如下命令：

```
cd ~/hardware_test/can/
./can_test CAN1
```

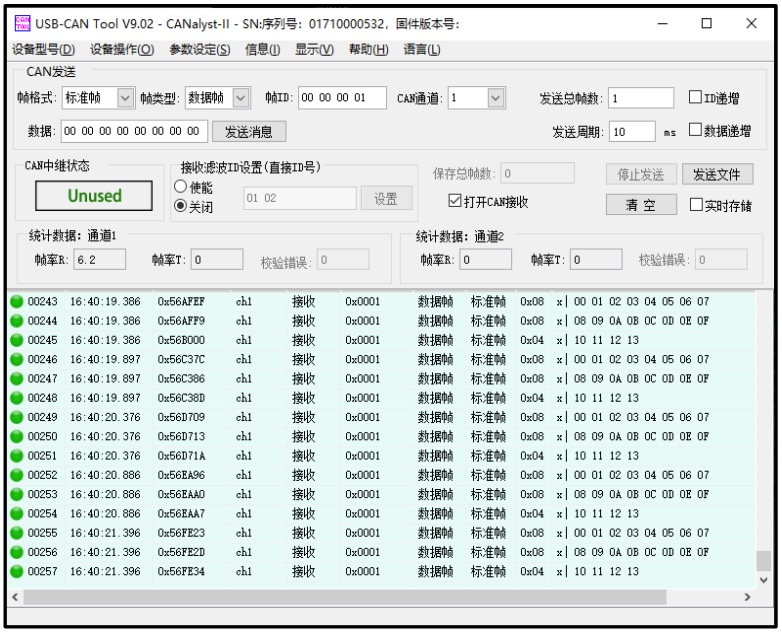
命令执行后，开发板 CAN1 上会一直对外发送数据：

```
root@xinluyao-imx6ull:~/hardware_test/can# ./can_test CAN1
flexcan 2090000.can can0: writing ctrl=0x0e312005
can write: length=8, packet: 00 01 02 03 04 05 06 07
can write: length=8, packet: 08 09 0A 0B 0C 0D 0E 0F
can write: length=4, packet: 10 11 12 13
can write: length=8, packet: 00 01 02 03 04 05 06 07
can write: length=8, packet: 08 09 0A 0B 0C 0D 0E 0F
can write: length=4, packet: 10 11 12 13
can write: length=8, packet: 00 01 02 03 04 05 06 07
can write: length=8, packet: 08 09 0A 0B 0C 0D 0E 0F
can write: length=4, packet: 10 11 12 13
```

在 Windows 下打开 CAN 分析仪进行测试，波特率设置为 125k bps。



打开 CAN 分析仪后，会接收到如下数据：



2.19 时钟设置

Linux 将时钟分为系统时钟(System Clock)和硬件时钟(Real Time Clock，简称 RTC)两种。系统时钟是系统维护的时钟，用户一般使用和看到的都是系统时钟。而硬件时钟则是由主板上的电池供电的时钟。系统时钟在系统断电后即会消失，但 RTC 时钟在主板电池有电的情况下会长期运行。因此每次上电时，Linux 内核都会读取主板上的 RTC 时钟，并将它同步到系统时钟。下面列出一些与时钟相关的命令：

1、设置系统时钟

使用 date -s 命令可以设置系统时钟：

如要将当前时钟设置为 2019-10-19 21:57:00，可以使用如下命令：

```
date -s "2019-10-19 21:57:00"
```

```
root@xinluyao-imx6ull:~# date -s "2019-10-19 21:57:00"
Sat Oct 19 21:57:00 UTC 2019
```

2、查看系统时钟

使用 date 命令可以查看系统时钟：

```
date
```

```
root@xinluyao-imx6ull:~# date
Sat Oct 19 21:57:22 UTC 2019
```

3、设置 RTC 时钟

使用 hwclock -w，可以将系统时钟写入 RTC 时钟：

```
hwclock -w
```

4、查看 RTC 时钟

使用 hwclock 命令可以查看 RTC 时钟：

```
hwclock
```

```
root@xinluyao-imx6ull:~# hwclock
Fri Nov 22 15:34:29 2019  0.000000 seconds
```

5、同步系统时钟

使用“hwclock -s”，可以将 RTC 时钟写入系统时钟。一般在系统刚开机时运行此命令。

```
hwclock -s
```

2.20 音频测试

HD-IMX6ULL-MB 使用了 i.MX6ULL 芯片的 MQS 音频输出功能。

在开发板/home/root/hardware_test/audio 目录下，运行 audio.sh 程序可以播放音乐。直接接入耳机即可测试。

2.21 USB WIFI 测试

由于 IMX6ULL 芯片只有两个 SDIO 控制器，这两个控制器一个连接 eMMC Flash，一个连接 SD 卡。因此，IMX6ULL 一般使用 USB 接口的 WIFI 模块。开发板板载了一个 RTL8188FTV USB Wifi 模块，并提供了驱动程序。

2.21.1 查看 USB 设备

插入 USB WIFI 后，查看是否识别到 USB WIFI

```
root@xly-imx6ull:~# lsusb
Bus 001 Device 003: ID 0483:5750 STMicroelectronics
Bus 001 Device 004: ID 0bda:f179 Realtek Semiconductor Corp.
Bus 001 Device 002: ID 0424:2514 Standard Microsystems Corp. USB 2.0 Hub
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

对于 RTL8188FTV 模块，其 ID 为 0bda:f179。

2.21.2 检查驱动

```
dmesg | grep "rtl8188fu"
```

```
root@xly-imx6ull:~# dmesg| grep rtl8188fu
[  659.131047] RTW: rtl8188fu v5.7.4.1_35666.20191106
[  660.029644] usbcore: registered new interface driver rtl8188fu
```

dmesg 命令可以用来查看命令行中曾经打印的消息。我们在打印消息中搜索“rtl8188fu”查看驱动是否已加载成功。看到“registered new interface driver rtl8188fu”表示驱动加载成功。

2.21.3 配置 wifi 密码

在连接网络前，需要先设 WIFI 网络名称和密码。使用 wpa_passphrase 生成需要的配置文件。当 WIFI 名称为 xinluyao，密码为 123456 时：

```
root@xly-imx6ull:~# wpa_passphrase "xinluyao" > wifi.conf
123456
```

2.21.4 连接网络

使用 wpa_supplicant 命令可以连接 WIFI 网络。在运行 wpa_supplicant 命令前，先确定没有一个正在运行的 wpa_supplicant。

```
ps | grep wpa_supplicant
```

```
root@xly-imx6ull:~# ps |grep wpa_supplicant
root@xly-imx6ull:~#
```

如果没有搜索到任何结果，表明没有正在运行的 wpa_supplicant，可以使用如下命令运行 wpa_supplicant。

```
wpa_supplicant -B -c wifi.conf -i wlan0
```

“-B”指定在后台运行，“-c wifi.conf”指定配置文件，“-i wlan0”指定网口设备。

```
root@xly-imx6ull:~# wpa_supplicant -B -c wifi.conf -i wlan0
Successfully initialized wpa_supplicant
rfkill: Cannot open RFKILL control device
IPv6: ADDRCONF(NETDEV_UP): wlan0: link is not ready
root@xly-imx6ull:~# IPv6: ADDRCONF(NETDEV_CHANGE): wlan0: link becomes ready
```

2.21.5 获取 IP 地址

```
udhcpc -i wlan0
```

```
root@xly-imx6ull:~# udhcpc -i wlan0
udhcpc: started, v1.30.1
udhcpc: sending discover
RTW: rtl8188f_fill_default_txdesc(wlan0): SP Packet(0x0800) rate=0x0 SeqNum = 29
RTW: rtw_set_ps_mode(wlan0) Leave 802.11 power save - WIFI-LPS_CTRL_SPECIAL_PACKET
RTW: OnAction_back
RTW: OnAction_back, action=0
RTW: rtl8188f_set_FwPwrMode_cmd(): FW LPS mode = 0, SmartPS=2
RTW: issue_addba_rsp_wait_ack(wlan0) ra=34:ce:00:52:ba:d0 status:=0 tid=0 size:64,
udhcpc: sending select for 192.168.1.80
RTW: rtl8188f_fill_default_txdesc(wlan0): SP Packet(0x0800) rate=0x0 SeqNum = 31
udhcpc: lease of 192.168.1.80 obtained, lease time 43200
/etc/udhcpc.d/50default: Adding DNS 192.168.1.1
```

从上图看到，已经成功分配 IP 地址“192.168.1.80”。现在，我们就可以通过这个地址，使用 SSH 登录到开发板了。

2.21.6 设置 DNS

如果需要访问互联网，可以设置 DNS 地址。

```
echo "nameserver 114.114.114.114" > /etc/resolv.conf
```

2.22 设置 WIFI 自动连接

上一章介绍了如何测试 USB WIFI，不过很多时候，我们需要 Wifi 在系统开机就启动，这就需要我们设置 WIFI 开机启动。

2.22.1 新建配置文件

新建一个配置文件，并写入账户密码信息，账户名为"xinluyao"，密码为"123456"。用户根据实际情况修改 WIFI 账户和密码。

```
mkdir /etc/wpa_supplicant
touch /etc/wpa_supplicant/wpa_supplicant-wlan0.conf
wpa_passphrase xinluyao 123456 > /etc/wpa_supplicant/wpa_supplicant-wlan0.conf
```

2.22.2 启动 wps_supplicant service

```
systemctl enable wpa_supplicant@wlan0
```

```
root@xly-imx6ull:~# systemctl enable wpa_supplicant@wlan0
Created symlink /etc/systemd/system/multi-user.target.wants/wpa_supplicant@wlan0.service → /
lib/systemd/system/wpa_supplicant@.service.
```

2.22.3 为 WIFI 设备创建新配置文件

```
touch /etc/systemd/network/00-wireless-dhcp.network
```

编辑文件内容为如下内容：

```
[Match]
Name=wlan0

[Network]
DHCP=yes
```

2.22.4 启动网络服务

```
systemctl enable systemd-networkd.service
```

启动网络服务后重启下 wpa_supplicant 服务和 systemd-networkd 服务

```
systemctl restart wpa_supplicant@wlan0
systemctl restart systemd-networkd.service
```

这样设置好后，再次重新启动，开发板会自动启动 WIFI 并连接上路由器。

3 开发准备

通过以上章节，相信大家已经对开发板有一定的熟悉，已经迫不及待想要进行开发了。在开发前还需要做好如下准备工作。

3.1 Linux 环境安装

进行 linux 开发需要安装一个 linux 系统环境，一般使用 VMWare 安装 Ubuntu 系统进行开发。

3.1.1 VMware 软件安装

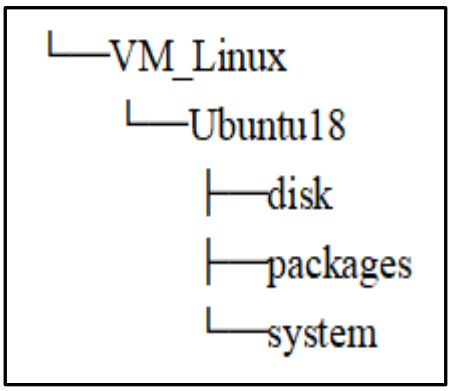
VMWare 软件存放在“工具/Windows/VMWare”目录下，文件名为“VMware-workstation-full-15.5.0-14665864.exe。”

安装完成后，输入“VMware 密钥.txt”文档中的密钥进行激活。

注：仅用于学习用途，如需商用，请购买正版，或使用 Vitrual Box 替代。

3.1.2 Ubuntu 安装

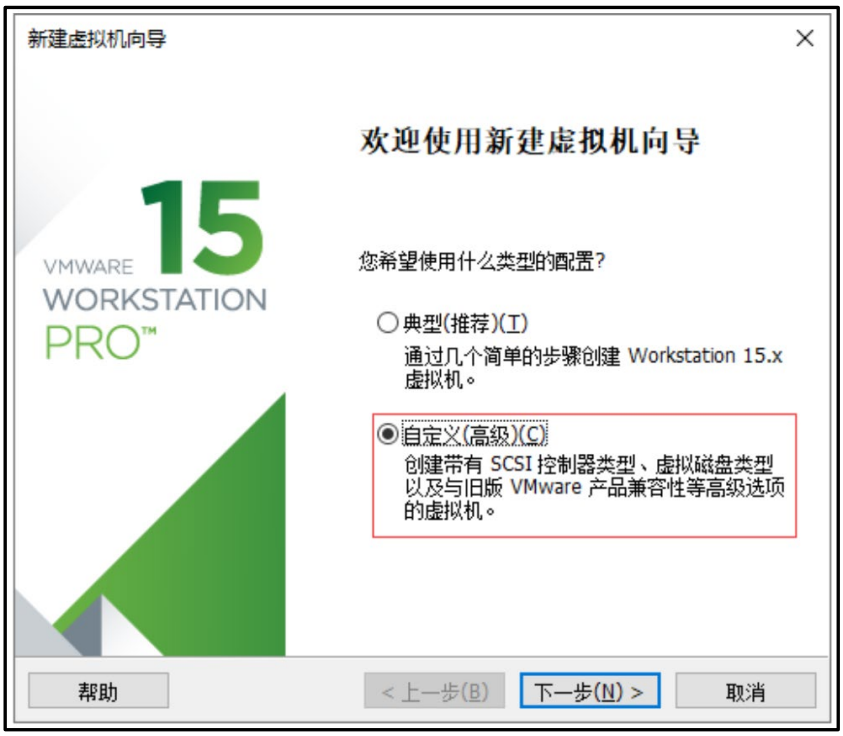
笔者电脑 C 盘是固态硬盘，而且容量较大，所以一般将虚拟机的配置文件存放在 C 盘。在 C 盘新建 system、packages 和 disk 文件夹，文件夹结构如下：



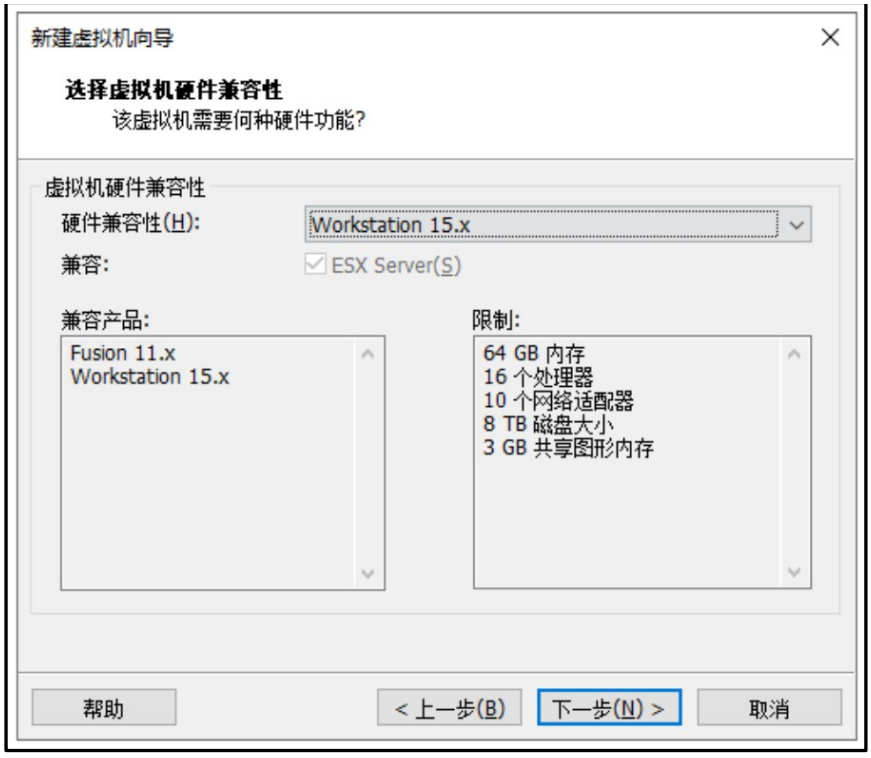
其中 system 用来存放虚拟机的配置文件，packages 用来存放一些安装包，disk 用来存放虚拟硬盘文件。

下面是详细安装过程：

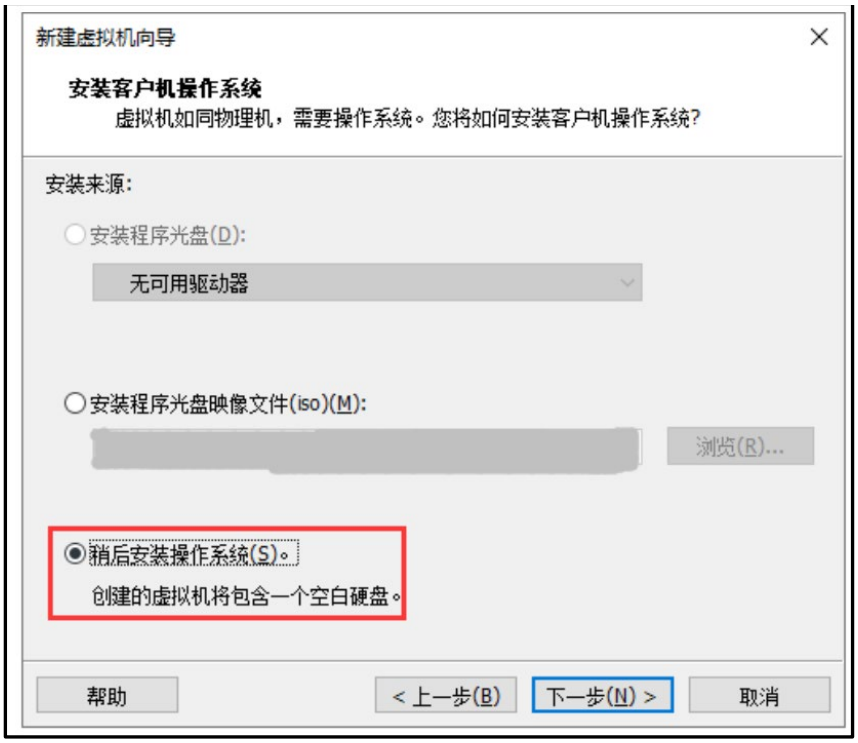
1） 打开 VMware 软件，新建虚拟机，选择自定义模式进行安装。



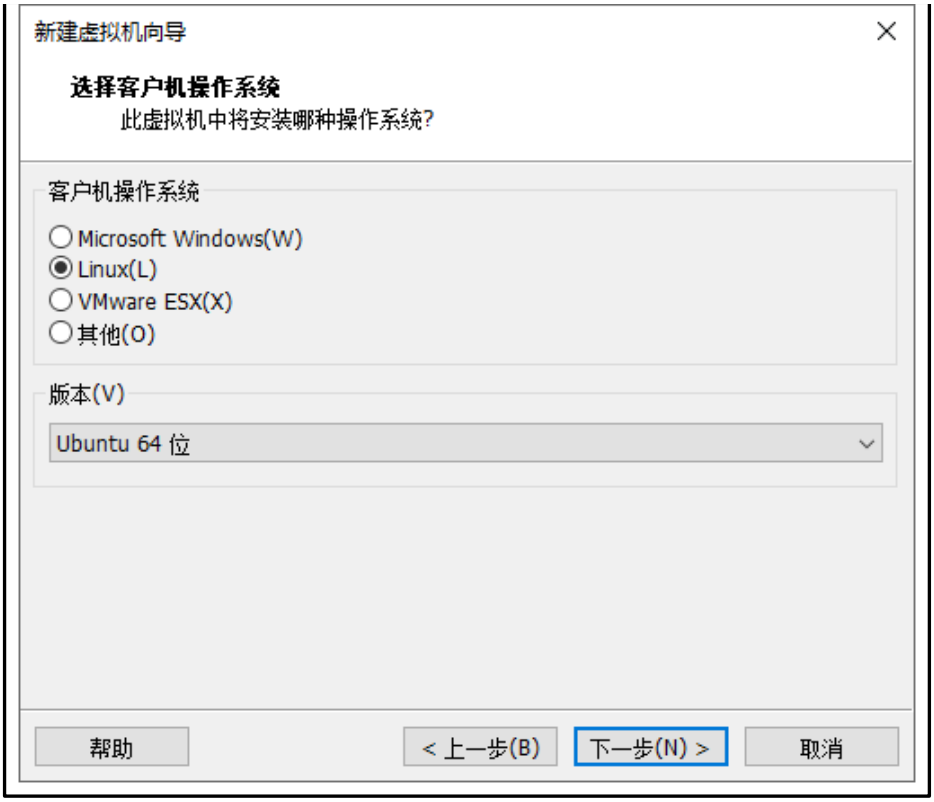
2） 硬件兼容性选择 Workstation 15.x 即可。



3） 选择“稍后安装操作系统”。



4） 客户操作机系统选择 Linux(L)，版本选择 Ubuntu 64 位。

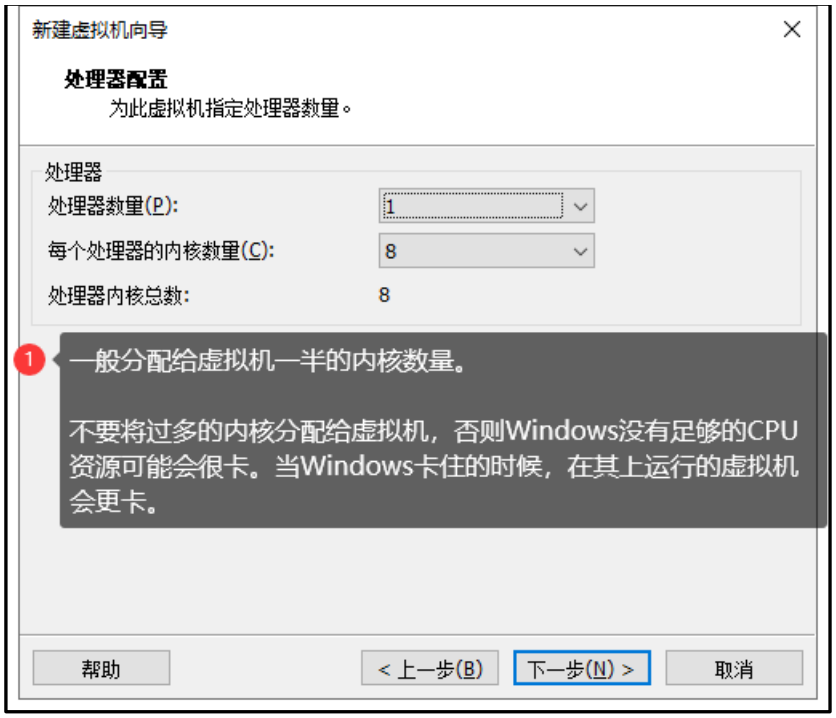


5） 虚拟机名称可自由设置，笔者这里设置为“Ubuntu18”。“位置(L)”是设置存放虚拟机配置和备份文件的位置，选择之前新建的“C:

\VM_Linux\Ubuntu18\System”目录。

注：此文件夹不会存放 Ubuntu 系统，占用磁盘 10GB 左右大小。

6） 给 Ubuntu 系统分配处理器资源，笔者建议将一半内核资源分配给 Ubuntu。

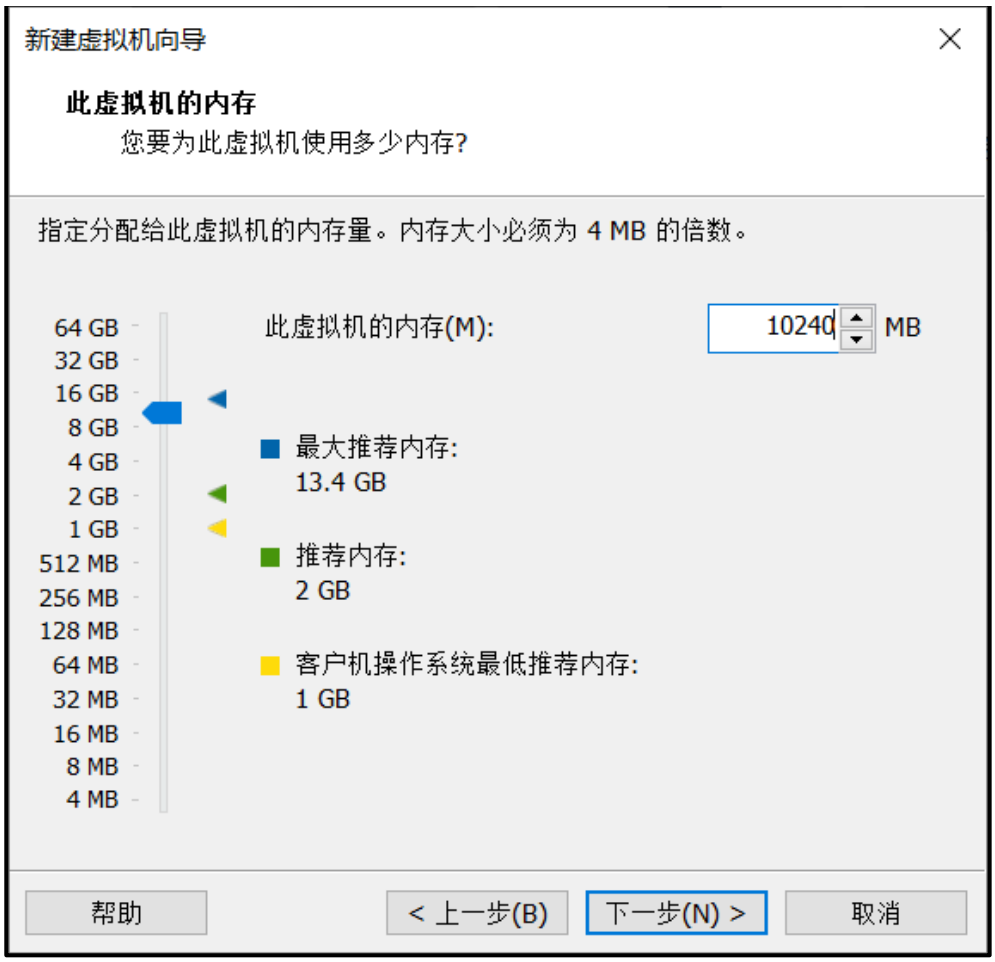


7） 给 Ubuntu 系统分配内存资源。

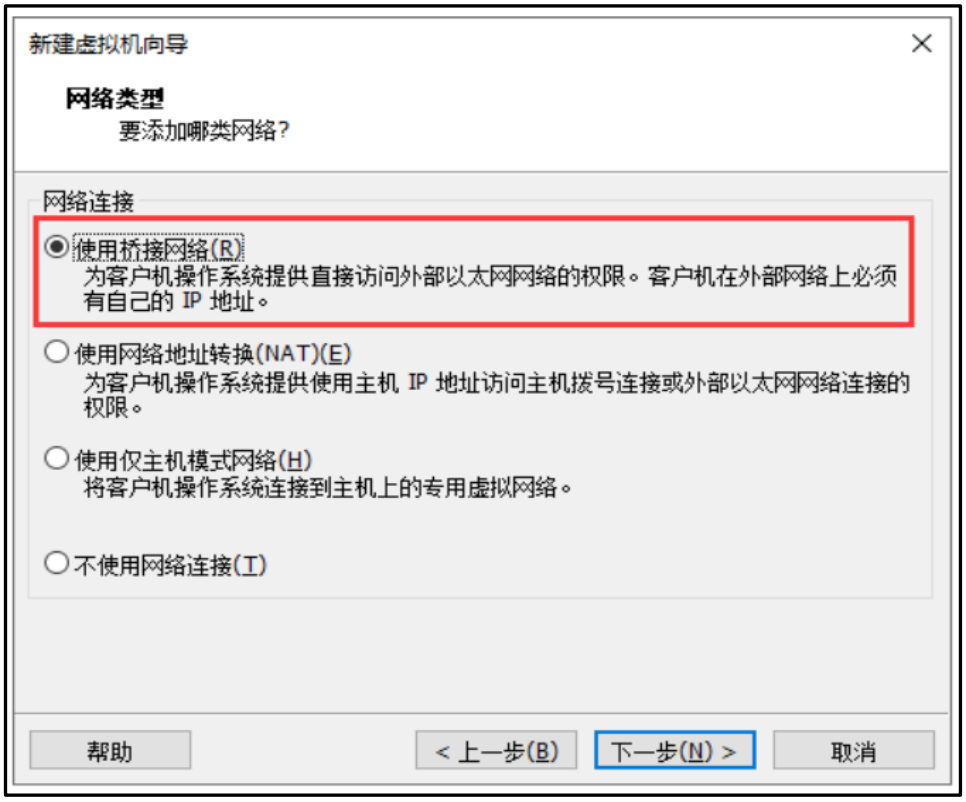
这一步需要注意：请给 Windows 系统预留足够的内存，否则宿主机(Windows)比较卡，虚拟机有再多内存也一样会卡。笔者电脑一共 16G 内存，这里

给虚拟机分配 10G 内存，Windows 系统预留 6G 内存。

注意：编译 Yocto 需要至少 8G 内存，当使用 VMWare 虚拟机时，显存会占用 768M 内存，所以如果需要编译 Yocto 至少分配 9G 内存。



8) 设置 Ubuntu 的网络连接方式。笔者建议将电脑连接至路由器，将 Ubuntu 的网络连接模式设置为桥接模式，这样 Windows 系统和 Ubuntu 系统都可以访问外网，并且有独立的 IP 地址，方便后面开发。



9） 设置 Ubuntu 的 I/O 控制器类型，默认即可。

10） 设置虚拟机磁盘类型

由于是虚拟磁盘，所以这个类型错误并不会导致磁盘出现无法访问，只是会降低访问性能。一般建议使用机械硬盘将虚拟磁盘设置为 SATA(A)， NVMe 接口固态硬盘将虚拟磁盘设置为 NVMe(V)。

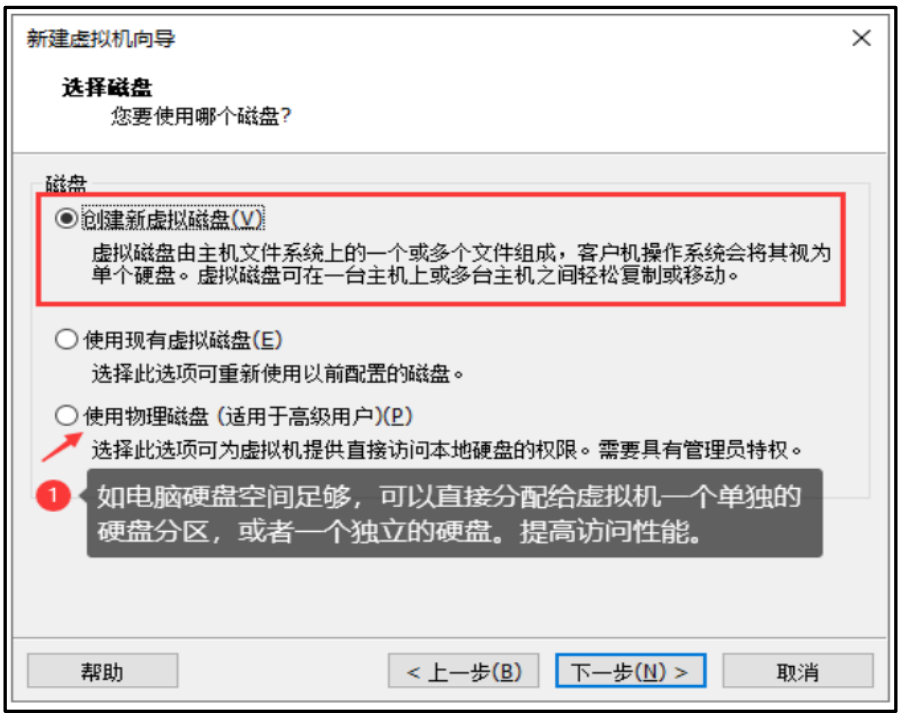
当磁盘设置为 NVMe，磁盘在/dev 目录下文件名为/dev/nvme0n1。当为 SATA 类型的磁盘时，磁盘为/dev/sdx(一般为 sda)。

因为 U 盘等外接设备在/dev/目录下文件名也为/dev/sdx（x 为 a、b、c 等），当磁盘类型为 SATA 时，操作 SD 卡或者 U 盘要仔细确认设备名。

11） 创建虚拟磁盘。

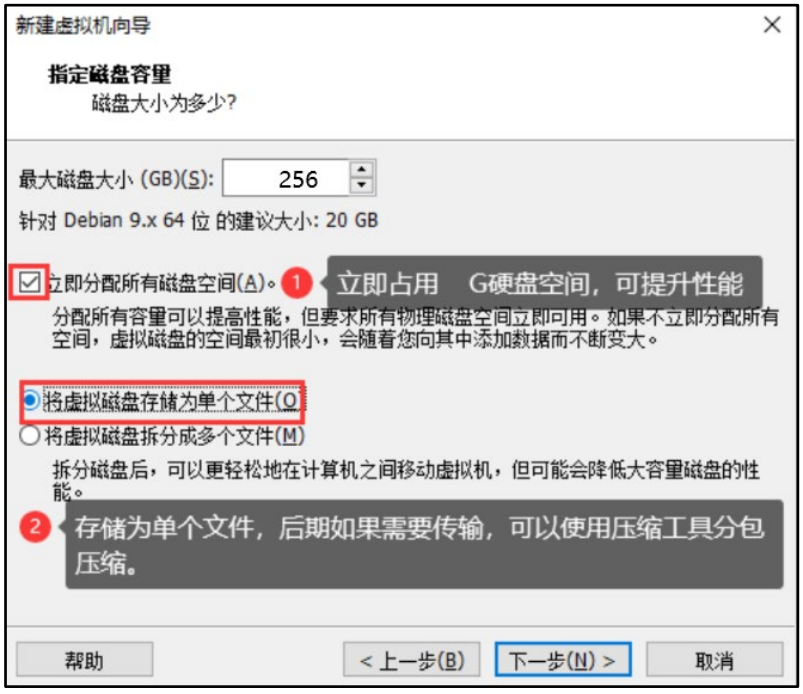
这个操作会在 Windows 管理的文件系统中创建 disk 文件，并占用较大磁盘空间。

如有空闲的硬盘或分区，可以将整个硬盘或分区分配给虚拟机 Ubuntu 使用，这时 Ubuntu 独占分配的整个硬盘或分区，可以提高 Ubuntu 硬盘访问性能。



12） 分配给虚拟机 Ubuntu 磁盘空间

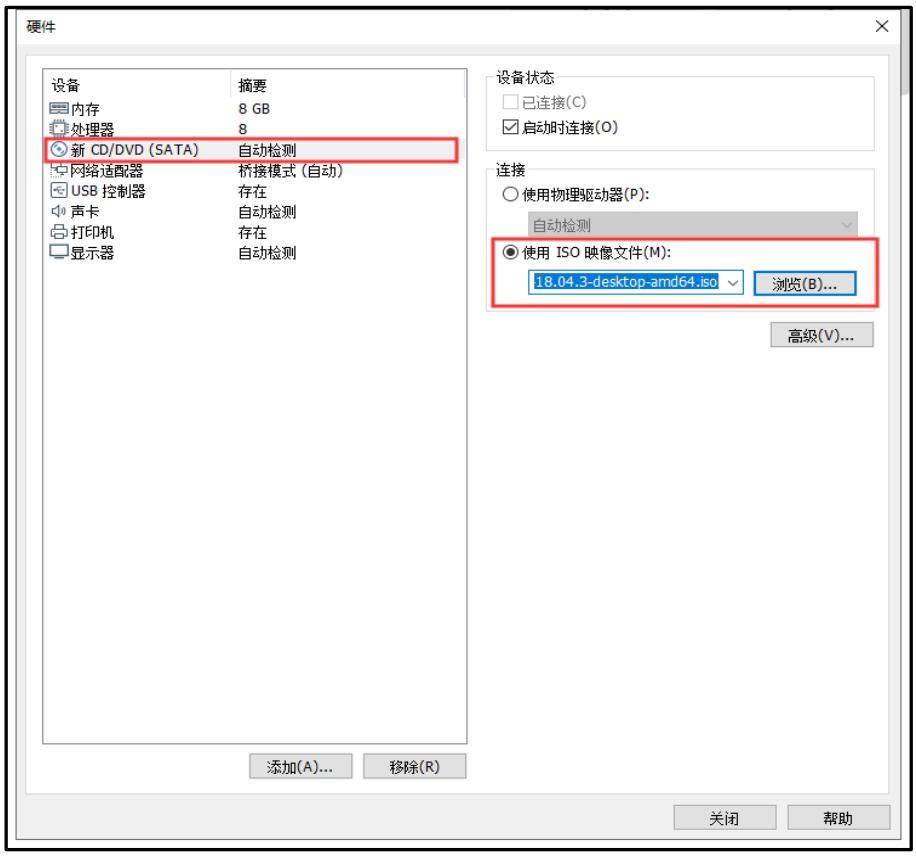
如硬盘剩余空间充足，可以适当增加分配的磁盘空间。



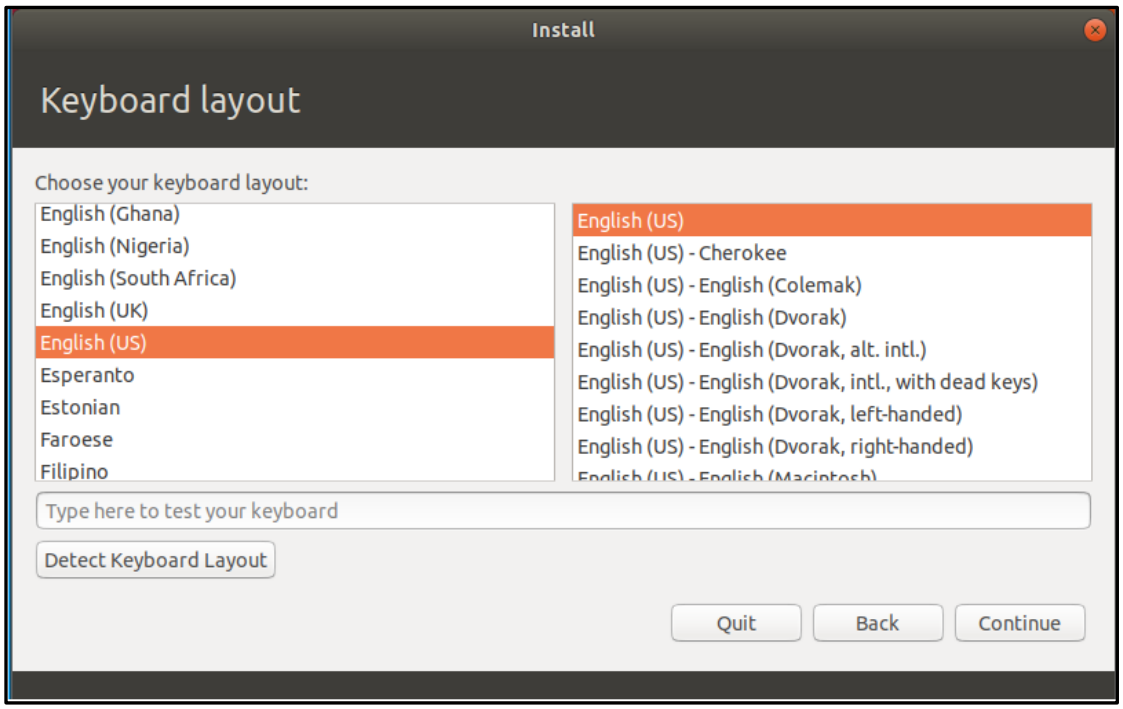
笔者这里选择直接分配所有磁盘空间，这样能提高虚拟磁盘性能，但这样会立即占用所有分配给虚拟机的磁盘空间。如您磁盘空间比较紧张，可不勾选“立即分配所有磁盘空间”选项。这样起初只会占用很少的磁盘空间，随着使用逐渐扩大对磁盘的占用，但是不会超过设置的最大磁盘大小。

13） 指定磁盘文件存储路径。

14) 自定义硬件，添加 Ubuntu 镜像 iso 文件。

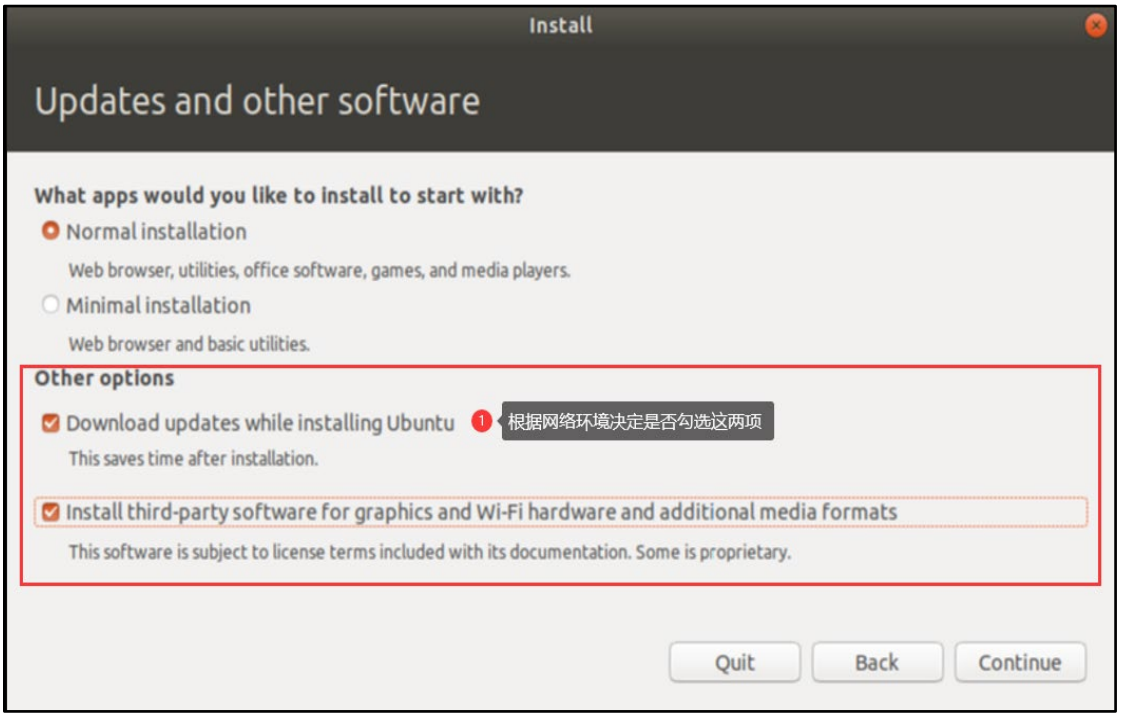


15） 启动虚拟机，开始安装 Ubuntu。

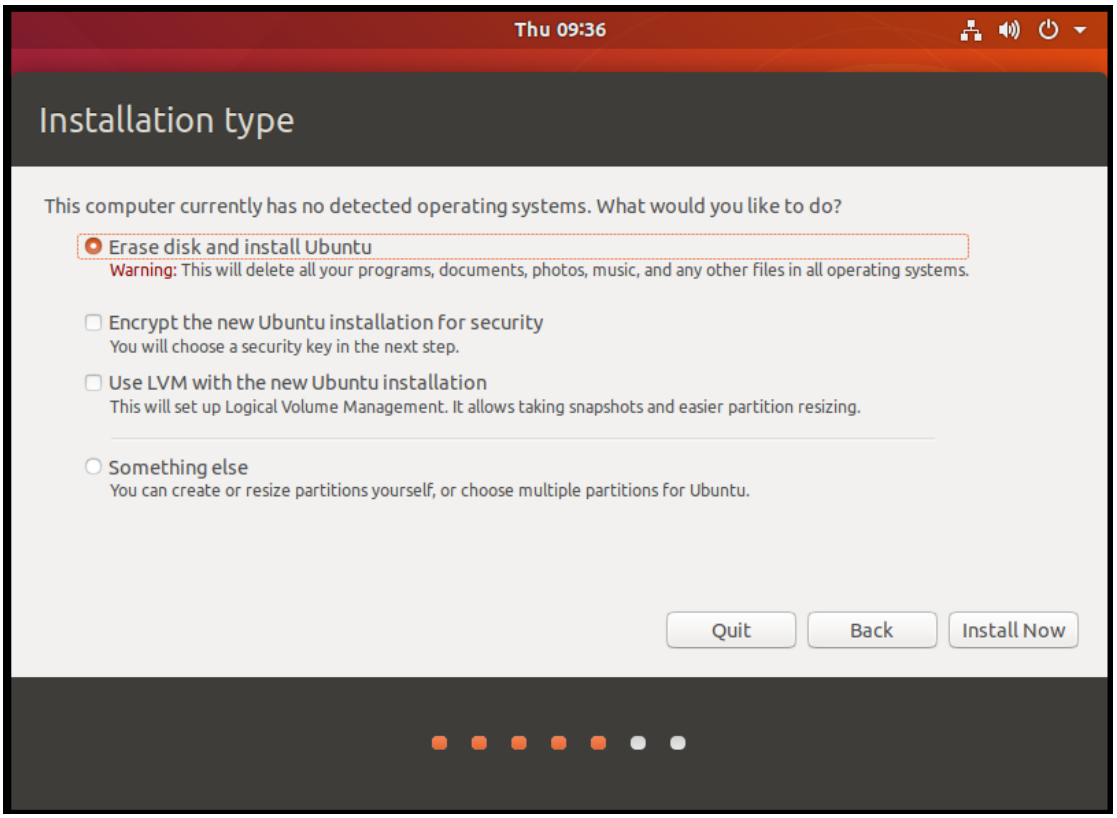


16） 安装时更新

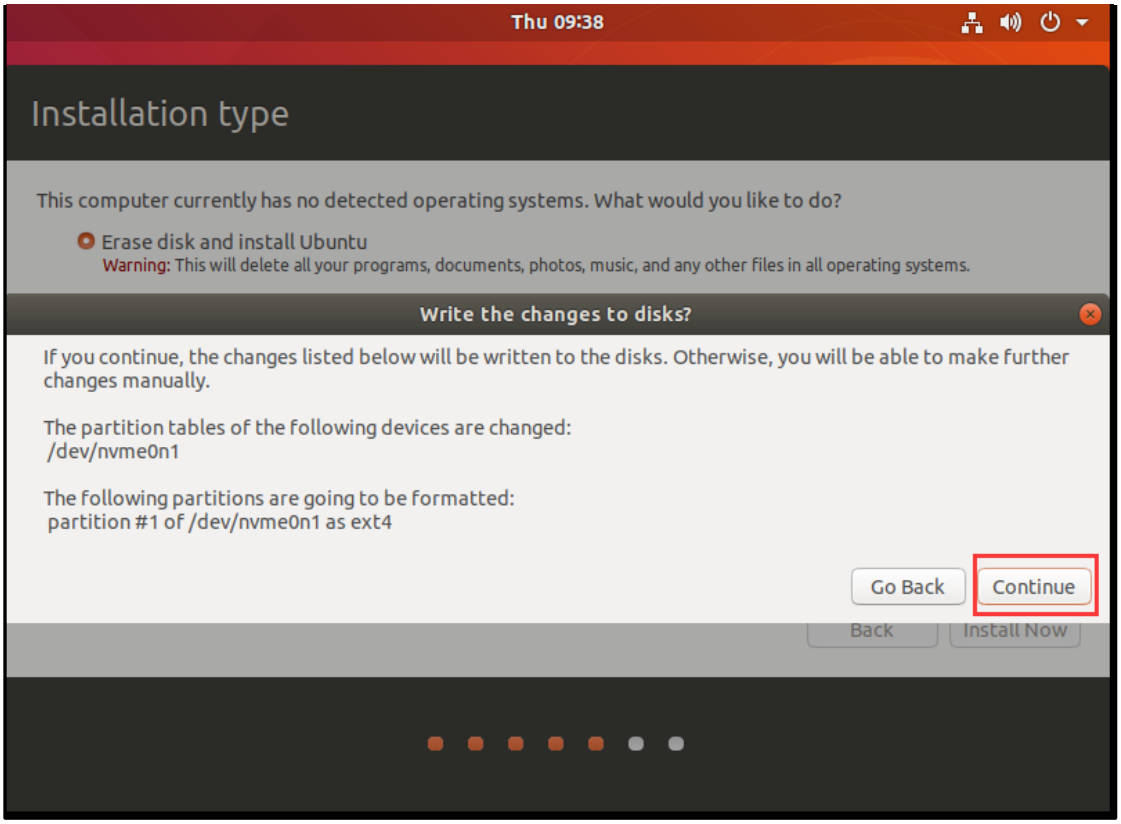
对于初学者，建议在安装时就下载更新，方便安装完成后使用。对于有经验用户，可不勾选 Other Options，在系统安装完成后自行安装相应组件。



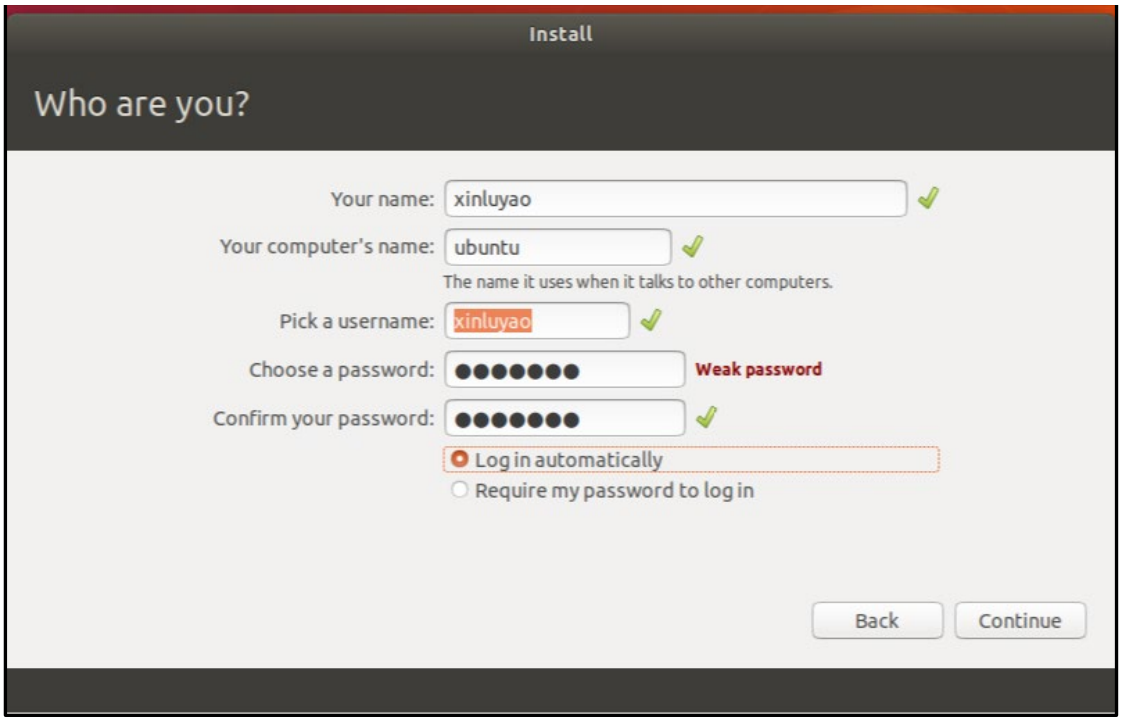
17） 擦除整个虚拟磁盘安装 Ubuntu。



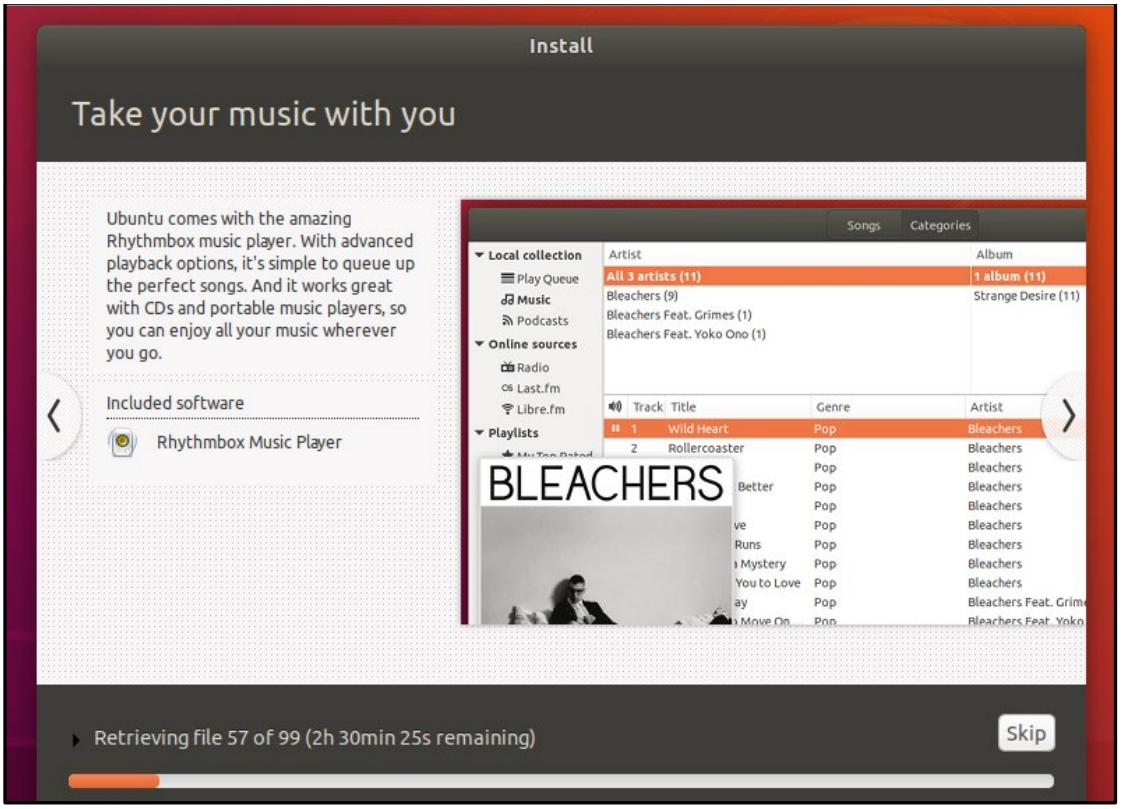
18） Ubuntu 会将虚拟磁盘格式化为 ext4 格式。



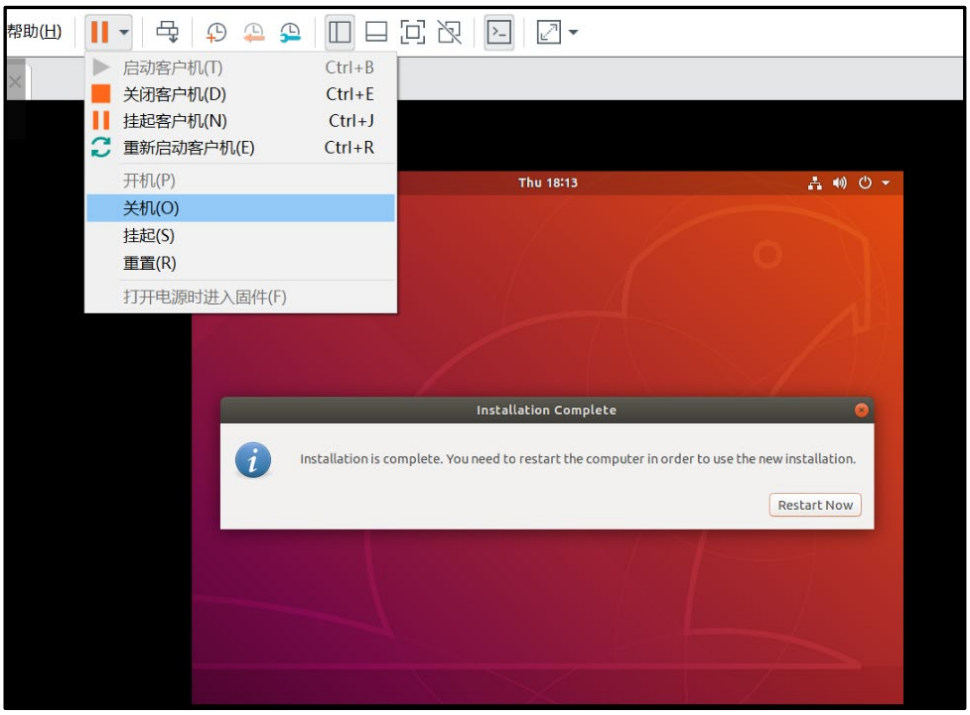
19） 设置用户名和密码。



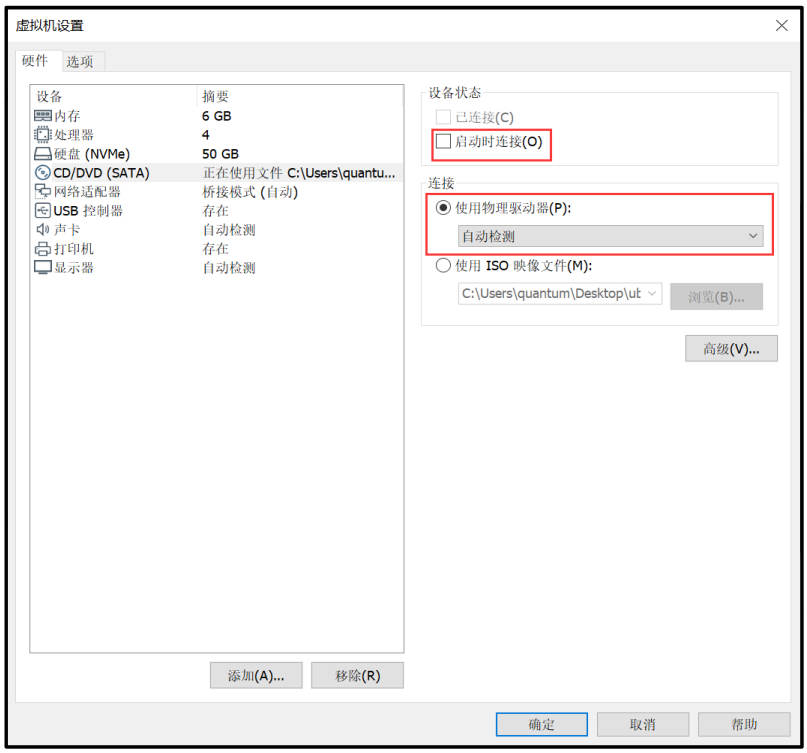
20） 等待系统安装组件。



21） 安装完成后，提示重新启动计算机。这里不选择重启，而是直接关闭计算机。



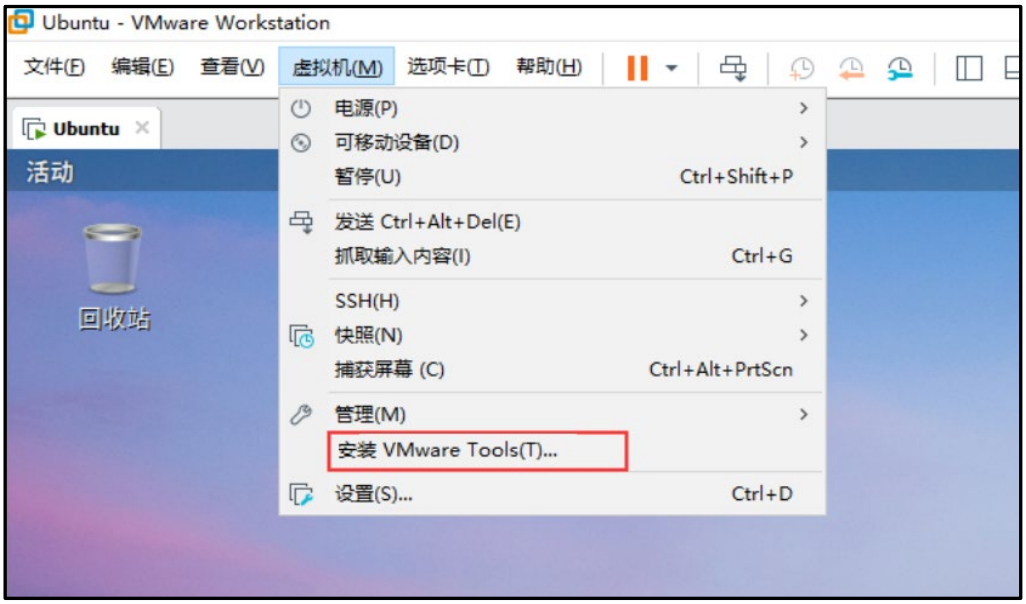
22） 选择 CD/DVD，不再使用 ISO 映像文件，防止开机后再次直接进入系统安装界面。



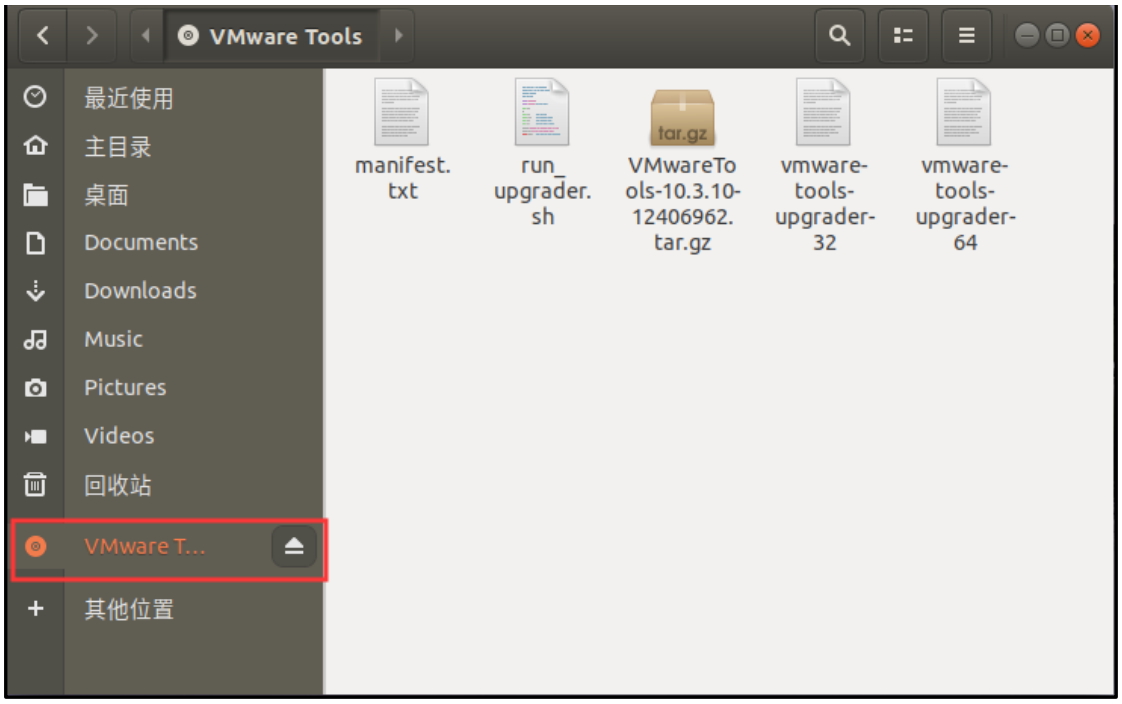
3.1.3 VMware tools 安装

VMware Tools 的主要作用是虚拟硬件的驱动。安装 VMware 后，鼠标能从 Windows 无缝移入和移出 Ubuntu，Windows 与 Ubuntu 剪贴板共享，虚拟机显示效果会优化，并且能使用拖动的方式在 Windows 和 Ubuntu 之间拷贝文件，十分方便。

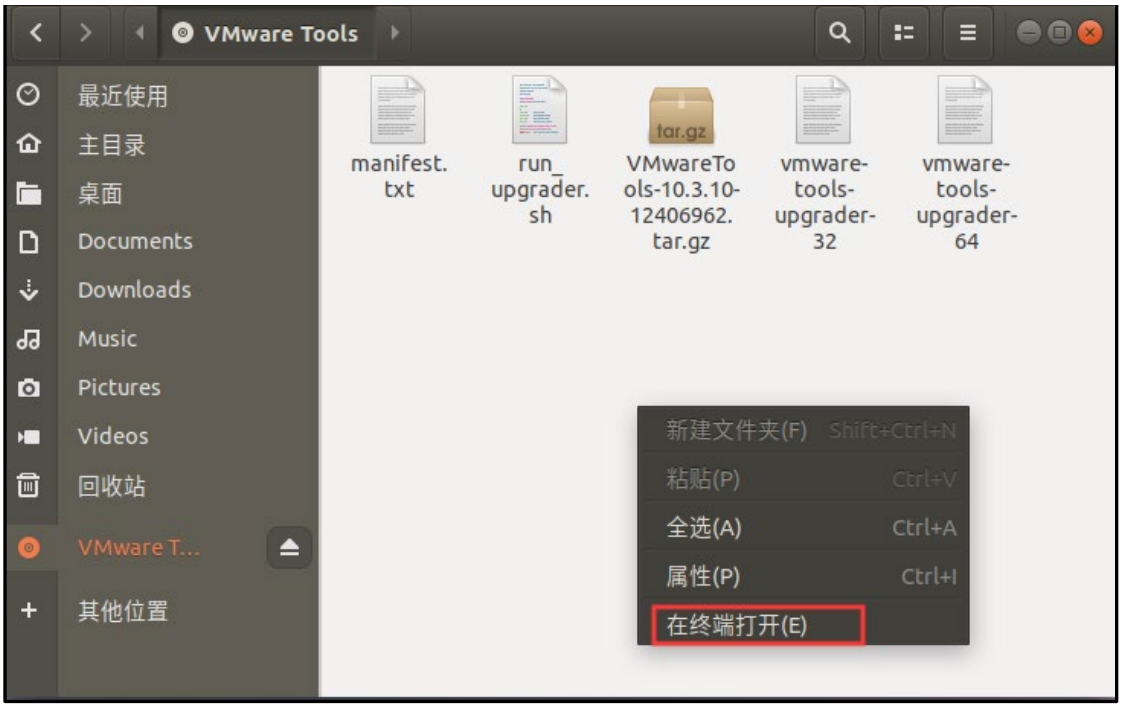
1） 选择虚拟机->安装 VMware Tools。



2） VMWare 中会自动挂载上 VMWareTools。



3) 打开 VMware Tools 文件夹，在空白处单击鼠标右键，选择“在终端打开(E)”。



4) 在弹出终端中输入命令进行安装。

```
xinluyao@ubuntu:/media/xinluyao/VMware Tools$ cp VMwareTools-10.3.10-12406962.tar.gz ~/Downloads/
xinluyao@ubuntu:/media/xinluyao/VMware Tools$ cd ~/Downloads/
xinluyao@ubuntu:~/Downloads$ tar xzf VMwareTools-10.3.10-12406962.tar.gz
xinluyao@ubuntu:~/Downloads$ cd vmware-tools-distrib/
xinluyao@ubuntu:~/Downloads/vmware-tools-distrib$ sudo ./vmware-install.pl
```

第一行直接将安装压缩包拷贝至“~/Download/”目录，第 3 行解压压缩包。最后一行执行安装命令。

由于 VMwareTools 会一直更新，读者安装时，版本号可能已经发生变化，安装前请核对版本号并修改，不要直接复制。

执行完上述命令，会弹出对话框，询问是否要安装，输入 y 后回车确认。

```
xinluyao@ubuntu:~/Downloads/vmware-tools-distrib$ sudo ./vmware-install.pl
open-vm-tools packages are available from the OS vendor and VMware recommends
using open-vm-tools packages. See http://kb.vmware.com/kb/2073803 for more
information.
Do you still want to proceed with this installation? [yes] y
```

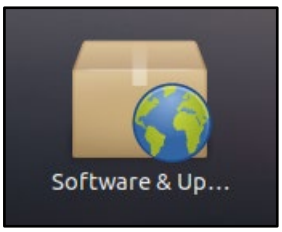
开始安装后，命令行中会出现很多选项，询问“yes or no”，读者可以在需要回答的地方全部直接回车，使用默认选项进行安装。

安装成功出现如下界面，Ubuntu 桌面自动会铺满整个 VMware 窗口。如果界面没有自动发生变化，您可以在命令行输入 reboot 重启下 Ubuntu 系统来使 VMware Tools 生效。

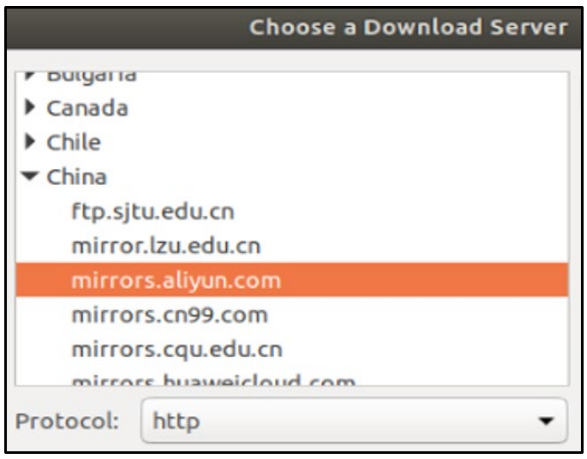
```
The configuration of VMware Tools 10.3.10 build-12406962 for Linux for this
running kernel completed successfully.

You must restart your X session before any mouse or graphics changes take
effect.

To enable advanced X features (e.g., guest resolution fit, drag and drop, and
file and text copy/paste), you will need to do one (or more) of the following:
1. Manually start /usr/bin/vmware-user
2. Log out and log back into your desktop session
3. Restart your X session.

^[^AFound VMware Tools CDROM mounted at /media/xinluyao/VMware Tools. Ejecting
device /dev/sr0 ...
Enjoy,

--the VMware team
```

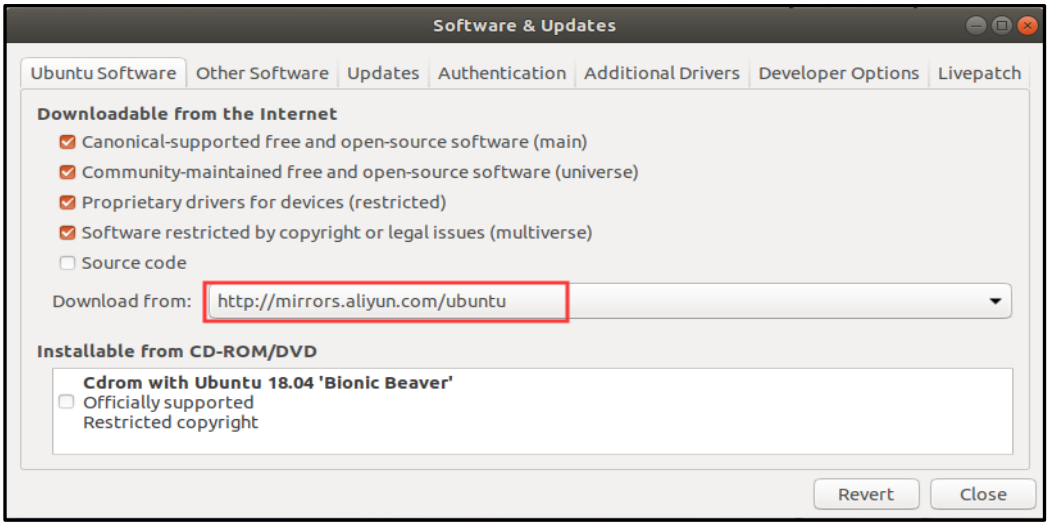
3.1.4 Ubuntu 软件源设置

1） 在所有应用中找到“Software & Update”(软件和更新)



1） 点击 Download from 下拉选框，将更新服务器设置为 China -> mirros.aliyun.com。这是阿里云的服务器，速度更快。





2） 选好更新服务器后，对系统进行更新。

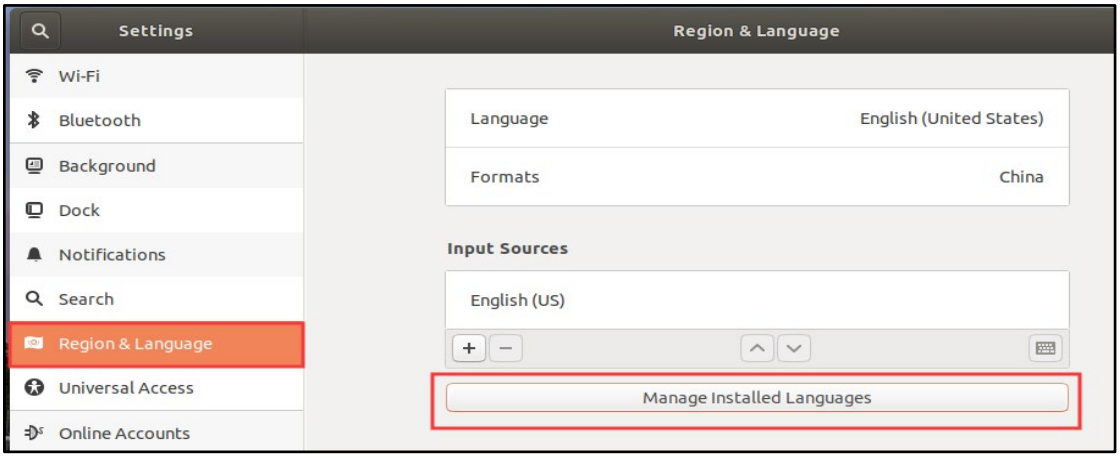
打开终端(Ubuntu 下快捷键 Ctrl+Alt+T)，输入如下命令，更新系统。

```
sudo apt update
sudo apt upgrade
```

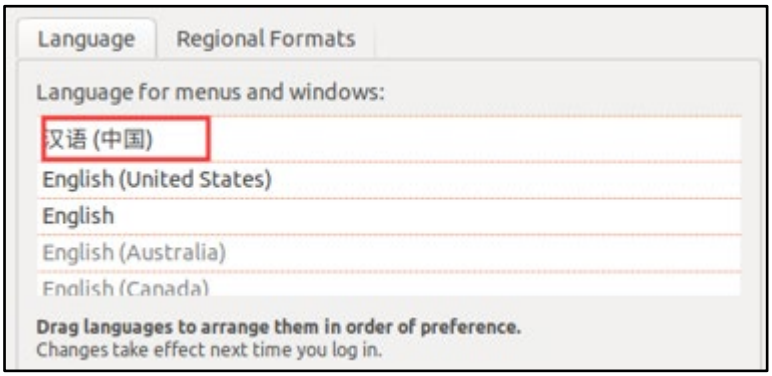
3.1.5 中文支持

1） 在所有应用中找到“Settings(设置)”

2) 设置“Region & Language(区域和语言)”。进入“Manage Installed Languages（管理已安装的语言）”

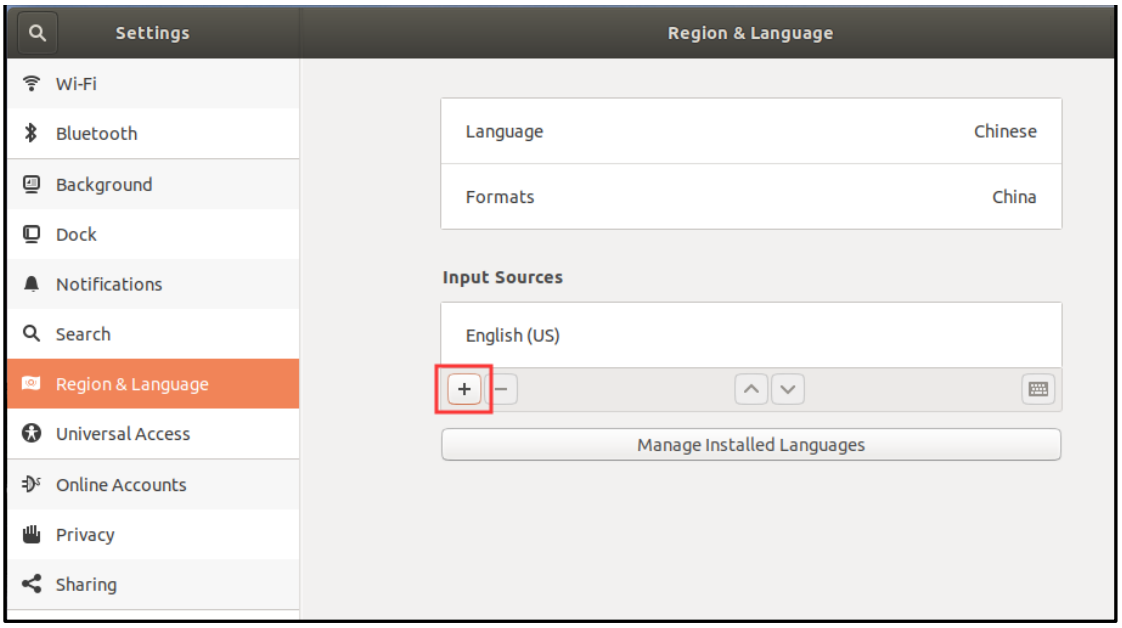


3) 将“汉语(中国)”拖动到“Language for menus and windows（菜单和窗口的语言）”选项卡的最上方，设置菜单和窗口显示为中文。



4) 设置好后，重启一次系统，使语言设置生效。

5) 重新进入“Settings->Region & Language”，设置“Input Sources”（输入源）。



6) 添加中文输入法 Chinese(Intelligent Pinyin)。

如界面中没有出现“Chinese(Intelligent Pinyin)”选项，请再次确认在设置完成后，需要重启下系统。

3.1.6 安装必要软件包

在开发过程中需要使用到一些工具，可以使用提供的脚本安装。

1） 将 Congigure_ubuntu.tar.bz2 拷贝至 Ubuntu Downloads 目录。

2） 解压 Congigure_ubuntu.tar.bz2。

```
tar xvjf Congigure_ubuntu.tar.bz2
```

因 Ubuntu 和 Windows 对于回车处理不一致，请不要在 Windows 下解压，否则脚本执行会报错。

3） 将 Configure_ubuntu.sh 拷贝至 Ubuntu Downloads 目录。

4） 打开命令行（快捷键为“Ctrl+Alt+T”，即同时按下 Ctrl、Alt、T 三个键）

5） 给文件添加执行权限：

```
chmod +x Configure_ubuntu.sh
```

6） 执行脚本

```
xinluyao@xinluyao:~/Downloads$ ./Configure_ubuntu.sh
Ubuntu Configure Tool V1.1.0
开始自动配置Ubuntu
Network OK.
[sudo] xinluyao 的密码：
```

提示输入密码时，输入 Ubuntu 密码（为了安全，ubuntu 系统中输入密码时，不会有任何显示，直接输正确密码回车即可。），系统会自动开始安装常用软件。

7） 安装完成

安装完成后，输出如下信息：

```
============================================================
==  Configure for iMX6ULL Linux development complete!  ==
============================================================
TFTP         PATH: /tftpboot
IMX6ULL NFS  PATH: /srv/imx6ull/rootfs
IMX6DL  NFS  PATH: /srv/imx6dl/rootfs
STM32MP NFS  PATH: /srv/stm32mp/rootfs
SAMBA PATH: /
```

3.1.7 交叉工具链安装

linux 系统下一般使用 gcc 等一系列工具来编译 c 程序，这一些列工具我们称为工具链。PC 系统自带的工具链仅能编译 PC 使用的程序。而嵌入式系统资源匮乏，系统性能较低，编译速度会很慢。所以一般会安装一个交叉工具链。使用交叉工具链编译的软件，可以在嵌入式系统上运行。

3.1.7.1 安装工具链

芯路遥团队提供了 crosstools.tar.bz2 压缩包，执行如下命令可安装交叉工具链：

```
tar xvjf crosstools.tar.bz2
cd crosstools
./install_crosstools.sh
```

执行 crosstools 脚本，选择需要安装的工具链，脚本会自动安装。

```
xinluyao@ubuntu:~/Downloads/crosstools$ ./install_crosstools.sh
Please select the board use:
    1. Install iMX6ULL Qt Cross Toolchain
    2. Install iMX6DL  Qt Cross Toolchain
    3. Install STM32MP Qt Cross Toolchain
    4. Quit
please input your choice:1 1 输入1，选择安装imx6ull的工具
sdk/
sdk/fsl-imx-x11-glibc-x86_64-meta-toolchain-qt5-cortexa7t2hf-neon-toolchain-xly-warrior.target.manifest
sdk/fsl-imx-x11-glibc-x86_64-meta-toolchain-qt5-cortexa7t2hf-neon-toolchain-xly-warrior.sh
sdk/fsl-imx-x11-glibc-x86_64-meta-toolchain-qt5-cortexa7t2hf-neon-toolchain-xly-warrior.host.manifest
sdk/fsl-imx-x11-glibc-x86_64-meta-toolchain-qt5-cortexa7t2hf-neon-toolchain-xly-warrior.testdata.json
NXP i.MX Release Distro SDK installer version xly-warrior
==========================================================
The directory "/opt/imx6ull/fsl-imx-x11/xly-warrior" already contains a SDK for this architecture.
If you continue, existing files will be overwritten! Proceed [y/N]? Extracting SDK.....................
.....................................................................................................
.....................................................................................................
.............................done
Setting it up...done
SDK has been successfully set up and is ready to be used.
```

由于包比较大，安装时间可能会比较久，请耐心等待。安装完成会有如下提示：

```
Install toolchain for imx6ull Ok.
==========================================================
==    Install iMX6ULL Cross Toolchain complete!    ==
==========================================================
```

3.1.7.2 选择工具链

安装完工具链后，下次启动终端，会出现工具链选择界面。可根据需要选择合适的工具链，对于 imx6ull 选择 1 即可，若直接回车，则不使能任何交叉工具链。若需要编译 Yocto 源代码，则需要直接回车。

```
Please select the board use:
    1. Configuring for iMX6ULL Linux development
    2. Configuring for iMX6Q Linux development
    3. Configuring for STM32MP157 Linux development
Enter. Quit
please input your choice:1
=============================================================
====    Configuring for iMX6ULL Linux Dev complete!    ====
=============================================================
```

3.1.7.3 测试工具链

输入$CC -v 查看，如成功显示交叉工具链版本则表示安装成功：

```
xinluyao@ubuntu:~$ $CC -v
Using built-in specs.
COLLECT_GCC=arm-poky-linux-gnueabi-gcc
COLLECT_LTO_WRAPPER=/opt/imx6ull/fsl-imx-x11/4.19-warrior/sysroots/x86_64-pokysdk-linux/usr/libexec/arm-poky-linux-gnueabi/gcc/arm-poky-linux-gnueabi/8.3.0/lto-wrapper
Target: arm-poky-linux-gnueabi
Configured with: ../../../../../../work-shared/gcc-8.3.0-r0/gcc-8.3.0/configure --build=x86_64-linux --host=x86_64-pokysdk-linux --target=arm-poky-linux-gnueabi --prefix=/opt/fsl-imx-x11/4.19-warrior/sysroots/x86_64-pokysdk-linux/usr --exec_prefix=/opt/fsl-imx-x11/4.19-warrior/sysroots/x86_64-pokysdk-linux/usr --bindir=/opt/fsl-imx-x11/4.19-warrior/sysroots/x86_64-pokysdk-linux/usr/bin/arm-poky-linux-gnueabi --sbindir=/opt/fsl-imx-x11/4.19-warrior/sysroots/x86_64-pokysdk-linux/usr/bin/arm-poky-linux-gnueabi --libexecdir=/opt/fsl-imx-x11/4.19-warrior/sysroots/x86_64-pokysdk-linux/usr/libexec/arm-poky-linux-gnueabi --datadir=/opt/fsl-imx-x11/4.19-warrior/sysroots/x86_64-pokysdk-linux/usr/share --sysconfdir=/opt/fsl-imx-x11/4.19-warrior/sysroots/x86_64-pokysdk-linux/etc --sharedstatedir=/opt/fsl-imx-x11/4.19-warrior/sysroots/x86_64-pokysdk-linux/com --localstatedir=/opt/fsl-imx-x11/4.19-warrior/sysroots/x86_64-pokysdk-linux/var --libdir=/opt/fsl-imx-x11/4.19-warrior/sysroots/x86_64-pokysdk-linux/usr/lib/arm-poky-linux-gnueabi --includedir=/opt/fsl-imx-x11/4.19-warrior/sysroots/x86_64-pokysdk-linux/usr/include --oldincludedir=/opt/fsl-imx-x11/4.19-warrior/sysroots/x86_64-pokysdk-linux/usr/include --infodir=/opt/fsl-imx-x11/4.19-warrior/sysroots/x86_64-pokysdk-linux/usr/share/info --mandir=/opt/fsl-imx-x11/4.19-warrior/sysroots/x86_64-pokysdk-linux/usr/share/man --disable-silent-rules --disable-dependency-tracking --with-libtool-sysroot=/home/xinluyao/yocto/imx-yocto-bsp/build/tmp/work/x86_64-nativesdk-pokysdk-linux/gcc-cross-canadian-arm/8.3.0-r0/recipe-sysroot --with-gnu-ld --enable-shared --enable-languages=c,c++ --enable-threads=posix --enable-multilib --enable-c99 --enable-long-long --enable-symvers=gnu --enable-libstdcxx-pch --program-prefix=arm-poky-linux-gnueabi- --without-local-prefix --enable-lto --disable-libssp --enable-libitm --disable-bootstrap --disable-libmudflap --with-system-zlib --with-linker-hash-style=gnu --enable-linker-build-id --with-ppl=no --with-cloog=no --enable-checking=release --enable-cheaders=c_global --without-isl --with-gxx-include-dir=/not/exist/usr/include/c++/8.3.0 --with-build-time-tools=/home/xinluyao/yocto/imx-yocto-bsp/build/tmp/work/x86_64-nativesdk-pokysdk-linux/gcc-cross-canadian-arm/8.3.0-r0/recipe-sysroot-native/usr/arm-poky-linux-gnueabi/bin --with-sysroot=/not/exist --with-build-sysroot=/home/xinluyao/yocto/imx-yocto-bsp/build/tmp/work/x86_64-nativesdk-pokysdk-linux/gcc-cross-canadian-arm/8.3.0-r0/recipe-sysroot --without-long-double-128 libgcc_cv_powerpc_float128=no --enable-poison-system-directories --enable-nls --with-glibc-version=2.28 --enable-initfini-array
Thread model: posix
gcc version 8.3.0 (GCC)
```

3.1.8 SSH

Secure Shell(SSH) 是由 IETF(The Internet Engineering Task Force) 制定的建立在应用层基础上的安全网络协议。传统的网络服务程序，如 FTP、Pop 和 Telnet 其本质上都是不安全的；因为它们在网络上用明文传送数据、用户帐号和用户口令，很容易受到中间人攻击方式的攻击。

SSH 分为客户端 openssh-client 和服务器 openssh-server，我们需要在 ubuntu 系统下安装 ssh 服务器，以便使用 ssh 客户端登录到系统。

1） 安装 ssh

```
sudo apt update
sudo apt install openssh-server
```

2） 启动 ssh

```
sudo service ssh start
```

3） 使用“ip addr show”命令查看 ubuntu 系统 ip

```
xinluyao@ubuntu:~$ ip addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:0c:29:50:27:20 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.118/24 brd 192.168.1.255 scope global dynamic noprefixroute ens33
       valid_lft 36338sec preferred_lft 36338sec
    inet6 fe80::362e:1a02:5429:8e5c/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

可以看出，当前 ubuntu IP 地址为 192.168.1.118

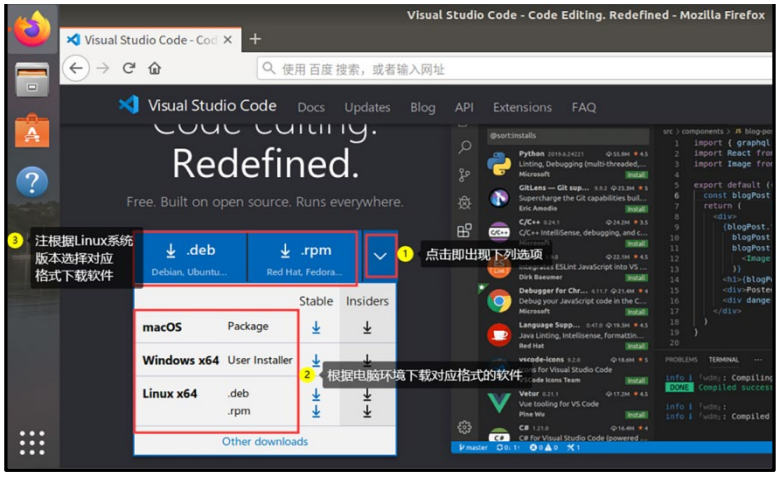
4） 测试 ssh 登录

在 Ubuntu 上安装完成 ssh 服务器后，接下来在 Windows 系统下使用 MobaXterm 进行连接，测试 ssh 服务器是否安装成功。具体连接步骤可以参考网络登录。

3.1.9 安装 Visual Studio Code

Visual Studio Sode（简称 VSCode）是微软出的一款免费的编辑器，有 Windows、Linux 和 macOS 三个版本，是一款跨平台编辑器。VSCode 下载地址如下：https://code.visualstudio.com/。读者根据自己的电脑环境下载自己需要的版本，本教程针对 Linux 版本讲解。

1） 获取 VSCode



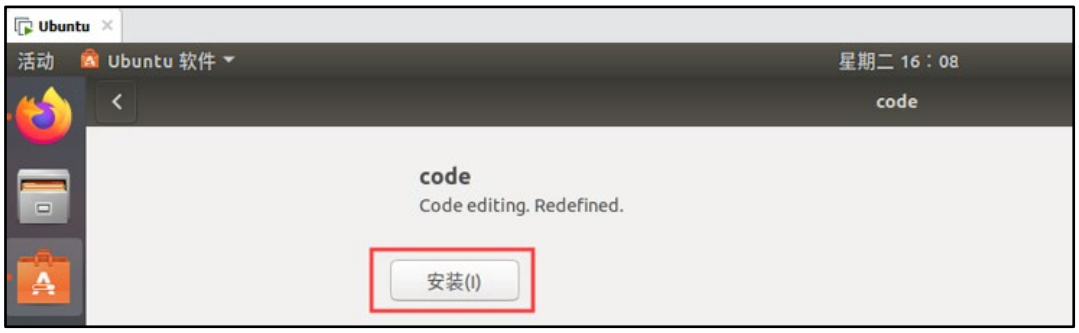
2） 安装 VSCode

使用如下命令安装：

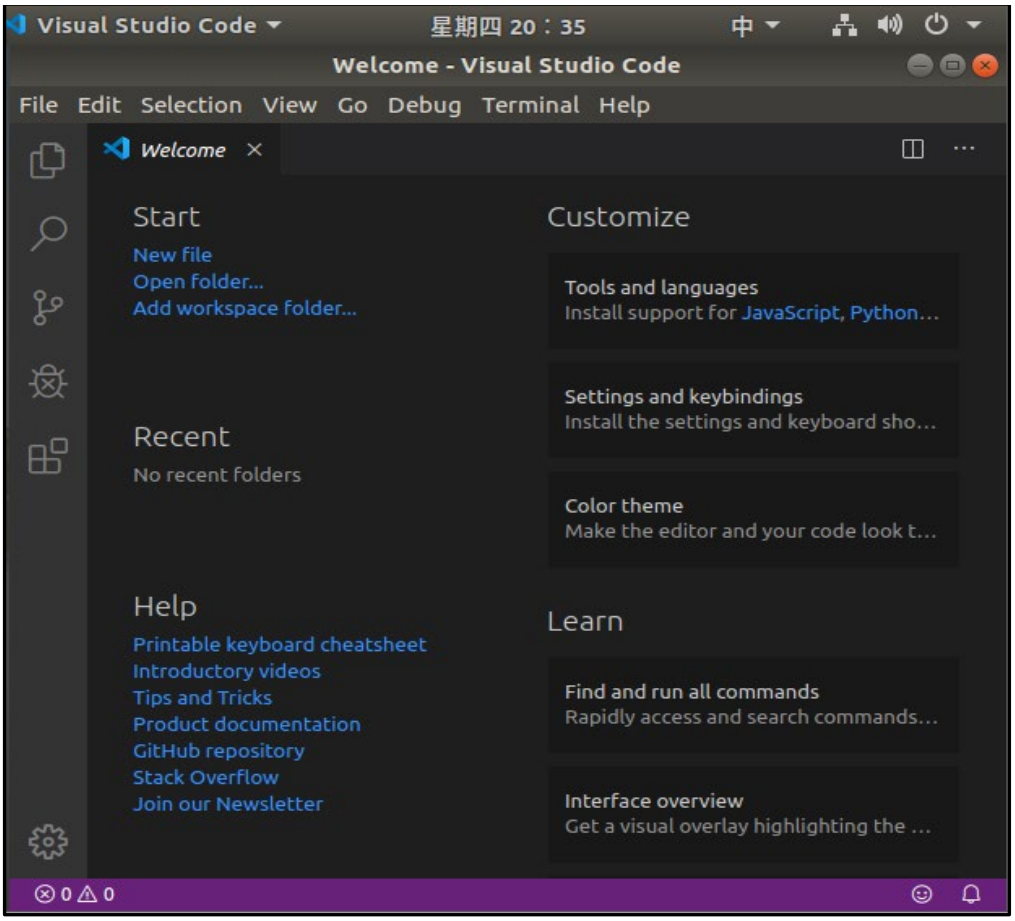
```
sudo dpkg -i code_1.35.3-1552606978_amd64.deb
```

如果后期版本更新，code 后面版本号可能变化，读者请自行修改。输入命

令后，会出现如下界面，直接点击安装即可。

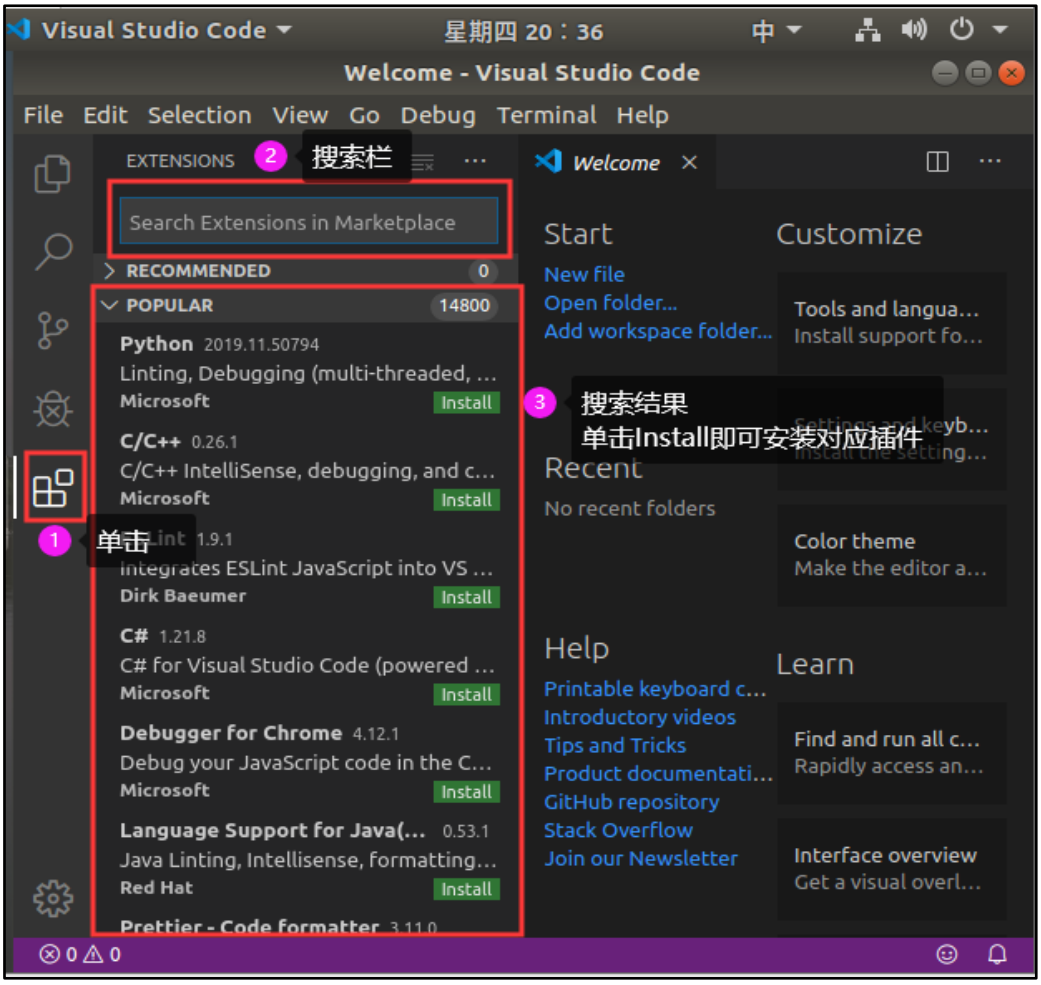


安装完成后，打开软件界面如下：



3） VSCode 插件安装

VSCode 支持多种语言，如 C/C++、Python、C#等，本教程我们主要用来编写 C/C++程序，需要安装 C/C++扩展包，安装步骤见下图：



常用的有如下插件需要安装：

- C/C++。
- C/C++ Snippets，即 C/C++重用代码块。
- C/C++ Advanced Lint,即 C/C++静态检测 。
- Code Runner，即代码运行。
- Include AutoComplete，即自动头文件包含。
- Rainbow Brackets，彩虹花括号，有助于阅读代码。
- One Dark Pro，VSCode 的主题。
- GBKtoUTF8，将 GBK 转换为 UTF8。
- ARM，即支持 ARM 汇编语法高亮显示。
- Chinese(Simplified)，即中文环境。
- vscode-icons，VSCode 图标插件，主要是资源管理器下各个文件夹的图标。
- compareit，比较插件，可以用于比较两个文件的差异。

- DeviceTree，设备树语法插件。
- Markdown Preview Enhanced，markdown 预览插件。
- Maridown pdf，将.md 文件转换成其他格式。

4）VS Code 添加中文支持

截至目前，VSCode 界面都是英文环境，我们已经安装了中文插件了，最后将 VSCode 改为中文环境，使用方法如下图所示：

根据上图提示，按下“Ctrl+Shift+P”打开搜索框，在搜索框里面输入“config”，然后选择“Configure Display Language”在打开的 local.json 文件中将 locale 修改为 zh-cn，

修改完成以后保存 local.json，然后重新打开 VSCode，VSCode 就变成了中文的了。效果如下图：


Visual Studio Code
星期六 23：01
英
欢迎使用 - Visual Studio Code
文件(F) 编辑(E) 选择(S) 查看(V) 转到(G) 调试(D) 终端(T) 帮助(H)
资源管理器
欢迎使用
打开的编辑器
欢迎使用
无打开的文件夹
尚未打开文件夹。
打开文件夹
大纲
启动
新建文件
打开文件夹...
添加工作区文件夹...
最近
无最近使用文件夹
帮助
快捷键速查表(可打印)
入门视频
提示与技巧
产品文档
GitHub 存储库
Stack Overflow
接收我们的新闻稿
自定义
工具和语言
安装对 JavaScript, Python, PHP, Azure, Docker 和更多 的支持
设置和按键绑定
安装 Vim, Sublime, Atom 和 其他 的设置和快捷键
颜色主题
使编辑器和代码呈现你喜欢的外观
学习
查找并运行所有命令
使用命令面板快速访问和搜索命令 (Ctrl+Shift+P)
界面概览
查看突出显示主要 UI 组件的叠加图
⊗0 △0

用户手册

4 四、U-Boot 入门

4.1 Bootloader 简介

与桌面通用 Linux 一样，嵌入式 Linux 也需要一段引导代码引导其运行，这段引导代码被称为 Bootloader。在 PC 机上通常是主板 BIOS，而在嵌入式领域由于处理器的多样性，Bootloader 也不尽相同，如 S3C24x0 系列处理器的 vivi，PXA2x0 系列的 Blob，都是比较有名的 Bootloader，但是这类 Bootloader 主要针对于特定的处理器，通用性较差。目前使用比较广泛的是 u-boot，它能支持绝大部分主流处理器。

Bootloader 有两种工作模式：系统引导模式和程序下载模式，他们分别对应 Bootloader 的两个基本功能，即引导和下载。

在引导模式下，Bootloader 根据设定的参数直接引导操作系统启动。引导操作系统启动是 Bootloader 最基本的核心功能。

在程序下载模式下，Bootloader 能够完成内核、根文件系统的固化和更新，甚至实现 Bootloader 的自我更新。至于通过何种方式来完成文件下载和固化，则与处理器以及 Bootloader 的具体实现相关，没有统一的标准。可以通过串口下载，也可以通过以太网下载，甚至可以通过 USB 或者 SD 卡等接口完成这些功能。

4.1.1 U-Boot 介绍

U-Boot 全称 Universal Boot Loader，是德国 DENX 软件工程中心 Wolfgang Denk 工程师维护的一个遵循 GPL 条款的开放源码项目，它从 FADSROM、8xxROM、PPCBOOT 逐步发展演化而来。U-Boot 既能支持多种处理器，包括： PowerPC 系列的处理器、 MIPS、 X86、 ARM、NIOS、 XScale 等，也能引导 Linux、 BSD、 Solaris、 VxWorks、 LynxOS、 pSOS、 QNX、 RTEMS、ARTOS 等众多操作系统。

U-Boot 的其源码目录与 Linux 类似，甚至于编译方式也与 Linux 内核相似。U-Boot 的源码，特别是很多驱动部分的源码都是 Linux 内核源码的简化。

U-Boot 有如下特点：

- 开放源码，自由使用；

- 支持多种嵌入式操作系统内核，如 Linux、NetBSD、VxWorks、QNX、 RTEMS 等；
- 支持多个处理器系列，如 PowerPC、ARM、x86、MIPS 等；
- 可靠性和稳定性都较好；
- 支持命令行，有自己的 Shell；
- 配置灵活，使用方便；
- 支持外设丰富，如串口、以太网、SDRAM、FLASH、LCD、NVRAM、EEPROM、RTC、键盘等；
- 文档较多，网络技术支持方便。

4.1.2 U-Boot 工作原理

U-Boot 的启动过程分为两阶段。第一阶段由汇编语言实现，与具体硬件平台相关；第二阶段由可读性和可移植性较好的 C 语言实现，完成 U-Boot 的主要功能。这样设计的优点在于可以把基于硬件的代码与系统的通用代码划分开，使得系统的移植工作主要针对第一阶段代码进行修改，而无需或只需少量修改第二阶段代码，简化了移植过程，提高了系统开发效率。

U-Boot 第一阶段代码实现的主要功能有：

- 硬件设备的初始化；
- 为加载 bootloader 第二阶段准备 RAM 空间；
- 复制 bootloader 第二阶段代码到 RAM 空间（U-Boot 拷贝其全部代码到 RAM）；
- 设置堆栈；
- 跳转到第二阶段 C 代码入口处。

当系统完成代码搬运并设置好 C 语言使用的堆栈等环境后，就会跳转到内存中的第二阶段代码 C 语言入口处继续运行。第二阶段代码完成的主要功能有：

- 继续初始化相关硬件设备（如串口、系统时钟及定时器等）；
- 检测系统内存映射；
- 加载内核映像及根文件系统映像；
- 设置内核启动参数；

- 调用内核。

第二阶段的 U-Boot 在设置好相应的终端设备后会停止等待若干秒，如果在该时间段内串口有输入，则 U-Boot 进入交互下载模式，循环读取串口命令并执行；如果串口没有输入，则 U-Boot 执行启动加载模式代码，将操作系统内核加载到内存并启动系统。

4.2 U-Boot 基本命令

U-Boot 自带命令行接口，在 U-Boot 启动期间，按任意键可进入 U-Boot Shell 命令行，在 Shell 界面可输入 U-Boot 支持的命令，使用 U-Boot 提供的各种功能：

```
U-Boot 2019.04 (May 16 2020 - 22:33:47 +0800)

CPU:   Freescale i.MX6ULL rev1.1 528 MHz (running at 396 MHz)
CPU:   Industrial temperature grade (-40C to 105C) at 62C
Reset cause: POR
Model: Freescale i.MX6 ULL 14x14 EVK Board
Board: MX6ULL 14x14 EVK



MMC:   FSL_SDHC: 0, FSL_SDHC: 1
Loading Environment from MMC... OK
Display: TFT_800x480 (800x480)
Video: 800x480x16
In:    serial
Out:   serial
Err:   serial
switch to partitions #0, OK
mmc1(part 0) is current device
flash target is MMC:1
Net:
Warning: ethernet@020b4000 (eth1) using random MAC address - 8a:1e:dc:26:
d7:9a
eth1: ethernet@020b4000 [PRIME]
Warning: ethernet@02188000 (eth0) using random MAC address - 4e:f7:6b:1a:
a8:34
, eth0: ethernet@02188000
Fastboot: Normal
Normal Boot
Hit any key to stop autoboot:  1
```

下面将针对 U-Boot 中的一些常用命令进行介绍。

4.2.1 查看命令列表

在 U-Boot 命令提示符下，输入 ？ 或者 help 可以查看 U-Boot 所支持的全部命令。

```
=> help
?          - alias for 'help'
base       - print or set address offset
bdinfo     - print Board Info structure
blkcache   - block cache diagnostics and control
bmode      - sd1|sd2|qspi1|normal|usb|sata|ecspi1:0|ecspi1:1|ecspi1:2|ecspi1:3
1|esdhc2|esdhc3|esdhc4 [noreset]
bmp        - manipulate BMP image data
boot       - boot default, i.e., run 'bootcmd'
bootd      - boot default, i.e., run 'bootcmd'
bootefi    - Boots an EFI payload from memory
bootelf    - Boot from an ELF image in memory
bootm      - boot application image from memory
bootp      - boot image via network using BOOTP/TFTP protocol
bootvx     - Boot vxWorks from an ELF image
bootz      - boot Linux zImage image from memory
clocks     - display clocks
cmp        - memory compare
coninfo    - print console devices and information
```

如需要查看单个命令的帮助信息，可以输入

```
help 命令
```

如：

```
help bootz
```

```
=> help bootz
bootz - boot Linux zImage image from memory

Usage:
bootz [addr [initrd[:size]] [fdt]]
    - boot Linux zImage stored in memory
        The argument 'initrd' is optional and specifies the address
        of the initrd in memory. The optional argument ':size' allows
        specifying the size of RAW initrd.
        When booting a Linux kernel which requires a flat device-tree
        a third argument is required which is the address of the
        device-tree blob. To boot that kernel without an initrd image,
        use a '-' for the second argument. If you do not pass a third
        a bd_info struct will be passed instead
```

U-Boot 命令中，有一些命令归类在某一个子类中，如 I2C、NAND FLASH、UBI 、MMC 等，直接输入命令类名称，将会显示该类下面的子命令及使用说明。例如要显示与 mmc 相关的子命令，可在命令行下执行：

```
mmc
```

```
=> mmc
mmc - MMC sub system

Usage:
mmc info - display info of the current MMC device
mmc read addr blk# cnt
mmc write addr blk# cnt
mmc erase blk# cnt
mmc rescan
mmc part - lists available partition on current mmc device
mmc dev [dev] [part] - show or set current mmc device [partition]
mmc list - lists available devices
mmc hwpartition [args...] - does hardware partitioning
  arguments (sizes in 512-byte blocks):
    [user [enh start cnt] [wrrel {on|off}]] - sets user data area attributes
    [gp1|gp2|gp3|gp4 cnt [enh] [wrrel {on|off}]] - general purpose partition
    [check|set|complete] - mode, complete set partitioning completed
  WARNING: Partitioning is a write-once setting once it is set to complete.
  Power cycling is required to initialize partitions after set to complete.
mmc bootbus dev boot_bus_width reset_boot_bus_width boot_mode
 - Set the BOOT_BUS_WIDTH field of the specified device
mmc bootpart-resize <dev> <boot part size MB> <RPMB part size MB>
 - Change sizes of boot and RPMB partitions of specified device
mmc partconf dev [boot_ack boot_partition partition_access]
 - Show or change the bits of the PARTITION_CONFIG field of the specified device
mmc rst-function dev value
 - Change the RST_n_FUNCTION field of the specified device
   WARNING: This is a write-once field and 0 / 1 / 2 are the only valid values.
mmc setdsr <value> - set DSR register value
```

为执行这些子命令，可在命令行下输入“类 子命令”，例如要显示可用的 mmc 设备，可在命令行下执行：

```
mmc info
```

```
=> mmc info
Device: FSL_SDHC
Manufacturer ID: 15
OEM: 100
Name: 8GTF4
Tran Speed: 52000000
Rd Block Len: 512
MMC version 4.0
High Capacity: Yes
Capacity: 7.3 GiB
Bus Width: 8-bit
Erase Group Size: 512 KiB
```

4.2.2 环境变量

1） 查看环境变量

U-Boot 支持环境变量。使用 printenv 命令可查看用户在 U-Boot 中配置的全部环境变量：

```
printenv
```

当要查看某个具体环境变量时，只需要在 printenv 命令后面加上环境变量名参数即可。例如要查看 serverip 环境变量，可执行如下命令：

```
printenv serverip
```

```
=> printenv serverip
serverip=192.168.1.118
```

如未设置 serverip，则结果为：

```
=> printenv serverip
## Error: "serverip" not defined
```

2） 设置环境变量

使用 setenv 命令可以设置环境变量。setenv 命令的格式如下：

```
setenv 环境变量名 环境变量值
```

当使用 setenv 命令时，如果环境变量名不存在，该命令会添加这个新的环境变量；若环境变量名已经存在，该命令会修改环境变量的值；如果环境变量名已经存在，但新设置的变量值为空，则该命令删除环境变量。

如要设置 serverip 环境变量的值为 192.168.1.118，可执行如下命令：

```
setenv serverip 192.168.1.118
```

要删除 severip 环境变量，可执行如下命令：

```
setenv serverip
```

注意 setenv 命令设置的环境变量仅保存在内存中，当 U-Boot 重新启动时，设置内容将丢失。若需要保存所设置的环境变量，可使用 saveenv 命令：

```
saveenv
```

```
=> saveenv
Saving Environment to MMC...
Writing to MMC(1)... done
```

saveenv 命令会把用户设置的所有环境变量都保存在非易失性储存器中。

```
=> saveenv
Saving Environment to MMC... Writing to MMC(1)... OK
```

3） 使用环境变量

环境变量的作用是字符串替换。在 U-Boot 中使用环境变量的方法为：

```
${环境变量名}
```

例如已经在 U-Boot 中定义了 serverip 环境变量，那么引用该环境的方法为：

```
${serverip}
```

假如要在 U-Boot 下探测网络上是否有 192.168.1.118 的 IP 时，可以使用如下命令：

```
ping 192.168.1.118
```

如已经将 serverip 值设置为 192.168.1.118，上述 ping 命令也可以为：

```
ping ${serverip}
```

```
=> ping $serverip
Using ethernet@020b4000 device
host 192.168.1.118 is alive
=> ping ${serverip}
Using ethernet@020b4000 device
host 192.168.1.118 is alive
```

4） 常用环境变量

U-Boot 自定义了很多环境变量，这些环境变量有特殊用途。此处列出 U-Boot 中经常用到的一些环境变量，如下表所示：

环境变量	说明
bootdelay	U-Boot 启动后会延迟若干秒，等待用户按键进入 U-Boot 的命令行。至于等待的秒数则由 bootdelay 环境变量设定。
bootcmd	U-Boot 启动后，若没有进入命令行，则自动执行 bootcmd 环境变量保存的命令组合，以启动内核。
bootargs	该环境变量保存了内核启动时，传递内核的启动参数。
fdt_file	设备树名称，启动后在指定分区查找的设备树的名称
nfsroot	使用 nfs 作为根文件系统时，需要使用此变量指定文件系统在 PC 上的路径

4.2.3 网络命令

1） 设置网络 IP

U-Boot 使用 ipaddr 环境变量用于设置本地 IP 地址；使用 serverip 环境变量用于设置服务器 IP 地址。例如要把本地 IP 地址和服务器 IP 地址分别设置为 192.168.1.160 和 192.168.1.118，可使用如下命令：

```
setenv ipaddr 192.168.1.160
setenv serverip 192.168.1.118
```

2） ping 命令

U-Boot 提供了 ping 命令用于探测网络是否畅通或者网络上是否有指定 IP 地址的主机。ping 命令的用法示例如下：

```
ping 192.168.1.118
```

```
=> ping 192.168.1.118
Using ethernet@020b4000 device
host 192.168.1.118 is alive
```

如果 ping 命令打印"host xxx.xxx.xxx.xxx is alive"信息，表示开发板和目的主机之间网络畅通；如果 ping 命令打印"ping failed; host xxx.xxx.xxx.xxx is not alive"，表示开发板和目的主机之间网络异常。

3) 使用 tftp 命令下载文件

使用 tftp 命令可以把 tftp 服务器中的文件下载到内存的指定位置。在使用之前，首先应设置开发板和 tftp 服务器的 IP 地址，并确保网络畅通。该命令的格式为：

```
tftp loadAddress filename
```

其中“loadAddress”表示要下载到的开发板内存的地址；“filename”表示需要从 tftp 服务器下载的文件名。例如，当需要从 IP 地址为 192.168.1.118 的 tftp 服务器下载 zImage 文件到本地内存的 loadaddr(0x80800000)地址时，应执行如下命令：

```
tftp ${loadaddr} zImage
```

```
=> tftp $loadaddr zImage
Using ethernet@020b4000 device
TFTP from server 192.168.1.118; our IP address is 192.168.1.160
Filename 'zImage'.
Load address: 0x80800000
Loading: ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ############
         1 MiB/s
done
Bytes transferred = 7785240 (76cb18 hex)
```

当 tftp 正在下载数据包时，会打印“#”符号；当打印“T”符号时，表示网络出现停顿。

4.2.4 mmc 命令

1) mmc rescan 命令

mmc rescan 命令用于扫描当前开发板上所有的 mmc 设备，包括 emmc 和 SD 卡。

2) mmc list 命令

mmc list 命令用于查看当前开发板一共有几个 mmc 设备。

```
mmc list
```

```
=> mmc list
FSL_SDHC: 0
FSL_SDHC: 1 (eMMC)
```

上述信息表示当前开发板有两个 MMC 设备，FSL_SDHC:0 和 FSL_SDHC:1(eMMC)。此时正使用 eMMC 设备。

3) mmc dev 命令

mmc dev 命令用于切换当前 MMC 设备，命令格式如下：

```
mmc dev [dev] [part]
```

[dev]用来设置要切换的 MMC 设备号，[part]是分区号。如果不写分区号的话默认分区为 0。使用如下命令切换到 SD 卡：

```
mmc dev 0
```

```
=> mmc dev 0
MMC: no card present
=> mmc dev 0
switch to partitions #0, OK
mmc0 is current device
```

4) mmc part 命令

有时候 SD 卡或者 EMMC 会有多个分区，可以使用 mmc part 命令来查看其分区，比如我们可以使用如下命令查看 emmc 分区的情况：

```
mmc dev 1
mmc part
```

```
=> mmc dev 1
switch to partitions #0, OK
mmc1(part 0) is current device
=> mmc part

Partition Map for MMC device 1  --   Partition Type: DOS

Part    Start Sector    Num Sectors     UUID            Type
  1     20480           262144          504d043d-01     0c
  2     282624          14987264        504d043d-02     83
```

如要将 emmc 的分区 2 设置为当前分区，可以使用如下命令：

```
mmc dev 1 2
```

```
=> mmc dev 1 2
switch to partitions #2, OK
mmc1(part 2) is current device
```

5) mmc read 命令

mmc read 命令用于读取 mmc 设备的数据，命令格式如下：

```
mmc read addr blk# cnt
```

addr 是数据读取到 DRAM 中的地址，blk 是要读取的块起始地址(十六进制)，emmc 一个块是 512 字节，cnt 是要读取的块数量(十六进制)。比如从 EMMC 的第 1792(0x700)个块开始，读取 16(0x10)个块的数据到 DRAM 的 0X80800000 地址处，命令如下：

```
mmc dev 1 0
mmc read 80800000 700 10
```

```
=> mmc dev 1 0
switch to partitions #0, OK
mmc1(part 0) is current device
=> mmc read 80800000 700 10

MMC read: dev # 1, block # 1792, count 16 ... 16 blocks read: OK
```

读取出的数据可以通过 md.b 命令显示出来。

```
md.b 80800000 2000
```

```
=> md.b 80800000 2000
80800000: 4e e4 0f c9 62 61 75 64 72 61 74 65 3d 31 31 35    N...baudrate=115
80800010: 32 30 30 00 62 6f 61 72 64 5f 6e 61 6d 65 3d 45    200.board_name=E
80800020: 56 4b 00 62 6f 61 72 64 5f 72 65 76 3d 31 34 58    VK.board_rev=14X
80800030: 31 34 00 62 6f 6f 74 5f 66 64 74 3d 74 72 79 00    14.boot_fdt=try.
80800040: 62 6f 6f 74 61 72 67 73 3d 63 6f 6e 73 6f 6c 65    bootargs=console
80800050: 3d 74 74 79 6d 78 63 30 2c 31 31 35 32 30 30 20    =ttymxc0,115200
80800060: 72 6f 6f 74 3d 2f 64 65 76 2f 6d 6d 63 62 6c 6b    root=/dev/mmcblk
80800070: 31 70 32 20 72 6f 6f 74 77 61 69 74 20 72 77 00    1p2 rootwait rw.
80800080: 62 6f 6f 74 63 6d 64 3d 72 75 6e 20 6e 65 74 62    bootcmd=run netb
80800090: 6f 6f 74 00 62 6f 6f 74 63 6d 64 5f 6d 66 67 3d    oot.bootcmd_mfg=
808000a0: 72 75 6e 20 6d 66 67 74 6f 6f 6c 5f 61 72 67 73    run mfgtool_args
808000b0: 3b 69 66 20 69 6d 69 6e 66 6f 20 24 7b 69 6e 69    ;if iminfo ${ini
808000c0: 74 72 64 5f 61 64 64 72 7d 3b 20 74 68 65 6e 20    trd_addr}; then
808000d0: 69 66 20 74 65 73 74 20 24 7b 74 65 65 7d 20 3d    if test ${tee} =
808000e0: 20 79 65 73 3b 20 74 68 65 6e 20 62 6f 6f 74 6d     yes; then bootm
808000f0: 20 24 7b 74 65 65 5f 61 64 64 72 7d 20 24 7b 69     ${tee_addr} ${i
80800100: 6e 69 74 72 64 5f 61 64 64 72 7d 20 24 7b 66 64    nitrd_addr} ${fd
80800110: 74 5f 61 64 64 72 7d 3b 20 65 6c 73 65 20 62 6f    t_addr}; else bo
80800120: 6f 74 7a 20 24 7b 6c 6f 61 64 61 64 64 72 7d 20    otz ${loadaddr}
80800130: 24 7b 69 6e 69 74 72 64 5f 61 64 64 72 7d 20 24    ${initrd_addr} $
80800140: 7b 66 64 74 5f 61 64 64 72 7d 3b 20 66 69 3b 20    {fdt_addr}; fi;
80800150: 65 6c 73 65 20 65 63 68 6f 20 22 52 75 6e 20 66    else echo "Run f
```

6) mmc write 命令

要将数据写到 MMC 设置里面，可以使用命令 mmc write，格式如下：

```
mmc write addr blk# cnt
```

addr 是要写入数据在 DRAM 中的地址

blk 是写入 MMC 的块起始地址(十六进制格式)

cnt 是要写入的块数目

使用 tftp 将 uboot 文件下载到内存 80800000 处

```
tftp 80800000 u-boot-imx6ull-ddr512-emmc.imx
```

```
=> tftp 80800000 u-boot-imx6ull-ddr512-emmc.imx
Using FEC1 device
TFTP from server 192.168.6.5; our IP address is 192.168.6.31
Filename 'u-boot-imx6ull-ddr512-emmc.imx'.
Load address: 0x80800000
Loading: ##################################################
         1.9 MiB/s
done
Bytes transferred = 760832 (b9c00 hex)
```

文件大小为 760832Bytes，是 760832/512=1486=0x5CE 个块。

切换 emmc 设备为 SD 卡的分区 0

```
mmc dev 0 0
```

```
=> mmc dev 0 0
switch to partitions #0, OK
mmc0 is current device
```

```
mmc write 80800000 2 5CE
```

```
=> mmc write 80800000 2 5CE

MMC write: dev # 0, block # 2, count 1486 ... 1486 blocks written: OK
```

7) mmc erase 命令

要擦除 MMC 设备的指定就用命令 mmc erase，格式如下：

```
mmc erase blk# cnt
```

blk 是要擦除的起始块

cnt 是要擦除的数量

4.2.5 FAT 命令

1. fatinfo

```
usage: fatinfo <interface> <dev[:part]>
```

在使用该命令的时候，直接指定需要查看的设备和编号即可，如：

```
fatinfo mmc 1
```

```
=>  fatinfo mmc 1
Interface:  MMC
  Device 1: Vendor: Man 000015 Snr c7df8c3a Rev: 0.6 Prod: 8GTF4R
            Type: Removable Hard Disk
            Capacity: 7456.0 MB = 7.2 GB (15269888 x 512)
Filesystem: FAT32 "NO NAME    "
```

fstype

```
usage: fstype <interface> <dev>:<part>
```

可以使用如下命令查看 emmc 上 3 个分区的格式：

```
fstype mmc 1:0
```

```
=> fstype mmc 1:0
Failed to mount ext2 filesystem...
** Unrecognized filesystem type **
```

```
fstype mmc 1:1
```

```
=> fstype mmc 1:1
fat
```

```
fstype mmc 1:2
```

```
=> fstype mmc 1:2
ext4
```

2. fatls

```
usage: fatls <interface> <dev[:part]> [directory]
```

使用该命令的时候需要指定设备及其编号，默认目录为根目录，如果想查看其它目录则可以通过 [directory] 参数指定。

可以使用如下命令查看 mmc 中 fat 分区内容

```
fatls mmc 1:1
```

```
=> fatls mmc 1:1
  5811064    zimage
    36578    imx6ull-xinluyao-nand-c.dtb
    35630    imx6ull-xinluyao-emmc-c.dtb

3 file(s), 0 dir(s)
```

3. fatload

fatload 命令用于将指定的文件读取到 DRAM 中，命令格式如下：

```
usage: fatload <interface> [<dev[:part]> [<addr> [<filename> [bytes [pos]]]]]
```

interface 为接口，比如 mmc， dev 是设备号， part 是分区， addr 是保存在 DRAM 中的起始地址， filename 是要读取的文件名字。 bytes 表示读取多少字节的数据，如果 bytes 为 0 或者省略的话表示读取整个文件。 pos 是要读的文件相对于文件首地址的偏移，如果为 0 或者省略的话表示从文件首地址开始读取。我们将 EMMC 分区 1 中的 zImage 文件读取到 DRAM 中的 0X80800000 地址处，命令如下：

```
fatload mmc 1:1 80800000 zImage
```

4. fatwrite

fatwirte 命令用于将 DRAM 中的数据写入到 MMC 设备中，命令为：

```
usage: fatwrite <interface> <dev[:part]> <addr> <filename> <bytes>
```

interface 为接口，比如 mmc， dev 是设备号， part 是分区， ad

dr 是要写入的数据在 DRAM 中的起始地址， filename 是写入的数据文件名字， bytes 表示要写入多少自己的数据。

可以使用如下命令查看 mmc 中 fat 分区内容

```
fatls  mmc  1:1
```

```
=> fatls mmc 1:1
  7738224    zImage
    36239    imx6ull-xly-emmc-lcd.dtb
    36811    imx6ull-xly-emmc-hdmi.dtb

3 file(s), 0 dir(s)
```

4.2.6 EXT 命令

1. extls

```
ext4ls <interface> <dev[:part]> [directory]
```

查看 emmc 中 ext 分区内容命令如下：

```
ext4ls  mmc  1:2
```

```
=> ext4ls  mmc  1:2
<DIR>       4096 .
<DIR>       4096 ..
<DIR>      16384 lost+found
<DIR>       4096 media
<DIR>       4096 bin
<SYM>          8 tmp
<DIR>       4096 home
<DIR>       4096 boot
<DIR>       4096 lib
<DIR>       4096 run
<DIR>       4096 mnt
<DIR>       4096 .cache
<DIR>       4096 sys
<DIR>       4096 opt
<DIR>       4096 proc
<DIR>       4096 dev
<DIR>       4096 etc
<DIR>       4096 .aiur
<DIR>       4096 sbin
<DIR>       4096 usr
<DIR>       4096 .config
<DIR>       4096 var
```

4.2.7 启动命令

bootz 命令可以用于启动 zImage 镜像文件，bootz 命令的格式为：

```
bootz [addr[initrd[:size]] [fdt]]
```

命令 bootz 有三个参数，addr 是 Linux 镜像文件在 DRAM 中的位置，initrd 是 initrd 文件在 DRAM 中的地址，如果不适用 initrd 则使用“-”代替即可，fdt 是设备树文件在 DRAM 中的地址。

要启动 Linux 系统，可以使用如下命令：

```
bootz ${loadaddr} - ${fdt_addr}
```

4.2.8 组合命令

在 U-Boot 中可以使用 setenv 设置任意环境变量，如果为环境变量赋值为一串命令的组合，则可称之为“组合命令”。组合命令的各个命令直接以;号分隔。执行这类命令的方法是“run 组合命令”。

例如 U-Boot 自定义的环境变量 mmcboot，其值为很长的组合：

```
=> printenv mmcboot
mmcboot=echo Booting from mmc ...; run mmcargs; if test ${tee} = yes; then run loac
fdt; run loadtee; bootm ${tee_addr} - ${fdt_addr}; else if test ${boot_fdt} = yes |
| test ${boot_fdt} = try; then if run loadfdt; then bootz ${loadaddr} - ${fdt_addr}
; else if test ${boot_fdt} = try; then bootz; else echo WARN: Cannot load the DT; f
i; fi; else bootz; fi; fi;
```

这个命令中使用到 hush shell 脚本。hush 脚本和我们常用的 bash shell 稍有区别，上述内容格式化后如下：

```
1  echo Booting from mmc ...;
2  run mmcargs;
3  if test ${tee} = yes; then
4          run loadfdt; run loadtee; bootm ${tee_addr} - ${fdt_addr};
5  else
6          if test ${boot_fdt} = yes || test ${boot_fdt} = try; then
7                  if run loadfdt; then
8                          bootz ${loadaddr} - ${fdt_addr};
9                  else
10                         if test ${boot_fdt} = try; then
11                                 bootz;
12                         else
13                                 echo WARN: Cannot load the DT;
14                         fi;
15                 fi;
16
17         else
18                 bootz;
19         fi;
20 fi;
```

4.2.9 重启命令

在 U-Boot 命令行下，执行 reset 命令可以重启系统：

```
reset
```

4.3 U-Boot 设置网络启动

在实际开发调试时，需要经常修改固件，每次都使用 U 盘或 SD 卡拷贝十分不方便，所以在开发时，一般从 tftp 服务器获取内核 image，使用 nfs 挂载文件系统。

HD-IMX6ULL-MB 开发板进行网络启动需要进行如下设置：

4.3.1 设置参数

网络启动前，需要设置一些网络相关的参数。

```
setenv ethaddr 00:01:02:03:04:05
setenv ip_dyn no
setenv fdt_file imx6ull-xly-emmc-lcd.dtb
setenv serverip 192.168.1.118
setenv ipaddr 192.168.1.160
setenv nfsroot "/srv/imx6ull/rootfs"
setenv bootcmd "run netboot"
saveenv
```

ethaddr 设置开发板的 MAC 地址， fdt_file 设置设备树名称，serverip 设置服务器 ip 地址（这个地址不同电脑不一样，请核实后填写），nfsroot 设置根文件系统的目录。这些参数可以根据用户实际情况进行修改。

4.3.2 网络环境准备

使用开发板 ETH2(靠近 HDMI 接口的那个网口)将开发板连接至路由器(需和 PC 在同一路由器网络下)，在服务器 tftp 目录中存放好 zImage 和 dtb 文件，并确认 rootfs 文件系统。

4.3.3 恢复 MMC 启动

设置好参数后，输入"boot"命令，开发板会从网络启动。由于保存了参数，之后每次重启，依然会从网络启动。若需要恢复 eMMC 启动，仅需设置：

```
setenv bootcmd "run loadimage;run mmcboot"
saveenv
boot
```

run loadimage 在出厂时已设置为" fatload mmc ${mmcdev}:${mmcpart} ${loadaddr} ${image}"，会从指定的 mmc 设备中读取 zImage。mmcboot 则读取设备树，启动内核。

4.4 编译 uboot

本节介绍如何编译 uboot。编译 uboot 需要安装好交叉工具链。

在“5-源代码”目录中，包含了 uboot 源代码，文件名为 uboot-imx6u

ll-2019.04.tar.xz。编译 uboot 操作系统的步骤如下：

1．将 uboot 内核源码文件 uboot-imx6ull-2019.04.tar.xz 拷贝到 Ubuntu 虚拟机中，将这两个文件拷贝到/home/xinluyao/work/imx6ull/bsp 中，并对其进行解压：

```
cd /home/xinluyao/work/imx6ull/bsp/
tar xvJf uboot-imx6ull-2019.04.tar.xz
```

2．清除前一次的编译结果。执行如下命令：

```
cd /home/xinluyao/work/imx6ull/bsp/uboot-2019.04
make distclean
```

3．使用开发板默认的配置文件生成编译内核时所需要的.config 文件。执行如下命令：

```
make mx6ull_14x14_ddr512_emmc_defconfig
```

注：第一次解压一定要配置一次，否则会编译失败。

4．编译。执行如下命令：

```
make all
```

编译完成后，会在 uboot 源码顶层目录（即 uboot-imx6ull-2019.04 目录）下生成 u-boot.bin 和 u-boot-dtb.imx 文件。

5 Linux 内核

5.1 内核简介

5.1.1 概述

Linux 是全球最受欢迎的开源操作系统。它是一个由 C 语言编写的，符合 POSIX 标准的类 UNIX 操作系统。操作系统是一个用来和硬件打交道并为用户程序提供一个有限服务集的低级支撑软件。Linux 内核的主要模块（或组件）分以下几个部分：进程管理、内存管理、虚拟文件系统、网络接口、进程间通信。

Linux 内核作为一个开放、自由的操作系统内核，有如下特点：

- Linux 是一个一体化内核。“一体化内核”是也称“宏内核”，其特点是效率高，数据吞吐量高；
- 可移植性强。Linux 目前已经成为支持硬件平台最广泛的操作系统，如 X86、IA64、ARM、MIPS、AVR32、M68K、S390、Blackfin、M32R 等；
- Linux 是一个可裁剪的操作系统内核。Linux 极具伸缩性，内核可以任意裁剪，可以大至几十或者上百兆，也可以小至几百 K，从超级计算机、大型服务器到小型嵌入式系统、掌上移动设备或者嵌入式模块，几乎都可以看到它的身影；
- 模块化。Linux 内核采用模块化设计，很多功能部件都可以编译为模块，可以在内核运行时动态加载/卸载而无需重启系统；
- 网络支持完善。Linux 内核集成了完整的 POSIX 网络协议栈，网络功能完善；
- 稳定性强。运行 Linux 内核的服务器可以做到几年不用复位重启；
- 安全性好。Linux 源码开放，由众多黑客参与 Linux 的开发，一旦发现漏洞都能及时修复；
- 支持的设备广泛。Linux 源码中，设备驱动源码占了很大比例，几乎能支持任何常见设备，无论是很老旧的设备还是最新推出的硬件设备，几乎都能找到 Linux 下的驱动。

5.1.2 Linux 内核源码

Linux 内核源码很复杂，包含多级目录，形成一个庞大的树状结构，通常称为 Linux 源码目录树。HD-IMX6ULL-MB 开发板使用 Linux 4.19.35 内核版本，本节以该版本为例介绍 Linux 源码的目录结构：

目录	说明
arch	包含各体系结构特定的代码，如 arm、x86、ia64、mips 等，在每个体系结构目录下通常都有： －boot 内核需要的特定平台代码 －kernel 体系结构特有的代码 －lib 通用函数在特定体系结构的实现 －math-emu 模拟 FPU 的代码，在 ARM 中，使用 mach-xxx 代替 －mm 特定体系结构的内存管理实现 －include 特定体系的头文件
block	存放块设备相关代码
crypto	存放加密、压缩、CRC 校验等算法相关代码
Documentation	存放相关说明文档，很多实用文档，包括驱动编写等
drivers	存放 Linux 内核设备驱动程序源码。驱动源码在 Linux 内核源码中站了很大比例，常见外设几乎都有可参考源码，对驱动开发而言，该目录非常重要。该目录包含众多驱动，目录按照设备类别进行分类，如 char、block、input、i2c、spi、pci、usb 等
firmware	存放处理器相关的一些特殊固件
fs	存放所有文件系统代码，如 fat、ext2、ext3、ext4、ubifs、nfs、sysfs 等
include	存放内核所需、与平台无关的头文件，与平台相关的头文件已经被移动到 arch 平台的 include 目录，如 ARM 的头文件目录<arch/arm/include/asm/>
init	包含内核初始化代码
ipc	存放进程间通信代码
kernel	包含 Linux 内核管理代码
lib	库文件代码实现
mm	存放内存管理代码
net	存放网络相关代码
samples	存放提供的一些内核编程范例
srcipts	存放一些脚本文件，如 menuconfig 脚本
security	存放系统安全性相关代码
sound	存放声音、声卡相关驱动
tools	编译过程中一些主机必要工具
usr	cpio 相关实现
virt	内核虚拟机 KVM

5.1.3 Linux 内核配置系统

Linux 内核的配置系统由三个部分组成：

1、Makefile：分布在 Linux 内核源代码根目录及各层目录中，定义 Linux 内核的编译规则；

2、配置文件 Kconfig：分布在 Linux 内核源代码根目录及各层目录中，为用户提供每个源码目录可以使用的内核配置菜单；

3、配置工具：包括配置命令解释器和配置用户界面。配置用户界面主要有：基于字符界面(make config)、基于 Ncurses 图形界面(make menuconfig)以及基于 Xwindows 图形界面(make xconfig)。

下面将对这三部分分别加以介绍：

5.1.3.1 内核 Makefile 文件

源码目录树顶层 Makefile 是整个内核源码管理的入口，对整个内核的源码编译起着决定性作用。编译内核时，顶层 Makefile 会按规则递归遍历内核源码的所有子目录下的 Makefile 文件，完成各子目录下内核模块的编译。

在内核源码的子目录中，几乎每个子目录都有相应的 Makefile 文件，管理着对应目录下的代码，对该目录的文件或者子目录的编译进行控制。Makefile 中有两种表示方式来控制某个文件是否编译，一种是默认选择编译，用 obj-y 表示，如：

```
obj-y += usb-host.o  # 默认编译 usb-host.c 文件
obj-y += gpio/  # 默认编译 gpio 目录
```

另一种表示则与内核配置选项相关联，编译与否以及编译方式取决于内核配置，例如：

```
obj-$(CONFIG_WDT) += wdt.o # wdt.c 编译控制
obj-$(CONFIG_PCI) += pci/  # pci 目录编译控制
```

是否编译 wdt.c 文件，或者以何种方式编译，取决于内核配置后的变量 CONFIG_WDT 值：如果在配置中设置为[*]，则静态编译到内核，如果配置为[M]，则编译为 wdt.ko 模块，否则不编译。

5.1.3.2 Kconfig 文件

内核源码树每个目录下都还包含一个 Kconfig 文件，用于描述所在目录源代码相关的内核配置菜单，各个目录的 Kconfig 文件构成了一个分布式的内核配置数据库。通过 make menuconfig（make xconfig 或者 make gconfig）命令配置内核的时候，从 Kconfig 文件读取菜单，配置完毕保存到文件名为.config 的内核配置文件中，供 Makefile 文件在编译内核时使用。

5.1.3.3 内核配置工具

用户可以使用如下命令对内核进行配置：

1、make config：基于文本模式的交互式配置；

2、make menuconfig：基于文本模式的菜单型配置；

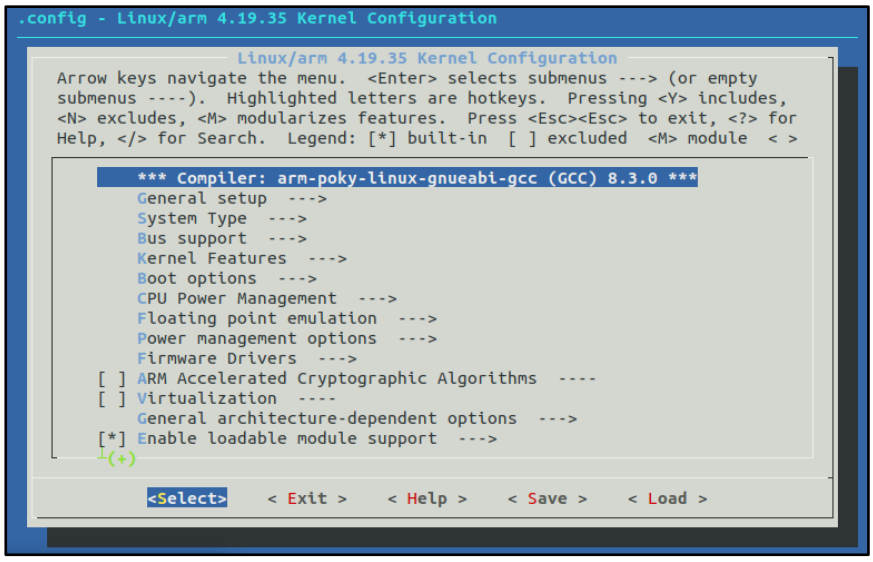
4、make xconfig：图形化的配置（需安装图形化系统）；

其中 make menuconfig 是最为常用的内核配置方式。下面将主要介绍 make menuconfig 的使用。

在运行 make menuconfig 前，主机必须先安装 ncurses 相关的库。在 Ubuntu 下执行如下命令完成 ncurses 的安装：

```
sudo apt install libncurses5-dev
```

在 Linux 内核源码顶层目录下输入 make menuconfig 命令，可进入 Linux 内核配置主界面，如下图所示：



内核配置主界面由三部分组成：

- 最上面部分为基本操作的简要介绍；
- 中间部分为内核配置的各个菜单项；
- 最下面部分为功能菜单

基于 Ncurses 的 Linux 内核配置界面不支持鼠标操作，必须用键盘操作。基本操作方法如下：

1、使用上、下箭头可以选择内核配置菜单项；使用左、右箭头可以选择下排的功能菜单项；

2、当功能菜单中的“Select”项被选中时，在某个具有子菜单的内核菜

单项上按**回车键**可进入该菜单的子菜单；

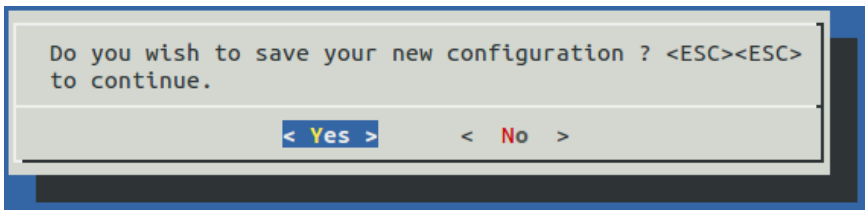
3、对于某个菜单项，可以直接按“？”键查看该菜单项的帮助，或者用左右箭头键将功能菜单移到“Help”上，并按回车键查看帮助；

4、根据菜单项的类型，其配置方式略有不同。可以分成三种情况，具体说明如下：

- 对于以[ ]开头的配置项，[*]表示选中，[ ]表示未选中。可以用空格键进行切换或者使用键盘快捷键（Y-选中，N-未选中）进行切换；
- 对于以<>开头的配置项，<*>表示静态编译，<M>表示编译为模块，<>表示未选中。可以使用空格键进行切换或者使用键盘快捷键（Y-静态编译，M-编译为模块，N-未选中）进行切换；
- 对于()开头的配置项，表明该配置是数值或者字符串，可以按回车键直接编辑。

5、按斜线（/）可启用搜索功能，填入关键字后可搜索全部菜单内容；

6、连续按两次 ESC 键或者用左右箭头键将功能菜单移到“Exit”上，按回车键，将退回到上一级菜单。如果当前已经位于顶层菜单界面（即内核配置主界面），则将弹出确认修改的提示画面，如下图所示：



选择“Yes”将保存对内核配置的修改并退出，选择“No”将不保存修改直接退出，连续按两次 ESC 键将重新返回内核配置画面，允许用户继续对内核配置进行修改。

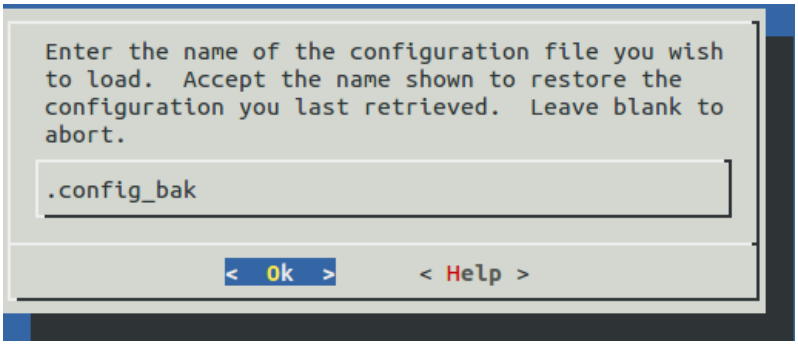
内核配置完成后，其配置信息保存在内核源码顶层目录的.config 文件中。用户可以直接在命令行下将.config 文件备份为其他的文件名，以备日后使用，例如：

```
cp .cofnig .config_bak
```

为了使用以前备份的内核配置文件，可以直接将备份的内核配置文件覆盖内核源码顶层目录的.config 文件，例如：

```
cp .cofnig_bak .config
```

或者执行 make menuconfig 进入内核配置主界面，并用左右箭头选中功能菜单上的“Load”项，并按回车键，弹出配置文件加载画面，如下图所示：



在该画面中输入需要加载的备份文件的名字，选择“Ok”按钮后按回车键，此时会加载用户选择的备份文件，用户可对其进行修改并保存设置产生新的.config 文件。

在 arch/arm/configs/目录下有很多*_defconfig 文件，这些都是内核预设的配置文件，分别对应各种不同的开发板。如果要使用其中的配置文件作为内核编译配置，可用“make xxx_defconfig”命令来完成。对于已经设定好的内核配置，也可以命名为某个文件名，放到 arch/arm/configs/目录下，在以后直接用 make 来调用该配置即可。芯路遥团队提供的默认内核配置文件为：imx_xly_defconfig，只需执行下列命令即可使用该配置：

```
make imx_xly_defconfig
```

5.2 编译内核

本节介绍如何编译 Linux 操作系统映像文件 zImage。编译 zImage 需要安装好交叉工具链。

在“5-源代码”目录中，包含了一个已经配置好的 Linux 内核源码文件 linux-4.9.88.tar.xz。编译 Linux 操作系统的步骤如下：

1. 将 Linux 内核源码文件 linux.tar.xz 拷贝到 Ubuntu 虚拟机中，此处假定将这两个文件拷贝到/home/xinluyao/work/imx6ull/bsp 中，并对其进行解压：

```
cd /home/xinluyao/work/imx6ull/bsp/
tar xvJf linux.tar.xz
```

2. 清除前一次的编译结果。执行如下命令：

```
cd /home/xinluyao/work/imx6ull/bsp/linux
```

```
make distclean
```

3．使用开发板默认的配置文件生成编译内核时所需要的.config 文件。执行如下命令：

```
make imx_xly_defconfig
```

注：第一次解压一定要配置一次，否则会编译失败。如果已经生成了.config 文件，或已通过 make menuconfig 对配置进行了修改，则可跳过该步骤。

4．使用 make menuconfig 对内核进行配置。执行如下命令：

```
make menuconfig
```

注：如果不需要自行对内核配置进行了修改，则可跳过该步骤。

5．编译内核。执行如下命令：

```
make all
```

编译完成后，会在内核源码顶层目录（即 linux 目录）的 arch/arm/boot 子目录下生成 zImage 文件。

为简化内核编译操作，在内核源码顶层目录下，提供了一些 shell 脚本，简要说明如下：

shell 脚本	说明
build-all.sh	清除以前的编译结果，全部重新编译
build-dtb.sh	编译设备树
build-modules.sh	编译内核模块
build-zImage.sh	编译内核映像
build-menuconfig.sh	使用 menuconfig 配置内核

5.3 内核模块

Linux 内核是一个单内核系统，它的最大有点是效率高，其缺点是可扩展性和可维护性相对较差。内核模块的机制就是为了弥补这一缺点而产生。

模块是具有独立功能的程序，它可以被单独编译，但不能独立运行。它在运行时被链接到内核作为内核的一部分在内核空间运行。Linux 驱动常常以内核模块的形式存在。

5.3.1 最简单内核模块

下面是一个最简单的内核模块（摘自《Linux 设备驱动开发详解》）：

```
1  /*
2   * a simple kernel module: hello
3   *
4   * Licensed under GPLv2 or later.
5   */
6
7  #include <linux/init.h>
8  #include <linux/module.h>
9
10 static int __init hello_init(void)
11 {
12         printk(KERN_INFO "Hello World enter\n");
13
14         return 0;
15 }
16
17
18 static void __exit hello_exit(void)
19 {
20         printk(KERN_INFO "Hello World exit\n");
21 }
22
23 module_init(hello_init);
24 module_exit(hello_exit);
25
26 MODULE_AUTHOR("Barry Song <21cnbao@gmail.com>");
27 MODULE_LICENSE("GPL v2");
28 MODULE_DESCRIPTION("A simple Hello World Module");
29 MODULE_ALIAS("a simplest module");
```

这个最简单的内核模块只包含内核模块加载函数、卸载函数和对 GPL v2 许可权限的声明以及一些描述信息。在加载内核模块时会执行 module_init 中声明的函数，即 hello_init。卸载内核模块时会执行 module_exit 声明的函数，即 hello_exit。

5.3.2 编译内核模块

编译内核模块需要使用交叉工具链并使用 Makefile 进行编译。下面是一个简单的 Makefile。

```
KERN_DIR := ${HOME}/work/imx6ull/bsp/linux-imx6ull-4.19
CURDIR := ${shell pwd}

# Kernel modules
obj-m := hello.o

build: kernel_modules

kernel_modules:
	${MAKE} -C ${KERN_DIR} M=${CURDIR} modules

clean:
	${MAKE} -C ${KERN_DIR} M=${CURDIR} clean
```

其中 KERN_DIR 需要设置为存放内核源代码的目录，由于我们编译的模块源代码为 hello.c，则 obj-m 值设置为 hello.o。

编写好 Makefile 文件后，直接执行 make，即可编译内核模块，生成 hello.ko 文件：

```
xinluyao@ubuntu:~/work/imx6ull/driver/001_hello$ make
make -C /home/xinluyao/work/imx6ull/bsp/linux-imx6ull-4.19 M=/home/xinluyao/work/imx6ull/driver/001_hello modules
make[1]: 进入目录“/home/xinluyao/work/imx6ull/bsp/linux-imx6ull-4.19”
  CC [M]  /home/xinluyao/work/imx6ull/driver/001_hello/hello.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/xinluyao/work/imx6ull/driver/001_hello/hello.mod.o
  LD [M]  /home/xinluyao/work/imx6ull/driver/001_hello/hello.ko
make[1]: 离开目录“/home/xinluyao/work/imx6ull/bsp/linux-imx6ull-4.19”
```

如需清理编译结果重新编译，可执行“make clean”，效果如下：

```
xinluyao@ubuntu:~/work/imx6ull/driver/001_hello$ make clean
make -C /home/xinluyao/work/imx6ull/bsp/linux-imx6ull-4.19 M=/home/xinluyao/work/imx6ull/driver/001_hello clean
make[1]: 进入目录“/home/xinluyao/work/imx6ull/bsp/linux-imx6ull-4.19”
  CLEAN   /home/xinluyao/work/imx6ull/driver/001_hello/.tmp_versions
  CLEAN   /home/xinluyao/work/imx6ull/driver/001_hello/Module.symvers
make[1]: 离开目录“/home/xinluyao/work/imx6ull/bsp/linux-imx6ull-4.19”
```

这里仅对内核模块进行简单介绍，详细驱动开发可参考相关教程。

5.4 动态加载模块

5.4.1 insmod

insmod 命令动态加载模块。例如需要加载/lib/modules/$(uname -r)/hello.ko 模块，在命令行下执行如下命令：

```
insmod /lib/modules/$(uname -r)/hello
```

5.4.2 modprobe

modprobe 可载入指定的个别模块，或是载入一组相依的模块。modprobe 会根据 depmod 所产生的相依关系，决定要载入哪些模块。若在载入过程中发生错误，在 modprobe 会卸载整组的模块。所以更推荐使用 modprobe 来替代 insmod。

如需要使用 modprobe 安装，需要先将模块放入/lib/modules/$(uname -r)目录，并先执行 depmod。

```
depmod
modprobe hello
```

```
root@xly-imx6ull:~# depmod
root@xly-imx6ull:~# modprobe hello
[ 5343.663259] Hello World enter
```

5.4.3 lsmod

lsmod 命令查看当前系统已经加载的模块。在命令行下执行如下命令：

```
lsmod
```

5.4.4 rmmod

rmmod 命令删除已经加载的模块。例如需要删除已经加载的 hello.ko 模块，在命令行下执行如下命令：

```
rmmod hello
```

6 文件系统

6.1 文件系统介绍

从文件组织结构上来说，嵌入式 Linux 文件系统与普通 PC/服务器 Linux 的文件系统是一样的，只是嵌入式 Linux 文件系统根据产品功能进行过裁剪，在内容多少和体积大小上不同。进行嵌入式 Linux 产品开发，构建一个合适的文件系统是不可或缺的，可以基于已有文件系统进行裁剪或者定制，也可以从头开始构建。

6.1.1 根文件系统目录结构

Linux 文件系统是树状结构的，其根即为“/”。

/bin

存放一些可执行文件，如常见的 ls、cp 等命令就是存放在此

/sbin

存放系统命令，如 modprobe、hwclock 等，这些命令大多涉及系统管理命令。

/dev

设备文件存储目录，应用程序通过对这些文件的读写和控制以访问实际的设备。

/etc

系统配置文件

/lib

系统库文件存放目录

/mnt

这个目录一般是用于存放挂载存储设备的目录。

/opt

opt 是“可选”的意思，主要存放一些用户级的程序。一般安装到/opt 的程序，其库文件等都一起放在/opt 下

/proc

操作系统运行时，进程以及内核信息(比如 CPU，硬盘分区，内存信息等)存放在这里。proc 并不是真正的文件系统，它存在于内存中。

/tmp

用于运行程序的临时文件存放于此。

/usr

usr 不是 user 的缩写，user 其实是 Unix Software Resource 的缩写，也就是 Unix 操作系统软件资源所放置的目录，而不是用户的数据，所有系统默认的软件都会放置到/usr

/var

var 表是变化的意思，这个目录主要针对常变动的文件，比如 cache、logfile 等文件

/sys

sys 目录用于映射 sysfs 文件系统。Linux 设备驱动模型中的总线，驱动和设备都可以在 sysfs 文件系统中找到对应的节点。当内核检测到在系统中出现了新设备后，内核会在 sysfs 文件系统中为该设备生成一项新的记录。

6.1.2 根文件系统类型

如果文件系统已经布局完成，就可以将其发布到目标系统中了。通常会制作成一个镜像文件，然后通过某种方式固化到目标系统中，具体采用什么样的形式发布，需要根据系统资源状况、内核情况和系统需求等方面进行裁决：

- 硬件方面：需要考虑主存储介质的类型和大小，如 Flash 是 NAND Flash 还是 eMMC，RAM 的大小等；
- 内核方面：需考虑所裁剪后的内核支持哪些文件系统，采用哪种文件系统最合适，能满足性能、速度等要求；
- 系统需求方面：需要考虑运行速度、是否可写、是否压缩等方面因素。

常见的可用于根文件系统的文件系统类型有 ramdisk、jffs2、ubifs 和 ext3 等，各个文件系统的特点如下表所示：

类型	介质	是否压缩	是否可写	掉电保存	存在于 RAM 中
ramdisk	-	是	是	否	是
jffs2	NOR Flash	是	是	是	否
yaffs/yaffs2	NAND Flash	否	是	是	否
ubifs	NAND Flash	是	是	是	否
ext4	eMMC Flash	否	是	是	否

6.1.2.1 JFFS/JFFS2

Journalling Flash File System（闪存设备日志型文件系统，JFFS）是由瑞典的 AxisCommunication AB 为嵌入式设备开发的文件系统。

JFFS2 是 JFFS 的后继者，由 Red Hat 重新改写而成，全名为 Journalling Flash File System Version 2 （闪存日志型文件系统第 2 版），其功能就是管理在 MTD （Memory Technology Device）设备上实现的日志型文件系统。

JFFS2 直接在 MTD 设备上实现日志结构的文件系统。 JFFS2 会在安装的时候，扫描 MTD 设备的日志内容，并在 RAM 中重新建立文件系统结构本身，所以启动时间依赖于文件系统大小，时间通常比较长。

JFFS2 还实现了 MTD 设备的“损耗平衡”和“数据压缩”等特性。

JFFS2 最初只支持 NOR Flash，后来也增加了 NAND Flash 支持，但是在 NAND Flash 上的表现不好，不推荐在 Nand Flash 上使用 jffs2。

6.1.2.2 YAFFS/YAFFS2

YAFFS（Yet Another Flash File System）是由 Aleph One 公司开发的，第一个在 GPL 协议下发布的、基于日志的、专门为 NAND Flash 存储器设计的、适用于大容量的存储设备的嵌入式文件系统。

YAFFS 是基于日志的文件系统，提供磨损平衡和掉电恢复的健壮性。它还为大容量的 Flash 芯片做了很好的调整，针对启动时间和 RAM 的使用做了优化。它适用于大容量的存储设备，已经在 Linux 和 WinCE 商业产品中使用。

YAFFS2 和 YAFFS 主要差异在于页面读写尺寸的大小，YAFFS2 可支持到 2K 页面，远高于 YAFFS 的 512 字节，因此对大容量 NAND Flash 更具优势。另外， YAFFS2 不再写 NAND Flash 的 Spare Area，seque

nceNumber 用 29 bits 表示。

6.1.2.3 UBIFS

UBIFS 文件系统(Unsorted Block Image File System, UBIFS)是用于固态硬盘存储设备上，并与 LogFS 相互竞争，作为 JFFS2 的后继文件系统之一。真正开始于 2007 年，并于 2008 年 10 月第一次加入 Linux 2.6.27 稳定版内核。

UBIFS 最早在 2006 年由 IBM 与 Nokia 的工程师 Thomas Gleixner、Artem Bityutskiy 所设计，专门为了解决 MTD 设备所遇到的瓶颈。由于 NAND Flash 容量暴涨，YAFFS 等都无法在有效管理 NAND Flash 的空间。UBIFS 通过 UBI 子系统处理与 MTD 设备之间的动作。与 JFFS2 一样，UBIFS 建构于 MTD 设备之上，因而与一般的块设备不兼容。

UBIFS 在设计与性能上均较 YAFFS2、JFFS2 更适合 MLC NAND Flash。例如：UBIFS 支持回写（write-back），写入的数据会被缓存，直到有必要写入时才写到 NAND，大大降低分散小区块数量并提高 I/O 效率。UBIFS 文件系统目录存储在 Flash 上，UBIFS 挂载时不需要扫描整个 FLASH 的数据来重建文件目录，所以启动速度很快。

另外，UBIFS 支持文件数据压缩，而且可选择性压缩部份文件，UBIFS 也是日志型文件系统，使用日志可减少对 Flash 的更新频率。

6.1.2.4 EXT3

EXT3 是第三代扩展文件系统(Third extended filesystem)，它是一个日志文件系统。适用于硬件、eMMC、SD 卡这样的 Block Device，而不适合 NAND 这样的 Flash 设备，是 eMMC 设备上最常见的根文件系统。

6.2 设置开机自启动程序

6.2.1 rc.local

对于 HD-IMX6ULL-MB，启动脚本为/etc/rc.local 文件，因此如果某个程序或者脚本需要开机自动运行，只需将该程序或脚本添加到/etc/rc.local 即可。例如需要开机自动运行/home/demo/ledtest 程序，可以在/etc/rc.local 文件的最后一行"exit 0"前添加如下内容：

```
/home/demo/ledtest &
```

其中最后的"&"表示程序在后台运行。

6.2.2 systemd

systemd 是一个 init 程序，用来替代 System V 初始进程。虽然早期 systemd 出现时存在一些争议，但现在被越来越多发行版接受，比如我们熟知的 Ubuntu 就已经默认使用 systemd。

对于嵌入式环境来说，systemd 有一个最大的好处，就是实现了系统初始化服务的并行启动，可以加快启动速度。

HD-IMX6ULL-MB 在开机时会启动一个 Qt 桌面程序，就是使用 systemd 进行管理，其命令如下：

6.2.2.1 启动 qtdemo

```
systemctl start qtdemo
```

稍等片刻，便会启动 qtdemo。

6.2.2.2 关闭 qtdemo

```
systemctl stop qtdemo
```

稍等片刻，便会启动 qtdemo。关闭后，屏幕显示内容会停留在关闭时的样子。

6.2.2.3 关闭 qtdemo 开机自启动

```
systemctl disable qtdemo
```

6.2.2.4 开启 qtdemo 开机自启动

```
systemctl enable qtdemo
```

7 系统更新

7.1 eMMC 分区

一般 emmc 在出厂时已经进行了物理分区，分为 user 分区、boot0 分区、boot1 分区和 rpmb 分区，对于普通用户来说，我们可以忽略 boot0、boo1 和 rpmb 分区，直接对 user 分区进行逻辑分区，将 uboot、内核和根文件系统存放于 user 分区中。

对于 8G eMMC user 分区如下：

分区	分区名	地址范围(单位：扇区)	分区 ID/类型	类型	用途
		0 - 2047	无	无	U-Boot
p1	boot	2048 - 133119	0x0C	FAT32	存放 dtb、zImage 等文件
p2	rootfs	133120 -	0x83	Linux	根文件系统

7.2 烧写系统映像

可使用 MfgTools 烧写工具通过开发板的 USB 烧写系统映像。MfgTools 烧写工具位于“9-烧录工具_镜像/mtgtool”目录下。相关的系统映像文件位于“mfgtool/Profiles/Linux/OS Firmware/files”目录，用户可以根据实际需要替换为自己的映像文件。HD-IMX6ULL-MB 目录中各个文件的说明如下：

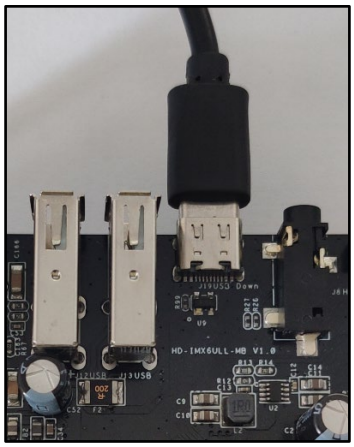
文件名	描述
u-boot-imx6ull-ddr512-emmc.imx	U-Boot 映像文件
zImage	内核文件
imx6ull-xly-emmc-lcd.dtb	设备树文件
rootfs.tar.bz2	根文件系统

下面介绍怎样烧写镜像到开发板：

（1）关闭 HD-IMX6ULL-MB 的电源开关；

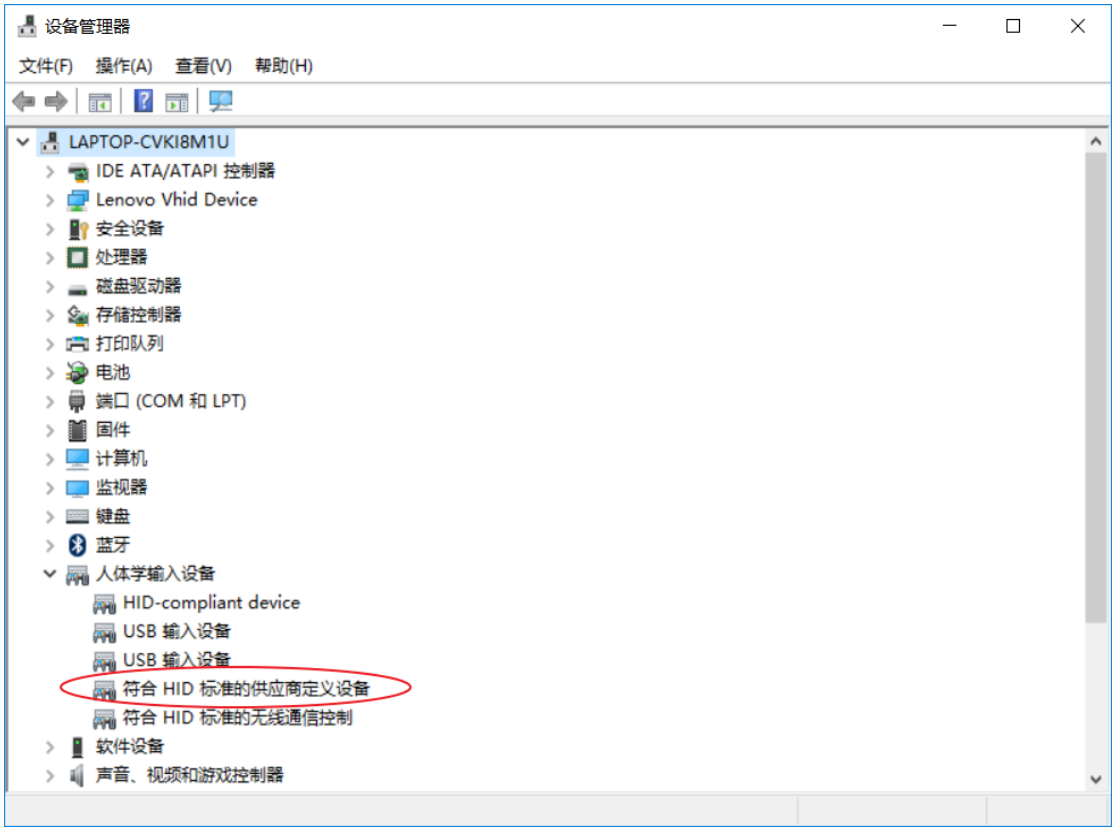
（2）把 HD-IMX6ULL-MB 开发板设置为 Upgrade eMMC 启动方式，详见启动选择。

（3）使用 USB 线缆将 HD-IMX6ULL-MB 开发板的 USB 端口与计算机的 USB 端口相连；



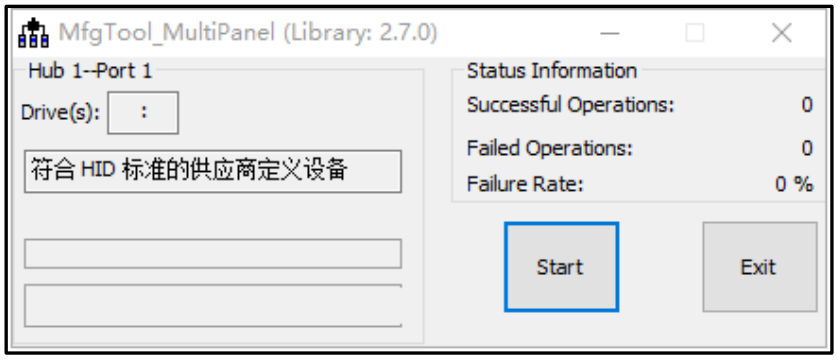
（4）重新给 HD-IMX6ULL-MB 开发板上电。

注：若上述 4 个步骤操作正确，则 HD-IMX6ULL-MB 重新上电之后，计算机应该能够检测到新的设备接入，并且可以在“设备管理器”中看到新的项目“人体学输入设备/符合 HID 标准的供应商定义设备”，如图所示：

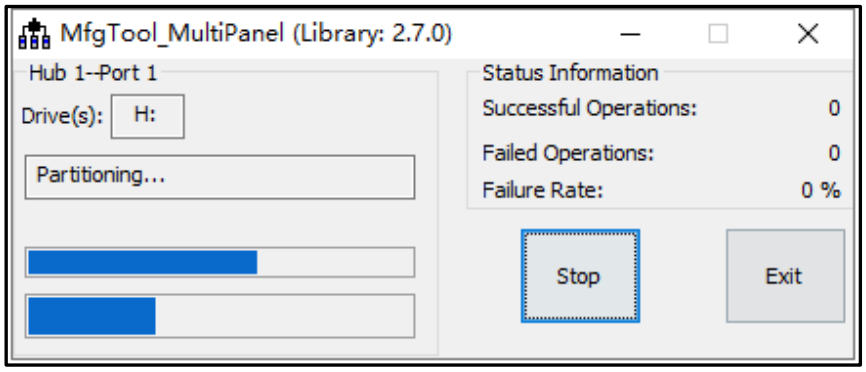


（5）使用 MtgTools 软件进行烧写

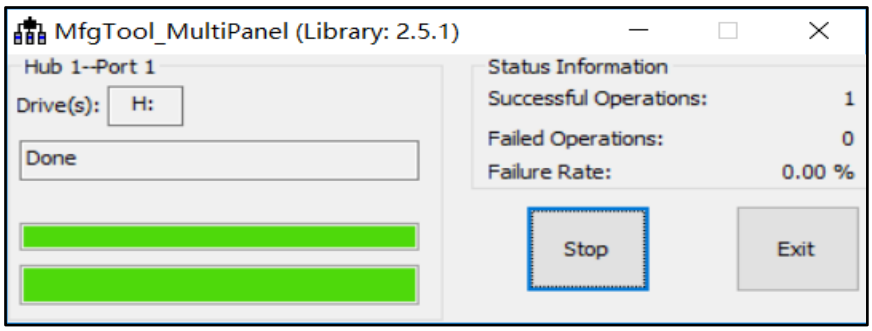
双击“Mfgtool2-eMMC-ddr512-eMMC.vbs”运行 mfgtools 文件夹中的 MfgTool2.exe 软件，打开 Mfgtool 软件的初始界面，界面显示正在监听“符合 HID 标准的供应商定义设备”，如图所示：



点击 Start 按钮开始进行固件的烧写工作，烧写过程如下图所示：



在烧写过程中，可以在调试串口上看到烧写过程。待烧写完毕后，如下图所示：



点击“Stop”按钮后，再点击“Exit”按钮退出软件。

接着断开 HD-IMX6ULL-MB 开发板供电电源，并设置为 eMMC 启动启动方式。重启开发板，即可从 eMMC 启动系统。

7.3 单独更新 zImage

本节介绍在操作系统命令行下，单独更新（zImage）的步骤：

1） 将操作系统映像文件 zImage 拷贝到开发板任意目录下，笔者使用~/Downloads 目录。

2） 进入到 mmcblk1p1 分区：

```
cd /run/media/mmcblk1p1
```

3） 将操作系统映像拷贝到 eMMC 中：

```
cp ~/Downloads/zImage /run/media/mmcblk1p1
```

4） 确保文件写入 MMC

```
sync
```

5） 重启系统

7.4 单独更新 DTB

本节介绍在操作系统命令行下，单独更新设备树文件 DTB 的步骤：

1） 将 dtb 拷贝到开发板任意目录下，笔者使用~/Downloads 目录：

2） 进入到 mmcblk1p1 分区：

```
cd /run/media/mmcblk1p1
```

3） 将操作系统映像拷贝到 eMMC 中：

```
cp ~/Downloads/imx6ull-xly-emmc-lcd.dtb /run/media/mmcblk1p1
```

4） 确保文件写入 MMC

```
sync
```

5） 重启系统

7.5 单独更新内核模块

本节介绍在操作系统命令行下，单独更新内核模块到开发板的步骤：

1） 使用脚本编译内核模块：

```
./build-modules.sh
```

编译完成后，会在内核源代码的下的 modules_install 目录下生成 modules.tar.bz2 文件。

2） 将 modules.tar.bz2 拷贝到开发板任意目录下，笔者使用~/Downloads 目录：

3) 拷贝 modules 到开发板/lib/modules:

```
cd /lib/modules
cp ~/Downloads/modules.tar.bz2 /lib/modules
```

4) 删除之前的内核模块:

```
rm 4.9.88 -rf
```

5) 解压新的内核模块:

```
tar xvjf modules.tar.bz2
```

6) 确保文件写入 MMC

```
sync
```

7) 重启系统

8 联系我们

武汉芯路遥科技有限公司

公司地址：武汉东湖新技术开发区大学园路长城园路 8 号海容基孵化园 B 栋 5 楼 503

售后邮箱： support@xinluyao.xyz

wiki 地址： www.xinluyao.wiki

公司网址： www.xinluyao.xyz

武汉万象奥科电子有限公司

公司地址：武汉东湖新技术开发区大学园路长城园路 8 号海容基孵化园 B 栋 5 楼 503

公司电话：027-59218026

公司邮箱：sales@vanxoak.com

售后邮箱：support@vanxoak.com

公司网址：www.vanxoak.com